Cyber Threat Intelligence
Analysis of adversaries and their methods

Griffioen, H.J.

DOI
10.4233/uuid:37f7367f-bc5e-4cde-a7fd-47d12621f853

Publication date
2022

Document Version
Final published version

Citation (APA)
Griffioen, H. J. (2022). *Cyber Threat Intelligence: Analysis of adversaries and their methods*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:37f7367f-bc5e-4cde-a7fd-47d12621f853

Cyber Threat Intelligence

Analysis of adversaries and their methods

Cyber Threat Intelligence

Analysis of adversaries and their methods

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
vrijdag 30 september 2022 om 12:30 uur

door

Harm Jonathan Griffioen

Ingenieur in de Informatica, Technische Universiteit Delft, Nederland
geboren te Gouda, Nederland.

Dit proefschrift is goedgekeurd door de promotors.

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. R. L. Lagendijk	Technische Universiteit Delft, promotor
Prof. dr. C. Doerr	Hasso Plattner Institut Potsdam, promotor
Onafhankelijke leden:	
Prof. dr. C. Meinel	Hasso Plattner Institut Potsdam
Prof. dr. ir. H. J. Bos	Vrije Universiteit Amsterdam
Prof. dr. M. J. G. van Eeten	Technische Universiteit Delft
Prof. dr.-ing. M. Fischer	Universität Hamburg
dr. F. A. Oliehoek	Technische Universiteit Delft

Keywords: Cyber Threat Intelligence, Network Security, Internet Measurement

Printed by: IPSKAMP printing

Front & Back: Hannah Griffioen

ISBN 978-94-6384-369-0

Een elektronische versie van dit proefschrift is beschikbaar op
http://repository.tudelft.nl/.

They say nothing is impossible, but I do nothing every day.

Winnie the Pooh

CONTENTS

SUMMARY

The growing dependency on interconnected devices makes cyber crime increasingly lucrative. Together with the rise of premade tools to perform exploits, the number of cyber incidents grows rapidly each year. Defending against these threats becomes increasingly difficult as organizations depend heavily on the Internet and have many different connected devices, all with their own protocols and vulnerabilities. The rise in cyber crime and plethora of devices make it difficult for organizations to detect and mitigate all attacks targeting their business.

Cyber Threat Intelligence (CTI) provides defenders with information about cyber threats and thus the ability to scope the defensive efforts towards the areas with the highest risk of damages. This information comes in different forms, from lists if indicators that are direcly ingestible into the defensive infrastructure of a company to documents describing the Tactics, Techniques and Procedures (TTPs) of adversaries.

A major challenge in CTI is identifying indicators that describe more abstract features of adversaries, such as the tools that are used, to automatically detect mitigation attempts in defensive infrastructure. Furthermore, the identification of adversarial campaigns remains challenging, but the analysis on the campaigns that are identified proves to provide valuable information about actor capabilities and the threat landscape.

In this thesis, we focus on improving CTI by getting a better understanding of adversarial behavior and evolution. We first create metrics to measure the quality of CTI feeds and address some measurement bias in network-based measurements. To obtain better understanding of adversaries we focus on tool fingerprinting, adversarial evolution and campaign analysis.

We find a surprising lack of sophistication and evolution of adversaries. But we also find that the quality of CTI feeds is poor with on average a response time of 21 days before an indicator is added to a feed after it is active. We show that by fingerprinting adversarial tools and performing campaign analysis on individual attacks, we can learn the sophistication of adversaries and obtain a better understanding of the threat landscape. In addition, following attacker campaigns over time allows us to better understand the evolution of actors and their objectives. To allow for this campaign analysis in DDoS attacks, we introduce a new model to describe attacks and cluster these on behavior. Finally, we utilize adversarial TTPs to devise a method to disrupt malware propagation and evaluate this method on a real-world botnet.

Samenvatting

De groeiende afhankelijkheid van onderling verbonden apparaten maakt cybercriminaliteit steeds lucratiever. Samen met de opkomst van kant-en-klare tools om exploits uit te voeren, groeit het aantal cyberincidenten elk jaar snel. Verdediging tegen deze bedreigingen wordt steeds moeilijker omdat organisaties sterk afhankelijk zijn van internet en veel verschillende aangesloten apparaten hebben, allemaal met hun eigen protocollen en kwetsbaarheden. De toename van cybercriminaliteit en de overvloed aan apparaten maken het voor organisaties moeilijk om alle aanvallen op hun bedrijf te detecteren en de gevolgen te beperken.

Cyber Threat Intelligence (CTI) biedt verdedigers informatie over cyberdreigingen en daarmee de mogelijkheid om de defensieve inspanningen te richten op de gebieden met het hoogste risico op schade. Deze informatie komt in verschillende vormen, van indicatoren in lijsten die direct opneembaar zijn in de defensieve infrastructuur van een bedrijf tot documenten die de tactieken, technieken en procedures (TTP's) van tegenstanders beschrijven.

Een grote uitdaging in CTI is het identificeren van indicatoren die meer abstracte kenmerken van tegenstanders beschrijven, zoals de tools die worden gebruikt om automatisch mitigatiepogingen in defensieve infrastructuur te detecteren. Bovendien blijft het identificeren van vijandige campagnes een uitdaging, terwijl de analyse van de geïdentificeerde campagnes blijkt waardevolle informatie te verschaffen over de capaciteiten van actoren en het dreigingslandschap.

In dit proefschrift richten we ons op het verbeteren van CTI om een beter begrip te krijgen van vijandig gedrag en evolutie. We maken eerst meetbare statistieken om de kwaliteit van CTI-feeds te begrijpen en pakken meetafwijkingen aan in netwerkgebaseerde metingen. Om een beter begrip van tegenstanders te krijgen, richten we ons op het herkennen van tools, vijandige evolutie en campagne-analyse.

We vinden een verrassend gebrek aan verfijning en evolutie van tegenstanders. Maar we vinden ook dat de kwaliteit van CTI-feeds slecht is met een responstijd van gemiddeld 21 dagen voordat een indicator wordt toegevoegd aan een feed nadat deze actief is. We laten zien dat door het identificeren van fingerprints van vijandige tools en het uitvoeren van campagneanalyses van individuele aanvallen, we de verfijning van tegenstanders kunnen leren en een beter begrip kunnen krijgen van het dreigingslandschap. Bovendien stelt het volgen van aanvallerscampagnes ons in de loop van de tijd in staat om de evolutie van actoren en hun doelstellingen beter te begrijpen. Om deze campagne-analyse bij DDoS-aanvallen mogelijk te maken, introduceren we een nieuw model om aanvallen te beschrijven en te clusteren op gedrag. Ten slotte gebruiken we vijandige TTP's om een methode te bedenken om de verspreiding van malware te verstoren en testen deze methode op een real-world botnet.

1

Introduction

In 1986, the first computer virus hit the newly introduced MS-DOS and spread from system to system using floppy disks [1]. Because the Internet did not yet exist, the virus could only infect a machine when a floppy disk containing the malicious code was inserted into a device. With the inception of the Internet, computers became more interconnected, removing the need for some physical action to infect another system. The Morris worm was one of the first pieces of malicious code taking advantage of the Internet to spread the infection, reportedly infecting 10% of the entire Internet at the time. With the increasing number of systems being connected to the Internet, estimated at 42 billion in 2022 [2], the potential for malicious software to spread has increased dramatically.

Over time, the motivation behind malicious activities that target digital systems changed from being for 'fun' to profit. With the scale of the current Internet, organizations conduct malicious activity like a business, with many criminals working together to extort victims for money, use infected systems to 'mine' cryptocurrencies, or use any other of many ways to make money from infecting other systems. By 2022, cyber business interruptions are in the top three top global business risks [3] and damages from cybercrime are expected to grow even further to 10.5 trillion USD by 2025 [4]. Additionally, cyber warfare is becoming a more prominent issue as critical systems could be disrupted through cyberattacks [5].

Defending against the increasing number of cyber threats is not a trivial task. To mount an effective defense, one needs information about these attackers, their capabilities, commonly used techniques, and how to defend against them. This so-called Cyber Threat Intelligence is an essential part of cyber security in organizations, as it helps identify and subsequently mitigate threats.

In this chapter, we introduce the concept of CTI that forms the basis of this thesis. Section 1.1 introduces Cyber Threat Intelligence, the types of intelligence, and the essential concepts this thesis addresses. Section 1.2 identifies the key challenges in the current state-of-the-art, which are addressed in this thesis. The problem statement and research questions this thesis addresses are discussed in section 1.3. Finally, section 1.4

lists the contributions and outline of this thesis.

1.1. CYBER THREAT INTELLIGENCE

Cyber Threat Intelligence (CTI) consists of information that enables an organization to deploy a proactive and timely response to cyber threats. CTI comes in many forms: from PDF documents detailing a specific threat to lists of identifiers such as IP addresses that automatically load in defensive infrastructure such as a firewall, all the information about a cyber threat is essentially CTI. While the main use of CTI is to identify and mitigate threats, there are three forms of CTI that can together improve the defense of organizations when they are all sufficiently addressed [6]:

1. **Strategic**: Focuses on the *who* and *why* behind threats. Understanding *why* the organization is vulnerable, and *who* would be interested in performing an attack allows executives to make management decisions minimizing the attack surface or incentive.

2. **Operational**: Identifies the *how* and *where*, with the aim of identifying the way in which adversaries will attack. As it is hard to catch malicious actors when a defender does not know where to look, this type of CTI focuses the attention on where an adversary is most likely to attack.

3. **Tactical**: Addresses the *what* question. Tactical CTI identifies Indicators of Compromise (IOCs), including IP addresses, file hashes, domain names, and other actionable information that can be used in defensive infrastructure to detect and mitigate threats.

In broad terms, strategic CTI identifies the threat landscape, operational CTI focuses on the Tactics, Techniques, and Procedures (TTPs) of adversaries, and tactical CTI allows the identification of threats. A successful deployment of CTI depends on all three uses. In the remainder of this section, we discuss several concepts and models that are at the core of CTI.

CYBER KILL CHAIN

In cyber security, attackers have the upper hand on defenders, as the attempts from an attacker to perform an exploit only have to succeed once. On the other hand, the defenders have to repel every incoming attack. Unlike in regular crime, where a failed robbery could quickly lead to the arrest of a suspect and therefore mitigation of future crimes, cybercriminals are much harder to track down, and they can try again. However, a successful attack does not consist of a single action but rather a chain of activities leading to a system's compromisation. Lockheed Martin developed the Cyber Kill Chain model to provide defenders with better proactive defense mechanisms [7]. The model contains the chain of activities that adversaries must go through before a system is successfully compromised. Figure 1.1 shows this model. Instead of treating an attack as a single data point, defenders can now identify an attack in any of the stages and subsequently stop the entire attack. The first stage in the chain is *Reconnaissance*, for which the adversary gathers the information required to perform a successful attack. For example, where are

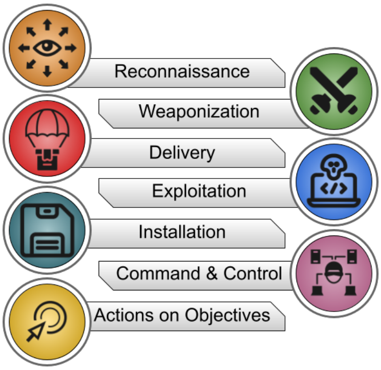


Figure 1.1: The Cyber Kill Chain as introduced by Lockheed Martin [7]. Adversaries have to traverse all stages for a successful attack.

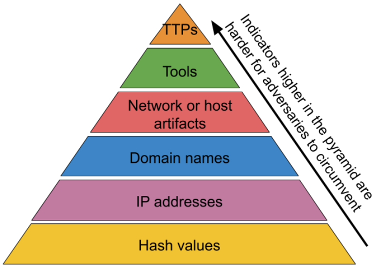

TTPs
Tools
Network or host artifacts
Domain names
IP addresses
Hash values
Indicators higher in the pyramid are harder for adversaries to circumvent

Figure 1.2: The pyramid of pain showing different levels of indicators. The higher the indicator is listed in the pyramid, the harder it is for an adversary to avoid detection from defenses that are based on this indicator.

the company's servers located, or which employees could be sent an email containing a malicious file. In the *Weaponization* stage, the adversary tests an exploit and creates a deliverable payload. The adversary places the payload in the *Delivery* stage. In the *Exploit* stage, the payload is triggered, and the adversary can execute code on the victim's system. After obtaining the initial foothold, an adversary can install malware on the asset in the *Installation* stage. A command channel can be opened to the adversary in the *Command & Control* stage to allow the attacker to perform arbitrary commands. After all these stages, an adversary can finally perform the *Actions On Objectives* to achieve the goal of the entire exploit.

The Cyber Kill Chain model can be oxymoronic for securing a system: stopping the adversary as quickly as possible in this chain would minimize the potential damage the adversary can inflict in the current attack, but catching an adversary later in the chain will provide more information about infection methods and the vulnerabilities. It might help in mitigating more attacks in the future. By gathering enough data about the stages in this chain, defenders can cluster separate attacks into attack campaigns [8] to also provide valuable threat intelligence and learn about the motives behind the attacks.

Pyramid of Pain

Indicators of Compromise (IOCs) may consist of many different identifiers, ranging from file hashes to specific behaviors known to be associated with a specific adversary. These IOCs are not equally valuable for detecting emerging threats, as some may be easier for the adversary to change to prevent detection. For example, a file hash can be trivially modified, while a specific tool that the adversary uses is much harder to change for every attack. The pyramid of pain classifies different levels of IOCs in terms of how hard it is for an adversary to change [9]. Figure 1.2 shows the pyramid. The pyramid consists of several levels of defensice capabilities, and the higher defenders get in the pyramid, the harder it will be for adversaries to adapt and avoid detection based on the indicators used by the defenders. On the top of the pyramid, we find adversaries' Tactics, Techniques, and Procedures (TTPs), which refer to *how* these specific cybercriminals operate their exploits and their specific goals. To change their TTPs would require adversaries to completely overhaul their operations, which is infeasible to perform for every attack. Fighting cybercrime is largely a game of economics. The higher the level of indicators in the pyramid of pain we achieve, the more effort an attacker must make before successful exploitation. This means that adversaries will be priced out of the system if defenses are built on the indicators from a high enough layer in the pyramid.

THREAT INTELLIGENCE FEEDS

Organizations do not solely rely on the CTI that they gather themselves. If an organization uses solely their own data, it is much harder to respond to threats that might already have attacked other organizations. Instead, organizations ingest threat information from many sources to provide the best view of the threat landscape. A common way to share CTI is via IOC lists, mainly consisting of unordered lists of “known” indicators observed by measurement infrastructure or organizations [10]. Some of these lists are available for free. In contrast, others come at a hefty premium from commercial intelligence vendors with large measurement infrastructures and many analysts to provide detailed information about threats observed in their systems. When discovering a new threat, an automatically ingested CTI feed can deliver IOCs in real-time and mitigate future attacks using these indicators as soon as possible. The information shared on feeds ranges from lists of simple IOCs such as IP addresses and domain names that can be automatically ingested into security appliances to PDF documents describing specific threat actors and their TTPs.

ACTOR TYPES

Not all actors have the same knowledge and resources to use in their attacks. Understanding how these actors behave and what their goals are is an essential aspect of CTI. Improving the understanding of threat actors and their behavior is integral to threat assessments and the prioritization of security controls [11]. Actor types vary from inexperienced attackers that use off-the-shelf tools to actors backed by nations [12]. Classifying actors that attack a system is often tricky, as many actors have similar methods depending on the level of detail with which actors are classified. For example, Timothy Casey identifies 22 different actor types based on eight attributes [13]. With this level of detail, it is hard to distinguish between ‘Radical Activists’ and ‘Sensationalists’ as their profiles are nearly identical. Other taxonomies rely on the types of attacks to classify an actor, assuming that actors only perform one type of attack [14].

To provide the necessary understanding of actor types that might perform cyberattacks, we use a classification of actors in five general types based on their intent, such as provided in various ontologies [12, 15]. These five types are a generalization of the types in related works and broadly define adversaries:

- **Script kiddie** - An actor that has limited knowledge and resources and thus mainly uses off-the-shelf tools. This type of actor is mainly in it ‘for fun’ and will not always have a monetary interest.

- **Criminals** - These actors range from individuals to full-fledged business-like groups that operate malware or extort people. The intent of these actors is monetary.

- **Hacktivists and terrorists** - These actors want to make political statements and disrupt the functioning of businesses or even countries. This actor type intends to show people their beliefs.
- **Insiders** - Disgruntled employees, corporate spies, or employees that make critical errors of oversights that are inside an organization. The intent of these actors can range from simple mistakes to corporate espionage or sabotage.
- **State sponsored actors** - The most sophisticated and persistent actor is state-sponsored. These actors will have custom tools and exploits and do not have a monetary incentive. Instead, they will be interested in espionage or undermining other countries.

As mentioned before, these five actor types are broadly defined, and some detailed actor types are not included.

Campaign analysis

Campaign analysis aims to discover clusters of activities of malicious actors. The term was first introduced from the analysis of SPAM [16] but is increasingly used for other cyber threats [17]. The cluster of malicious activity originating from an actor is called a ‘campaign’, and by tracking this cluster, we better understand the adversary and their TTPs. There is no standard way to perform this analysis, as it is hard to cluster attacks into single actor campaigns. However, the information obtained by doing so provides excellent insights into actor motivation and their TTPs, making it a significant source of Strategic threat intelligence.

1.2. Challenges in Cyber Threat Intelligence

As CTI is such a broad field, there are many challenges that can be solved to improve the state-of-the-art of CTI. This section will list the major challenges we see in CTI in the near future. This thesis addresses challenges involving CTI quality, identifying new IoCs, measuring TTPs, and campaign analysis. This does not mean that the challenges not addressed by this work such as CTI sharing are of lesser importance. It is vital to improve all factors of CTI to mount a more effective defense, and in this thesis we set a first step in improving CTI in various directions.

Measuring CTI quality

Depending on external CTI as a data source to prevent attacks can be challenging, as there needs to be some level of trust towards the data supplier to provide good quality data that is usable in practice. However, the understanding of CTI feed quality often ends with a quantification of the alerts generated by a feed and whether this is feasible for analysts to address [18]. Trust in CTI feeds should instead be based on the how well indicators can be used to defend against emerging threats and whether they can be used in practice and do not give too many false-positives. This is a complex question, as quality metrics for CTI do not yet exist, and thus the overall quality of CTI feeds is unknown.

Attack indicators

While CTI proves to be a helpful tool to identify malicious behaviors, adversaries are evolving to avoid detection by CTI information. With the increases in the number, variety, and severity of cyberattacks, it becomes increasingly important to recognize IOCs and extract information about attack methods [19]. On the other hand, it becomes increasingly crucial for adversaries to change their attack patterns to avoid detection. When defenders climb the pyramid of pain, it will be harder for adversaries to adapt to the defenses, but climbing the pyramid is also increasingly complex for defenders [20].

At the bottom of the pyramid, hash Values, IP addresses, and Domain Names are easy to identify and block by a defender. We can create lists and ingest them directly into some security apparatus from these values [21]. For this purpose, sharing taxonomies exist that provide a common ontology for sharing information [22]. For the higher layers of the pyramid, we need to investigate how to measure and describe indicators such that they can be used to identify threats.

Describing TTPs

When going up in the pyramid, information becomes increasingly abstract and thus harder to describe in a way that can be automatically ingested in security appliances. For the top level of the pyramid, TTPs, there is currently no method describing the IOC in a machine-readable format [23]. Furthermore, for the description of tools, there is a need for research in identifying and describing these tools in a way that is automatically ingestible and hard for an adversary to circumvent. Ideally, we would automatically identify and characterize adversarial tools and TTPs, and create machine-readable signatures that mitigate future attacks from the same adversary.

Campaign analysis

Another considerable challenge in CTI is the identification of attack *campaigns*, a group of attacks performed by the same actor or group that share behavioral traits [24]. The identification of these campaigns will provide *strategic* CTI and help identify the intent and capabilities of adversaries [25]. This information increases the understanding of the current threat landscape, but the identification of campaigns has proven to be a challenging problem. Adversaries can change part of their attack chain, making it hard to identify whether two attacks originate from the same actor. Some studies analyze campaigns and gain valuable insights into adversarial behavior [16, 17, 26–28], but this is not widely deployed as it requires many data points to cluster attacks based on their entire attack chain. As adversaries can attack multiple organizations, they would have to work together to identify attack clusters and subsequently analyze campaigns. However, even when all the data is available, clustering campaigns is still not trivial, and more research is needed to understand campaigns and what we can learn from them.

CTI Models

CTI models do not apply to all attack forms. The Cyber Kill Chain is, for example, mainly focused on attacks targeting devices in an organization, but not always on other threats. To this end, many alterations of the kill chain model exist that focus on a specific type of threat, such as social engineering [29], IoT attacks [30], industrial control systems [31, 32], and even niche subjects such as multimedia service environments [33]. Tailoring the CTI models to specific attacks does not only allow for campaign analysis on these threats, but it will also lead to better situational awareness frameworks and attack ontologies that build on these models [34–38]. A major attack vector that does not yet have a model that identifies the stages an adversary has to go through is the DDoS attack [39]. With the increase in DDoS attacks and the introduction of DDoS as a Service by using so-called booter services [40], there is an increasing need to understand the steps behind these attacks better.

CTI Dissimination

Even when CTI is available, it does not mean that attackers can no longer use the attack vector. In the case of EternalBlue [41], an exploit developed by the NSA and leaked in April 2017, was successfully exploited a month later by the WannaCry ransomware because many systems were not yet updated even though patches were already available for a month [42, 43]. The WannaCry ransomware attack cost nearly 4 billion USD in damages, according to Symantec [44], which could, for a large part, have been prevented as the information about the exploit was widely known for a month. The exploit was later also used as part of the NotPetya attack in June [45], again infecting thousands of machines where administrators neglected to deploy the security patches available for over two months. These events are not only occurring in systems owned by small businesses or consumers. In 2016, the US democratic committee discovered a breach in their systems. On inspection of the IP address used to perform the attack, it was discovered that simply alerting on IP addresses that were known to perform attacks like this would have identified the breach much earlier [46]. The dissemination of CTI and its usage in organizations can thus be improved.

CTI sharing

Learning about adversaries should be a community effort. If an attacker targets an organization, it is useful to share indicators and TTPs with other organizations that might target the same actor. There are many opportunities for sharing CTI information between organizations, but also many challenges such as the privacy of organizations [47]. While there are platforms that allow for CTI sharing [48, 49], the adoption of these methods remains low [18]. New methods propose to share machine learning models instead of indicators, providing a better privacy-preserving method for sharing IOCs and allowing for the detection of threats at a higher level in the pyramid of pain [50]. Despite the current sharing models, sharing CTI remains an important challenge, both technical and non-technical.

1.3. Problem Statement

This thesis aims to better understand adversarial behavior by monitoring cyber threats high up in the pyramid of pain and performing in-depth campaign analysis. We investigate adversarial tools, TTPs, and the evolution of actors. While there are many tools and attack vectors that this thesis does not consider, we focus on improving CTI by adding to the understanding of TTPs and campaign analysis as these topics are not explored thoroughly in other works. For instance, there are no metrics for measuring the quality of CTI feeds and thus no general knowledge about the efficacy of these feeds. Furthermore, existing works in campaign analysis generally group different actors into the same threat as there is no behavioral classification leading to the identification of a campaign. How threat actors evolve and the differences between threat actors performing various malicious activities such as building a botnet or sending a DDoS attack is still largely unknown. Related works only discuss the sophistication of adversaries to a limited extent. Research on campaign analysis shows that the Cyber Kill Chain is usable to cluster singular attacks, but the lack of such a model for other types of attacks such as DDoS limits the behavioral analysis for these types of attacks.

Considering the research gaps of the existing literature on CTI, TTPs, and campaign analysis, this thesis will aim to answer to following:

To what extent can we increase the quality of CTI by learning about adversaries and how can we use this information to create better defenses?

This question is split into several subquestions that focus on specific parts of the pyramid of pain for different attack vectors:

- *How can we measure the quality of Cyber Threat Intelligence feeds?*
- *How do actors differ in their TTPs, and how can we better measure these TTPs?*
- *What is the feasibility of using information about adversarial tactics to disrupt malicious activities?*

1.4. CONTRIBUTION OF THE THESIS

This thesis analyzes the current state-of-the-art of CTI, identifies attack campaigns and investigates the TTPs of actors in DDoS attacks and those operating a botnet. It provides solutions for improving this information by understanding what biases can occur in this data, what actor sophistication can change in measurements, and how actors evolve. In all chapters, we rely on real-world data collections that are larger than similar studies and therefore provide a better picture of the online threats. Justification for this claim is provided in the individual chapters. The contributions of this thesis are the following:

- We provide the first in-depth study on the quality of Cyber Threat Intelligence feeds. The metrics discussed in **Chapter 2** are, to the best of our knowledge, the first published metrics that defenders can use to measure the quality of these feeds.
- Our proposal in **Chapter 3** to measure Internet-wide IP churn using botnet traffic is, to the best of our knowledge, the first method that can be used to *passively* measure IP churn in networks. Additionally, this method can identify a part of the NATs in a network using the same passive measurements. The understanding of IP churn in networks can help improve the quality of network-based studies and ultimately improve the understanding of Internet threats.

- To the best of our knowledge, **Chapter 4** provides the first campaign analysis of many SIP scanners. It introduces a novel fingerprinting method to identify scanners targeting SIP using only a single packet.

- We propose a new CTI model to address the shortcomings of existing models with respect to DDoS attacks, detailing attack phases an adversary has to carry out. Using this model, we show in **Chapter 5** that we can behaviorally classify DDoS attacks and perform campaign analysis.

- By deploying a large honeynet, we are the first to identify the different behaviors of adversaries carrying out DDoS attacks to the best of our knowledge. In **Chapter 5** we provide insight into the sophistication of various adversaries and show that there is a small number of highly sophisticated actors that go to great lengths to maximize their attack efficacy.

- We introduce a novel way to use structural weaknesses in the Random Number Generator of the Mirai botnet to perform better measurements on the lifetime of infections in **Chapter 6**.

- We provide the most extensive honeypot study of the Mirai botnet to date and show how different strains of this malware compete against each other. We show in **Chapter 6** that adversaries specialize towards specific victim groups to gain a competitive advantage and observe the competition to copy their good ideas.

- **Chapter 6** provides the first epidemiological quantification of the Mirai botnet and shows that the infection is not self-sustaining. Adversaries thus constantly have to spread infections to keep their botnet from dying out.

- We show that adversaries have a deliberate effort to circumvent detection from specific network ranges. Large public monitoring infrastructures are generally avoided entirely to avoid measurements about the extent of the threat.

- We are the first to test the effects of a network tarpit at scale in a real-world setting and show that this can be an effective way of stopping a botnet from spreading. To the best of our knowledge, our simulator in **Chapter 7** is the first experimentally verified tarpit simulator that estimates the result of a tarpit operation.

1.4.1. Outline

The thesis is structured as follows:

Chapter 2
State of Cyber Threat Intelligence

Improving CTI is only possible if the field's current state is known and the quality of the shared information is measurable. Therefore, in Chapter 2, we introduce and evaluate a set of metrics with which the quality of CTI feeds is measurable. Afterwards, we provide an evaluation of 17 open source CTI feeds in terms of *timeliness*, *Sensitivity*, *Originality*, and *Impact*. We find that the current state of these CTI feeds is poor for all four metrics, as information is often shared late, feeds do not show a complete picture of the threats, and blocking some IP addresses can have a large impact on the number of domains that are unreachable due to the block. We conclude with a discussion about the adoption of CTI and the scope of the malicious activities listed in these feeds. This chapter has been published as *"Quality Evaluation of Cyber Threat Intelligence Feeds"* by **Griffioen, H.**, Booij, T. and Doerr, C. in the International Conference on Applied Cryptography and Network Security (2020).

CHAPTER 3

ADDRESSING BIASES IN MEASUREMENTS OF INTERNET DATA

Measuring the quality of CTI, and performing Internet measurements in general, can be challenging due to the volatility of IP address leases and the presence of Network Address Translation (NAT). Measuring some phenomenon based on IP addresses could, therefore, lead to significant bias, as changing IP addresses will lead to an over-estimation of the threat. In contrast, the presence of a NAT will lead to an under-estimation. Chapter 3 introduces a novel way to measure the churn rate of network ranges with subnet accuracy. Using this method, we can identify which IP ranges of an ISP are used for enterprise customers that require static IP addresses and which ranges are dedicated for personal use. Additionally, we quantify the measurement bias that will occur when measuring, for example, a botnet by using the IP addresses. This chapter has been published as *"Quantifying Autonomous System IP Churn using Attack Traffic of Botnets"* by **Griffioen, H.**, Doerr, C. in the International Conference on Availability, Reliability, and Security (2020).

CHAPTER 4

FINGERPRINTING ADVERSARIAL TOOLS

CTI becomes more useful for a defender if the type of IOC is located high up in the pyramid of pain. Additionally, understanding adversarial campaigns will further improve CTI information as it shows adversaries' evolution and activities. Chapter 4 introduces a method to identify and fingerprint SIP scanning tools by only looking at a single packet. We identify 21 tools used for scanning for open SIP servers in 5 years of Internet traffic and show only limited evolution of actors who scan for SIP servers. Using behavioral traits of the scanners, we can cluster individual scanners into large scanning campaigns. By identifying campaigns, we show that actors scan for SIP periodically and that they, over time, lower the number of packets sent by a scanner to avoid detection. This chapter has been published as *"SIP Bruteforcing in the Wild - An Assessment of Adversaries, Techniques and Tools"* by **Griffioen, H.**, Hu, H. Doerr, C. in the IFIP Networking Conference (2021).

CHAPTER 5

UNDERSTANDING ADVERSARIAL TTPS

The Cyber Kill Chain is used to describe adversarial steps that lead to a successful exploit, but it does not apply to all cyber threats. For DDoS attacks, the model does not work as much of the activity leading up to a successful attack takes place outside of the organization. While a model that fits DDoS might thus not help with defending against this threat, it does allow for the analysis of attack campaigns. It provides valuable information about the sophistication of adversaries. In Chapter 5 we show a new model that identifies the steps an adversary has to take before a successful DDoS attack. Using this model, we characterize real-world DDoS attacks and find a high variance in attacker sophistication, targets, and TTPs. Because the model captures the behavioral traits of the attackers, we can identify and analyze attack campaigns and analyze adversarial TTPs. This chapter has been published as *"Scan, Test, Execute: Adversarial Tactics in Amplification DDoS Attacks"* by **Griffioen, H.**, Oosthoek, K., van der Knaap, P.C.F., Doerr, C. in the ACM SIGSAC Conference on Computer and Communications Security (2021).

Chapter 6

Understanding the evolution of actors

The Mirai botnet hit the Internet like a storm in 2016 and launched the (at the time) heaviest DDoS attack ever recorded. After its inception and media attention, the author shared the source code for this botnet on a public forum for anyone to download, allowing others to create and deploy a similar botnet. Over the years, many different botnets emerged that mimicked the Mirai botnet, and all of them are competing for the same devices. Chapter 6 monitors the different botnets to identify what actors are doing to obtain a high number of bots from the devices that everyone is competing over. We find clear evolution of actors that specialize their botnet towards certain ISPs or Geographical regions and even skip large parts of the Internet to focus on a smaller number of IP addresses. This chapter has been published as *"Examining Mirai's Battle over the Internet of Things"* by **Griffioen, H.**, Doerr, C. in the ACM SIGSAC Conference on Computer and Communications Security (2020).

Chapter 7

Taking down malicious activity

Knowing how an adversary spreads its infections allows defenders to look for and block the malicious activity. Chapter 7 identifies a 'tarpit' vulnerability in the scanning method deployed by many botnets, allowing defenders to not only detect and mitigate exploitation attempts but also to occupy the botnet such that others will not be exploited as well. We evaluate the effectiveness of such measures on the Mirai botnet and show that we can cut exploitation attempts in half in some cases. Parts of this chapter will be published as *"Could you clean up the Internet with a Pit of Tar? Investigating tarpit feasibility on the Mirai malware"* by **Griffioen, H.**, Doerr, C in the IEEE Symposium on Security and Privacy (2023).

Chapter 8

Discussion

Chapter 8 concludes this work and discusses the research questions in detail. We also look forward to future work and provide ideas and research gaps that we have learned from the works included in this thesis.

1.4.2. List of Excluded Publications

In the following, we list the papers that have been published during this Ph.D. but are not included in this work because they are out of scope for the main line of this thesis.

1. Ghiette, V., **Griffioen, H.**, Doerr, C., *Fingerprinting tooling used for SSH compromisation attempts*, 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019).

2. **Griffioen, H.**, Doerr, C., *Taxonomy and adversarial strategies of random subdomain attacks*, 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS 2019).

3. Bouwman, X., **Griffioen, H.**, Egbers, J., Doerr, C., Klievink, B., van Eeten, M., *A different cup of TI? The added value of commercial threat intelligence*, 29th USENIX Security Symposium (USENIX Security 2020).

4. **Griffioen, H.**, Doerr, C., *Discovering Collaboration: Unveiling Slow, Distributed Scanners based on Common Header Field Patterns*, IEEE/IFIP Network Operations and Management Symposium (NOMS 2020).

5. **Griffioen, H.**, Doerr, C., *Quantifying TCP SYN DDoS resilience: A longitudinal study of Internet services*, IFIP Networking Conference (IFIP Networking 2020).

6. Taniguchi, T., **Griffioen, H.**, Doerr, C., *Analysis and Takeover of the Bitcoin-Coordinated Pony Malware*, ACM Asia Conference on Computer and Communications Security (ACM AsiaCCS 2021).

1.4.3. About the Thesis

This thesis consists of integral copies of publications divided into several technical chapters. All chapters are based on one publication with only minor changes. The original article's title is included on the first page of every chapter. Because these are integral copies of published works, some parts might overlap between chapters, such as the introduction, background, or datasets used.

References

[1] *Brain-virus: 25e verjaardag van een computervirus,* https://archive.is/20130703125140/http://www.govcert.nl/actueel/Nieuws/brainvirus.html.

[2] *Iot devices worldwide,* https://www-statista-com.tudelft.idm.oclc.org/statistics/471264/iot-number-of-connected-devices-worldwide/.

[3] *Top business risks,* https://www.allianz.com/en/press/news/studies/220118_Allianz-Risk-Barometer-2022.html.

[4] *Cyber security almanac 2022,* https://cybersecurityventures.com/cybersecurity-almanac-2022/.

[5] M. Robinson, K. Jones, and H. Janicke, *Cyber warfare: Issues and challenges,* Computers & security **49**, 70 (2015).

[6] A. Dehghantanha, M. Conti, T. Dargahi, *et al., Cyber threat intelligence* (Springer, 2018).

[7] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, *Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,* Leading Issues in Information Warfare & Security Research (2011).

[8] F. Falconieri, *Approximate automated campaign analysis with density based clustering,* (2018).

[9] *The pyramid of pain,* https://detect-respond.blogspot.com/2013/03/the-pyramid-ofpain.html.

[10] M. Bromiley, *Threat intelligence: What it is, and how to use it effectively,* SANS Institute InfoSec Reading Room **15**, 172 (2016).

[11] E. C. working group, *Enisa threat landscape 2021,* (2021).

[12] J. Andress and S. Winterfeld, *Cyber warfare: techniques, tactics and tools for security practitioners* (Elsevier, 2013).

[13] T. Casey, *Threat agent library helps identify information security risks,* Intel White Paper **2** (2007).

[14] M. d. Bruijne, M. v. Eeten, C. H. Gañán, and W. Pieters, *Towards a new cyber threat actor typology,* (2017).

[15] A. Stankovska *et al., Cyber threat actors and cyber threat management,* Entrepreneurship **4**, 174 (2016).

[16] S. Dinh, T. Azeb, F. Fortin, D. Mouheb, and M. Debbabi, *Spam campaign detection, analysis, and investigation,* Digital Investigation **12**, S12 (2015).

[17] H. L. Bijmans, T. M. Booij, and C. Doerr, *Inadvertently making cyber criminals rich: A comprehensive study of cryptojacking campaigns at internet scale,* in *28th USENIX Security Symposium (USENIX Security 19)* (2019) pp. 1627–1644.

[18] X. Bouwman, H. Griffioen, J. Egbers, C. Doerr, B. Klievink, and M. Van Eeten, *A different cup of ti? the added value of commercial threat intelligence,* in *29th USENIX Security Symposium (USENIX Security 20)* (2020) pp. 433–450.

[19] M. Conti, T. Dargahi, and A. Dehghantanha, *Cyber threat intelligence: challenges and opportunities,* in *Cyber Threat Intelligence* (Springer, 2018) pp. 1–6.

[20] K. Oosthoek and C. Doerr, *Cyber threat intelligence: A product without a process?* International Journal of Intelligence and CounterIntelligence **34**, 300 (2021).

[21] M. Kührer, C. Rossow, and T. Holz, *Paint it black: Evaluating the effectiveness of malware blacklists,* in *International Workshop on Recent Advances in Intrusion Detection* (Springer, 2014) pp. 1–21.

[22] S. Barnum, *Standardizing cyber threat intelligence information with the structured threat information expression (stix),* Mitre Corporation **11**, 1 (2012).

[23] V. Mavroeidis and S. Bromander, *Cyber threat intelligence model: an evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence,* in *2017 European Intelligence and Security Informatics Conference (EISIC)* (IEEE, 2017) pp. 91–98.

[24] F. Skopik and T. Pahi, *Under false flag: Using technical artifacts for cyber attack attribution,* Cybersecurity **3**, 1 (2020).

[25] N. Pitropakis, E. Panaousis, A. Giannakoulias, G. Kalpakis, R. D. Rodriguez, and P. Sarigiannidis, *An enhanced cyber attack attribution framework,* in *International Conference on Trust and Privacy in Digital Business* (Springer, 2018) pp. 213–228.

[26] H. S. Lallie, L. A. Shepherd, J. R. Nurse, A. Erola, G. Epiphaniou, C. Maple, and X. Bellekens, *Cyber security in the age of covid-19: A timeline and analysis of cybercrime and cyber-attacks during the pandemic,* Computers & Security **105**, 102248 (2021).

[27] D. U. Case, *Analysis of the cyber attack on the ukrainian power grid,* Electricity Information Sharing and Analysis Center (E-ISAC) **388**, 1 (2016).

[28] R. Ottis, *Analysis of the 2007 cyber attacks against estonia from the information warfare perspective,* in *Proceedings of the 7th European Conference on Information Warfare* (Academic Plymouth, 2008) p. 163.

[29] K. Shin, K. M. Kim, and J. Lee, *A study on the concept of social engineering cyber kill chain for social engineering based cyber operations,* Journal of The Korea Institute of Information Security & Cryptology **28**, 1247 (2018).

[30] J. Haseeb, M. Mansoori, and I. Welch, *A measurement study of iot-based attacks using iot kill chain,* in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)* (IEEE, 2020) pp. 557–567.

[31] M. J. Assante and R. M. Lee, *The industrial control system cyber kill chain,* SANS Institute InfoSec Reading Room **1** (2015).

[32] X. Zhou, Z. Xu, L. Wang, K. Chen, C. Chen, and W. Zhang, *Kill chain for industrial control system,* in *MATEC Web of Conferences*, Vol. 173 (EDP Sciences, 2018) p. 01013.

[33] H. Kim, H. Kwon, and K. K. Kim, *Modified cyber kill chain model for multimedia service environments,* Multimedia Tools and Applications **78**, 3153 (2019).

[34] Y. Wang, T. Zhang, and Q. Ye, *Situation awareness framework for industrial control system based on cyber kill chain,* in *MATEC Web of Conferences*, Vol. 336 (EDP Sciences, 2021) p. 02013.

[35] S. Cho, I. Han, H. Jeong, J. Kim, S. Koo, H. Oh, and M. Park, *Cyber kill chain based threat taxonomy and its application on cyber common operational picture,* in *2018 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA)* (IEEE, 2018) pp. 1–8.

[36] M. Mohsin and Z. Anwar, *Where to kill the cyber kill-chain: An ontology-driven framework for iot security analytics,* in *2016 International Conference on Frontiers of Information Technology (FIT)* (IEEE, 2016) pp. 23–28.

[37] B. D. Bryant and H. Saiedian, *A novel kill-chain framework for remote security log analysis with siem software,* computers & security **67**, 198 (2017).

[38] A. Hahn, R. K. Thomas, I. Lozano, and A. Cardenas, *A multi-layered and kill-chain based security analysis framework for cyber-physical systems,* International Journal of Critical Infrastructure Protection **11**, 39 (2015).

[39] A. Srivastava, B. Gupta, A. Tyagi, A. Sharma, and A. Mishra, *A recent survey on ddos attacks and defense mechanisms,* in *International Conference on Parallel Distributed Computing Technologies and Applications* (Springer, 2011) pp. 570–580.

[40] J. J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Z. Granville, and A. Pras, *Booters—An analysis of DDoS-as-a-service attacks,* in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (IEEE, 2015).

[41] H. Kettani and P. Wainwright, *On the top threats to cyber systems,* in *2019 IEEE 2nd international conference on information and computer technologies (ICICT)* (IEEE, 2019) pp. 175–179.

[42] T. Stier and J. Greve, *An analysis of wannacry and eternalblue,* (2019).

[43] K. Da-Yu, S.-C. Hsiao, and T. Raylin, *Analyzing wannacry ransomware considering the weapons and exploits,* in *2019 21st International Conference on Advanced Communication Technology (ICACT)* (IEEE, 2019) pp. 1098–1107.

[44] *Wannacry malware damages,* https://symantec-enterprise-blogs.security.com/blogs/feature-stories/wannacry-lessons-learned-1-year-later.

[45] A. Greenberg, *The untold story of notpetya, the most devastating cyberattack in history,* Wired, August **22** (2018).

[46] C. Doerr and K. Oosthoek, *Six cti challenges and their solutions - reaching cti's full potential,* https://www.youtube.com/watch?v=EIkjpl7XOZE.

[47] T. D. Wagner, K. Mahbub, E. Palomar, and A. E. Abdallah, *Cyber threat intelligence sharing: Survey and research directions,* Computers & Security **87**, 101589 (2019).

[48] D. W. Chadwick, W. Fan, G. Costantino, R. De Lemos, F. Di Cerbo, I. Herwono, M. Manea, P. Mori, A. Sajjad, and X.-S. Wang, *A cloud-edge based data security architecture for sharing and analysing cyber threat information,* Future Generation Computer Systems **102**, 710 (2020).

[49] D. Homan, I. Shiel, and C. Thorpe, *A new network model for cyber threat intelligence sharing using blockchain technology,* in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (IEEE, 2019) pp. 1–6.

[50] D. Preuveneers and W. Joosen, *Sharing machine learning models as indicators of compromise for cyber threat intelligence,* Journal of Cybersecurity and Privacy **1**, 140 (2021).

2

State of Cyber Threat Intelligence

To mount an effective defense, information about likely adversaries and their techniques, tactics, and procedures is needed. This so-called cyber threat intelligence *helps an organization to understand its threat profile better. Next to this understanding, specialized feeds of indicators about these threats downloaded into a firewall or intrusion detection system allow for a timely reaction to emerging threats.*

These feeds, however, only provide an actual benefit if they are of high quality. In other words, if they provide relevant, complete information in a timely manner. Incorrect and incomplete information may even cause harm, for example, if it leads an organization to block legitimate clients or if the information is too unspecific and results in an excessive amount of collateral damage.

This paper evaluates the quality of 17 open source cyber threat intelligence feeds for 14 months and 7 additional feeds over 7 months. Our analysis shows that most indicators are active for at least 20 days before they are listed. Additionally, we have found that many lists have biases towards certain countries. Finally, we also show that blocking listed IP addresses can yield large amounts of collateral damage.

This chapter has been published as *"Quality Evaluation of Cyber Threat Intelligence Feeds"* by **Griffioen, H.**, Booij, T. and Doerr, C. in the International Conference on Applied Cryptography and Network Security (2020).

2.1. Introduction

To effectively protect a system, one needs information. The most notable is the need for details on possible attackers, their capabilities, commonly used tactics, techniques, and feasible countermeasures. This information can be gathered by a company itself or obtained from providers specialized in providing this information. At first glance, the idea of gaining a head start and an upper hand to combat the activities of malicious adversaries seems like an oxymoron. After all, if an intelligence provider distributes information about the activities, used tools or domains, and IP addresses from which attacks appear to a broad public, adversaries would immediately recognize having been uncovered and correspondingly change their activities, negating the benefits a defender would have from this information in the first place. The use of cyber threat intelligence is thus a race against the clock, where published data is prone to lose its value soon. This means that the distribution of information about malicious activities as quickly as possible is vital. While it is in principle possible for organizations to assemble and grow such a body of knowledge, most shy away from the associated costs and complexity in building such *cyber threat intelligence* (CTI) themselves, but instead turn to commercial and open-source threat intelligence providers for help.

In the very recent past, this shift to intelligence-driven defense has led to the emergence of a plethora of companies providing CTI feeds, often at a heavy price tag. Threat intelligence feeds range from complex sitreps, sector analysis, and trend reports in essay formats to machine-readable lists of indicators of compromise such as traffic signatures or malicious IP addresses that organizations can download and install into their firewall, thereby catering to the entire spectrum of organizational cyber security maturity levels. In addition to such for-profit services, a variety of open-source alternatives have sprung up, provided by security companies as a marketing instrument or driven by a community effort of aggregating information across defenders. While commercial feeds may add unique results, for example, from an internal forensic investigation, many commercial providers have been known to repackage, curate and resell other (open source) lists [1].

By definition, cyber threat intelligence is, however, a highly perishable good, since as soon as it is discovered and distributed to clients, adversaries will also know that one of their tools or assets has been found and would try to replace this “burnt” artifact as soon as possible. Thus, to be effective, threat intelligence has to be timely and highly accurate. With inaccurate listings, automatic download and application of indicator information could lead to undesired effects such as blocking traffic from benign clients. Overall, this collateral damage may do more harm than good, which leads to the question of how effective these feeds are.

In this paper, we aim to answer the question *How effective are open source Cyber Threat Intelligence feeds, and how can we measure their quality?* To do this, we create suitable metrics to evaluate the quality of 24 open source cyber threat intelligence feeds and estimate the utility and risk each of these services provides to an organization. With our work, we make the following three contributions:

- We introduce a taxonomy to evaluate the quality of cyber threat intelligence feeds to assess the utility the user may gain from such a feed.

- We evaluate the indicators reported on 24 open source threat intelligence feeds across four dimensions and benchmark using NetFlow data and zone transfers these feeds' timeliness, sensitivity, originality, and impact.

- We empirically analyze the impact a listing of an indicator on an intelligence feed has on its activity after that. This allows us to evaluate the adoption of these feeds in practice and estimate whether a feed can "save" clients and networks from future harm.

The remainder of this paper is structured as follows: Section 2.2 discusses existing work into cyber threat intelligence and its evaluation. In Section 2.3, we develop evaluation criteria for a quality assessment of cyber threat intelligence that we will use within this paper. Section 2.4 describes open source intelligence feeds collected for the analysis, while Section 2.5 describes their utility in terms of relevance, timeliness, completeness and accuracy. Section 2.6 evaluates the adoption of these sources in practice and the benefit they bring to networks. Section 2.7 summarizes our findings and concludes our work.

2.2. Related work

Cyber threat intelligence feeds are widely used in industry and are a valuable tool to mitigate attacks. Despite significant commercial interest in these feeds, initial surveys indicate that the quality of these feeds might not be as high as one would like. In a study in 2015 [2], authors state that most intelligence was not specific enough. Additionally, 66% of respondents note that the information is not timely.

On the same note, Tounsi et al. [3] state that there are still many limitations when it comes to threat intelligence. One of the limitations is that there is too much information, with 250 million indicators per day. Another finding in this paper is that threat intelligence available to enterprises is often out of date due to the short lifespan, and therefore not always helpful. Limitations of blocklists are also apparent in [4], in which the authors show the incompleteness of the evaluated blacklists. To further measure the shortcomings, several works have been focused on empirical evaluation of these feeds [5–7], as well as test suites with specific goals in testing blocklists [8].

In 2014, Kührer et al. published a paper in which multiple blocklists are empirically analyzed [7]. In this paper, the authors identify the active, parked, and sinkholed domains in several domain lists. The analysis in the paper gives insight into domain blocklists, measuring their accuracy, completeness and estimating the timeliness of the blocklists. As the study's goal was not to create metrics to generalize the evaluation of cyber threat intelligence feeds, no metrics have been created and evaluated in this topic.

A Defcon presentation by Pinto and Maxwell [9] aims to measure the effectiveness of threat intelligence feeds in two dimensions. In this presentation, the authors show evaluations for the scope and accuracy of these feeds. Their research has been further complemented by Pawliǹski and Kompanek [10], which state that there are eight criteria in which the quality of a threat intelligence feed can be measured. However, these metrics are not evaluated on a large number of CTI feeds, and some of these metrics are hard to evaluate automatically.

This paper proposes four quality metrics for CTI feeds that can be automatically analyzed and examines 24 different open source CTI feeds using these metrics.

2.3. Quality Criteria for Threat Intelligence

This section will describe four criteria, which we will use in the following to evaluate the quality of open-source threat intelligence feeds. As discussed in the related work, Pawliǹski and Kompanek [10] have proposed at an industry forum a taxonomy to benchmark threat intelligence along the dimensions of (a) relevance, (b) accuracy, (c) completeness, (d) timeliness, and (e) ingestibility. We find this classification, however, problematic, as several of the criteria are entangled: for example, in the machine learning and pattern recognition domains, relevance is usually measured by precision and recall. In other words, how many of the selected items in a dataset are correctly identified, and how many of the relevant items are found in the dataset, respectively. Recall, however, also partially assesses similar aspects as completeness so that quantification results would contain some degree of correlation. Along the same lines, accuracy is also a widely used concept in machine learning, and in binary classification measures the ratio of accurate results to all examined data, or $\frac{TP+TN}{TP+TN+FP+FN}$. While threat intelligence is a classification task, classifying activity as either malicious or non-malicious, threat intelligence *feeds* are not classification tasks but should mainly contain information from one label. Therefore, a binary accuracy characterization does not work well due to an imbalance of the data present in the feeds. For these reasons, we propose complementary metrics to measure the quality of these feeds.

A Taxonomy for CTI Quality

To evaluate the quality of cyber threat intelligence, we propose a set of four metrics: timeliness, sensitivity, originality, and impact, which we will describe in further detail in the following:

1. *Timeliness.* The goal of subscribing to a threat intelligence feed is to obtain early warning of some emergent, malicious activity so that infections in the local area network can be stopped in time before significant losses are incurred. Hence, the earlier indicators such as IP addresses or domain names are flagged, the higher the utility of the feed is to the subscriber. In turn, we can also conclude the better the quality of the provided information. One essential quality criteria of a threat intelligence feed is thus the timeliness of the information posted, in other words, how soon a domain or IP address is included in such lists after it has started malicious activities. High timeliness will minimize the amount of damage that could be incurred as part of a compromise. It shortens the time window during which hosts may be under adversarial control, and the time an adversary may exfiltrate data or abuse the infected client.

2. *Sensitivity.* To be included in a feed, the threat intelligence provider has to observe some malicious activity in the first place. This is typically done using a variety of sensors, recording network traffic patterns, DNS lookups, and, for example, based on the forensic analysis of malware samples. Suppose a particular malware instance, C&C server, or maliciously acting host shows only low, sporadic activity. In

that case, there is a high likelihood that a provider would not see it and thereby go by unmitigated until the problem grows above a certain threshold.

With sensitivity, we assess what volume is necessary for the intelligence provider to notice malicious activity. In other words, what are the average and typical minimum thresholds at which detection will take place. In addition to quantifying the overall per-feed threshold, we can also measure the sensitivity of a threat intelligence feed with respect to a geographical focus: if a provider predominantly has sensors in a specific region, detection will be biased against threats emerging or deployed in this particular area, while comparatively insensitive towards threats originating outside of the measurement coverage. As Internet threats by definition operate worldwide, heavy geographical biases signal a significant risk of getting hit unprepared.

3. *Originality.* In practice, an organization, would likely subscribe to several threat intelligence feeds, as CTI providers often specialize towards a particular type of threats. We also see this behavior in threat intelligence providers themselves, who – as we have said earlier – are often aggregating, curating, and repacking other sources to be marketed as their own service. An essential metric of a cyber threat intelligence feed is originality, in other words, the amount of information is unique to this particular source and that could not be obtained otherwise.

   While originality measures the contribution made by one specific feed, it can also be used as a metric to quantify an ecosystem of intelligence feeds as a whole. Consider several k feeds, which all report malware C&C servers. Suppose all indicators provided by these feeds are highly unique, in other words. In that case, there is no or only limited overlap between them, this also means that even their union provides only an insufficient peak at the population of C&C servers. We can thus say that in the case of high ecosystem-wide originality, each feed only draws for samples from a large problem space. In these cases, the set of intelligence feeds is unsuited to provide sufficient defense against this particular type of threat.

4. *Impact.* When an organization applies the information obtained from the threat intelligence feeds, this should lead to a mitigation of a particular threat, as connections to and from a malicious host are suppressed, and no command & control activity or an initial infection should happen anymore. Based on this positive impact, applying the threat information can also have negative consequences, especially if the information is not specific enough or contains false positives.

   The former is particularly of concern if feeds only provide IP address information, such as the IP address where a command & control server is currently hosted. While in times of domain-generation algorithms (DGAs) indicators such as domain names have an extremely short lifetime, in many circumstances, an actor will not host malicious infrastructure on a dedicated machine but rather employ the services of commercial vendors as this offer much higher flexibility and incurs no loss (except for the forfeiture of prepaid service) such as the seizure of own hardware. However, this also means that at a particular IP address that is flagged as malicious, other services may be present, which are then blocked as collateral damage.

Our metric impact measures the consequences to an organization if the information from a threat intelligence feed is applied, for example, by blocking IP addresses in the firewall. This can have both positive and negative consequences, and we care whether all of the malicious activity will be suppressed given the feed's data and whether it *only* covers malicious activity or the application will also cause harm to benign services. For example, suppose a malware communicates with its C&C server using 10 IP addresses. In that case, the blockage is only successful and helpful if all ten addresses are included in the feed. Otherwise, the activity continues using an alternative channel, and only these ten addresses are blocked.

2.4. DATASETS

This paper aims to evaluate the quality of cyber threat intelligence feeds, which we will do based on the criteria described in the previous section. For this purpose, we have monitored a total of 24 open source feeds that list domain names and IP addresses based on detected malicious activities, annotated into major categories such as botnet C&C server activity, usage as a phishing domain, etc. These feeds were continuously monitored over seven months from August 1, 2018, until February 28, 2019, and when available, all historical records back until January 1, 2018. This yielded a total of 1,383,040 indicators that we are going to use for this evaluation.

We monitored 17 threat intelligence feeds for our analysis for 14 months and seven feeds for 7 months. In table 2.1, we will briefly enumerate each of the feeds included in this analysis.

To evaluate the available cyber threat intelligence feeds for timeliness, accuracy, completeness, and relevance, we make use of two auxiliary datasets:

- **Active domain crawls** Based on zone transfers on registered domains from ICANN and national domain registries, we have crawled approximately 277 million unique domains across 1151 generic and country-code top-level domains daily. This data shows which IP address was connected to which domain on any given day.

- **NetFlows of a tier 1 operator** To detect whether an IP address is receiving traffic or not and to investigate the response of networks to a blocklisting, and we leverage NetFlow data collected at the backbone of a tier 1 network operator. These NetFlows were recorded at each of the operator's core routers at a sampling rate of 8192:1 and thus allowed the reconstruction of activity towards specific IP addresses. We provided a list of IP addresses flagged as malicious by the threat intelligence feeds and received an anonymized list of IP addresses connected to the suspicious targets. To preserve the privacy of the customers, the IP addresses of the clients were anonymized by the ISP using the technique described in [11] and obfuscated at the level of autonomous systems. This allowed quantifying the activity of malicious endpoints without learning anything about the identity of the actual users.

2.4.1. ANONYMIZATION

To preserve the privacy of users in the NetFlow dataset, the IP addresses of senders and receivers are randomized to mask their identity. While for NetFlow datasets, only a deterministic, random one-to-one mapping of original to anonymized IP addresses is necessary to match outgoing requests with returning answers, in such blind randomization, the relationship information of networks is lost. Thus, it is impossible to preserve information locality information such as a C&C activity realized by several hosts in the same /24 subnet, as these hosts would be scattered across the entire IPv4 space.

This paper uses the method introduced by Xu et al. [12], who introduces a random one-to-one mapping while preserving network information. If we represent network addresses in a binary tree, each bit of the IP address, when read left to right, will result in a transition to the left or right subtree under a node, an IP block under a shared prefix will be expressed as an entire subtree under one specific node. Consider the example in 2.1(a), all IP addresses in the prefix P_1 start with the digits “00’ in their address, while IP addresses in the adjacent address block begin with “01”. Under each leaf node, marked in grey, are all IP addresses associated with this particular IP allocation.

In Xu et al.’s “Cryptography-based Prefix-preserving Anonymization” (Crypto-PAn) [12] scheme, the bit value of every non-leaf node is flipped randomly. This means that if two IP addresses share a k-bit prefix, the anonymized IP addresses will share an identical but now randomized k-bit prefix. Within each netblock, IP addresses can now be scrambled without losing information about the logical coherence of the addresses to one provider, and prefix-preserving anonymization comes in handy for the evaluation of threat intelligence feeds as related activity is often located in adjacent IP addresses or subnet blocks as we will show later. The randomness in Crypto-PAn is drawn from the AES block cipher, and a short encryption key is thus sufficient to provide a practical IP randomization function. The authors prove in [12] that this scheme delivers semantic security.

The anonymization of NetFlows was done on-site of the Tier1 operator using a secret key chosen by the operator. Only obfuscated data were analyzed within the context of this project, thus preserving the identity of Internet users. To match the information on

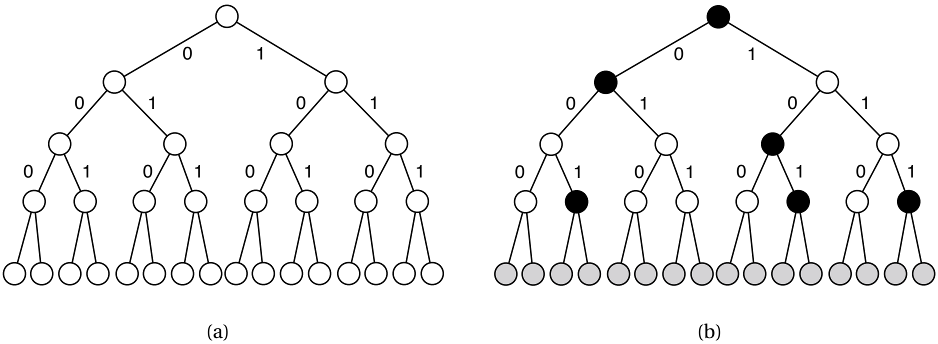


Figure 2.1: Prefix-preserving randomization after Xu et al. [12]

Table 2.1: List of evaluated open source feeds

TI Feed	Automated	Period	Amount of IPs
Badips	Yes	14 months	95
Bambenek	Yes	14 months	1,796
Blocklist.de	Hybrid	14 months	944,622
BotScout Bot List	No	14 months	1,564
Botvrij	No	14 months	95
BruteForceBlocker	No	14 months	4,663
CI Army IP	Hybrid	14 months	181,439
CINSscore	Hybrid	14 months	250,153
Charles the Haleys	No	7 months	38,999
Cruzit	No	7 months	49,911
Danger.rulez	No	7 months	3,099
Dshield	No	14 months	106
Emerging Threats	No	14 months	10,464
Greensnow	No	14 months	116,748
MalwareConfig	Yes	14 months	19
Malwaredomainlist	Yes	14 months	1,011
Myip	No	7 months	55,936
Nothink	Yes	7 months	42
Phishtank	Yes	14 months	2,708
Ransomwaretracker	Hybrid	14 months	383
Rutgers	Yes	14 months	112,898
Talos	Hybrid	7 months	2683
Tech. Blogs and Reports	Yes	14 months	6,151
Zeustracker	Yes	7 months	112

malicious activity from the threat intelligence feeds to the traffic patterns in the NetFlow dataset, the operator additionally provided us with a lookup table of the malicious IP addresses from the feeds to the anonymized counterpart in the dataset to enable the analysis presented in the remainder of this paper.

2.5. Quality evaluation of feeds

Based on the criteria introduced in section 2.3, in this section, we discuss the results of the quality evaluation of the 24 tested cyber threat intelligence feeds. The following subsections will first review their performance in terms of timeliness, sensitivity, originality, and impact before in section 2.6 we will in further detail analyze the question of their overall utility and adoption in practice.

2.5.1. Timeliness

This section assesses the timeliness of cyber threat intelligence feeds based on the amount of traffic a particular destination has received before and after a feed included it.

To visualize the process, figure 2.2 depicts connections within the Tier1 network to

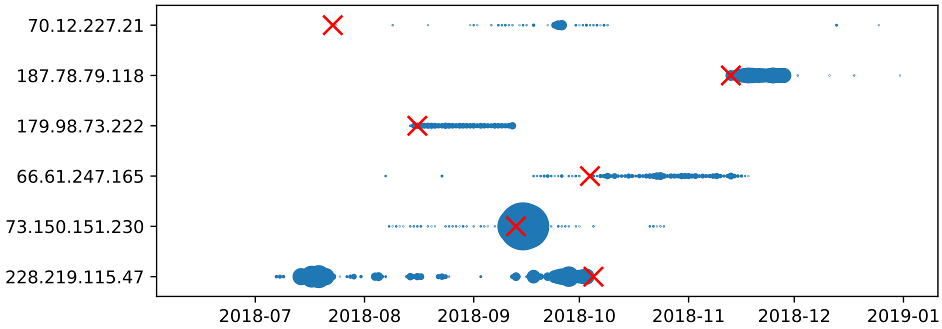


Figure 2.2: Scatter plot of netflow activity. The size of the line shows the amount of traffic observed. Crosses denote when the IP address was blacklisted.

seven example destination IP addresses between July 2018 and January 2019 that was in the second half of 2018 flagged as malicious. For each day, we aggregated flows from distinct clients towards each destination. The size of each circle shows on a logarithmic scale the total number of recorded flows. Note that the IP addresses are anonymized as discussed in section 2.4: while the anonymization protocol matched the feed indicators to the IP addresses, the shown IP addresses are randomized at the level of prefixes. Thus, no conclusion can be taken about the concrete IP addresses at hand or their location in the world.

As we see in the graph, we find that IP addresses and their appearance in intelligence feeds frequently diverges significantly in practice. The first three IP addresses in the figure are examples of a very timely detection – the IP addresses are reported as soon as the first activity arises. In the first case, even months before significant botnet traffic appears towards this C&C server. Not every intelligence report is, however, as successful. In the fourth and fifth cases, the IP addresses are active for several weeks before reporting, and in the case of 73.150.151.230, it is only marked as malicious after a significant traffic volume emerges. An even worse outcome is shown just below in the case of 228.219.115.47. While after the including of the IP address in the threat feeds activity abruptly stops, the IP address had been active for almost three months prior and been engaged in thousands of connections with clients.

While we could mark the onset of a significant number of flows to a flagged IP as the beginning of the illicit activity, this procedure would result in overestimations, for example, when IP addressed legitimately before is reallocated by a provider to a customer that starts to abuse them. However, we can identify the starting point and transition phases such as IP churn or hacked hosts by comparing the sets of client IP addresses that connect to a flagged destination. Intuitively, we exploit here that a server running as a reporting point for ransomware or as a command & control instance for bots or would be contacted regularly by infected clients [13]. In contrast, a regular website would attract a more diverse group of visitors that would connect at unspecific times than the same set of clients coming back in similar regular intervals. In the left part of figure 2.3, there exists a significant overlap of the client IP addresses making contact for the entire period

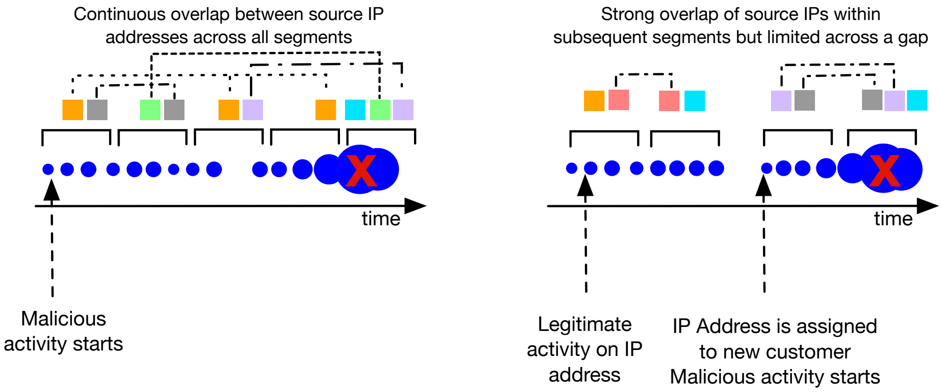

Continuous overlap between source IP addresses across all segments
time
Malicious activity starts
Strong overlap of source IPs within subsequent segments but limited across a gap
time
Legitimate activity on IP address
IP Address is assigned to new customer Malicious activity starts

Figure 2.3: By comparing the set of IP addresses that are connecting to an IP address flagged as malicious, we can approximate the start of the malicious activity.

beginning the start of traffic and the IP being reported malicious. In the right part, we can identify a break in this pattern, with a consecutive sequence of windows showing significant overlap until being marked. In contrast, earlier activity shows only little overlap with this period. When analyzing the IP addresses flagged as malicious by the feeds, we can obtain a conservative lower rather than a loose upper bound for the start of the malicious activity.

We analyzed timeliness for all 1,383,040 indicators across the 24 threat intelligence feeds and counted the number of consecutive days IP addresses had received activity from clients before they were included in a list. Based on this analysis, we find that the first examples of successful indication in figure 2.2 are the exception rather than the norm, surprisingly we find that it takes, on average, 21 days before indicators are included in a list.

Figure 2.4 splits this analysis out, where each curve shows in a probability density function the number of active days until listed by an individual provider. As can be seen in the graph, a handful of feeds are leading the pack, with the response time of CI Army being about 50% better than the overall average. In the bulk of the feeds, we find surprisingly high homogeneity and overall slow inclusion of malicious sources into the feeds, with turnaround times of, on average, approximately one month. Even lists commonly praised by practitioners as “high quality” or “industry standards”, such as the widely used *Emerging Threats* score surprisingly average in this respect. At the lower end of the scale, we have already observed activity for on average 65 to 80 days before the slowest to respond feeds – Talos and PhishTank – include these IP addresses in their reports as malicious.

This lag between the emergence of malicious activity and the inclusion in the threat intelligence feeds might be due to a slow update frequency. To investigate this hypothesis, we analyzed the inter-arrival time when information was included across the 24 feeds, figure 2.5 shows a cumulative density function of the time in between list updates. As we can see from the graph, information is pushed at a very high frequency to the port-

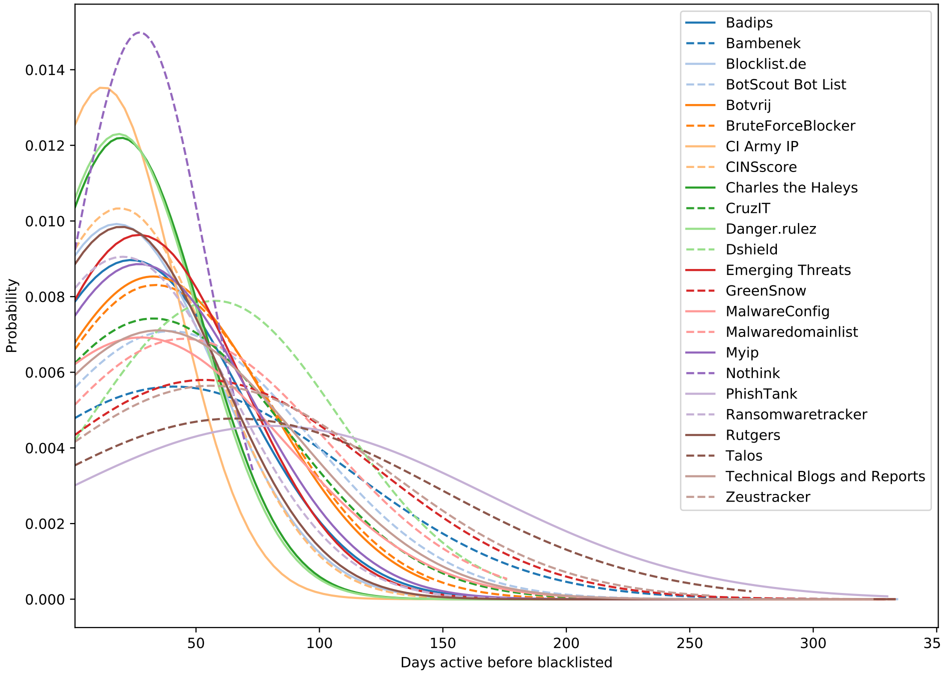


Figure 2.4: PDF of the times it takes a list to blacklist a host after it became active. From the plot can be seen that hosts are routinely active for multiple weeks before a destination is marked as malicious by threat intelligence feeds.

folio of lists. In one-third of the cases, updates occur at least hourly, while approximately two-thirds of items are updated at a granularity of at least once per day. Thus, it is not the processing of the lists where reporting latency occurs but during the selection and preparation of indicators.

2.5.2. SENSITIVITY

As we have already seen above, there seems to be a significant deviation between feeds in how soon indicators are included after the first sign of network activity. Figure 2.6 lists a cumulative density function of the number of connections we observe before an address is included in a particular intelligence feed as malicious. While most lists are surprisingly homogeneous in their sensitivity – we see that the bulk of them triggers with 50% likelihood if at least 6*8192 - 10*8192 flows are recorded –, also here many drastic outliers emerge. Shield's sensitivity is across the board 1.5 - 2 orders of magnitude lower than the rest, here indicators are almost exclusively listed only when they show major activity. When we refer back to figure 2.4, we notice this feed to also perform sub-average with respect to timeliness. Other feeds that are also not very sensitive, like Danger.rulez, have better timeliness.

Sensitivity is however not only dependent on the types of sensors a threat intelli-

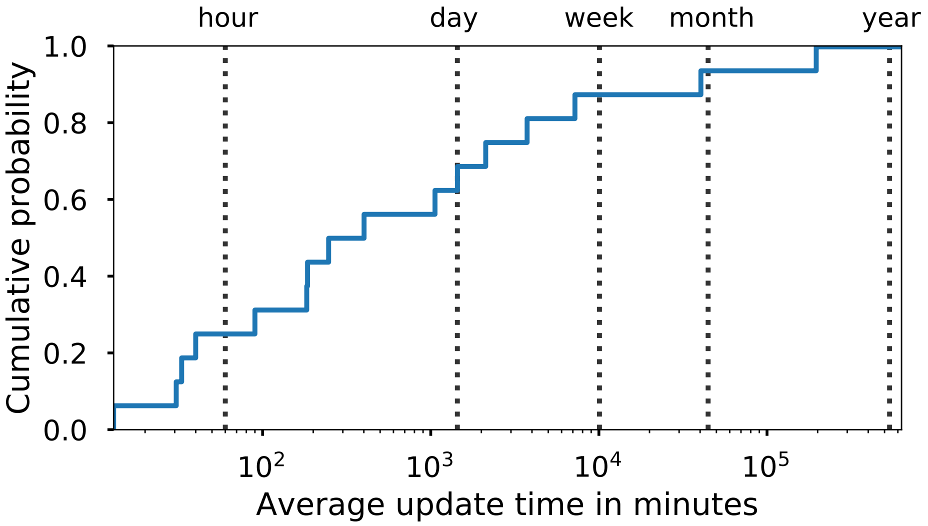


Figure 2.5: CDF of update frequency of the evaluated lists. Two thirds of the threat intelligence feeds are updated a least once a day, one thirds even includes new indicators in hourly intervals.

gence provider utilizes, but also where these sensors are located. Threats emerging in specific geographic areas might hence be under-or overestimated, leading to an overall bias in sensitivity. As there is no ground truth on where in the world threats are located, we can only do a relative evaluation on the position of the listed indicators – based on their IP prefix information – for each intelligence feed. Figure 2.7 shows this relative geographical distribution of indicator per feed, which reveals major differences in reporting between the providers and likely the location of their sensor infrastructure. For instance, more than 40% of all reports made by Bambenek are located in the United States. In contrast, more than 40% of reports on MalwareConfig originate from Turkey, and around 40% of the data provided by GreenSnow relates to China.

These biases become even more apparent when we normalize the IP addresses reported as malicious by the number of IP addresses allocated within that region. Assuming that malicious activity is not strongly concentrated within individual countries, we thus obtain a normalized geographical reporting as shown in figure 2.8, this indicates that, for example, CI Army and CINSscore are heavily leaning towards reports from Turkmenistan, which is nearly absent in the reports from all the other threat intelligence providers.

On a positive note, the distribution of reports by Emerging Threats, Badips, Blocklist.de, BruteForceBlocker, CruzIT, and Myip show no clear geographical preference, which seems to lend to the conclusion that their measurement infrastructure is sufficiently diverse.

2.5.3. Originality

To investigate the uniqueness of the provided information, we traced for each of the 1.38 million indicators when it was first emerging on a list and whether individual indicators

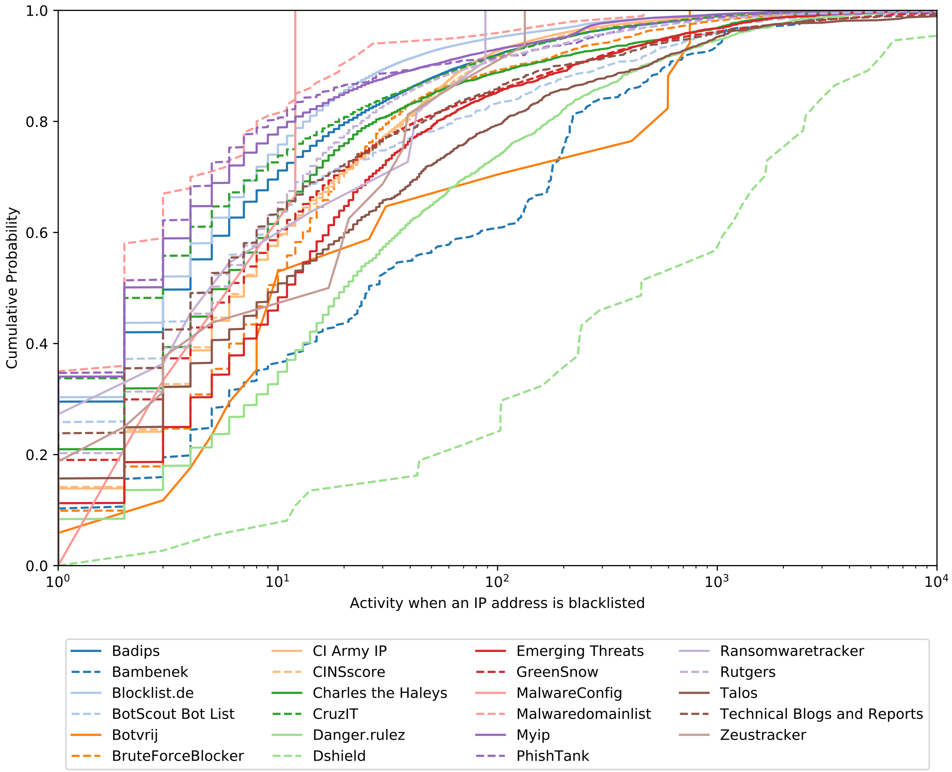


Figure 2.6: Cumulative density function of the minimum activity before an IP addresses is included in a threat intelligence feed.

were afterward also included on other lists. Besides the result of independent original research, such reuse might also indicate that a particular feed would import the data provided by others. Figure 2.9 shows the later reuse of indicator information across lists, where an arrow indicates that information first originated at the source of the arrow and was later included in the list it points to. The arrow's thickness and the corresponding label correspond with the percentage of information on the receiving list that earlier appeared somewhere else. For readability, only flows where more than 5% potentially originated from a different list are shown.

While commercial threat intelligence providers often only consolidate and curate information, as discussed above, we also see some repackaging – although at highly varying degrees – in the case of open source feeds. The indicators first appearing on the feed Blocklist.de routinely appear on other lists, and in two cases, a fifth of all indicators are shared with Blocklist.de where they appear earlier. Similarly, a quarter of the indicators on Danger.Rulez previously appeared on Emerging Threats; however, at a global scale, such repacking is comparatively rare, and only 11 out of the 24 lists showed such relationships. Across the entire dataset, indicators reappear comparatively seldom, in total

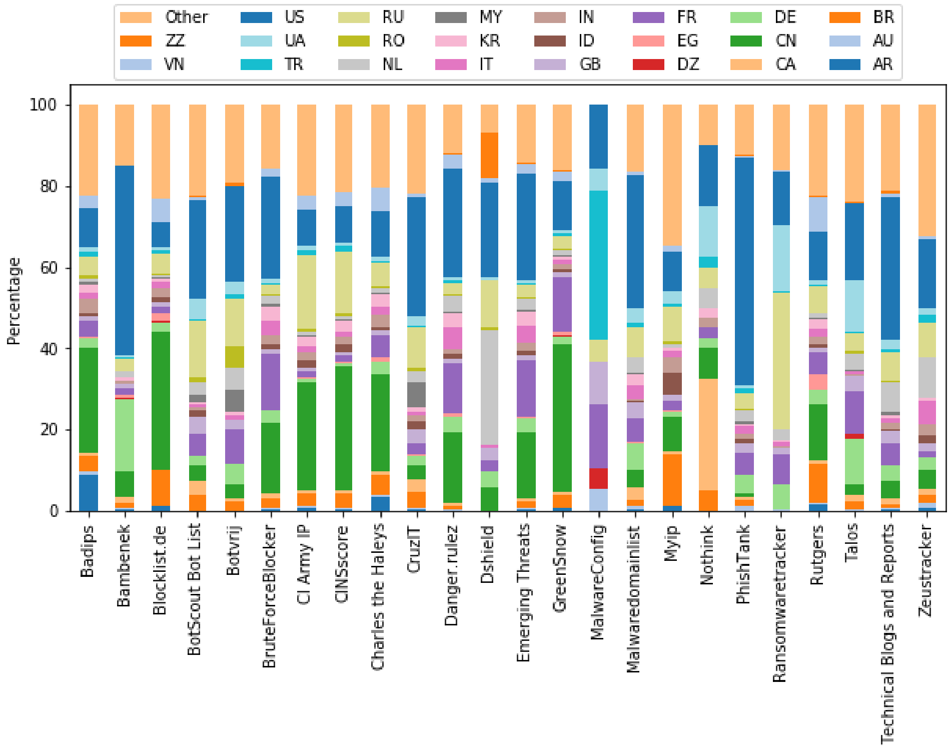


Figure 2.7: Geographical distribution of indicators per list.

only 85,906 out of the total 1.38 million entries were also listed on another feed (6.2%).

2.5.4. IMPACT

To evaluate the level of potential collateral damage, we resolved the IPv4 and IPv6 A records of 277 million domain names across 1151 top-level domain zones that we received from the TLD operators daily. For every threat intelligence feed, we then analyze how many domain names were pointing to a particular IP address on the day it was marked as malicious, as all of these domain names would no longer be resolvable if a customer would apply the ruleset provided by the threat intelligence provider in for example a firewall. Figure 2.10 shows the cumulative density function of the number of domain names resolving to the indicated IP addresses by threat feed.

As we can see in the graph, there are drastic differences in collateral damage between feeds. A homogeneous set of feeds – among them BruteForceBlocker, Talos, CruzIT, CI Army IP, and Rutgers – are comparatively targeted, more than half of their entries are not affected by any other domain names. In contrast, the 80% most targeted indicators affect less than six other domain names if applying an IP-based block. This is somewhat logical for a list that focuses on brute-forcing, which is typically not happening from servers that a shared web hoster operates host websites or, this is however not the case for the

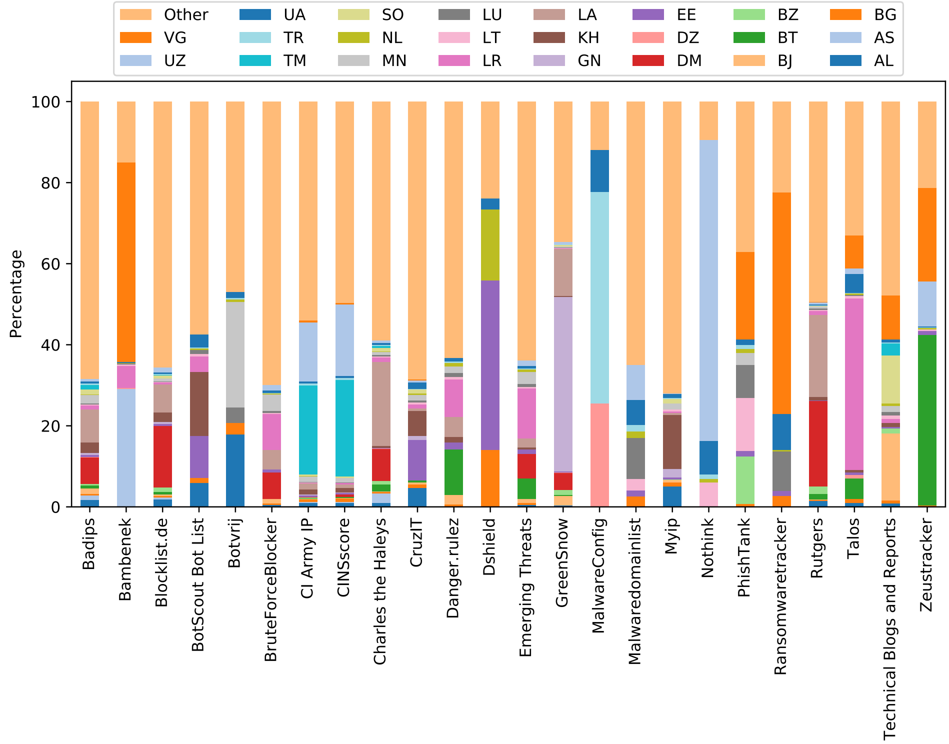


Figure 2.8: Relative geographical distribution of indicators, normalized by total number of IP addresses in a country.

information included on CruzIT, CI Army IP or Rutgers, which include IP addresses used to attack or probe certain networks. For Talos, we know that it is curated by Cisco, which makes it likely that this is the reason for the low amount of collateral damage.

This is, however, not true for all of the feeds. In the case of Bambenek, only 16% of the best-performing indicators will block less than 50 live domains hosted at these websites, where the 50% worst-performing indicators even affect 100 or more domains as collateral damage. While some of the blocked domains may indeed also contain malicious activity, some of the instances included large shared hosters, in one case with more than 900,000 domains pointing to the blocklisted IP address. While such issues could be explained due to the automatic collection of indicators, we also found a surprisingly poor track record in the case of "Technical Blogs and Reports", a curated list of human analyst reports. This indicates that human-made feeds are not necessarily better, or at least suggests that feeds do not filter records that could potentially harm regular system operation.

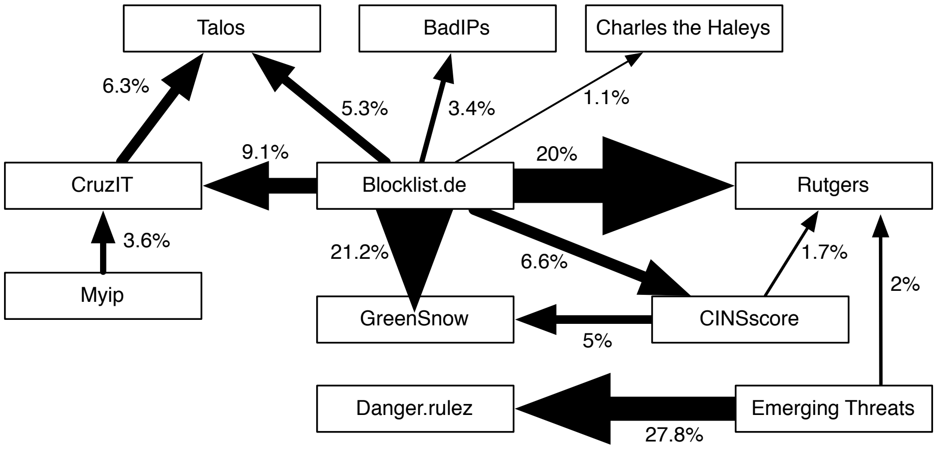


Figure 2.9: Indicator reuse across feeds occurs only sporadically, with the expection of two threat intelligence feeds.

2.6. Discussion

After evaluating each of the 24 feeds across the four dimensions timeliness, sensitivity, originality, and impact, we will take a step back in this section and consider the ecosystem of intelligence feeds as a whole. Specifically, we will investigate how widely network owners and operators adopt these feeds in practice and review the issue of the surprisingly high level of originality for the ecosystem as a whole.

2.6.1. Adoption of Intelligence Feeds

As cyber threat intelligence feeds are meant to alert and empower network owners to block malicious activity, their application should reduce network traffic towards the hosts flagged as malicious. This observation provides us with an angle to investigate the adoption of threat intelligence feeds worldwide. After all, as soon as an indicator has appeared on a list, we would expect a significant drop in activity – if not the absence of requests – from subscribers.

When networks apply the information provided in cyber threat intelligence feeds at scale, we should ideally see the activity from infected clients to malicious destinations drop and eventually die out. As we have seen above, intelligence feeds are not universally adopted but have a specific regional footprint, and there exists only a marginal overlap between lists; thus, even if a network would subscribe to and apply the information from every single intelligence feed, we cannot expect all activity to seize immediately.

Figure 2.11 shows a probability density function of how long activity towards a particular destination continued after an indicator was listed on a specific intelligence feed. As we see from the graph, the inclusion of indicators on, for example, BruteForceBlocker seems to be an effective deterrent, as, on average, activity stops within two weeks. Other lists such as Botvrij are less successful: more than 50% of all hosts reported as malicious

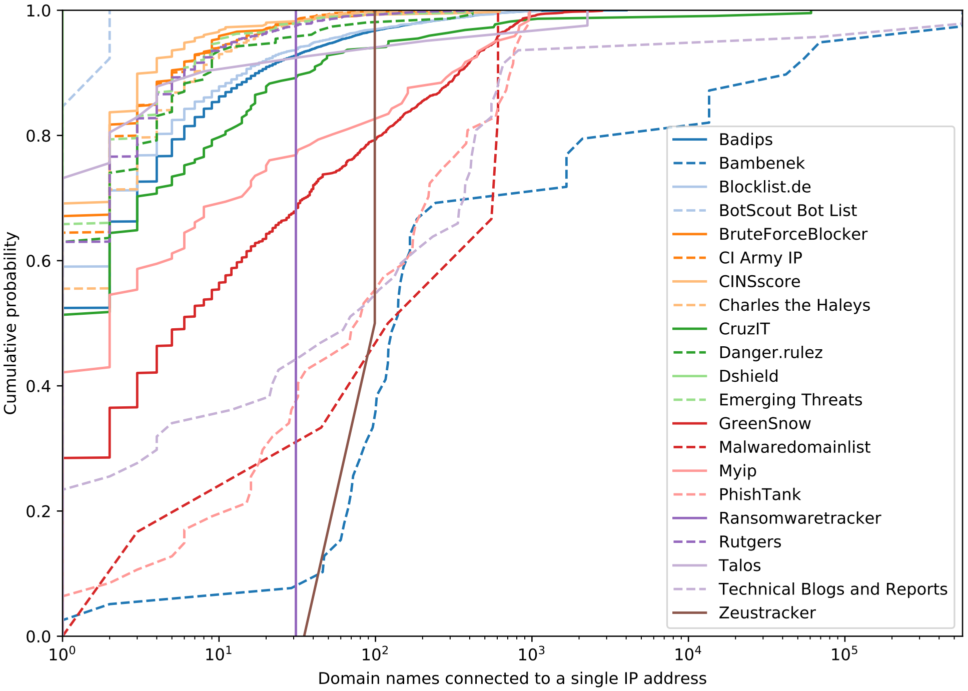


Figure 2.10: Cumulative probability function of the domain names associated with the IP addresses indicated as malicious per threat intelligence feed.

by this feed continue their activity for at least 79 days, with an extremely long tail. Thus a listing on this blocklist seems to have almost no impact on the criminal activity itself. Like in the case of timeliness until detection (see section 2.5.1), the threat intelligence feeds are also surprisingly homogeneous with respect to the continuation of activities, and we can see in figure 2.11 two main clusters, with activity termination peaking around 20 days after listing and 60 days after reporting.

2.6.2. DO WE HAVE ENOUGH COVERAGE?

There remain, however, questions about the quality of the ecosystem of cyber threat intelligence providers as a whole. Although it is desirable for a customer that CTI feeds have a significant degree of originality as otherwise, a customer would subscribe – and pay for – redundant information, we have seen that the amount of overlap between the entire spectrum of analyzed feeds was remarkably low. This, on the one hand, is commendable as it maximizes the value of CTI users. On the other hand, it also raises questions about whether the cyber threat intelligence feeds provide sufficient information to stop malicious activity in their tracks.

As discussed in section 2.4, the 24 evaluated open-source feeds spanned the entire ecosystem of malicious activity from brute-forcing activity, ransomware, and other malware, to botnets. As each type of malicious activity was covered by multiple feeds, we actually would expect *some* overlap in reported indicators. The fact that there is almost

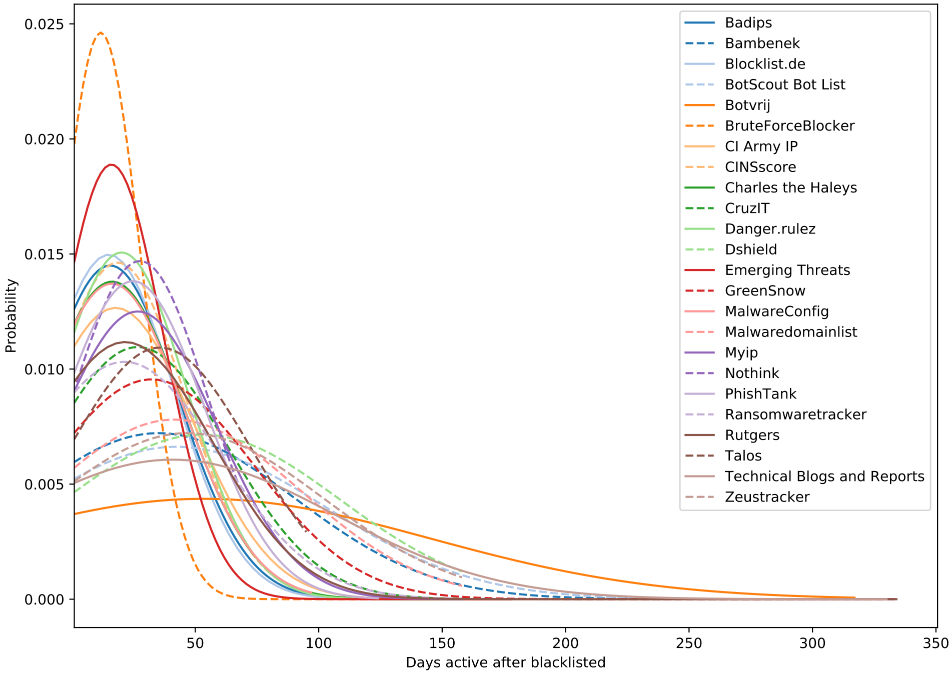


Figure 2.11: Continuation of activity to flagged destinations after listing in a feed.

no overlap between lists of similar scope could be the result of two reasons: first, all individual lists, for example, targeting botnets or ransomware, rely on orthogonal detection methods and provide complementary information. Second, the lists monitor malicious activity similarly. Still, the overall volume of malicious activity is so large that effectively each list only obtains a tiny sample. Such low rate sampling from a large universe would statistically lead to a very low chance for duplicates.

Conceptually, we can see that we are probably dealing with the latter than the former reason, after all methods to for example detect ransomware C&C servers are limited, and most likely all providers would employ off-the-shelf tools such as an analysis of network activity across malware samples or a forensic analysis thereof. This, unfortunately, drives us to the conclusion that cyber threat intelligence feeds cover much less malicious activity than we would expect and require to apply intelligence feeds and confidently expect that with a very high degree of certainty, malicious activity will be stopped through these indicators.

2.7. Conclusions

Effective protection requires insights into the activities of adversaries, commonly referred to as cyber threat intelligence. In order to protect against evolving threats, networks can subscribe to CTI feeds which list indicators, such as domain names, IP addresses, or hashes, related to malicious activity.

In this paper, we have analyzed 1.38 million indicators provided by 24 open source cyber threat intelligence feeds over a period of 14 months, and analyzed whether the information provided by these lists is timely, original, and estimated how sensitive the detection of the intelligence providers are as well as the positive and negative impacts a utilization of these feeds would have in practice. We find large variations between the performance of these lists, some are providing indicators within a few days while others only report activity months after it has commenced. This variation is surprising as we find all feeds to be relatively homogeneous in sensitivity, in other words the threshold beyond which they pick up undesired activity.

Nearly all of the analyzed lists are able to provide a significant intelligence contribution. Although lists contain a small degree of overlap, these lists are not merely subsets or repackaged versions of each other. This on the one hand is valuable to defenders as each feed does provide benefit with limited redundancy, at the same time the little overlap across all CTI feeds raises the question how much the ecosystem of cyber threat intelligence feeds as a whole really covers, and whether the current feeds will provide defenders with a suitable defense posture if applied.

REFERENCES

[1] EclecticIQ, *Intelligence-powered defences,* https://www.eclecticiq.com/dss.

[2] *The second annual study on exchanging cyber threat intelligence: There has to be a better way,* Retrieved from: https://www.ponemon.org/blog/the-second-annual-study-on-exchanging-cyber-threat-intelligence-there-has-to-be-a-better-way.

[3] W. Tounsi and H. Rais, *A survey on technical threat intelligence in the age of sophisticated cyber attacks,* Computers & security **72**, 212 (2018).

[4] A. Ramachandran, N. Feamster, and S. Vempala, *Filtering spam with behavioral blacklisting,* in *Proceedings of the 14th ACM conference on Computer and communications security* (ACM, 2007) pp. 342–351.

[5] S. Sheng, B. Wardman, G. Warner, L. F. Cranor, J. Hong, and C. Zhang, *An empirical analysis of phishing blacklists,* in *Sixth Conference on Email and Anti-Spam (CEAS)* (California, USA, 2009).

[6] S. Sinha, M. Bailey, and F. Jahanian, *Shades of grey: On the effectiveness of reputation-based "blacklists",* in *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)* (IEEE, 2008) pp. 57–64.

[7] M. Kührer, C. Rossow, and T. Holz, *Paint it black: Evaluating the effectiveness of malware blacklists,* in *International Workshop on Recent Advances in Intrusion Detection* (Springer, 2014) pp. 1–21.

[8] A. Oest, Y. Safaei, A. Doupé, G.-J. Ahn, B. Wardman, and K. Tyers, *Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists,* in *PhishFarm: A Scalable Framework for Measuring the Effectiveness of Evasion Techniques against Browser Phishing Blacklists* (IEEE, 2019) p. 0.

[9] *Measuring the iq of your threat intelligence,* Retrieved from: https://www.slideshare.net/AlexandrePinto10/defcon-22-measuring-the.

[10] *Evaluating threat intelligence feeds,* Retrieved from: https://www.first.org/resources/papers/munich2016/kompanek-pawlinski-evaluating-threat-ntelligence-feeds.pdf.

[11] M. Foukarakis, D. Antoniades, S. Antonatos, and E. P. Markato, *Flexible and high-performance anonymization of netflow records using anontool,* in *Third International Conference on Security and Privacy in Communications Networks and the Workshops* (2007).

[12] J. Xu, J. Fan, M. Ammar, and S. B. Moon, *On the design and performance of prefix-preserving ip traffic trace anonymization,* in *ACM SIGCOMM Workshop on Internet Measurement*, IMW '01 (2001).

[13] J. Abbink and C. Doerr, *Popularity-based detection of domain generation algorithms,* in *2nd International Workshop on Malware Analysis* (2017).

3

Addressing biases in measurements of Internet data

To connect to the Internet, hosts are assigned an IP address by their network provider by which they exchange data. As such, IP addresses are frequently used as a proxy metric to count the number of hosts on a network or quantify particular phenomena such as botnets' size or malware's infection statistics. Although a single host is typically linked to a single IP address at a given moment, this relationship is frequently not stable over time due to IP churn. As network operators dynamically assign IP addresses to clients for a specific lease duration, a host obtains a new IP address after the expiry of this lease, thereby leading to overestimating active host counts or malware infections.

This paper presents a novel method to detect and quantify IP churn in autonomous systems on the Internet by exploiting a weakness in the packet generation algorithm and random number generation of the Mirai IoT malware. These design shortcomings allow us to re-identify the same IoT infection when the host resurfaces on the Internet with a different IP address with very high confidence and thereby characterize how IP addresses in provider netblocks churn over time. As Mirai is widespread with hundreds of thousands of infected devices worldwide and uses the faulty RNG output to scan the Internet, our methods enable worldwide IP churn measurements to be done efficiently and completely passively.

This chapter has been published as *"Quantifying Autonomous System IP Churn using Attack Traffic of Botnets"* by **Griffioen, H.**, Doerr, C. in the International Conference on Availability, Reliability and Security (2020).

3.1. Introduction

In the Internet and Internet Protocol-based networks, networked hosts are assigned an IP address as an identifier to exchange data packets. As every host has to have an IP address to participate in the network, the number of IP addresses is often taken as a proxy to count the number of hosts in a particular network. Such measurements include Internet surveys on deployment statistics in-home, enterprise or cloud environments [1, 2], studies on how the address space that is assigned to network operators is actively used in the Internet [3], or to obtain insights into the spread and installation sizes of botnet infections worldwide [4, 5].

While every host requires an IP address to send and receive packets, this relationship between IP address and host is not strictly bijective – especially not beyond the short term. The is primarily due to two reasons: first, networks may issue IP addresses dynamically to their hosts, for example, using the Dynamic Host Configuration Protocol (DHCP). Suppose a host reconnects after the lease has expired. In that case, it gets issued a new network address. If counting hosts with a specific characteristic based on the number of network addresses, we would thus overestimate the actual occurrence of the phenomenon given this so-called IP churn. Second, also in the reverse direction, there is not a solid one-on-one relationship: as the IPv4 address space is exhausted, network operators – especially in mobile networks – place connecting customers behind a carrier-grade network address translation (NAT), and all hosts behind such NAT appear to others as a single host as they all have the same IP address. (Carrier-grade) NATs would therefore lead to a significant underestimation.

These two operational practices in networks in result mean that IP addresses cannot be used as an effective surrogate to quantify the behavior and characteristics of hosts on the Internet. For example, in [6] the authors showed using sink holing that the number of bots in the Torpig network is overestimated by order of magnitude. Similarly, [7] showed that bots based in Brazil had on average 1.3 IP addresses per day. However, these measurement biases could be addressed if we would have detailed information per autonomous system (AS) or network address block on the presence of IP churn.

Despite this importance, only relatively little work has been done on measuring and characterizing IP churn in networks, primarily since assessing this phenomenon at Internet scale is challenging and obtaining a ground truth from the plethora of network operators is difficult. While some works have empirically observed IP churn [1, 8–11], only two works to date provide a methodology that can be used to test networks for the presence of churn by actively sending sequences of ICMP and TCP/UDP probing packets [12, 13]. However, active probing is costly and often frowned upon by network operators, for whom large scale measurements mean additional – and in their view unnecessary – system load on their network infrastructure and may lead to false alarms in the network monitoring systems.

In this paper, we present a novel mechanism to quantify IP churn efficiently and NAT aggregation across the Internet *without* active measurement probes. We accomplish this by leveraging scanning patterns from malware-infected IoT devices, which due to the prevalence of insufficiently secured IoT devices and the many malware strains that descended from the IoT malware Mirai are omnipresent through autonomous systems worldwide. For our measurement method, we utilize a flaw in the random-number generation of Mirai and how it populates port scanning packets using this data to reliably link and re-identify ongoing infections even when the IP address of the host has changed, as well as pinpoint if multiple infected hosts sit behind the same public IP address. This allows us to continuously obtain reliable estimates for IP churning and the location of NATs in autonomous systems across the Internet.

In this paper, we make the following three contributions:

- We show that using a flaw in the random number generation and packet generation of Mirai, it is possible to link together infections over time and across different IP addresses to the same host with very high confidence, and thereby identify IP churn behavior in networks.
- We demonstrate that this flaw can be exploited at scale across the Internet given the prevalence of infected IoT devices. With this, we introduce the first technique for passive measurement of IP churn on the Internet. Passive assessment has a significant advantage over currently used methods, as it does not cost network operators any computational resources or bandwidth, nor does it trigger alarms in their network monitoring infrastructure. The technique proposed in this paper has thus no adverse effects on networks at all.
- We show that we can identify churn rates for a large portion of the Internet due to the widespread of Mirai infections and how we can use this to demonstrate how autonomous systems allocate their IP blocks, implement NATs, and how network prefixes can drastically change due to churn.

The remainder of this paper is structured as follows: section 3.2 provides an overview of related work on IP churn and current methods to quantify it in communication networks. Section 3.3 describes the port scanning behavior of the IoT botnet Mirai, the random number generation used, and how flaws allow us to re-identify a particular infected host even if it has been assigned a new IP address. Section 3.4 describes the data set used for our study. Section 3.5 explains the methodology we put forward in this paper, which section 3.6 validates against established techniques in terms of accuracy. In Section 3.7 we show results from the application of our measurement technique on the usage of IP churn in networks across the Internet. Section 3.8 summarizes and concludes our work.

3.2. Related work

The first work attempting to automatically quantify the volatility of IP addresses in ASes was the work of Xie et al. [10], where the authors propose a methodology that relies on IP-user mappings and routing information, where IP-user mappings are used as identifiers of which device is being used, even after a routing change. A similar methodology, leveraging uniquely identifiable features after an address change, is used in several other studies measuring churn rates on the Internet [9, 14, 15]. However, the proposed methods are dependent on software logs and do not provide a way to perform these measurements without access to these logs.

[13] employs an active method to measure IP volatility, enabling the measurement of any AS. They create and verify a high-performance, scalable approach to probe a large

portion of the Internet, where the authors consider an IP address as changed if it does not respond to their probes anymore. The authors show their method can correctly estimate 72.3% of the DHCP churn rates in a mid-sized ISP network.

While successful, active probing measurements are often frowned upon by network operators, as it generates an unwanted stream of packets that they deem unnecessary. But passive methods have not yet been able to measure a large part of the total network space. The most extensive semi-passive measurement study is done by Padmanabhan et al. [8], who used data gathered from 3,038 RIPE Atlas probes hosted across 929 ASes and 156 countries to give an overview of the reasons Internet addresses change, and how different operator policies show in the changes. However, the proposed methodology depends on device logs located in several ASes, requiring access to such a data source.

This work proposes a method based solely on passive traces that can be observed from any network and does not require additional data sources such as logs. We use uniquely identifiable features of the widespread Mirai botnet that remain constant after an address change. We extend on current works and provide a new method to passively measure this churn, which can investigate over 12,000 networks in our study of 9 months. We show that due to a large number of probes within networks, we can accurately map the structure of different networks to prefix allocations and are therefore able to provide insight into differences between ASes in terms of network block allocation.

3.3. The Mirai botnet and its packet generation

Over the past decade, computer networks, especially mobile networks, have experienced a drastic transformation with the pervasive introduction of low-cost devices of limited, dedicated functionality. Often referred to as the "Internet of Things" (IoT), this development has often been seen with concern, as the economics of the IoT, such as the low cost and profound supply chain, mean that security is frequently of little to no consideration during system design, deployment and operation. Given the plethora of devices now connected to the Internet and their relative state of insecurity, this has created the perfect storm for massive compromises.

In 2016, this issue became mainstream media when the IoT-targeting malware Mirai launched a major distributed denial-of-service (DDoS) attack on major Internet infrastructure operators, surpassing all previously known attack volumes by a factor of two, only to double this record again a few weeks later. Soon after the Mirai source code had been posted on the Internet, copycats entered the scene, a behavior also seen in other DDoS vectors [16]. In this case, various actors recycled Mirai's source and introduced minor alterations to create their own IoT botnets. While new bot masters typically changed how the malware identified itself, the passwords it used, or the ports it targeted, they all effectively left how their IoT botnet generated the scan and probe packets unchanged from the original Mirai. This means that today, an entire ecosystem of IoT malware shares behavioral characteristics that we exploit in this paper to quantify IP churn.

The way Mirai generates its scan and attack packets exhibits some particularities. After starting the non-persistent malware, the software initializes a custom-built random number generator (RNG) based on the current epoch time, the process ID(s), and the time in clock ticks since the program has started. Mirai then spins off another thread responsible for scanning the entire Internet for potentially vulnerable hosts, where the

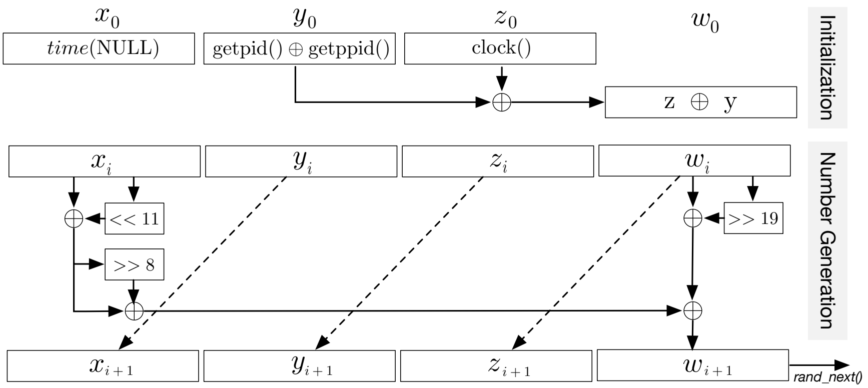


Figure 3.1: Random number generation of Mirai.

destination addresses are chosen from the output of the random number generator. The source port and window size of all scanning probes are randomly selected from the RNG but fixed throughout the entire execution of the malware. In other words, all scanning packets from the same host will feature the same header values until the device is cleaned up or rebooted, as Mirai only exists in memory and does not store its state persistently. On the other hand, if the host gets assigned a new IP address, we can recognize and link a host to an earlier IP address as it will keep sending probes with the same configuration values.

While packet features such as a session-constant source port and window size and a sequence number identical to the destination IP address are necessary conditions that a probe packet was generated by Mirai, it is not sufficient to conclusively link it to a particular infection. Thus, to establish that a train of packets comes from the same host – especially when we are aiming to connect hosts assuming new IP addresses over time due to IP churn –, we need to establish that the data which was used to populate the packets came from the same instance and initial seed.

As discussed above, Mirai's RNG output is used to choose the session-permanent source port and window size, and for each probe, two numbers are drawn to set the packet's IP ID and destination IP. While the internal state of Mirai's RNG is four times a 32-bit value, in practice, the algorithm does not start with 128 bits of entropy. As shown in the schematic representation in figure 3.1, the state is seeded using the epoch, process IDs, and clock ticks since startup. As Mirai is aggressively scanning the Internet at high speed, we experimentally determined that we will receive a scanning packet in our measurement infrastructure after approximately 15 minutes. Still, even if we conservatively estimate that scan probes arrive only within a day after the startup of the malware, the overall entropy in the epoch value is merely 16 bits. Process IDs are limited to 16 bits. Clock(), when called in a thread, returns the number of ticks since the fork(), which has only taken place four instructions earlier and is thus of limited value as well. In glibc before v2.18, which is often used in IoT devices, clock()'s resolution was limited to a granularity of 10,000 ticks, thus this value is either 0 or 1. This means that the overall complexity of the internal state is a mere 33 bits, which we can even trivially brute-force and determine for incoming scanning packets which the internal seed of the random number sequence is.

3.4. Dataset

To quantify IP churn using Mirai's flawed random number generation process, we need a measurement infrastructure to capture the connection attempts by Mirai bots. As Mirai connects to random IP addresses in the IPv4 space, this only requires passive listening on any IP address for incoming frames matching the signature used by Mirai. Of course, while a quantification by monitoring one IP address would, in principle, work, it will take long before bots from all autonomous systems in the world will be visible.

While a single IP address is sufficient to quantify churn, the measurement speed and reliability grow exponentially with the number of IP addresses used for monitoring. [17] derives an estimation for the number of IP addresses required and the resulting measurement confidence, which shows that already with a few hundred IP addresses, we can obtain estimates below 1% error margin. In this study, we rely on a network telescope of some 65,000 unused IP addresses, which can quantify the Mirai botnet's infection within an error margin below 0.1%.

To quantify churn over time, we collected probe packets for 9 months from March until December 2018 across these 65,000 IP addresses. As discussed in the previous section, Mirai's brute-forcing behavior can be distinguished based on specific header values for the TCP sequence number, destination ports, and random but session-fixed values for the window size and source port. Based on these filters, we received 2.5 billion packets during the observation period from 12,112 autonomous systems worldwide. As after reboot and reinfection, the IoT devices choose a new random window size and source port, we can cluster these 2.5 billion packets into 24,042,833 individual flows and thus unique infections. 4,034,454 of these infections continued at different IP addresses, allowing us to quantify churn across autonomous systems worldwide.

3.5. Passively measuring churn

As discussed in section 3.3, separate Mirai infections can be distinguished based on the source port and the window size of scanning packets sent out by the malware. These two session-constant values are both 16-bit in length and generated by the Mirai PRNG, creating 32-bits of entropy with which separate infections can be distinguished from one another. As we can distinguish between infections using the session constants, we can also identify whether a session stops on an IP address and continues on another.

Figure 3.2 shows three major scenarios of the progression of Mirai infections on IP addresses. Consider the case of an IP address A, which shows a particular configuration in source port/window size and thus RNG seed, and sends scanning packets towards random Internet hosts using these values. If we observe these flows to stop and later resume using a different configuration, we consider this a separate infection on the same IP address.

Consider now the case of infection on IP address B with a particular configuration. Suppose packets stop originating from B but continue using the same structure from an IP address C. In that case, we can link these two IP addresses to the same infected

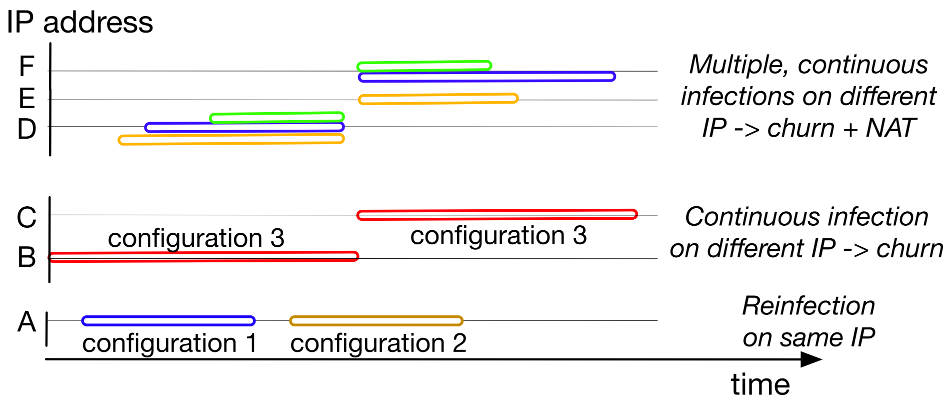

IP address
F
E
D
Multiple, continuous infections on different IP -> churn + NAT
C
B
configuration 3
configuration 3
Continuous infection on different IP -> churn
A
configuration 1
configuration 2
Reinfection on same IP
time

Figure 3.2: Three cases of infections and churn. For IP address *A*, no churn occurred, whereas IP address *B* shows a churn. IP address *D* is located in a NAT, as multiple infections are running on the same IP.

IoT device, as this situation could only happen with a chance of one in 4 billion, not even considering the clear temporal relationship. Our dataset finds pairs of IP addresses where a session stops at one IP address but shows up at another within 2 hours. Assuming that a device scans the Internet with 25 packets per second would give us a 95% chance of this device hitting our telescope within these 2 hours. In reality, we observe consecutive packets arriving on average merely 421 sec apart, which means we will be able to capture most churns.

While the probability of finding a random matching between a session that stopped on one IP and another session that started on another is negligible, we take two additional verification steps to ensure that a session is continuous rather than randomly matched: (1) We only consider IP address changes within an AS, as IP addresses would not churn between different ASes. (2) We leverage seeding of the Mirai PRNG, as explained in section 3.3, to verify that the session-constant values could not have been generated within the last two hours and is indeed a churn.

The third scenario we consider is that of IP churn in combination with a NAT, as shown at the top of figure 3.2. Imagine three infections at hosts behind that NAT with the public IP address *D*, identifiable based on the different configuration values. As Mirai infections close their entry point, multiple infections originating from one IP address are unlikely, except for various Mirai variants that use other protocols to break into a device. While unlikely, this could still happen, and therefore multiple infections on one IP address alone are not enough to establish the use of a NAT. However, if numerous others would infect one device, an IP churn would change the location of all of these infections to a single IP address. When the infections instead churn to multiple IP addresses, the infections have to be in a NAT, located at different devices behind a single IP address. IP address *E* and *F* in figure 3.2 show this behavior, where the infections from *D* churn to two separate IP addresses. These scenarios thus allow us to identify an IP address in a NAT.

3.6. Methodology Validation

As discussed in the related work, there are already methods to quantify IP churn based on active measurements. The key difference to the technique proposed in this paper is that our methodology does not require the installation of any test probes or active testing of an AS, thereby avoiding any adverse effects for network operation. The assessment using backscatter from pervasive IoT infections also has the additional advantage that it quantifies IP churn continuously, at a much higher speed, and for all autonomous systems simultaneously. This section will first assess whether our proposed method delivers comparable results to active IP churn quantification presented before.

In [8], the authors characterized the IP churn behavior of 21 autonomous systems using 3,038 ATLAS probes in 2016. Table 3.1 presents the churn rate quantified in 2016 by [8] in hours in column $d*$, together with the churn time that we obtained through our passive measurements in 2020 as column d. As we see from the table, these times are across the board identical, and we see only minor differences in the case of the Orange and Digi Tavkozlesi, where the churn time has decreased from 168 hours in 2016 to 24 hours in 2020. As both hour values are exact integer multiples of days, we attribute this to changes in operator policy over the 5 years and not measurement errors. Five of the autonomous systems studied in [8] did not exist anymore but are still included in the table for transparency.

In a typical operator network, not all IP addresses would be subject to IP churn, however. IP addresses in blocks associated with enterprise customers would not exhibit churn behavior. A similar difference in churn behavior might also exist between different customer groups. Table 3.1 hence also tabulates the share of flows in each autonomous system that demonstrated churn behavior, which provides an overall estimation of the number of IP addresses allocated to the DHCP pools. Column $f \leq d*$ lists the percentage of IP addresses found in 2016 to be churning within a particular AS, the column $f \leq d$ tabulates the percentage in 2020 per our method. As can be seen in the table, not only does our method reliably detect IP churn duration compared to previously established methods, it also accurately performs these quantifications down to the level of IP blocks. The only noticeable differences are again the networks of Orange and Digi Tavkozlesi, which likely changed their network operations within the 5 years.

Based on these results, we can conclude that our proposed passive measurement technique generates results on par with established active IP churn quantification methods.

3.7. IP Churn across the Internet

Using the methodology described and evaluated in the previous sections, we assessed the IP churn behavior of autonomous systems across the Internet. In total, we have observed 24 million infections across 12,112 autonomous systems, of which 4 million changed their IP address during the lifetime of the infection. In this section, we describe our observations on the utilization of this practice in the wild, demonstrate that we can distinguish clear demarcation lines between churning and non-churning IP blocks, and finally provide some quantification on how much IP-based estimations of phenomena such as botnets would be biased.

AS	**ASN**	**Country**	d	d*	N	N*	Flows	$f \leq d$	$f \leq d$*	*MAX*
Orange	3215	France	24	168	7,913	122	78,259	73%	98%	6113
DTAG	3320	Germany	24	24	111,445	63	182,967	90%	90%	6466
Telefonica DE 1	13184	Germany	-	24	0	14	0	-	-	-
Telefonica DE 2	6805	Germany	24	24	8,482	17	15,242	99%	99%	1450
PJSC Rostelecom	8997	Russia	-	24	0	22	0	-	-	-
BT	2856	U.K.	-	337	3,813	67	28,748	-	79%	6364
Proximus	5432	Belgium	24	36	1,841	41	9,336	71%	79%	1418
A1 Telekom	8447	Austria	24	24	1,071	12	7,848	68%	68%	3506
Vodafone GmbH	3209	Germany	24	24	13,475	21	27,346	89%	89%	6562
Hrvatski	5391	Croatia	24	24	953	7	2,342	99%	99%	111
ISKON	13046	Croatia	-	24	28	6	89	-	97%	30
ANTEL	6057	Uruguay	12	12	64,856	6	94,430	96%	96%	2399
Global Village Telecom	18881	Brazil	48	48	104,244	6	250,393	99%	99%	3235
Mauritius Telecom	23889	Mauritius	24	24	4,981	6	11,433	93%	93%	252
JSC Kazakhtelecom	9198	Kazakhstan	24	24	1,616	15	11,103	80%	80%	3834
Orange Polska	5617	Poland	22	22	24,946	10	73,202	86%	86%	6355
VIPnet	31012	Croatia	-	92	91	7	236	-	100%	47
Digi Tavkozlesi	20845	Hungary	24	168	7,049	41	18,023	81%	99%	1243
Free SAS	12322	France	24	24	715	12	24,182	73%	73%	6392
SONATEL-AS	8346	Europe	24	24	1,792	7	5,090	93%	93%	231
Net by Net	12714	Russia	48	47	1,040	7	12,552	88%	87%	3123

Table 3.1: Measured churn times d compared to measured churn times d* from [8] (in hours). N shows the total amount of churn events observed per AS, N* shows the number of probes used in [8]. $f \leq d$ shows the percentage of flows with a lower duration than d and $f \leq$ d* shows the duration of flows gathered in 2016 smaller than d*. The largest flow duration in hours observed in an AS is given in *MAX*.

3.7.1. Churn rates in autonomous systems

Looking at the Internet in general, we find that IP churn is a very wide-spread practice. Earlier work usually mentions two reasons for operator-based IP churn: first, forceful disconnects were used by early operators to discourage the hosting of services on residential Internet connections [18], and second, operators want to accomplish a better utilization of generally scarce IP addresses [11]. With the development of the Internet landscape over the past 20 years, these reasons seem, however, less valid today: on the one hand, as IP transit bandwidth and IP hosting prices have been in a general decline for years [19], discouraging home-grown hosting might have been a sensible practice in the early Internet, but commercial hosting has become so inexpensive that it is commercially viable even for home users. On the other hand, with the move to pervasive and continuous connectivity and the resulting introduction of carrier-grade NATs, the urge to conserve publicly routable IP addresses would be less dire.

Commonalities of churning autonomous systems

Still, of the 12,112 autonomous systems that we observed to have Mirai infections, only 5,148 did not use this practice. IP churn is hence still an established technique, although its usage differs significantly by geographical regions. Table 3.2 shows the percentage of autonomous systems in a country that churn IP addresses, normalized by the total number of autonomous systems in that country that had Mirai infections and were therefore visible to us. The table shows that Poland has the least churning ASes, with only 5% of the observed ASes churning. However, in terms of IP churns, more than a third of the IP space in this country churns, as the ASes that churn are magnitudes larger than the non-churning ASes. The opposite is the case for Croatia, China, Iran, and Taiwan, as the percentage of churning ASes is much more significant than the percentage of churning IPs, so only small ASes, or small parts of ASes churn. Germany, a country that has been identified by related work as a country with a high churn rate [6], has the largest IP churn rate, but not so high of an AS churn rate. Interestingly, in Brazil, which has been identified by related work as a highly churning country [7], the number of churns in Brazil is 31% and 24% for IP and AS respectively, which does not rank it globally as one of the primary users.

As discussed above, one of the motivations to allocate IP addresses only dynamically is to limit the number of IPs that are “tied” up at a given moment, and thus potentially serve more customers with a smaller IP netblock. This could manifest itself in one of two ways: first, it could mean that we should over-proportionally see IP churn in smaller ASes that have a smaller allocation of IP addresses, or second, that IP churn is especially dominant in networks that have the bulk of their IP addresses in use and are hence short on resources. To test the first hypothesis, we analyzed the adoption of dynamic allocation for the number of IP addresses belonging to an autonomous system. We found no relation between the size of an AS and its churn rate ($R = 0.03$, $p < 0.001$).

To obtain an estimation of the IP addresses that are actually in use within an autonomous system, we turned to the Internet surveys of Censys that scan all IPv4 addresses for a selection of well-known TCP and UDP ports and perform ICMP ping tests. For this analysis, we characterize an IP address as “active” if it has at least one TCP or UDP port open or responds to ICMP requests. While this provides a lower bound on the

Country	IPs	ASes	Churning IP	Churning AS
PL	34352	567	37%	5%
US	69467	1192	30%	10%
UA	20305	653	12%	15%
RU	174237	1467	35%	15%
BG	23229	200	28%	16%
KR	77856	115	6%	30%
DE	69431	137	62%	31%
CN	1259021	150	20%	34%
IR	33309	139	19%	36%
TW	154880	55	13%	51%

Table 3.2: Selection of top and bottom five countries in terms of IP and AS churn. Similarly to other studies, we identify Germany as a highly churning AS [6], but we see that these churns actually originate from a smaller portion of ASes.

IP addresses in use, this is acceptable as an estimator here as we undersample everywhere and are only interested in relative differences between the autonomous systems. However, also for the second common hypothesis on why ASes would employ churning, we find no relation (R = 0.08, p < 0.001) between the fraction of the IP addresses used in an AS and the fraction of observed churns in the AS.

CHURN RATE OF AUTONOMOUS SYSTEMS

Clients, home and enterprise routers may connect and disconnect to their Internet Service Provider at arbitrary times. For example, a customer might close a laptop or unplug the router when no longer in use, and similarly, routers might reboot after a power outage or software crash, potentially obtaining a new public IP address from the ISP network. Aside from all of these “routine” activities, this paper is particularly interested in those moments when the Internet Service Provider breaks the connection and issues new IP addresses to its customers, which leads to predictable and large-scale IP churn across an autonomous system. To automatically distinguish random population events from policy-based operator activities, we first quantify the lifetime of Mirai botnet infections at a given IP within a particular autonomous system, which is depicted in figure 3.3.

Consider the blue line belonging to AS2516, KDDI Corporation, a Japanese telecom operator. As we see in the graph, in about 40% of all cases where clients disconnect and reappear on the Internet under a different IP address (but with the same ongoing Mirai infection), the client was online anywhere between seconds to almost one day. In the middle of the graph, we see a major spike: in approximately 45% of all situations where clients of KDDI disconnect and reappear with a new IP address on the Internet do so precisely in a 24-hour interval, which we identify due to its consistent and network-wide application as an operator policy. Other networks apply different thresholds: AS27925 resets its connections after 12 hours, while AS9924 shows consistent reconnections after 2 days. We find that ASes occasionally operate different policies across the various parts

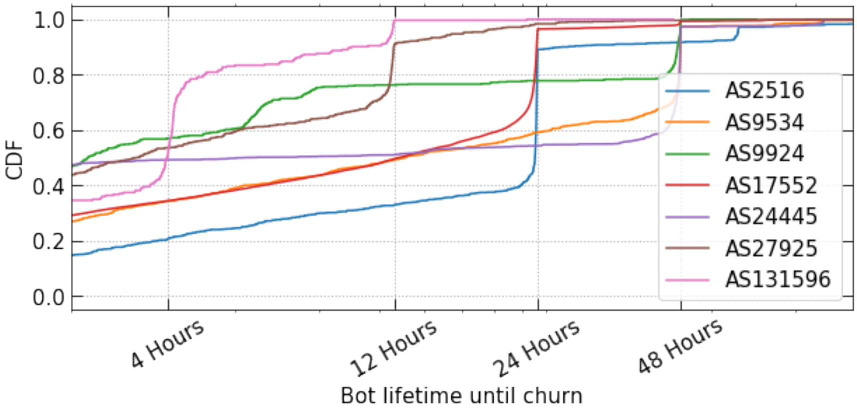


Figure 3.3: Churns for a selection of ASes, showing some ASes have multiple churn events.

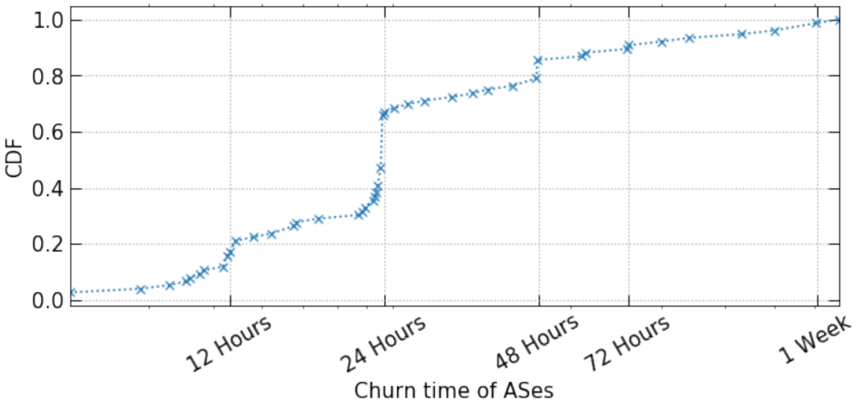


Figure 3.4: CDF of churn events for ASes. Most churn with a duration of 24 hours or a multiple of this.

of their networks. An excellent example of this is AS131596, in which most reconnects occur after precisely 4 hours, while some of its blocks churn in a 12-hour interval.

Using peak detection based on the relative growth of the churn CDF, we extract modes or the churn interval(s) in use by a particular autonomous system. Figure 3.4 depicts the cumulative density function of these churn intervals for the churning 6,964 autonomous systems in this study. The most common policy found in the wild is a reset once a day. Approximately 30% of autonomous systems disconnect their customers after 24 hours, to a lesser degree also for 2 days, 12 hours or 3 days. While these multiples of (half) day intervals seem to be in wide-spread use, in practice, we find many different and often highly unique policies across autonomous systems across the Internet. As can be seen by the dots and associated jumps in the CDF, we find churn intervals of every integer multiple of days and integer multiples of hours as a dominant policy somewhere on the Internet. Overall, there is significant turnover, and on average, clients move to a new IP address after 33 hours.

3.7.2. ENTERPRISE NETWORK BLOCKS

While residential IP addresses are often allocated with, for example, DHCP, this is not desirable for enterprise customers. To ensure continuity in IP addresses used in, for example, websites or other services, these customers are allocated static IP addresses, meaning that they are not prone to churn such as residential hosts. To do so, Internet Service Providers (ISPs) allocate blocks of IP addresses for their enterprise customers and do not use them in the dynamic pool. Insight into how these blocks are allocated inside an AS yields information on which netblocks do churn and which netblocks' IP addresses are static.

By aggregating churn rates of individual hosts in their /24 netblock, as enterprise networks are likely to be managed at /24 [20], and labeling these netblocks as churning when we observe churn in this netblock, we can plot a Hilbert curve that accurately shows the allocation of netblocks in an AS. A Hilbert curve maps a 1-dimensional array into two dimensions and preserves the distance information for a large part, where adjacent points on the curve are adjacent in the 1D array. Figure 3.5 shows this Hilbert curve for AS16135 across a number of its prefixes separated by white lines, located in Turkey owning 197 IPv4 prefixes. In the figure, we only show a selection of these prefixes for visibility. Every point in the Hilbert curve is mapped to one /24, which we have observed in our data. We omit /24 netblocks that we have not observed. The figure shows that this particular AS allocates its network based on prefixes and places its enterprise and residential customers in different prefixes. The allocation does not seem to depend on the prefix, as the 5.25/16 prefix is allocated to non-enterprise customers, but all adjacent prefixes are not.

While some networks segregate their network, such as in figure 3.5, not all operators allocate their network based on BGP prefixes. In AS3320, where 10% of flows exceed the churn time as shown in table 3.1 and thus are most likely located in enterprise netblocks, we do not see the clear distinction between netblocks as in AS16135 but rather enterprise blocks and residential IPs scattered through eachother, showing the difference in ISP policies for segregating their networks.

3.7.3. PREFIX ALLOCATION FOR CHURNING CLIENTS

When customers dynamically get assigned an IP address from a pool of IP addresses, the obvious question to ask is how this allocation strategy works. Imagine a larger network with multiple netblocks added over time and are therefore not consecutive in the IP address space. One way to realize this is to assign a pool of IP addresses to a particular network device that provides access to customers connecting through it. This would mean that IP address allocations are stable within netblocks and can be associated with customer groups based on region or how they connect to the network (DSL, fiber, cable, vs. dial-up). However, such a semi-fixed strategy is challenging to run if the goal is to reduce address usage and thus conservatively assign netblocks while providing enough addresses during connection surges. An alternative strategy is to have netblocks “float” across devices, handing them out to connecting customers, which provide the best utilization but is more challenging to do routing for.

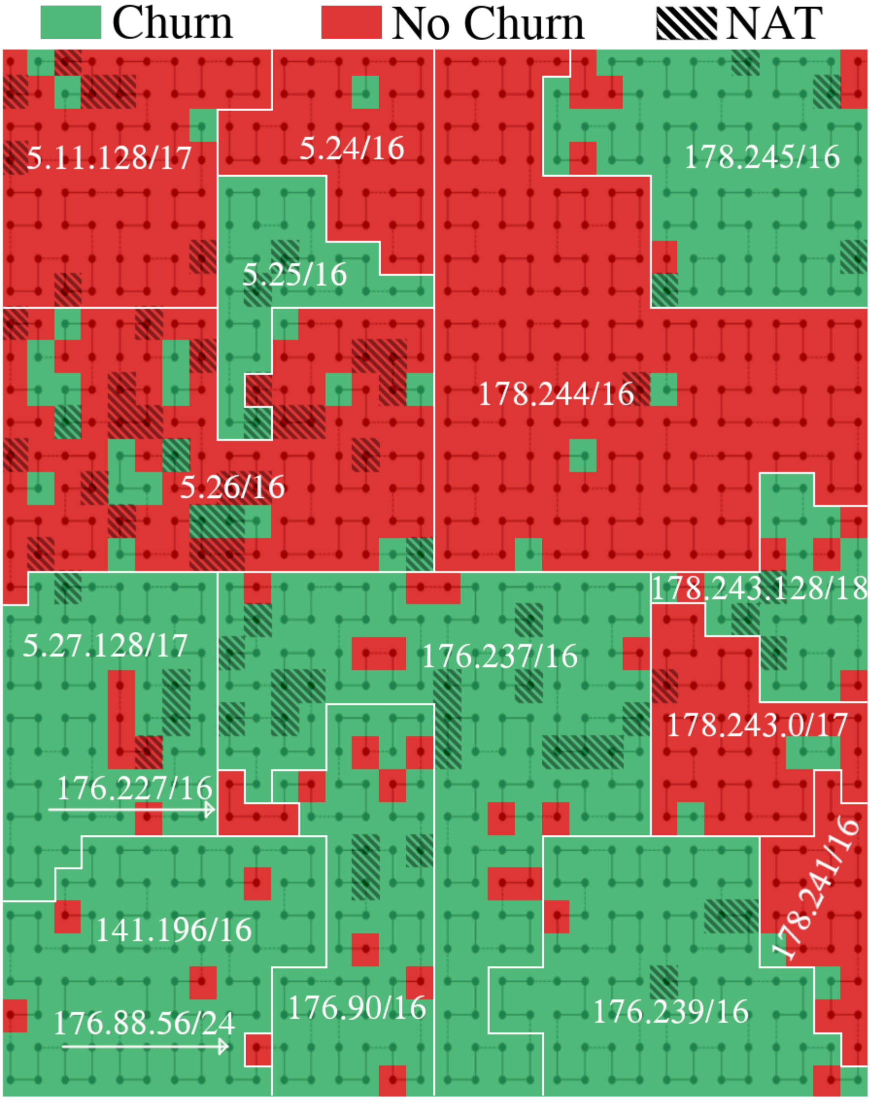


Figure 3.5: Hilbert curve of churn in netblocks of AS16135 per /24 netblock. Blocks are labeled with churn if there has been one or more churn events in the /24 throughout the whole study. The graph shows blocks of non-churning ranges where enterprise customers are located. The Hilbert curve is drawn in the figure with a dotted line.

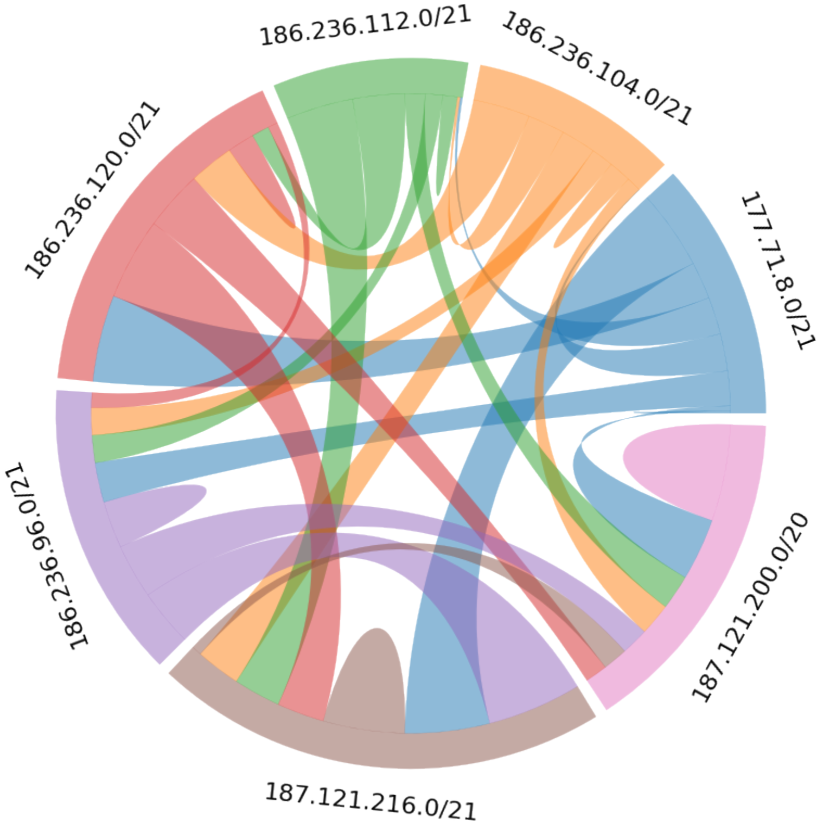


Figure 3.6: Churn behavior between netblocks in AS53131. Outgoing edges are colored by the originating netblock. Some netblocks, like the blue one, act at "sources", churning to other netblocks but never receiving from other blocks. Others act as "sinks", only receiving and churning within.

By tracing over time at which IP addresses returning customers reconnect to the Internet – which we can identify as they still run the same Mirai infection –, we can obtain some insight into how the autonomous systems manage the address pools. Although some address mobility is expected also in the first scenario if a customer for example goes on business travel and connects with the same device and account from a different part of the network, if these events happen en masse across an autonomous system we can link this behavior clearly to the second strategy in use. Figure 3.6 shows the movements of clients between seven different /21 netblock in the network of AS53131, where the outgoing edges are filled in with the color of the netblock they originated from. As we can see, for this particular operator, there appears no clear relationship between clients and IP pools they get addresses from, but allocations jump across ranges.

While previous work on IP churn has also found common prefix changes of Internet endpoints [8], the necessity to infer this by active measurements limited the number of autonomous systems that could be investigated. In [13] for example, the authors

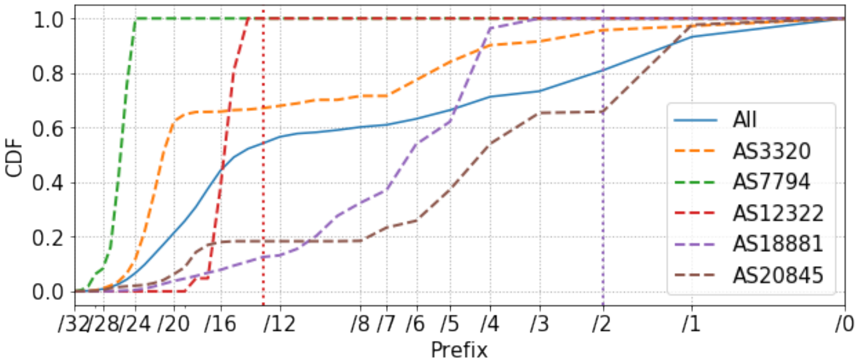


Figure 3.7: CDF of prefix change due to churn, vertical lines show the maximum jump of AS12322 and AS18881, the others can make jumps within /0. Different operator policies can be observed, with for example AS7794 only changing inside a /24 prefix.

found dynamic network-wide pools across four network operators, while [8] was able to test from 3,038 RIPE Atlas probes a total 929 ASes. The passive technique presented in this paper allows us to do such an investigation at a much broader scale, and figure 3.7 shows the distance between consecutive IP address assignments for endpoints across more than 12,000 autonomous systems that offers a multitude of different policies across the Internet. A prominent example of the first allocation strategy is AS7794, Execulink Telecom from Canada, which always assigns IP addresses from the same /24 netblock – customers reappearing with a new IP address never jump outside the block they connected from previously. This is not an artifact of operator size but operator policy, as AS7794 has a total of 136,192 IP addresses assigned to them. In the autonomous system AS12322, half of all new address assignments occur within a distance of /16, even though the operator has many netblocks that are both larger and spaced apart at a distance larger than this. When we look at IP address assignment across all autonomous systems, we see allocation mainly driven by custom, per-AS settings, and no universally applied practices appear across autonomous systems. This can, for example, be seen by the complete absence of peaks for the blue line in figure 3.7 representing all churn distances, which is void of any noticeable peaks at typical blocks used for netblock allocations and routing such as /24 or /16.

3.7.4. NAT behavior

As discussed before, due to the shortage and resulting high value of IP addresses, in many operator networks clients are no longer connected with a publicly routable IP address to the Internet, but placed behind a (carrier-grade) network address translation (NAT) where a large number of hosts share the same public IP address. This practice will lead to underestimating phenomena such as botnet infections on the Internet if prevalence is assessed based on IP address as a proxy. As the Mirai botnet binds a local port such as TCP23 on the host it has infected, it is not possible to have two concurrent infections on the same physical machine.[1] This, however, means that if we see Mirai scanning traffic with different session configurations originating from the same public IP address, we have clear evidence of the existence of at least two hosts behind the IP in a network address translation configuration. In this section, we will therefore use our methodology to quantify NAT deployments.

We find 9370 ASes containing IP addresses having concurrent infections, therefore most likely being part of a NAT. Figure 3.5 shows NAT allocation on AS16135, with NATs spread out over various subnets. While we have observed many NATs in some subnets, such as “5.26.x.x”, other subnets do not contain any sign of NATs being used.

NATs are used to connect many devices to the Internet instead of limiting the total users on the Internet to the entire IPv4 space. Intuitively, one would expect a NAT in a small AS, where the operators do not have enough IP addresses to cope with the number of devices connecting to the Internet. To verify this hypothesis, we compare the total number of IP addresses allocated to an AS, to the number of NATs we have observed normalized by the total number of observed IP addresses at an AS, and find there to be a slight negative correlation (R = -0.11, p < .00001). So indeed, smaller NATs are more likely to have NATs in their network, while larger networks can do without.

Even in a NAT, network operators churn the IP addresses placed behind an IP address. As more devices, and therefore infections can be located behind the same publicly routable IP address, infections in one NAT can be found in multiple other NATs after an operator churns the network as in the example of figure 3.2. We observe this behavior in 289 ASes and from these can identify 104 ASes where network operators churn IP addresses in a NAT-pool, where devices churn within a small pool of IP addresses.

3.7.5. Botnet estimation

Both churning IP addresses and the use of NATs have an effect on Internet based measurements where IP addresses are considered immutable single-source datapoints. While both effect the measurements, churn leads to an overestimation, while NATs lead to an underestimation in the measurements. In this section we will evaluate the effect of IP churn and NATs on measuring the Mirai botnet used in this study.

We have observed over 4 million churns over the 9 months of data collection, which would imply an overestimation of 20% when counting IP addresses infected by Mirai. As we have shown in this paper, ASes have different policies for when their network is churned, and thus how much we overestimate the size of a botnet in that AS. The largest overestimation we observe is in AS48738, which churns every 6 hours leading to an overestimation of 409%.

NATs lead to an underestimation of the total Mirai spread when solely counting IP addresses, as multiple infections are located behind the same publicly routable IP address. In our data, only looking at IP addresses will lead to an estimation of only 55% of the total number of infections. As smaller networks are more inclined to use NATs in their network, the underestimation in small ASes tend to be larger than in ASes having access to many IP addresses. The largest underestimation is for AS29914, a small AS with 15,616 IP addresses, and all the infections located behind a single NAT. So we would only

[1] There are corner cases to this rule, as there exist a Mirai descendant that bind not to telnet at the default port. However, these cases are rare, and the effect negligible when doing quantifications across ASes.

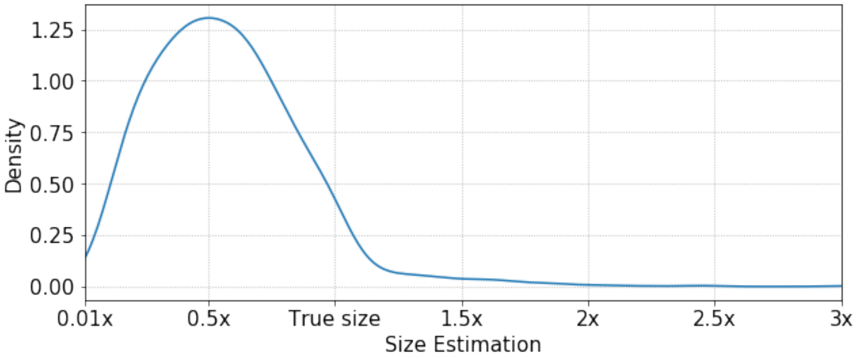


Figure 3.8: PDF of the estimation of botnet size per AS when using IP addresses as unique bot count.

count this as 1 infection, while in reality we observe 287 different infections behind this NAT, meaning we largely underestimate the number infections.

Figure 3.8 shows the PDF of infection size estimations of 2200 ASes divided by the actual size of the infection on the AS. We see that most infections are underestimated, due to the presence of a NAT, and that only few are overestimated due to churn. We see that NATs have a larger effect on the estimation than churns, as NATs can easily host a large number of devices while the churn rate is not high enough to compensate for this underestimation. Only 27 ASes were correctly estimated, of which 21 were void of churns and NATs, showing that IP addresses cannot be used as an effective surrogate to quantify behavior and characteristics of hosts.

3.8. CONCLUSION

This paper presents a method to quantify IP allocation behavior by leveraging the spread of a large botnet. The proposed method can distinguish in detail different subnets used by ASes for various purposes. While previous work has relied on active measurements to measure churn rates on a large scale, we can passively measure over 12,000 ASes.

We show we can measure churn rates in different networks by using only passive data collected in a network telescope. Using these measurements, we show that network operators segment their network in different ways, where some operators choose to divide their network based on BGP prefixes, other operators also segment within these prefixes. Additionally, we show that we can identify addresses used in (carrier-grade) network address translation and quantify the differences in over- and underestimation for IP-based measurements in different ASes.

REFERENCES

[1] L. Vu, D. Turaga, and S. Parthasarathy, *Impact of dhcp churn on network characterization,* ACM SIGMETRICS Performance Evaluation Review **42**, 587 (2014).

[2] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, *Going wild: Large-*

scale classification of open dns resolvers, in *ACM SIGCOMM Conference on Internet Measurement* (2015).

[3] M. Dooley and T. Rooney, *IPv6 Deployment and Management,* Vol. 22 (John Wiley & Sons, 2013).

[4] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, *My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging,* in *USENIX Workshop on Hot Topics in Understanding Botnets* (2007).

[5] D. Dagon, C. C. Zou, and W. Lee, *Modeling botnet propagation using time zones,* in *Network and Distributed System Security Symposium* (2006).

[6] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, *Your botnet is my botnet: analysis of a botnet takeover,* in *Proceedings of the 16th ACM conference on Computer and communications security* (2009) pp. 635–647.

[7] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, *Measurement and analysis of hajime, a peer-to-peer iot botnet,* in *Network and Distributed Systems Security (NDSS) Symposium* (2019).

[8] R. Padmanabhan, A. Dhamdhere, E. Aben, K. Claffy, and N. Spring, *Reasons dynamic addresses change,* in *Proceedings of the 2016 Internet Measurement Conference* (2016) pp. 183–198.

[9] M. Khadilkar, N. Feamster, M. Sanders, and R. Clark, *Usage-based dhcp lease time optimization,* in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (2007) pp. 71–76.

[10] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber, *How dynamic are ip addresses?* in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (2007) pp. 301–312.

[11] I. Papapanagiotou, E. M. Nahum, and V. Pappas, *Configuring dhcp leases in the smartphone era,* in *Proceedings of the 2012 Internet Measurement Conference* (2012) pp. 365–370.

[12] J. Heidemann, Y. Pradkin, R. Govindan, C. Papadopoulos, G. Bartlett, and J. Bannister, *Census and survey of the visible internet,* in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement* (2008) pp. 169–182.

[13] G. C. Moura, C. Ganán, Q. Lone, P. Poursaied, H. Asghari, and M. van Eeten, *How dynamic is the isps address space? towards internet-wide dhcp churn estimation,* in *2015 IFIP Networking Conference (IFIP Networking)* (IEEE, 2015) pp. 1–9.

[14] G. Maier, A. Feldmann, V. Paxson, and M. Allman, *On dominant characteristics of residential broadband internet traffic,* in *ACM SIGCOMM Conference on Internet Measurement* (2009).

[15] M. Casado and M. J. Freedman, *Peering through the shroud: The effect of edge opacity on ip-based client identification,* in *NSDI* (2007).

[16] V. Ghiette and C. Doerr, *How Media Reports Trigger Copycats: An Analysis of the Brewing of the Largest Packet Storm to Date,* in *ACM SIGCOMM Workshop on Traffic Measurements for Cybersecurity (WTMC)* (2018).

[17] N. Blenn, V. Ghiëtte, and C. Doerr, *Quantifying the spectrum of denial-of-service attacks through internet backscatter,* in *Proceedings of the 12th International Conference on Availability, Reliability and Security* (2017).

[18] D. Herrmann, *Beobachtungsmöglichkeiten im Domain Name System. Angriffe auf die Privatsphäre und Techniken zum Selbstdatenschutz* (Springer, 2016).

[19] W. B. Norton, *The Internet Peering Handbook* (DrPeering Press, 2014).

[20] X. Cai and J. Heidemann, *Understanding block-level address usage in the visible internet,* in *Proceedings of the ACM SIGCOMM 2010 conference* (2010) pp. 99–110.

4

FINGERPRINTING ADVERSARIAL TOOLS

Over the last two decades, Voice-over-IP (VoIP) and specifically SIP have become standard solutions to realize voice telephony in residential, commercial, and telecom environments. As of now, an abundance of SIP endpoints exist. It has become financially lucrative for cybercriminals to systematically search for VoIP installations, abuse them for billing fraud, or hide their criminal activities behind a legitimate connection and phone number. By now, this has made SIP one of the most scanned UDP protocols on the Internet.

In this paper, we take a look at the actors behind these attacks. Using a large network telescope, we collect over 822 million SIP brute-forcing attempts from 5,691 sources over 187 countries and analyze who is searching for and attacking VoIP endpoints. As each tool and campaign exhibits specific implementation differences, we can relate individual attempts into campaigns and can thereby provide a detailed view into different actors in the ecosystem, different techniques and tooling, and how these are developing over five years. We show that we can fingerprint different SIP scanning tools, show that actors hardly ever change their toolkit, and identify increases in highly distributed and coordinated scanning.

This chapter has been published as *"SIP Bruteforcing in the Wild - An Assessment of Adversaries, Techniques and Tools"* by **Griffioen, H.**, Hu, H. Doerr, C. in the IFIP Networking Conference (2021).

4.1. INTRODUCTION

Voice-over-IP or VoIP provides the technical means for carrying telephony calls over regular IP packets over data communication networks, eliminating the need for dedicated voice lines and the associated circuit switching. This drastically reduces required resources, starting with lower bandwidth per call and more straightforward and less expensive telecommunication infrastructure. As a result, charges for telephony have plummeted, and the widespread adoption of voice-over-IP makes it possible for all-you-can-call plans to exist for landlines and mobile handsets, sometimes at prices a long-distance call had been charged per minute in the 90s.

There are two main components to Internet telephony: first, protocols such as RTP or SRTP transmit the packetized audio and video data, but before a call can be made, a signaling protocol has to locate the user, set up a call, negotiate the connection and ultimately end the session. One of the most popular protocols of this second category is the Session Initialization Protocol (SIP). Standardized in RFC 3261, it is responsible for VoIP signaling in Internet telephony and adopted by the 3GPP for mobile phone calling over LTE. As VoIP has become the dominant solution to realize voice telephony in residential or commercial environments and has been adopted by telecommunication networks themselves, a plethora of endpoints exists listening for incoming connections signaled by the SIP.

This ubiquity of SIP-speaking endpoints however also opens up the possibility for abuse, as misconfigured systems or those insufficiently secured may be used by cybercriminals, for example, fraudulently placing long-distance calls via the phone lines of the victim [1]. Furthermore, by making malicious calls through a victim's systems, criminals could cloak their identities and origin and even impersonate authorities or legitimate users as part of a social engineering attack [2]. To discover potentially vulnerable systems, criminals scan public networks such as the Internet for hosts responding to SIP requests, and given the wide opportunities for abuse and the plethora of potential victims, SIP has become a wide-scanned protocol on the Internet today.

While we know already much about the vulnerabilities of SIP in general and how adversaries can abuse it, we know relatively little about the adversaries who abuse it. Knowledge of the actors and the overall ecosystem and the techniques used to scan and compromise hosts provides vital intelligence to detect malicious activities, modify systems to reduce incentives for an attack, and ultimately identify the criminals behind it. Furthermore, when we can effectively detect the tooling used by adversaries for a compromise, we can stop malicious activities early before causing significant damage.

In this paper, we address this gap and assess the adversaries, their techniques, and tools for exploiting SIP endpoints. Using a large network telescope of two partially populated /16 networks located in enterprise systems, we redirect and analyze incoming SIP requests for analysis. Based on 822 million SIP connection attempts sent by 5,691 sources over 5 years, we can provide a detailed view of this cybercrime ecosystem. With this work, we make the following main contributions:

- We create a method for fingerprinting SIP scanning tools and apply this method in the wild to identify tools used to scan for SIP servers.
- We show that actors do not change their tools but rather keep the same tool even

when running their operations over multiple years.

- We show most actors use standard tools and run these without any customization.
- We show that custom-made tools are used more in highly distributed and coordinated scanning.
- We identify an increase in distributed scanning over the years, where actors aim to stay undetected.

The remainder of this paper is structured as follows: Section 4.2 provides an overview of previous work and existing knowledge on SIP abuse. Section 4.3 briefly introduces the SIP protocol. Section 4.4 outlines our method used for data collection and highlights the methodology used to create tool-specific fingerprints. Section 4.5 analyzes a set of 21 tools found in-the-wild. In Section 4.6 we identify scanning campaigns conducted using these tools and show how actors are distributing their operations. Section 4.7 summarizes and concludes our work.

4.2. Related work

In 1999, Handley et al. introduced the Session Initiation Protocol (SIP) [3], used for initiating and maintaining real-time session for messaging applications including voice and video messaging. Over the years, the protocol has seen many scrutinous changes adding new functionalities and security improvements [4–8].

As SIP is a widely adopted protocol, a large body of research work exists to identify and solve the protocol's security vulnerabilities. Attacks on SIP are discovered impairing the availability of servers, leaking sensitive information [9], injecting data in SQL databases, or even using SIP servers as an amplification service to perform DDoS attacks [10]. To aid in finding SIP vulnerabilities, many researchers have created automated tools which automatically scan and identify problems in the software [11–14]. Along with automated vulnerability scanning tools, researchers introduced other tools to scan the Internet, identifying hosts running SIP such as NMap [15] and SIPVicious [16]. This so-called port scanning allows attackers to identify and consequently exploit hosts running SIP. To examine the threat-landscape of SIP, Nassar et al. [17] created a honeypot system capable of capturing attack traffic in the wild. In similar work, Hoffstadt et al. [1] deploy a honeypot system in the wild for almost one full year, capturing over 47.5 million SIP messages. Aziz et al. [18] record in a similar study 857 different attackers in their honeypots over 1.5 months. While these works report on exploitation behavior in the wild, the authors do not examine differences in scanning tools and the evolution of SIP scanning over multiple years.

In other work capturing SIP scanning in-the-wild, Dainotti et al. [19] analyze a large horizontal network scan targeting the SIP protocol, identifying the SIP payloads used in the attack and noting that the structure of the payload is similar to that of the SIPVicious scanning tool, which the malware authors most likely modified. In a follow-up paper, Raftopoulos et al. [20] identify exploits attempts following the large SIP scan in 2011 and identify a surge in IP addresses performing exploitations of SIP. While previous works have identified in-the-wild attack traffic, only one focused on detecting the Tools, Techniques, and Procedures (TTPs) used by attackers to perform these attacks [21]. While the

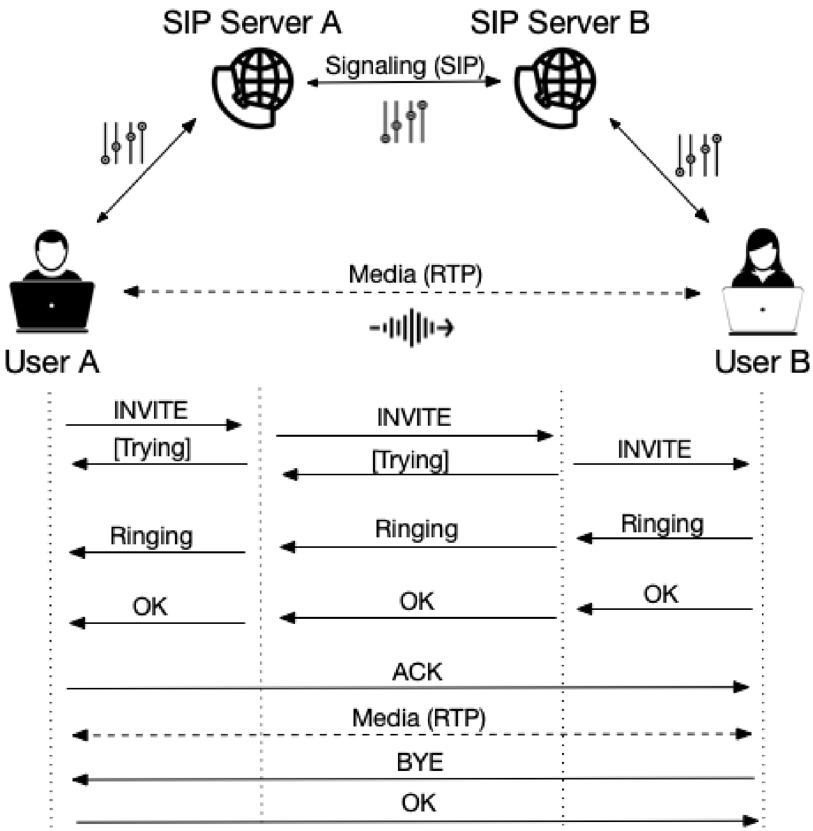


Figure 4.1: SIP Signaling and RTP media flow during a VoIP call.

authors identify tools used to scan and exploit their SIP servers and note that adversaries change SIP fields to circumvent IDS detection, they did not identify large scanning and exploitation campaigns or fingerprints of specific tools. This paper aims to address the gap by fingerprinting the tools used to perform SIP scanning and analyzing six years of in-the-wild scanning traffic to identify actor evolution.

4.3. THE SESSION INITIALIZATION PROTOCOL

Before we discuss our data collection and results, we will first briefly introduce the necessary flows in Voice-over-IP and specifically the Session Initialization Protocol as background for the messages we observe later and their interpretation in the protocol.

As discussed in the introduction, Internet telephony uses two components: first, the signaling protocol SIP communicates status messages, setups and terminates calls or exchanges configuration information, and second, a media transfer protocol such as RTP carries the actual voice and video payload between the two communicating endpoints. While for efficiency and latency reasons, media is transmitted directly between caller and callee, this requires knowledge of the location and network address of the remote party. When users are mobile or making a connection for the first time, we would not yet know where to find the other party; hence signaling messages are exchanged via proxy servers. VoIP user identities come in the form of *sip:username@domain*, which would tell the caller which server to contact to establish a connection with the desired endpoint. As shown in figure 4.1, a SIP endpoint requests its SIP proxy to make a call, which would contact the server responsible for the destination. As the called user would keep alive a connection with its SIP proxy, this server would know how to reach its user and pass along the request for a connection to the client. If the connection is accepted and passed back via the proxies, both parties would have all information necessary to start exchanging data directly.

The Session Initialization Protocol has a message structure comparable to HTTP. Requests are made from a client to a server, which returns a response message in the same connection. The protocol messages are text-based, the first line of the text contains the request or response (like an HTTP GET or POST), followed by headers and a message body. SIP monitors for timeouts and has the functionality for retransmits built into the protocol. SIP messages are preferably sent over UDP instead of TCP to reduce connection latency and overhead of the transport layer. There are six general types of requests in the protocol: *INVITE* messages request the establishment of a SIP session - thus a VoIP call -, an *ACK* confirms the receipt, a *BYE* terminates an existing connection, and *CANCEL* stops a transaction that is currently ongoing. An *OPTIONS* message asks the other side for its capabilities with starting a session, and a *REGISTER* performs a sign-in of an endpoint with the proxy server of its SIP registrar. This means that to get an insight into SIP abuse and brute-forcing in the wild, we need to open and listen to UDP port 5060 on publicly exposed IP addresses. An abuse will either start with a registration attempt through *REGISTER* or the request for a call in an *INVITE*, which we can harvest when monitoring a large number of IP addresses for incoming SIP traffic.

4.4. Dataset Collection and Processing

The data source used in this study is a network telescope of two partially populated /16 networks. All incoming data directed at the unused IP addresses are logged, which gives insights into all unsolicited traffic, such as port scans probing for available systems and backscatter traffic originating from current attacks on the Internet. As a network telescope consists solely of unused IP space, this dataset is void of user traffic. It, therefore, provides a clear overview of Internet-wide scanning for SIP services. This section provides an overview of the dataset collected from the network telescope and the method used to process SIP packets.

4.4.1. Dataset

As the network telescope consists of two partially populated /16 networks, it receives a large amount of data that needs to be processed. On average, in 2020, the telescope recorded 1.3 Tb of raw network traffic per month. To manage the size of the data, we aim our analysis at the last 6 months of 2020, providing a recent analysis of SIP scanning attempts in the wild. In addition, to allow for a longitudinal study of the TTPs used by attackers and their evolution, we include the first two months of each year from 2016 until 2020.

We select all packets sent to the SIP service running on the IANA default UDP port 5060 and discard 9.6 million packets that do not contain a valid SIP message. We observe 822 million packets containing a valid SIP header originating from 5,691 unique sources during the measurement period, sending on average 144,439 packets.

4.4.2. DATA PROCESSING

While SIP runs over UDP and is vulnerable to adversaries spoofing their source IP address, this would prevent the adversaries from receiving the response packet sent by a server. As scanners need the answer to the probe packet to establish whether a port is open, source IP addresses observed in the network telescope will be controlled by the scanning actor. We can thus group packets originating from an IP address into a scanning “flow”. Flows are aggregating the SIP packets originating from a single source IP address with an inactivity time-out of 20 minutes. After these 20 minutes of not observing a packet in the telescope, the flow is closed, and the next packet coming in from this IP address will create a new flow. These flows aim to identify artifacts inside the packets sent by a specific scanning tool.

To ensure we do not lose essential data while aggregating the packets, we leverage the structure of a SIP packet to devise a single-pass algorithm that can create a fingerprint for a scanning tool. The SIP and SDP header contain text-based variables in the form: *key:value*. To combine these text-based values, we compare the values by key and only keep the overlapping characters in the values corresponding to a key in two packets. For example if we compare the following two “from” fields: *<sip:nm@nm>;tag=1526* and *<sip:nm@nm>;tag=8711*, we keep the part that is an exact match: *<sip:nm@nm>;tag=* and replace the non-matching part with a non-Ascii character to ensure the location of all parts of the string remains the same. In this paper we use a dash to denote a non-matching character, the compared strings will thus return: *<sip:nm@nm>;tag=* - - - -. When all packets are compared and aggregated, only the constant part over all packets will be saved. By aggregating all keys of both the SIP and SDP header, we obtain a fingerprint containing parts of randomized fields and parts of fields that are constant in every packet. The characteristics captured in these fingerprints allow us to distill information about specific tools, as every implementation of a SIP scanner will generate these packets differently and, therefore, randomize header fields differently. For example, the standard NMap [15] SIP fingerprint is:

```
accept:application/sdp
call-id:50000
contact:<sip:nm@nm>
content-length:0
cseq:42 OPTIONS
from:<sip:nm@nm>;tag=root
max-forwards:70
to:<sip:nm2@nm2>
via:SIP/2.0/UDP nm;branch=foo;rport
```

As seen from the fingerprint NMap always sends the same headers in a SIP packets, as the aggragation of all packets does not show any randomizations in the header fields. A tool that randomizes more fields is SIPVicious [16], with the fingerprint:

```
accept:application/sdp
call-id:------------------------
contact:sip:100@SourceIP:SourcePort
content-length:0
cseq:1  OPTIONS
from:sipvicious<sip:100@1.1.1.1>;
            tag=38336234---------313363340\ldots
max-forwards:70
to:sipvicious<sip:100@1.1.1.1>
user-agent:friendly-scanner
via:SIP/2.0/UDP  SourceIP:SourcePort;
             branch=z9hG4bK----------------
```

In this fingerprint, the *call-id* field is always randomized together with the branch number and tag in the *from* field, but the *user-agent* is set to *friendly-scanner* in every packet. Additionally, we replace IP addresses and port numbers used inside the header with the strings “SourceIP”, “DestIP”, “SourcePort” and “DestPort” as they are commonly used in SIP headers and we will otherwise discard this characteristic if the source port is for example not constant between packets. The resulting fingerprints allow for the comparison of tools and are robust to small changes to a tool, as most other fields will still match. We have tested this method against four base- and modified versions of NMap and SIPVicious, and can uniquely identify all the different implementations.

4.5. SIP SCANNING TTPs

After applying our algorithm to the data, we obtain 187,327 separate flows, with an average of 4,393 packets per flow. Using these flows, this section will describe and analyze the behavior of SIP scanners. Using the implementation characteristics mined from the packets as described in the previous section, this section will list and analyze broadly used and custom-made tools used in SIP scanning.

4.5.1. SIP SCANNING TOOLS

As shown in section 4.4, tools generate packets with certain combinations of SIP header fields. A tool such as SIPVicious for example includes an optional header field *user-agent* and sets this to *friendly-scanner*. To detect scanning packets from this tool, analysts can match this field and filter all probes out. With an abundance of tutorials showing sysadmins how to filter probes based on this field, an actor might change this to prevent the probes from being dropped. To circumvent this detection, some actors might make minor changes to existing tools to alter the fingerprint to circumvent various scanning detection mechanisms rather than create a new tool. This section will differentiate between tools derived from known open-source tools and custom-made tools from scratch by comparing fingerprints to those of the open-source tools NMap and SIPvicious.

Tables 4.1 and 4.2 show the 21 most used tools we have been able to identify in our

	ID	Variant	First seen	Packets	Selection of SIP header fields used to identify the tool	Scan strategy	Known
Sipvicious	1	Base	2016	694 million	contact:sip:100@SourceIP:SourcePort from / to:sipvicious<sip:100@1.1.1.1> user-agent:friendly-scanner	Sequential	✓
	2	Changed user	2016	26 million	user-agent:[Random agent] from / to:sipvicious<sip:100@1.1.1.1>	Sequential	
	3	Changed fields	2016	19 million	user-agent:[Random agent] from / to:[Random name]<sip:100@1.1.1.1>	Sequential	
	4	AmooT	2018	26 million	user-agent:[Random agent] from / to:AmooT<sip:100@1.1.1.1>	Sequential	
NMap	5	Base	2016	3.6 million	call-id:50000 contact:<sip:nm@nm> from:<sip:nm@nm>;tag=root to:<sip:nm2@nm2>	Random sequential	✓
	6	Test	2020	238,623	contact:<sip:test1@test1> from:<sip:test1@test1>;tag=root to:<sip:test2@test2>	Random sequential	
	7	sip:a@a	2020	7.5 million	call-id:97 contact:<sip:test1@test1> from:<sip:a@a>;tag=1 to:<sip:b@b>	Random sequential	
	8	sip:ag@ag	2019	40,883	call-id:5000 contact:<sip:ag@ag> from:<sip:ag@ag>;tag=root content-lnegth (sic!):0 to:<sip:ag2@ag2>	Random sequential	
	9	100rel	2016	501,487	contact / to:<sip:[8 characters]@SourceIP:SourcePort> from:<[8 characters]<sip:[8 characters]@DestIP:DestPort>;tag=hetfgeeb max-forwards:30 supported:100rel	Random	
	10	Random user	2016	11 million	call-id:[10 characters]@SourceIP contact / from / to:[8 characters]@SourceIP:SourcePort user-agent:[8 characters]	Random	
	11	a123	2020	9 million	call-id:a123-bc2547-a159 user-agent:Avaya one-X Deskphone	Random	
	12	pplsip	2018	8 million	call-id:a123-bc2547-a159 contact:<sip:me@SourceIP:SourcePort> from:me<sip:me@DestIP> to:me<sip:me@DestIP> user-agent:pplsip	Sequential	✓

Table 4.1: Top 1-12 scanning tools used and a selection of their most prominent SIP header fields. Fields changing each packet are enclosed in square brackets. Header fields with exactly the same values are grouped together.

	ID	Variant	First seen	Packets	Selection of SIP header fields used to identify the tool	Scan strategy	Known
Other	13	StarTrinitySecurity	2019	360,363	call-id:startrinityvoipsecuritytestsuite contact:sip:s3@SourceIP:SourcePort from:sip:startrinityFriendlyScanner@SourceIP to:sip:penetrationTest@DestIP user-agent:StarTrinityFriendlyScanner	Random	
	14	a84b4c	2019	2 million	call-id:a84b4c76e66710 contact:<sip:alice@pc33.atlanta.com> from:Alice<sip:alice@atlanta.com> to:<sip:carol@chicago.com> via:pc33.atlanta.com	Random sequential	
	15	sip:b@localIP	2016	2,512	call-id:[32 characters]@127.0.0.1 contact / from:<sip:b@127.0.0.1> to:<sip:DestIP> date:[Current date] user-agent:Asterisk PBX	Random	
	16	sip:99999	2020	555,146	contact:<sip:99999@SourceIP:SourcePort> content-length:209 from:<sip:99999@DestIP:DestPort> to:<sip:[phonenumber]@DestIP>	Sequential	
	17	x-nat	2019	7,941	call-id:[9 numbers]@[5 letters] from / to:[5 numbers]<sip:[5 numbers]@SourceIP:SourcePort> x-nat:nothing x-serialnumber:[10 characters] server:o2-IAD_6741-2.6.3.32	Random	
	18	adoom	2018	2 million	contact:<sip:adoom@SourceIP:SourcePort> from:<sip:adoom@DestIP>[random string] to:<sip:adoom@DestIP> user-agent:Cisco-SIPGateway/IOS-12.x	Sequential	
	19	VOIP	2019	362,437	contact:sip:[phonenumber]@0.0.0.0:SourcePort from / to:[phonenumber]<sip:[phonenumber]@DestIP> user-agent:VOIP	Sequential	
	20	01146441408560	2019	3 million	contact:sip:DestIP:SourcePort from:DestIP<sip:01146441408560@DestIP> to:01146441408560<sip:01146441408560@DestIP> user-agent:[Random agent]	Sequential	
	21	Nessus	2017	5,778	contact:<sip:DestIP> from / to:Nessus<sip:DestIP:DestPort>	Sequential	✓

Table 4.2: Top 13-21 scanning tools used and a selection of their most prominent SIP header fields. Fields changing each packet are enclosed in square brackets. Header fields with exactly the same values are grouped together.

dataset, the year they first appeared, the number of packets reaching the telescope, a selection of SIP headers, their scanning strategy, and whether this tool is known when searching for it on the Internet. While SIPVicious is easily detected and the authors even provide a program that can be used to crash unsolicited scanning campaigns targeting your network [22], the standard version of SIPVicious is by far the most popular tool used to scan for open SIP servers. The alterations of the SIPVicious toolkit mostly change the *user-agent* field of the tool to either a name from a list of user-agents or a single genuine-looking SIP user-agent such as *Cisco* or *PBX*. While changing the *user-agent* will allow an attacker to circumvent some detection methods, tool 2 neglects to also change the other obvious mentions of SIPVicious in the *from* and *to* fields, where *sipvicious* is written into. Tool 3 and 4 are more thorough, as both change the fields to respectively a randomly chosen (but valid) agent name or the string "*AmooT*".

While tools based on SIPVicious change their fields and add more randomization, tools based on NMap keep the normal structure where no SIP header field is randomized and instead only change the values present in the fields. The sophistication of the tool's change seems to depend on the sophistication of the tool itself, as randomization is harder to implement in a tool statically sending the same packet than in a tool already including some form of randomization.

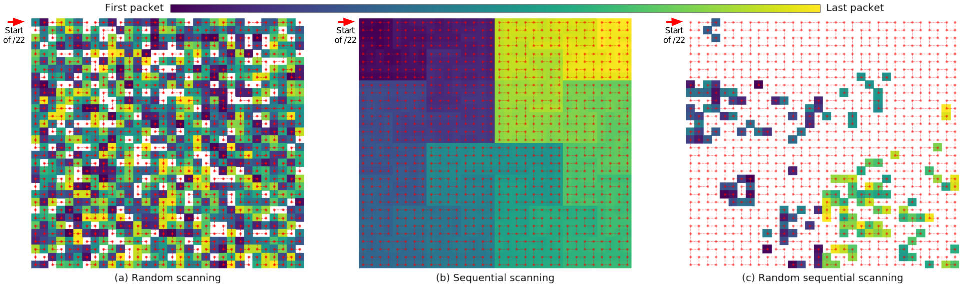
The best way to avoid detection of artifacts created by commonly used tools is to create a tool from scratch, or significantly change the tool such that no original fields are included in the packet anymore. 13 of the 21 tools discovered are not traceable back to a base version of NMap or SIPVicious and highly vary in implementation, with more customization in SIP header fields but also changes in scanning strategy. Most tools aim to spoof the *user-agent* field to something believable, but some tools such as tool 9 selects a random string for each packet. While this does not generate believable user-agents, it does circumvent any blacklisting approach as the string will be different on each probing attempt.

Most custom-made tools change header fields to expected values for a SIP packet and pick believable user agents that change every packet. However, some tools contain implementation details that make it trivial to spot that packets do not originate from a normal SIP client. NMap based tool 8 changed the SIP packet header details by including the misspelled key *content-lnegth* (sic!), which a normal SIP parser would not be able to match with an existing option in the protocol. Tool 14 even implements the SIP protocol using a dummy example found in many online tutorials, with names as *Alice* and *Carol*.

4.5.2. SCANNING ARTIFACTS

Scanning tools do not only create artifacts in SIP header fields but also fields in the IP and UDP header such as the destination IP address are set by the scanning tool. Suppose the packets are injected on the network interface without using an Operating System socket, which is mostly used for high-performance scanning [23]. In that case, fields such as the Time-To-Live or the IP identification number will also be set by the tool itself.

A tool can select the next target to scan in various ways, for example, by randomizing the destination IP address or increasing it by 1 every packet. To visualize these differences, we map our /16 ranges onto a Hilbert Curve, a 2D projection of a list that retains the distances between most points, where points close to each other in the list are also



(a) Random scanning (b) Sequential scanning (c) Random sequential scanning

Figure 4.2: Destination IP scanning patterns observed from different scanning tools in-the-wild mapped on a Hilbert curve containing a /22 netblock.

close to each other in the projection. We assign all points of the projection a color depending on when the particular IP address was scanned inside a flow. Figure 4.2 shows this projection for three different strategies seen in scanning tools. For visibility, we only include a /22 netblock in the figure.

Figure 4.2(a) shows a random scanning pattern implemented by some custom-made tools which randomize the destination IP address over the entire IPv4 space for every packet. By doing so, a scanner will hit a specific netblock with lesser intensity, as the probability of picking an IP address in this netblock is $\frac{netblock-size}{2^{32}}$. Figure 4.2(b) shows a scanning behavior where the next scanned IP address is sequential, which is implemented in SIPVicious and is thus also present in all tools based on this source code. This functionality can be changed in SIPVicious by providing a *–random* flag when running the program, we have not been able to identify any SIPVicious based activity that has used this functionality instead of the sequential scan. While a sequential scan is simple to implement, this method will scan all IP addresses in a netblock in a small time frame, making it easier for IDS systems to identify a scan is going on by matching the number of packets originating from a single source IP address in a small time window. Figure 4.2(c) shows the SIP scanning behavior of NMap and similar tools, where the IP address is randomized but generated for small netblocks in sequence, leading to all IP addresses in a netblock being scanned at once before moving on to another part of a netblock.

Another field a scanner can customize is the source port from which the packet is sent. For NMap based scanners, these source ports are randomized on every packet sent, without customization from users. However, in SIPVicious, a user can choose a port on startup of the scan with a default of 5060 where the program will bind to in the operating system. When the program cannot bind to a specific port for any reason, it will automatically move on to the next port until the program can bind to the port. Here there is again little customization as 94% of all flows are initialized with the default source port. 3% of the flows are initialized with port 5061, indicating that another process is already bound to the standard SIP port.

4.5.3. SIP MESSAGES

As described in section 4.3, there are six general types of SIP requests: *INVITE, ACK, BYE, CANCEL, OPTIONS* and *REGISTER*. Except for the *ACK* message, all of these messages

Message	Response
INVITE	100 Trying
REGISTER	401 Unauthorized
BYE / CANCEL	481 Call / transaction doesn't exist
OPTIONS	200 OK (Options in headers)

Table 4.3: SIP message types used for scanning and the responses from a server.

can be used for the detection of SIP servers, as a SIP server will respond. Table 4.3 shows responses of a plain SIP server to the message types. As every message type elicits a different response, the messages provide an adversary with different information after a server is identified. In our measurements, we have observed the *INVITE*, *CANCEL*, *OPTIONS* and *REGISTER* commands being used to scan for open services. Most scans use the *OPTIONS* message type, which is the default scan type of most tools, including NMap and SIPVicious, and notifies the scanner about server functionality.

While SIPVicious and NMap based scanners mainly use the *OPTIONS* type in the message, custom-made tools instead prefer to use *REGISTER* messages, with 6 out of the 13 custom tools solely using this message type. By using this message type, the scanner will immediately find whether an additional login is needed on the server, or that it can be used to perform for example call fraud without having the server credentials. *CANCEL* is the least often used with only tool 17 sending this message type. While a packet containing *CANCEL* could be used to stop a SIP connection forcefully, it is very unlikely for this to happen in a scan due to the entropy of call-ids. *INVITE* messages are only used by scanners 11, 12, and 20 and do not only obtain a reply from the server to identify new SIP servers, but they also instruct the server to request the establishment of a SIP session when a phone number is provided in the packet, which only occurs in 3.5 million packets. In total, 234 unique numbers are included located all over the world. We do not find any indication of why these specific numbers are called, except for two numbers included in over 100K packets each and reported to be scammers by a Norwegian phone number reputation database.

4.5.4. TOOL EVOLUTION

When scanning the Internet for a long time, actors might opt to redesign their tools to make them faster or change the payload to prevent their probes from being fingerprinted and subsequently dropped. To understand how SIP scanning actors have evolved, we identify the 21 tools found using the fingerprints in-network telescope data of the first two months of every year from 2016 to 2020 and compare the tools used by source IP addresses over the years. Figure 4.3 shows the evolution of tools used by source IP addresses, where only the IP addresses that are observed to be scanning for more than one year are included in our data. When scanning for periods spanning multiple years, they would be expected to update their tools over time, yet we see surprisingly little evolution in tooling. Most actors never change their tool even when newer tools are available that are better suited for circumventing detection systems by randomizing destination IP addresses. Even when IP addresses are observed years apart, such as an IP address

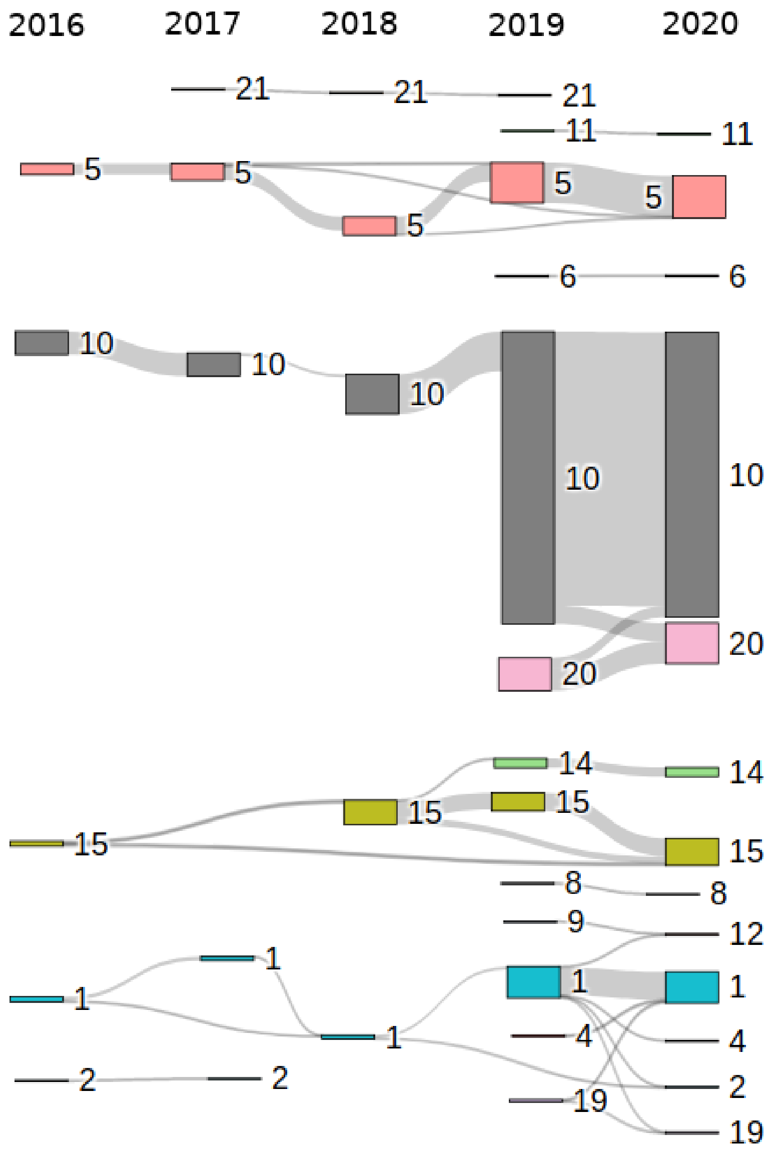


Figure 4.3: Sankey diagram of the yearly evolution of tools. Boxes show the relative number of IP addresses using a tool and the flows between boxes show the overlap between years.

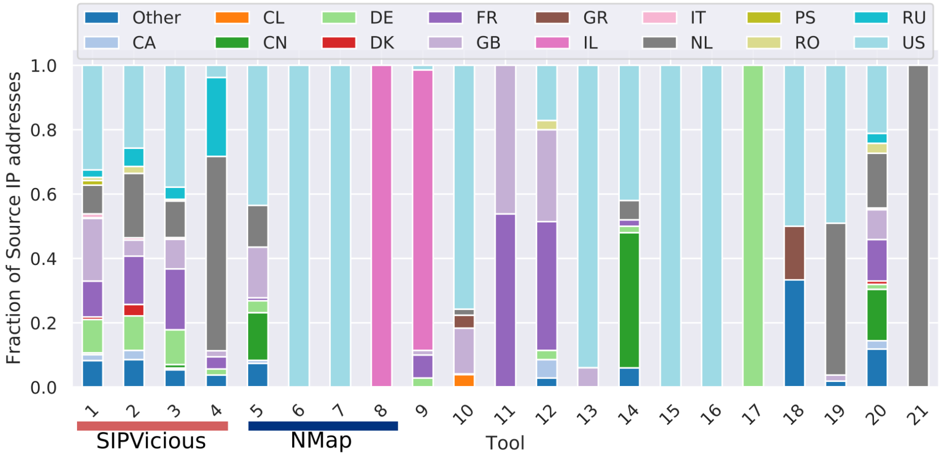


Figure 4.4: Geographical distribution of source IP addresses running specific tools.

that was only online in 2016 and 2020 running the *sip.b.local* tool, the same tool is used when they come online again.

4.5.5. Geographical distribution of tools

While scans originate from 75 countries, 65% of scanning traffic originates from only 5 countries: Germany, the United States, the United Kingdom, France, and the Netherlands. Interestingly, this does not follow usual IP allocation; China has a large address space but only 1.4% of hosts scanning SIP originates from China. Germany on the other hand has a much smaller address space while 30% of all SIP scanning hosts originate from this country. Overall scanning traffic mostly originates from countries such as China or the US [24], which is not the case for SIP scanning traffic.

Figure 4.4 shows the geographical distribution of sources using one of the tools we can fingerprint. The figure shows that tools based on SIPVicious are highly distributed, with countries with large IP space such as the US holding a significant part of the scanning activity. Tool #4 is also highly distributed in countries, but a disproportionate amount of sources is located in the Netherlands and Russia. We can trace back the scanning activity of this tool to datacenters and bullet-proof hosting providers. NMap-based tools show less geographical distribution, with only the base tool and not its derivations being used from multiple countries. Unlike SIPVicious, NMap is also used from Chinese sources. The only other two tools used from China are tool #14 which uses a standard tutorial packet and tool #20 which uses *INVITE* messages. As tool #20 is mostly distributed according to geographical IP allocation and only sends *INVITE* messages we suspect this tool to be used to place phone calls rather than discover SIP services, and therefore randomizing the Source IP address of each packet preventing the answer to these requests being observed by the actor. Custom tools are as expected much less distributed than common tools, as various actors worldwide use common tools, and custom tools are most likely used by a single actor or small groups of actors.

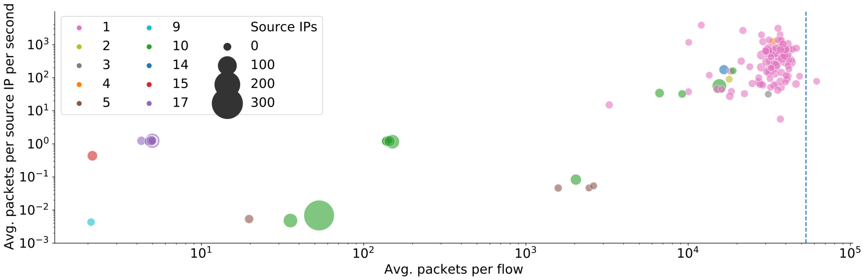


Figure 4.5: Bubble plot of scanning campaigns identified in the data colored by tool, with the observed amount of packets per IP address and the average speed observed in the telescope. A blue dotted line shows the addresses in the network telescope.

4.6. CAMPAIGN ANALYSIS

While the fingerprints identify a tool used by a source IP address, it does not allow for direct identification of scanning campaigns as the same tool might be used by different actors. In this section, we will identify behavioral traits allowing for the identification of scanning campaigns targeting SIP and analyze the behavior of these scanning campaigns.

4.6.1. BEHAVIORAL CLUSTERING

When scanning the Internet, an actor will either send all packets from one source IP address or farm out the operation over multiple controlled hosts. As creating a new tool for every collaborating host is impractical, the actor will most likely use the same tools on all machines. To verify this claim, we first create clusters of source IP addresses by grouping them based on their activity and verify that campaigns we find are conducting scans with a single tool.

A scanning campaign consists of multiple distributed hosts scanning the Internet at the same time. We cluster source IPs in three steps: First, we create a binary vector spanning our entire measurement period for every source IP address where a ‘1’ is added on every day the source IP has been observed and a ‘0’ when it is not observed. Second, we perform HDBScan clustering using pairwise Euclidean distances on the resulting vectors and select the clusters with 5 or more hosts. Third, we remove all clusters of IP addresses that are always active as they do not provide us with a distinct behavioral fingerprint but will rather cluster all campaigns scanning every day into one big campaign.

After clustering the 5,691 source IP addresses, we obtain 250 campaigns containing 2,787 (49%) IP addresses, all scanning the Internet using only one tool. Figure 4.5 visualizes these campaigns in a bubble chart where every bubble corresponds to a campaign, colored by the specific tool that was used to perform the scan. We find that common tools such as SIPVicious are used to scan the Internet at high speeds (mean of 520 packets per second in our telescope). Only limited actors are actively throttling their scanning speed to avoid detection by IDS systems. While only one campaign using SIPVicious is sending at a rate where we observe less than 5 packets per second, we observe highly distributed campaigns from custom tools that are actively scanning at very slow rates, with

4 campaigns sending even less than 200 packets per second if we extrapolate over the entire IPv4 space. We would expect that actors adapt the number of packets sent based on the number of hosts used to perform the scan and thus send half as many packets per host when scanning with 2 IP addresses instead of one. To test this, we correlate the size of the cluster with the number of packets sent per source IP address and find a much weaker relation than we expect with $R = -0.134$ ($p < 0.05$) as many campaigns scan all IP addresses from all of their hosts. For scanning speed, we do not find a significant relationship. Still, there is an indication of a small negative effect of the amount of source IP addresses used in a campaign and the average scanning speed ($R = -0.09$, $p = 0.13$), indicating a slight reduction in scanning speed when distributing scanning activities. Still, most distributed scanners are farming out their operation to obtain a significant speedup of the campaign. We identify the largest campaign uses tool #10 and is highly coordinated, evenly distributing the operation over 283 IP addresses and limiting scanning speed.

4.6.2. Scanning activity

While scanning the Internet once for open SIP servers might be enough for adversaries that aim to perform a one-time exploit, some actors will periodically update the list of open servers by performing regular scans. Campaigns using tools #10 and #17 are scanning the Internet every month, while tool #15 is used very sporadically, and no clear pattern emerges in its scanning behavior. From all IP addresses scanning SIP, 2,634 (46%) only scan the Internet once and never return. From the 54% of IP addresses generating multiple flows, only 642 (21%) are located in Germany, where the bulk of SIP scanning IP addresses are located. Instead, most recurring scanning activity originates from the US, where 1,345 (44%) of the IP addresses generating multiple flows are located. Additionally, we find that campaigns using custom tools are more likely to perform periodic scanning, with 76% of campaigns being periodic, whereas only 29% of SIPVicious-based scanning campaigns show recurrence. While over time Internet speeds increase, with only in 2017 already a global speedup of 30% [25], we do not see a significant increase in scanning speed over the years ($p = 0.79$). Instead, average scanning speeds remain constant as the variation between clusters becomes larger, where increases in scan rate are countered by campaigns actively throttling their operation. However, the number of packets sent per source IP address in a campaign sharply drops over the years ($R = -0.41$, $p < 0.001$), showing that the distribution of scanning campaigns becomes much more frequent.

4.7. Conclusion

This work analyzes tools used for SIP scanning from 2016 until 2020, finding that most actors perform large single-source scans using standard tools. We explore who is scanning and provide a method to identify which tools are used to perform a scan, uncovering 21 different tools used by various actors. We find evidence of distributed and persistent scanning campaigns, where actors scan the Internet periodically and actively reduce the number of probes sent per second by their measurement infrastructure. Over time, scanning campaigns become increasingly more distributed, making these scans harder to detect for defensive infrastructures.

REFERENCES

[1] D. Hoffstadt, A. Marold, and E. P. Rathgeb, *Analysis of sip-based threats using a voip honeynet system,* in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications* (IEEE, 2012) pp. 541–548.

[2] N. Miramirkhani, O. Starov, and N. Nikiforakis, *Dial one for scam: A large-scale analysis of technical support scams,* arXiv preprint arXiv:1607.06891 (2016).

[3] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, *Rfc 2543,* SIP: Session Initiation Protocol (1999).

[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, *Rfc3261: Sip: session initiation protocol,* (2002).

[5] A. Roach, *Rfc3265: Session initiation protocol (sip)-specific event notification,* (2002).

[6] R. Sparks, S. Lawrence, A. Hawrylyshen, and B. Campen, *Rfc5393: Addressing an amplification vulnerability in session initiation protocol (sip) forking proxies,* (2008).

[7] A. Roach, *Rfc6665: Sip-specific event notification,* (2012).

[8] S. Donovan *et al.*, *The sip info method,* (2000).

[9] D. Geneiatakis, T. Dagiuklas, G. Kambourakis, C. Lambrinoudakis, S. Gritzalis, K. S. Ehlert, and D. Sisalem, *Survey of security vulnerabilities in session initiation protocol,* IEEE Communications Surveys & Tutorials **8**, 68 (2006).

[10] U. U. Rehman and A. G. Abbasi, *Security analysis of voip architecture for identifying sip vulnerabilities,* in *2014 International Conference on Emerging Technologies (ICET)* (IEEE, 2014) pp. 87–93.

[11] H. J. Abdelnur, R. State, and O. Festor, *Kif: a stateful sip fuzzer,* in *Proceedings of the 1st international conference on Principles, systems and applications of IP telecommunications* (2007) pp. 47–56.

[12] H. Abdelnur, O. Festor, *et al.*, *Advanced fuzzing in the voip space,* Journal in Computer Virology **6**, 57 (2010).

[13] T. Alrahem, A. Chen, N. DiGiuseppe, J. Gee, S.-P. Hsiao, S. Mattox, T. Park, I. Harris, *et al.*, *Interstate: A stateful protocol fuzzer for sip,* Defcon **15**, 1 (2007).

[14] S. Taber, C. Schanes, C. Hlauschek, F. Fankhauser, and T. Grechenig, *Automated security test approach for sip-based voip softphones,* in *2010 Second International Conference on Advances in System Testing and Validation Lifecycle* (IEEE, 2010) pp. 114–119.

[15] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning* (Insecure, 2009).

[16] S. Gauci, *Sipvicious,* (), uRL: https://github.com/EnableSecurity/sipvicious.

[17] M. Nassar, R. State, and O. Festor, *Voip honeypot architecture,* in *2007 10th IFIP/IEEE International Symposium on Integrated Network Management* (IEEE, 2007) pp. 109–118.

[18] A. Aziz, D. Hoffstadt, E. Rathgeb, and T. Dreibholz, *A distributed infrastructure to analyse sip attacks in the internet,* in *2014 IFIP Networking Conference* (IEEE, 2014) pp. 1–9.

[19] A. Dainotti, A. King, K. Claffy, F. Papale, and A. Pescapé, *Analysis of a/0 stealth scan from a botnet,* IEEE/ACM Transactions on Networking (TON) **23**, 341 (2015).

[20] E. Raftopoulos, E. Glatz, X. Dimitropoulos, and A. Dainotti, *How dangerous is internet scanning?* in *International Workshop on Traffic Monitoring and Analysis* (Springer, 2015) pp. 158–172.

[21] J. M. Ceron, K. Steding-Jessen, and C. Hoepers, *Anatomy of sip attacks,* ; login:: the magazine of USENIX & SAGE **37**, 25 (2012).

[22] S. Gauci, *Svcrash,* (), uRL: https://github.com/topics/svcrash.

[23] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, *Zippier zmap: Internet-wide scanning at 10 gbps,* in *8th USENIX Workshop on Offensive Technologies (WOOT 14)* (USENIX Association, San Diego, CA, 2014).

[24] Z. Durumeric, M. Bailey, and J. A. Halderman, *An internet-wide view of internet-wide scanning,* in *23rd USENIX Security Symposium (USENIX Security 14)* (2014) pp. 65–78.

[25] Speedtest, *Internet speed increase in 2017,* https://www.speedtest.net/insights/blog/global-speed-2017.

5

Understanding adversarial TTPs

Amplification attacks generate an enormous flood of unwanted traffic towards a victim and are generated with the help of open, unsecured services, to which an adversary sends spoofed service requests that trigger large answer volumes to a victim. However, the actual execution of the packet flood is only one of the activities necessary for a successful attack. Adversaries need, for example, to develop attack tools, select open services to abuse, test them, and adapt the attacks if necessary, each of which can be implemented in myriad ways. Thus, to understand the entire ecosystem and how adversaries work, we need to look at the entire chain of activities.

This paper analyzes adversarial techniques, tactics, and procedures (TTPs) based on 549 honeypots deployed in 5 clouds that were rallied to participate in 13,479 attacks. Using a traffic shaping approach to prevent meaningful participation in DDoS activities while allowing short bursts of adversarial testing, we find that adversaries actively test for plausibility, packet loss, and amplification benefits of these servers, and show evidence of a "memory" of previously exploited servers among attackers. In practice, we demonstrate that even for commonplace amplification attacks, adversaries exhibit differences in how they work, and these behavioral differences allow us to cluster attacks to campaigns.

This chapter has been been published as *"Scan, Test, Execute: Adversarial Tactics in Amplification DDoS Attacks"* by **Griffioen, H.**, Oosthoek, K., van der Knaap, P.C.F., Doerr, C. in the ACM SIGSAC Conference on Computer and Communications Security (2021).

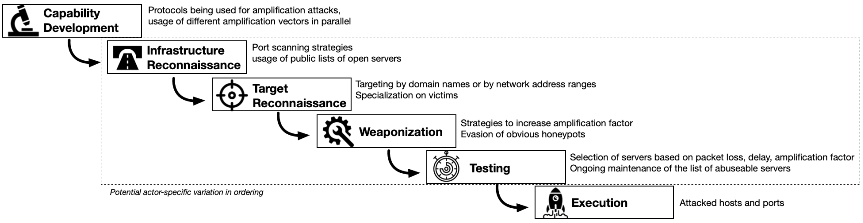

Capability Development
Protocols being used for amplification attacks, usage of different amplification vectors in parallel
Infrastructure Reconnaissance
Port scanning strategies usage of public lists of open servers
Target Reconnaissance
Targeting by domain names or by network address ranges Specialization on victims
Weaponization
Strategies to increase amplification factor Evasion of obvious honeypots
Testing
Selection of servers based on packet loss, delay, amplification factor Ongoing maintenance of the list of abuseable servers
Potential actor-specific variation in ordering
Execution
Attacked hosts and ports

Figure 5.1: To understand how adversaries work, we can separate DDoS attacks into separate phases and activities and investigate actor-specific techniques and commonalities in behavior between adversaries.

5.1. INTRODUCTION

In order to impair the availability of Internet services, adversaries commonly deploy distributed denial-of-service (DDoS) attacks. Such attacks typically take one of two forms: (1) they target communication protocols, for example, by exhausting a maximum number of concurrent connections, a classic example of this being the TCP SYN flood; or (2) they target the connection itself by flooding the victim with large volumes of unwanted data so other data cannot come through anymore. As such packet floods would require the attacker to generate lots of traffic, these assaults are expensive for the adversary. For this reason, they are typically executed as reflection attacks, where the attacker forges a request on behalf of the victim to a third party by address spoofing and lets the answer be delivered to the impersonated victim. If the request is much smaller than the resulting response, the cost and effort for the adversary are minimal.

Amplification attacks are conceptually easy, straightforward to implement, and enable large attacks using minimal resources. The attacker only needs to find open services running on a connectionless protocol with an attractive amplification ratio. The importance of this attack vector to DDoS attacks has led to several research projects that looked into the ecosystem of these amplification attacks over the past decades. Notable contributions to the field have been made by Paxson [1] and Rossow [2], who created *amplification honeypots* to capture attacks in-the-wild, or Krämer et al. who ran 21 nodes with their AmpPot to discover previously unknown details on amplification attacks and their victims [3].

While several studies look at amplification attacks in the wild, little is known to date about the activity behind these attacks. In other words: how do adversaries plan, prepare, and execute these attacks? For example, when attackers collect lists of abusable services, do they indiscriminately use any servers they encounter, or do they carefully test, select and curate a portfolio of amplifiers? Do adversaries go for “textbook” attacks, or do they make efforts to understand and adjust their actions to cause the most damage?

To help answer these questions, we can look at DDoS attacks not just in the context of the resulting packet flood but aim to understand the entire chain of events that led to the execution and (successful) completion of the attack. As shown in figure 5.1, before packets flow towards the victim, adversaries will have to go through a series of planning and preparatory steps, understanding which attack vector to use, locating infrastructure that could be abused for amplification, selecting how the victim should be targeted, implementing software and payloads to implement the attack, to eventual testing. Each of these steps can be implemented in various ways. By studying adversarial behavior, we gain better insights into the threat landscape of DDoS attacks assisting in developing better defenses.

In this paper, we perform an investigation into the different phases of DDoS attacks and in-depth analysis into the steps and techniques used by adversaries while developing, preparing, and executing such attacks. By deploying 549 honeypots in 5 public clouds around the world over a period of 3 months, we have collected a total of 605,181 reconnaissance and testing activities leading up to 13,479 attacks consisting of 720,995 individual attack flows on 8,315 victim systems. Through careful design of experimental conditions where we use a new traffic shaping approach to ensure we are not actively participating in these attacks, our study elicits adversarial behaviors by monitoring amplification attacks through an infrastructure an order of magnitude larger than previous work and provides significant new insights into the threat landscape that have been primarily investigated based on the attacks themselves. We show that a bulk of these 13,479 attacks can be linked together based on shared adversarial techniques, tactics, and procedures (TTPs). With this work, we make the following main contributions:

- We are the first to investigate the full ecosystem of amplification DDoS attacks and the impact of amplification power offered by a service on the level of abuse.
- We demonstrate a wide variety of ways how attackers maintain and curate their lists of abusable servers. We find that there is a “memory” on the Internet for some protocols that IPs once ran a service. Adversaries still return to these weeks after the service has stopped responding to requests.
- We show how sophisticated attackers optimize their bandwidth usage by conducting attacks in pulses rather than a constant packet stream, reducing the cost of attacks.
- We demonstrate that the number of honeypots needed to obtain sufficient attack coverage is much higher than previously shown in related work and that the threat landscape is more diverse than previously thought once we employ a large number of honeypots.
- We show that while the bulk of the threat landscape consists of unsophisticated actors with little capability, there are very knowledgeable adversaries who test services they abuse.

5.2. Related work

While DDoS attacks have historically been conducted using botnets as a means to amass enough firepower [4], the largest attacks to date were done as amplification attacks, with Google reporting attack peaks at 2.5 Tbps [5]. The ease with which these amplification attacks create large traffic floods popularized this method, and researchers report a median of 1,930 attacks per day since 2014 [6]. Many works exist analyzing observed amplification attacks by setting up *amplification honeypots*. In 2001, Paxson conducted the first work on capturing these amplification attacks “in the wild” by setting up these fake amplification services [1]. In more recent work, Rossow [2] provides an in-depth study into attacks performed in 2014, analyzing captured traces on various protocols, and evaluating open amplification services on the Internet. Subsequently Kührer et al. have performed significant work to reduce vulnerable NTP services and IP address spoofing [7], complemented by others that have focused on amplification using specific protocols, such as DNS [8–12] or NTP [13–16]. The largest study on amplification attacks to date has been performed by Thomas et al., who capture attack traces using a mean of 59.7 honeypots over 1010 days [6].

While the work on executed amplification attacks is abundant, research on the ecosystem behind these attacks is scarce. The first work to address this is from Krämer et al. [3], who deployed honeypots with different modes to capture differences in the behavior of adversaries using these systems. The authors used a fixed rate-limiting threshold to prevent their experiment causing serious damage, but identify the concern that this rate-limit might lead to honeypots being detected as such by scanners. While work by Krämer et al. [3] has tried to establish how many honeypots are needed to capture a significant part of these attacks by setting up 21 honeypots in different geographical regions, the lack of understanding on how "smart" adversaries select their amplification servers can significantly bias the results as these attackers might test the devices and consequently stop using them. To further understand the ecosystem of amplification attacks, Krüpp et al. analyzed the reconnaissance activity performed as part of DDoS operations [17]. In another experiment, Krüpp et al. used a set of 11 honeypots to attribute their use to certain booter services [18]. While the authors were able to link usage of their honeypots to actual booter services, they only found their results representative for DNS and NTP as they missed portions of self-attacks for SSDP and CHARGEN. Other works have also investigated the use of booter services in order to gain more understing of the ecosystem [19–22]. But only some of them have specifically explored the use of amplification DDoS honeypots in the context of these services, such as Kopp et al. [23] who have investigated spatial and temporal trends of DDoS attacks launched by booter services.

Contrary to honeypots aimed at compromisation attempts [24], it is currently unknown to which extent adversaries test and abandon amplification honeypots. While some works set initial steps into this direction by creating different configurations in the honeypots [3], none of the prior works has deployed a large enough network of honeypots to accurately capture these adversarial differences nor used a traffic shaping method to allow adversarial testing traffic. As prior works on DDoS attacks have mostly considered the execution and only look at a narrow set of the entire chain of attacks required for the attack to be successful, it is currently unknown to what extent adversaries test systems to use in their attack and whether they are actively weeding out known honeypots. The closest work to address the full chain of steps is by Krupp et al. [17] who analyze the scanning behavior between different attacks, but do not investigate other activities around the preparation or testing of an attack. Analyzing the entirety of these attacks will allow researchers to describe and discuss these attacks in more detail.

In this work, we go beyond adversarial scanning and set up various experiments using a configurable honeynet an order of magnitude larger than prior work. In a series of experiments, we not only investigate what how adversaries discover and abuse amplifiers, but study all steps adversaries take leading up to the DDoS attack itself. This allows us to study the sophistication of adversarial activities, and a deeper assessment of the threat landscape than before.

5.3. DDoS Attack Chain and System Setup

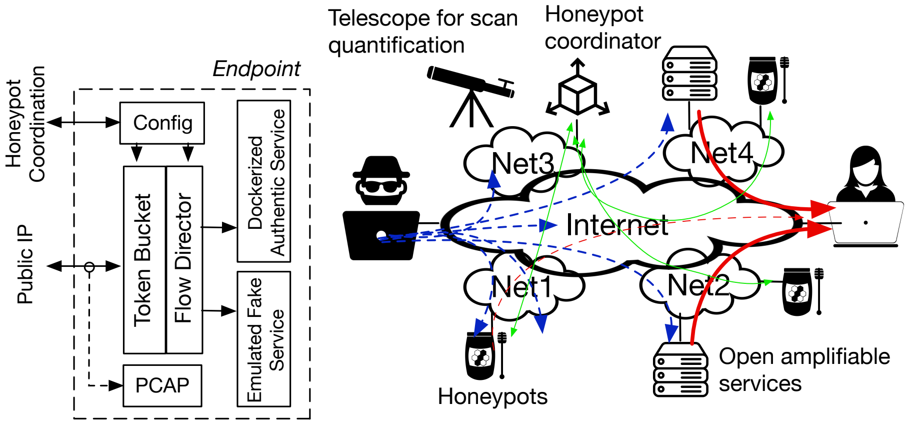
Previous studies discovered several methods used by adversaries to select amplification servers and victims. However, as discussed in the previous section, most research is limited to the attack itself and does not study the sequence of events before and during the attack nor investigates the adversarial characteristics behind the attack. In this study, we aim to address this gap through configurable, adaptive honeypots, which enable us to investigate how actors work towards attacks, how they prepare, what they look for in a service to abuse until the actual execution of the attack.

The DDoS Attack Chain. Amplification attacks typically do not suddenly emerge, but the onset of an attack on a victim results from a series of preparatory steps by the adversary that we can identify and measure. In this paper, we structure these activities along a sequence of six consecutive phases as depicted in figure 5.1, which allows for an effective discussion and thus a better understanding of how DDoS operations are realized. Although the concrete attack on a particular victim may be unique, adversaries can be assumed to recycle much of the supporting activities, such as the list of abusable servers or attack methods, which allows the recognition and linking of individual attacks, as well as to provide a better understanding of the ecosystem and an early warning system for upcoming activities.

As shown in figure 5.1, a pre-requisite for a DDoS attack is some *Capability Development* by the adversary. Depending on attacker sophistication, this can range from modifying standard scripts to tailor-made attack vectors. Next, the attacker would have to perform *Infrastructure Reconnaissance* to identify and collect systems that can be used as part of the attack. In terms of amplification attacks, the attacker would locate available amplifiers, but other attacks could recruit unsecured IoT devices or bots as a platform for the attack. The adversary then needs to know where to attack the victim. In *Target Reconnaissance* the fundamental details necessary for the attack, such as the victim's IP address, open ports, and vulnerabilities, are collected. During *Weaponization*, the output of the previous three phases is fused to craft the attack technique that will be used to perform the attack. To assess whether the developed weaponization is suitable to harm the target or even work with the identified infrastructure, some *Testing* may be required, before finally, the *Execution* commences, and is 'delivered' to the victim.

System Overview. In order to observe these adversarial TTPs along the DDoS attack chain, we developed a framework for the operation and management of honeypots. The system aims to enable deployment and exposure of commonly used services towards the Internet and allow us to facilitate attacker-dependent, configurable responses to incoming requests dynamically.

Figure 5.2(a) shows an architectural diagram of the honeypot. In order to provide an authentic appearance, each honeypot runs a selection of containerized services, for example, *BIND* to service DNS requests or *ntpd* for NTP queries. Incoming requests from the Internet are either proxied to these instances or to an emulated service that returns (obviously) fake answers. This emulated service does not implement all features the pro-



(a) Honeypot Design (b) System Architecture

Figure 5.2: Deployed honeypots maintain a control connection to a coordinator to perform effective rate-limiting.

tocol requires in order to test the interaction and reaction of adversaries to services. The Bandwidth Amplification Ratio (BAF) [2] of the honeypots are listed in Table 5.1. In addition to logging the interactions of adversary and backend services at the application layer, packets are also recorded at the link layer to discover reconnaissance actions like port scanning as well as implementation characteristics in the attack packets to potentially relate individual attacks and adversaries.

Our behavior differs depending on how the honeypots are contacted. As shown in figure 5.2(b), all honeypots are in constant communication with a central coordinator, which tracks the requests sent to all 549 honeypots. Suppose the incoming requests exhibit features of selected testing (e.g., only individual honeypots are triggered, short bursts instead of a packet flood are requested, requests are typical reconnaissance but no amplification packets, etc.). In that case, the coordinator instructs the honeypot to comply, if the requests show the characteristics of an attack, which we can detect due to our distributed setup, which will be explained in Section 5.5.2, the configuration is adjusted across *all* honeypots to reduce the data rate of all 549 servers towards the IP address to a maximum of 0.5 Mbps. This value was chosen to result in a minimal impact and ethical operation of the honeypot system. In several rounds of tests before the actual experiment, we verified whether adversaries would detect this traffic shaping and subsequently stopped using these systems. This was not the case. We used only honeypots at Digital Ocean for these tests and ran the experiments on different IP addresses. As we find no statistical differences between cloud locations, this verification did not bias our experimental setup.

Experimental Design. In order to discover how adversaries select their infrastructure and victims as well as perform their implementation and execution, we rolled out different configurations to parts of our honeynet. Specifically, we are interested in the influence of:

1. **Amplification Factor**: do adversaries search for and focus on servers with the highest amplification?
2. **Obvious Honeypots**: are adversaries monitoring and discarding servers if they do not fully implement the protocols or even explicitly announce to be a honeypot?
3. **Suspicious systems**: are adversaries actively investigating the system, and do they behave differently if an unbelievable number of open services is installed?
4. **Packet Loss**: do adversaries test and discard services if they are not constantly answering or drop requests?
5. **Packet Delay**: do adversaries select servers based on the response time and speed?

We have structured our system as follows: honeypots are split into four groups with low and high amplification ratios providing a correct response, and low and high amplification ratios providing a response that is obviously a honeypot by replying with a message such as *"This is a honeypot attack detector, usage is logged and rate-limited"*. So that size is not a confounding variable; these responses of the obvious honeypots are equally large as real responses. Additionally, we vary the number of services operating on the servers. To ensure that the results are not biased due to the location of the honeypots, we distributed groups equally over five different clouds and availability zones. Throughout the experiment, we alter the behavior of this setup, for example, by dropping packets or delaying the responses, to identify the reaction of the adversaries.

Ethical considerations. DDoS amplification honeypots run the risk of participating in attacks when being used by adversaries. In the past, honeypot systems were typically implemented to drop attack packets entirely [2], the same method is used for other works [3, 6, 25], however, this comes with the complication that the research community would not be able to observe the more sophisticated actors who would consequently abandon honeypots after testing. In order to balance the need to understand the threat landscape and actor actors while not irresponsibly participating in DDoS attacks, we implemented a token bucket as a flow shaper in front of the honeypots. Figure 5.3 visualizes the concept, where at a specific rate, tokens are being dropped into a bucket of maximum size. Every packet that leaves our amplification honeypots consumes a token, and if none are available, the packet is dropped. Token buckets have the advantage that they can be configured to throttle attack traffic aggressively but at the same time allow short temporary bursts of activity that we would see during testing. Adversaries would therefore consider the honeypot as a normal server. However, during an attack, we would not create a significant impact, leading to a loss of attack power for the adversary when using our system instead of a real amplification server. How we behave during testing and sustained attack is governed by the fill rate and bucket size. We chose a bucket depth for a peak of 25 packets per second, which we experimentally determined in our pre-study. This value was 20% above the maximum traffic we observed from adversaries during testing. Thus the honeypots would pass all testing procedures we observed. After the bucket is emptied, the *combined* attack force of our honeypot drops down to a maximum packet rate

Table 5.1: Response sizes of the honeypot setup in bytes and the Bandwidth Amplification Factor (BAF) [2] per protocol, the allocation over cloud providers and different experiments.

	Responses	Real small	Real large	Fake small	Fake large
RIP	Bytes	84	524	88	413
	Max BAF	3.5	21.8	3.7	17.2
CharGen	Bytes	94	1,406	94	1,450
	Max BAF	94	1,406	94	1,450
QOTD	Bytes	45-50	1,450	53	1,437
	Max BAF	45-50	1,450	53	1,437
SSDP	Bytes	272	430	277	429
	Max BAF	2.3	3.7	2.4	3.7
NTP	Bytes	Varies	Varies	Varies	347
	Max BAF	0	46+	0	43.4
DNS	Bytes	83	Varies	83	352
	Max BAF	1.6	1.6+	1.6	6.8

Cloud provider	Total servers	Real small	Real large	Fake small	Fake large
AWS	171	47	52	36	36
Azure	24	6	6	6	6
Digital Ocean	166	45	49	36	36
Google	124	32	34	29	29
OVH	64	16	16	16	16

Experiment	Total servers	Real small	Real large	Fake small	Fake large
Single protocol	120	30	30	30	30
Packet loss	60	15	15	15	15
Packet delay	60	15	15	15	15

of 0.5 Mbps, which is less than $\frac{1}{160}$ of the average Internet speed [26]. Sending a fraction of the traffic that a *single* Internet user could send, we are not actively hampering victim systems. At the same time, we are in practice able to collect data on adversarial behavior in DDoS attacks.

We have received approval from our IRB for the design, and additionally from our information security officer (ISO) whether the honeypots, rate limiting, and maximum data rates were responsible choices. No further requirements were put forth, and the experimental setup and data rates were confirmed as adequate choices.

5.4. Datasets

Our 549 honeypots were distributed in the public clouds of the five largest cloud providers Google, Amazon, OVH, DigitalOcean, and Microsoft, and placed in availability zones in North America, Europe, Australia, and Asia between 31 August 2019 and 30 November

Table 5.2: Timeline of data collection phases.

Phase	From	To
Passive (scanning baseline)	31-08-2019	07-09-2019
Active	07-09-2019	27-09-2019
Passive (attacker memory)	27-09-2019	30-11-2019

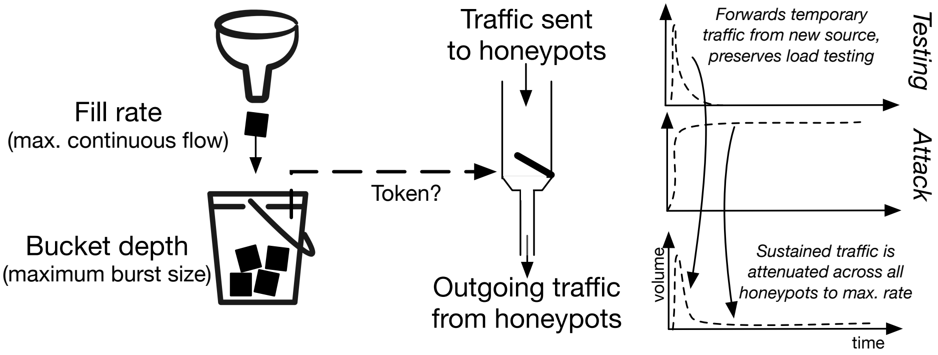


Figure 5.3: The token bucket limits the maximum outgoing flow, but preserves unseen temporary bursts and testing.

2019. The distribution of different servers over cloud providers and experiment groups is listed in Table 5.1. As we are interested to understand the entire chain of adversarial preparation and maintenance behavior and understand if adversaries work ad-hoc or build repositories of known servers, the honeypots were not activated immediately on the first day. Instead, for the first week, honeypots merely recorded all scanning activity without responding to establish a baseline. Subsequently, the honeypots actively answered requests for 20 days. Finally, we again switched to passively recording scan and attack packets that were directed towards us for another nine weeks (see Table 5.2) to understand the "memory" of the attack landscape, whether attackers use previously discovered amplifiers even though they are not active.

Aside from the application and link-layer traces from the honeypots, we used two additional datasets to further quantify how adversaries scan the Internet for open amplification services. While these datasets do not provide additional insights into amplification attacks, they do provide insights into the reconnaissance phases of an attack. First, a large network telescope of three partially populated /16 networks with 65,000 IP addresses providing a historical record of scan traffic for the past five years, and second, scan activity against the distributed telescope of the provider Greynoise. This allows us to differentiate whether attackers focus on select public clouds or perform broad scans across the entire Internet.

The number of honeypots needed to perform accurate measurements is much higher than previously believed. As we will show later, the threat landscape of actors per-

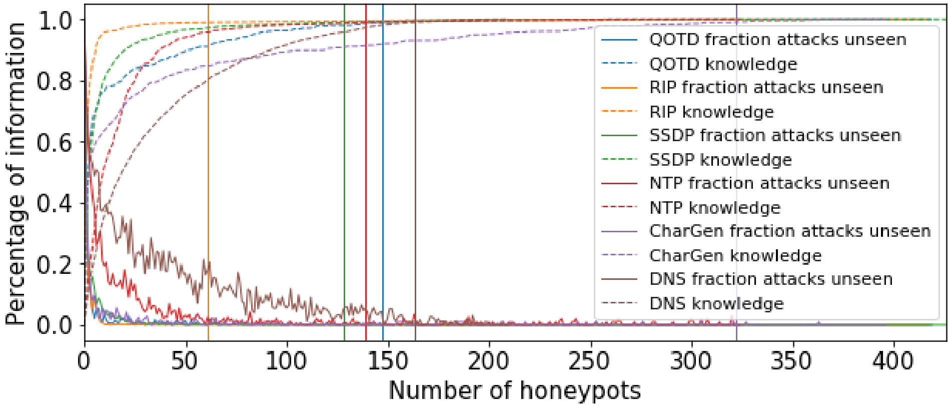


Figure 5.4: New information per extra honeypot. Dotted lines show the total number of attacks identified and regular lines show the extra information added per honeypot. Vertical lines indicate knowledge of 99% of attacks.

forming amplification DDoS attacks is highly heterogeneous regarding techniques and resources applied. For instance, adversaries often do not exhaustively search the Internet for all open, amplifiable services. However, they are content with small sets of amplification servers, since on average, only 41 of our 549 honeypots were used in a given attack campaign. This means that to obtain a good overview of the ecosystem, it is necessary to operate many honeypots to capture small attacks and avoid bias towards adversaries conducting massive trawling through the entire Internet. Figure 5.4 shows the convergence of the spectrum of attacks seen as a function of honeypots in operation. Even in comparatively simple protocols such as NTP and quote-of-the-day (QOTD) where DDoS attacks presumably all look similar, the heterogeneity when looking at the entire sequence of attack steps is so large that as many as 150 honeypots are needed to capture 99% of actor behavior. This shows the constant evolution of the ecosystem, as previous work from 2015 identified that the majority of attacks were captured in 21 honeypots [3]. The honeynet size of more than 500 servers – an order of magnitude more than in previous studies – is thus necessary to provide a good understanding of the DDoS threat landscape.

5.5. AMPLIFICATION DDOS IN THE WILD

When we activated our honeypots after the week-long baseline, it took mere hours for the first adversaries to abuse our infrastructure in attacks. During its 20-day activation, we recorded 13,479 separate attack campaigns, targeting 8,315 unique source IPs located in 4,340 /24 subnets. Altogether, our honeypots collected 448 GB in amplification requests attributed to attacks, for which we generated on average 0.12 Mbps from our system towards a victim. In this section, we discuss these attacks using the model from figure 5.1.

Table 5.3: Number of scanning IPs per protocol.

Protocol	Residential	Hosting	Research
NTP	855 (50.1%)	941 (24.8%)	489 (25.1%)
DNS	317 (38.3%)	179 (9.3%)	658 (52.4%)
SSDP	341 (63.3%)	138 (9.0%)	425 (27.7%)
CharGen	63 (62.5%)	8 (5.3%)	51 (32.2%)
RIP	31 (73.1%)	2 (4.7%)	32 (22.2%)
QOTD	24 (50.5%)	2 (2.0%)	28 (47.5%)

5.5.1. CAPABILITY DEVELOPMENT

To perform an attack, an adversary first needs to develop a capability to execute the attack with a specific vector. In this study, we focus on six commonly abused protocols in amplification attacks [2]: NTP, DNS, SSDP, CharGen, RIP, and Quote-of-the-Day (QOTD). Overall, we find adversaries are abusing these long-established protocols with conceptually similar vectors. We however also see glimpses of innovation and capability development where attackers are not bound to a specific protocol and perform attacks using multiple protocols simultaneously. This however only occurs infrequently, as only 252 of the 13,479 attacks leverage multiple protocols in their attack.

5.5.2. INFRASTRUCTURE RECONNAISSANCE

Before a server can be abused, an adversary needs to know about its existence. This is typically executed through port scanning, which our system can distinguish from attacks, as during a scan, only a few packets are sent to one honeypot, wherein an attack, many packets are sent from one IP address to multiple open services. To identify scanners, we apply our auxiliary telescope datasets and the honeypot servers that are only running a subset of services. If source IPs connect to dark IP addresses or honeypots where a service is not running, these requests are scanning and not part of attack usage. We experimentally derived that actors use up to 20 packets from the same source IP to test honeypots. In the following, we use this threshold to classify probes below this as scanning, while flows of more than 20 packets towards two or more of our honeypots are labeled as an attack.

Our honeypots report 3,650 distinct IP addresses sending scanning probes to our system, in which NTP, DNS, and SSDP are much more popular than the other protocols in terms of scanning activity. Based on the data in the telescopes, we can identify whether hosts scan the entire Internet or target the cloud providers in which our honeypots are located. Surprisingly, only 56% of all scans seem to target the Internet indiscriminately, and 44% was only observed on our honeypots in the cloud locations. Additionally, we find that 39% of IP addresses targeting only cloud instances are hitting only one cloud location, which we can largely attribute to scans originating from the same /24 subnets targeting their scans towards separate cloud locations spread across the netblock. The remainder of the scans targeting small parts of the network might originate from botnets segmenting their scanning activity or from attackers probing a single cloud provider. We observe a small fraction (6%) of all attacks only using amplification servers located in a single cloud, which we will show later, are not performed by sophisticated attackers.

Table 5.4: Distribution of connections per country.

Country	Residential	Hosting	Research
US	287 (28.8%)	361 (7.7%)	1377 (63.5%)
TR	500 (35.5%)	819 (64.5%)	0 (0%)
CN	291 (100%)	0 (0%)	0 (0%)
NL	90 (78.4%)	14 (5.4%)	92 (16.2%)
GB	43 (75.9%)	10 (18.3%)	108 (5.8%)
FR	49 (79.7%)	4 (6.0%)	32 (14.3%)
Other	377 (88.5%)	31 (7.3%)	18 (4.2%)

Research scans are prevalent, and we need to account for these to avoid biases. Not all scanning traffic is malicious, as services such as Shodan, Censys, and Rapid7 scan the Internet for research purposes and identify themselves as such using hostnames such as *worker-01.sfj.corp.censys.io* and *census6.shodan.io* and originate from a known block of IP addresses. To obtain an accurate view of scanning with malicious intent, we manually classified IP addresses either as research, hosting provider, or residential, based on reverse DNS, BGP data, and cloud customer IP ranges. Table 5.3 shows the result per protocol. It is striking that overall, research-based scanning accounts for more than 30% of all scans and 37% of all IP addresses. For DNS, research-based scans even amount to over half of all scanning. While not common in scanning research, it appears that careful curation is needed.

When considering the geolocation of scanning IP addresses, most of the hosts are located in the US. However, as listed in table 5.4, the large majority of IP addresses scanning from the US belong to research institutions. In our analysis, we thus excluded research scans from our dataset and instructed our honeypots not to respond to these projects. By excluding scanning institutions such as Shodan and Censys [27, 28] our honeypots will not appear on their publicly available lists, meaning that attacks can only be a result of scanners from the other categories: residential and hosting IP addresses.

Responsive IPs make scanners come back twice as fast. As shown in figure 5.5, the number of IP addresses scanning our honeypots sharply increased after the services started to respond after a week of passive listening. Richter et al. [29] have investigated this by comparing scanning traffic in a CDN against a network telescope and find that the presence of active services would trigger adversaries to intensify their activities. By comparing the difference of our baseline and the active experiment, we find that this increase is primarily driven by intensified rescanning – IP addresses come back to a vulnerable server on average two times faster than when the scanned protocol is not present on the machine – and to a lesser degree on new IP addresses emerging. After the infrastructure returns to passive mode, previously connecting IP addresses slowly go back to the baseline. As we will show later, the attacker's "memory" of which services used to be available at an IP easily spans months.

Initial testing happens during scanning. During a scan, adversaries are already interested in the first quantification of the system. We have seen that even during the initial scan, the amplification supplied by the server is tested; adversaries would use the same packets that will eventually be used in attacks to immediately establish whether a server

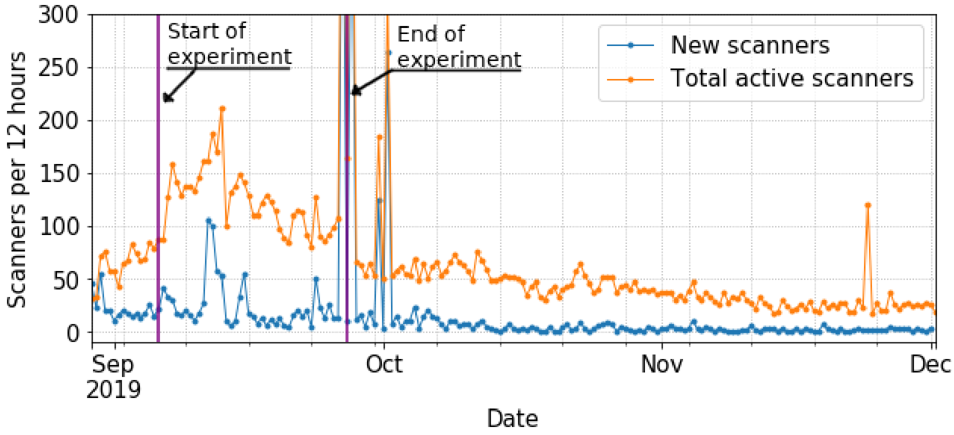


Figure 5.5: New/total active *non-research* scanners. After activation on Sep 7th, traffic from recurring scanners increased and only slowly declined after the shutdown on Sep 27.

can be used later on based on the observed amplification power. We found this for all protocols except DNS, where scans primarily request the BIND version running.

5.5.3. TARGET RECONNAISSANCE

Compared to DDoS attacks that exploit a particular vulnerability, the amount of target reconnaissance activity is minimal in case of amplification attacks. Though an attacker can direct any kind of data to the victim to consume the bandwidth, we still observe slight nuances in how victims are targeted in practice.

Victimization has changed since previous studies. The majority of victims are located in the US and China, with 846 and 602 subnets being attacked, consistent with previous works and logical given IP address allocation. Jonker et al. [25] found attacks mostly follow Internet usage patterns with exceptions of, for example, Japan, Russia, and France. We however observe a different disproportional share of attacks on countries such as South Africa, Poland, or Kuwait. We find that this disproportionality mainly stems from different services located in these countries, such as hosting providers which are extensively attacked during our study. As our experiments only ran for a limited time due to the scale of our honeypot system, we cannot identify seasonality in the attacks, and we could therefore have a bias towards large attacks that happened during our measurements. However, as we identify significant changes in attack traffic than previous work, future work should identify whether there are trends in DDoS attacks or are largely spurred by opportunistic attacks.

Only 51% of all targeted IP addresses had a domain name pointed to it during the attack based on passive DNS and a database of daily active domain crawls. While we expect most DDoS attacks to be targeted against servers that would commonly have a domain name pointed to them, we find that there are also a large number of DDoS attacks on residential IP address space without domain names. As the domains are not only pointing towards web servers but also to, for example, Minecraft servers, we queried the Shodan

API for active scanning data to find open ports on the victim devices. Shodan lists the open ports for 1,289 (9,6%) of the IP addresses that were attacked. While Jonker et al. [25] associate most UDP amplification attacks with online gaming, we find that only a small part of the attacks we have recorded are targeting game servers or ports used by multiplayer games.

Attacks towards large domains are rare. Load balancing interferes with an attacker's objective, as taking down a host leaves a service unimpaired, redirecting users to one of the hosts that are still online. Performing an attack on these services requires an attacker to attack all fronts, using the domain name resolution to resolve all IP addresses of this service. By identifying attacks on IP addresses hosting common domain names using passive DNS and active domain lookups, we can find attacks conducted on multiple IP addresses addressed by a single domain name. In total, we find 862 domain names for which more than one IP address was part of an attack and find that not only do attacks target multiple IP addresses for the primary domain but also multiple subdomains simultaneously. In an attack on the Discord service, providing chat rooms targeted at gaming communities, 61 servers were targeted, aiming to bring down a specific region indicated by domains such as *russia17.discord.gg*. Many of the attacks directed to multiple IP addresses hosting domains are targeting CDN providers such as *yunjiasu-cdn.net* or *googleusercontent.com*. As the attacks solely target the IP addresses where a domain is hosted and no other addresses belonging to the specific company, these attacks could only have been conducted using DNS lookups.

Subnet-based attacks are increasingly rare. Attacking an enterprise can be daunting, as the resources or large organizations allow them to switch between different IP addresses to mitigate an attack rapidly. To avoid this, attackers choose not only to attack the IP addresses running a service but also to attack entire /24 subnets. These attacks do not target hosts but exhaust the router's capacity in front of these hosts, rendering all hosts unreachable. We find these attacks are increasingly rare as opposed to [6], as we observe 12 complete subnets being attacked with another 29 subnets being partially attacked during our entire study while Thomas et al. identify on average 5.39 of these attacks per day. All full subnet attacks we have identified are aimed at shared hosting providers, for which a single set of amplifiers is used for the entire attack across all IP addresses. This is implemented using a round-robin for the IP address of the subnet, which are all hit consecutively in the full-subnet attacks observed.

Attacks are scheduled to hit during prime time. While related work has not considered the time a DDoS attack is conducted from the victim's perspective, we find that DDoS attacks are aimed at a victim during times that there would be the most impact. Figure 5.6 shows the start of attacks in the local timezone of the victims for the top 5 attacked countries. We find that attacks occur mainly in the afternoon and evening, where many people would be using the attacked services. Based on open ports listed in Shodan, attacks on websites occur evenly during the afternoon and evening. Interestingly, we find that 83% of attacks on game servers occur after 6 PM as adversaries hit these servers at their busiest times.

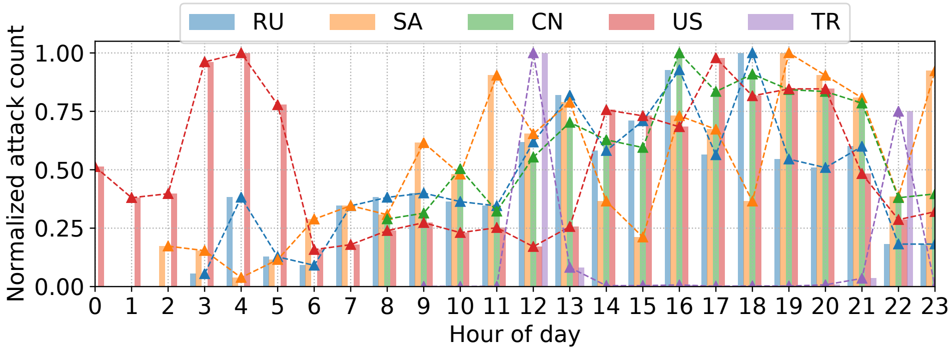


Figure 5.6: Local attack times for the top 5 victim countries. Attacks are mostly conducted in the afternoon and evening, except in the US where mainly data centers targeted at night.

5.5.4. WEAPONIZATION

After the reconnaissance phases, the adversary needs to create software and packet payloads to trigger the amplifiers. This might result in attacker-specific and tool-dependent implementations.

Packet content shows high homogeneity between attacks. In the first step of the analysis, we investigate the commands used to trigger the amplification, where we find as shown in table 5.5 very high homogeneity of what techniques adversaries use, even though we have observed 16,900 different payloads being directed to our honeypots. In NTP the monlist packet is present in almost all attacks, and similarly in RIP only one packet has been used for all attacks. Attacks using SSDP however show the use of multiple different strings, although the overall attack stays the same, where they all request *ssdp:discover* in the packet, albeit with different flags or different order in the fields sent. In QOTD however, 10% of the attacks were observed leaving the message *bigbo* in our honeypots, and another 8% of all attacks requested *getstatus*, which has no meaning in the quote of the day protocol. DNS attacks were mainly conducted with empty packets that did not request a server to perform a lookup and return some record, which results in a response containing a string of root servers, amplifying the DNS header approximately six times. Attackers could obtain much higher amplification rates when using servers enabling zone transfers, which allows attackers to make a DNS server pass a copy of its database to a victim. While we would expect adversaries to probe our systems on whether this is possible, none of the attackers has tried to identify non-restricted DNS servers.

Attackers care about the amplification power of a server - at least in some protocols. We placed honeypots with different amplification factors throughout our network to test the differences in how attackers located and used them. Intuitively, two things could happen when attackers find several servers with various amplification ratios: (1) The attacker is aware and consciously optimize based on amplification ratio, in which case we should see a selection bias towards high amplification machines, or (2) the attacker does not track this in which case we should see randomly sampling from the set of high and

Table 5.5: Most popular requests per protocol.

Protocol	Command	% of attacks
NTP	Monlist	99.96
DNS	Root lookup domain	75.61
SSDP	ssdp:discover Host:239.255.255.250:1900	67.16
CharGen	0x01	71.07
RIP	Standard request	100
QOTD	0x01	72.66

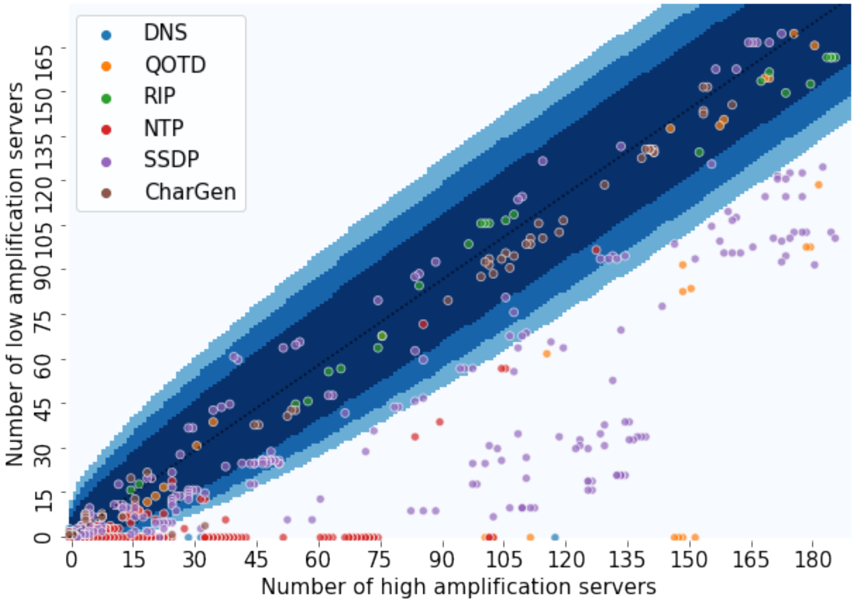


Figure 5.7: Selection of low and high amplification servers in attacks, showing a surprisingly little amount attackers actively select servers with higher amplification.

low amplification servers. Figure 5.7 shows the number of high and low amplification servers utilized in each attack, where each dot is an attack and the color indicates the protocol used. We configured an equal amount of machines to act as a high or low amplification server by letting them answer requests with a different response size. Suppose an adversary is not making any active selection. In that case, the same number of high and low amplification honeypots should statistically be picked, and the dot thus falls on the 45-degree line or in the blue shaded area for sampling from a hypergeometric distribution at 95%, 99%, and 99.9% confidence intervals.

As we see in the figure, there is active selection for "good" amplification servers in many attacks. These practices are differently pronounced depending on the protocol used in the attack. For example, RIP is never located outside the 95% confidence interval of non-selective behavior, while attacks using NTP and DNS are seldom located inside this interval. We see significantly different levels of heterogeneity for other protocols de-

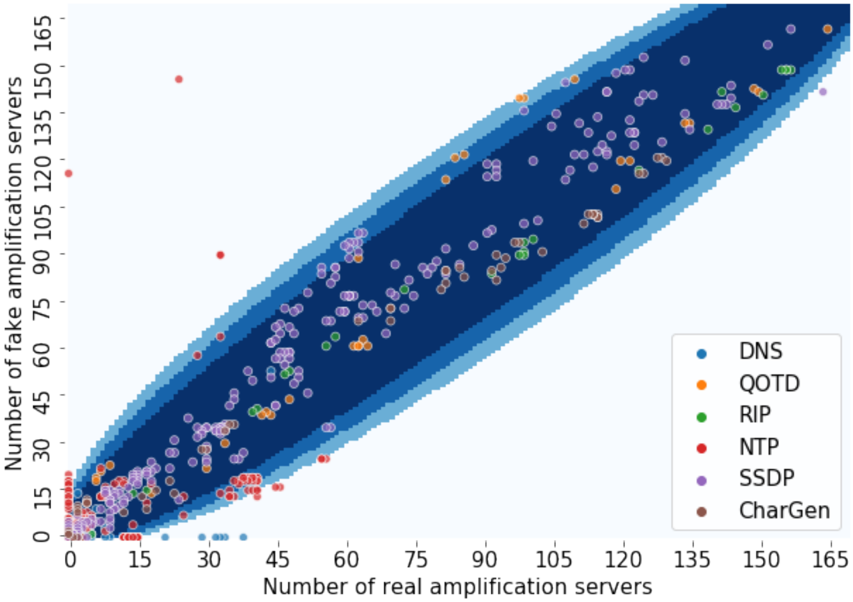


Figure 5.8: Attacks are largely indifferent about whether a service is real as long as it provides amplification.

pending on the attack campaigns, with some actors randomly picking servers and more sophisticated attackers concentrating on high amplification servers only. It is interesting to note that no attacks use a disproportionate amount of low amplification servers, confirming the hypotheses that attackers either randomly sample or select higher amplification servers.

Adversaries generally do not care whether a system is a honeypot. To investigate whether attackers only collect machines in an automated fashion or apply some level of post-processing or vetting of the discovered services, we allocated a new set of IPs to honeypots that would, as part of the response, clearly identify themselves as honeypots that are rate-limited. While this would require a human to look at the responses, we also made these honeypots behave non-protocol compliant, which meant that machine-based post-processing (if being done) should weed them out. To rule out a potential confounding based on the amplification as discussed in the previous section, the amplification ratio was identical to those of our real servers. Servers running fake responders were exclusively running fake services to prevent confounding.

Figure 5.8 shows the distribution of attacks between real and fake servers in an identical setup as figure 5.7, and we see that the bulk of the attackers is indifferent about the responses of the system. Only in DNS and NTP do we observe a handful of attacks being conducted without any fake services. However, in these rare situations, the adversaries did not show sensitivity to the honeypot identification string but expected a particular response to their query and would drop deviating servers from the list. Contrary to what one might think, adversaries do not make any effort to check the plausibility of servers, neither manually nor by scripts. Curiously, we also observed a series of NTP attacks that were *exclusively* using fake systems, the result of an adversary using an incorrect *Monlist* request packet that would be dropped by a regular NTP server implementation as an incorrect query.

Implausible amplifier setups do not matter to any adversary. Krämer et al. hypothesize that hosting multiple services on the same IP address might influence the behavior of an attacker when interacting with the device [3]. To establish whether such a difference exists, our honeypot system deployed 120 servers (24 per protocol) that are solely running one service. In contrast, the rest of the honeypots run all protocols in parallel, which is an implausible setup on a regular server. We do not find any statistically significant differences between those groups. Although anecdotal, we observe adversaries performing multi-vector attacks gladly using all services located at the same honeypot for their attack, even though in practice, the shared uplink would limit traffic at the amplifier.

5.5.5. TESTING

After locating candidate systems for abuse, an adversary might opt to test amplifiers whether they are heavily rate-limited or drop packets and thus not useful in an attack. Actors would not observe these behaviors when scanning using a single packet but need to test hosts actively. Additionally, as services might be moved to other IPs or be shut down, it would make sense for adversaries to frequently test if the amplifiers are still online to not waste reflection potential by sending packets to a server that does not respond.

Only a few attackers test whether a system is rate-limited before performing an attack. Evidently, operators do not want their servers being abused in performing attacks, as this unnecessarily drains one's resources, and the victim might blame the amplifier for the packet flood. Operators would therefore minimize the risk for attacks, for example, by establishing rate limits on requests towards a service. To establish the extent to which adversaries actively test servers before using them in an attack, our honeypot system contains 60 servers per protocol that are actively dropping one-third, one-half, or two-thirds of all outgoing packets. While all servers will amplify incoming traffic when an attacker crafts the correct packet, it is clear that the amplification potential decreases dramatically when two-thirds of the packets are dropped. To ensure these servers will be found by a scan and not bias the setup, the first packet in a connection will always receive a reply. With this setup, only attackers who have probed the system with more than a single scan will know these differences. Figure 5.9 shows the distribution of attacks across these services and the probability of these server choices occurring at random using the confidence intervals explained in the previous section. The image shows that overall packet loss seems of no concern to adversaries. Only in SSDP do we see attackers making an effort to steer clear of servers with packet loss. As there is no significant deviation in how many times servers were scanned, these choices have to be made deliberately.

While many attacks using SSDP use a disproportionate amount of loss-free servers, we would expect adversaries to completely steer away from them when having the choice between different servers that perform better. However, almost no attacks are void of any lossy servers, raising the question of how adversaries make this selection. As the probability that packet loss will occur when contacting a host is in our case $1-(1-$

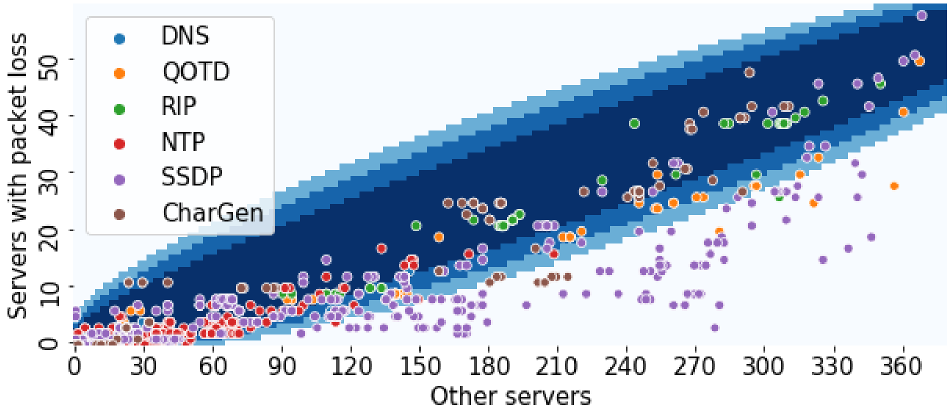


Figure 5.9: Usage of servers with packet loss in attacks, active selection only occurs in case of SSDP.

$droprate)^{k-1}$ where k is the number of packets sent during one connection, some scans will not be enough to establish that some of our hosts never reply to a portion of the scans coming in. Assuming a scan rate of 5 packets per host, which we observe in 9% of scanning traffic, an adversary will have a 20% chance that packet loss is not observed when we drop $\frac{1}{3}$ of the packets, making the packet loss invisible to the adversary. Let us only consider the servers with $\frac{2}{3}$ packet loss, which would be visible 99% of the time when sending five packets towards this honeypot. We indeed find adversaries testing the infrastructure with 150 attacks in NTP and 96 attacks in SSDP that do not use any of these servers.

Adversaries do not care about server latency. Aside from dropping packets, also considerable packet delays might pose an obstacle for adversaries when launching attacks. We placed 60 servers that delayed the response by an extra delay of 500 ms to investigate a potential selection strategy for this. With regular service times below 1 ms and worst-case round-trip network delay from Europe to Asia being in the order of 300 ms, this delay should significantly stick out. A significant delay in sending responses could indicate a highly loaded or overloaded system, making it less attractive for an adversary as it might get overwhelmed during the attack. We do, however, not find any statistically significant differences that adversaries prefer non-delaying servers over these servers.

Sophisticated adversaries do not blindly trust their amplifier lists. Over time, results of the scans performed to find open amplifiers might not be accurate anymore due to the reconfiguration of servers or even the decommissioning of the machines. To not waste effort on non-responsive machines, an adversary should repeat scans and curate the list of used servers. To investigate the extent to which adversaries rescan the hosts to verify their status, we continued to collect all network traffic towards our honeypots after we concluded the active experimentation. In this passive phase, all traffic was sink-holed, and no replies were sent out anymore. Figure 5.10 shows the attack rate while we actively responded to requests and the rate after the shutdown of the amplifiers. In the baseline period before the amplifiers actively respond to requests, no attacks were mea-

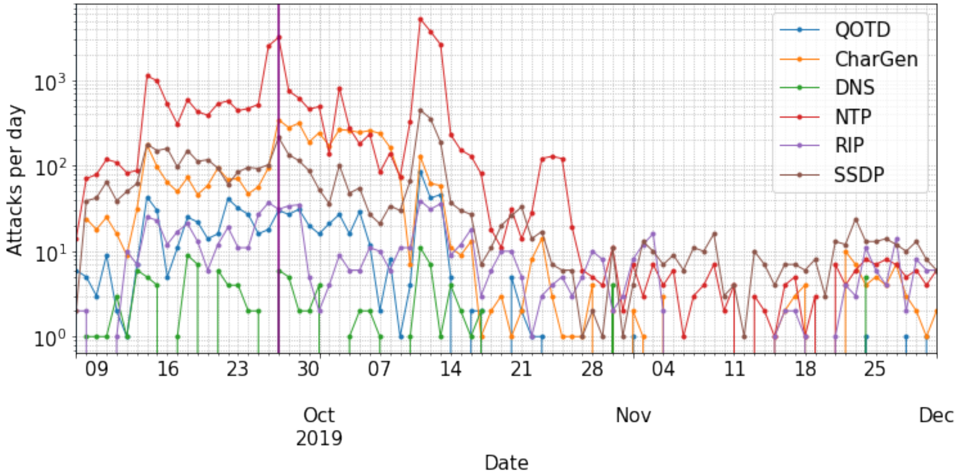


Figure 5.10: Incoming attacks during active replies and subsequent shutdown on September 27th.

sured. While we verified during the baseline establishment that the IPs were unknown to attackers and received no attack requests, reverting to this state and nullifying their use in amplification attacks did not cause the number of attacks conducted using the servers to go down drastically. On the contrary, attacks continued at roughly the same pace for all protocols for more than two weeks. Even after two months, attacks were still conducted using our infrastructure, which had been disabled for longer than it was ever enabled. We believe there can be two explanations for this behavior: (1) The attacks conducted with our servers originates from the same actors who have created lists of vulnerable servers and do not update these frequently, or (2) our servers have been listed in publicly available lists of amplification servers, and the people conducting attacks blindly trust these lists to be accurate and up to date. While we have not been able to find any notion of our servers being on amplifier lists, these lists might be shared inside closed communities. While many attacks are conducted after the system does not respond anymore, no large attacks are being conducted using the amplification servers. Additionally, after the servers are taken offline, the attacks conducted are only rarely using solely high amplification servers. The adversaries that are capable of performing large and lengthy DDoS attacks are thus updating their lists.

5.5.6. Execution

While a DDoS attack is a simple concept in which a victim's resources are drained, the ways in which these attacks are executed are diverse in practice. In this section, we will show various attack techniques used by adversaries captured in our honeypots.

Attack durations and modes differ between amplification protocols and targets. Amplification attacks sent to our honeypots had an average duration of 394 seconds, almost a factor of 5 longer than for TCP SYN floods, the other major type of DDoS attack vector [25, 30, 31]. The duration of attacks observed in our work falls between observed median

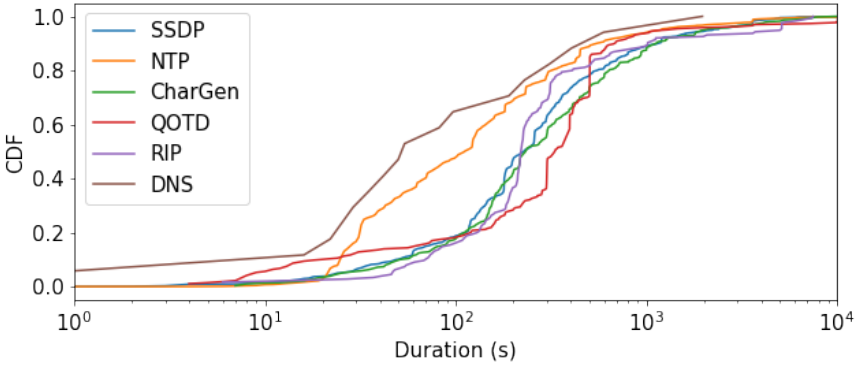


Figure 5.11: CDF for the duration of attacks per protocol.

Table 5.6: Source port usage of traffic coming into our system.

Protocol	Attacks	1. Single (%)	2. Static (%)	3. Rand (%)
CharGen	471	1	95	4
DNS	55	11	35	55
NTP	11,130	2	47	51
QOTD	264	6	94	0
RIP	184	2	98	1
SSDP	1375	30	68	1

durations in related work, which report a median of 658 [6] and 255 seconds [25]. Figure 5.11 shows how the durations even significantly differ per protocol, with QOTD having a median attack duration three times as high as NTP. In QOTD, we also see modes at 400 and 500 seconds that are not present in other attacks. The mode at 300 seconds is visible in QOTD, CharGen, NTP, and RIP but does not show in SSDP. The presence of attack durations and modes has earlier been identified by [30] for TCP SYN flooding attacks in 2017, where typical durations of 30 and 60 seconds were traced back to test attacks by booter services. For amplification DDoS attacks, these modes have been identified by [6]. In our study, we find that both the attacks take significantly longer and common modes are significantly higher opposed to TCP SYN flooding attacks, but both are lower compared to [6].

Given the differences between protocols and the way attacks are run, it is natural to wonder whether these are used for different purposes. When we look at attacks on two different types of services, those on game hosts and web servers based on their open ports using Shodan, we find that the type of target influences the used attack vector and the duration of the attack. The attacks on game servers are solely conducted with NTP and SSDP, whereas attacks across all protocols attack web hosts. In terms of duration, game hosting servers are attacked more rigorously than web servers, with an average attack duration of 12 minutes. In contrast, web servers only deal with attacks that, on average, last a bit more than 5 minutes. Between the two groups, there is no difference in the selection of servers to be used in the attacks.

Attacked ports cannot be used in the protection against attacks - except when attackers are targeting TCP ports instead of UDP. Various controls exist to filter DDoS attacks [2]. One of these techniques is to filter based on ports. Like most other services, our honeypots run their services on default UDP ports, which would not provide an angle for the victim to filter out the amplified data. On the other hand, the attacker's spoofed requests need to originate from a source port, to which (at the victim's IP) the response will be sent. We distinguish three cases for the ports used to request data: (1) All requests towards honeypots are made from one port, indicating a crafted, injected packet. (2) The requests are made using different ports per honeypot. (3) For every request, the attacker generates a random port number, evenly distributing the traffic over all ports. All three cases show distinct implementation choices an attacker can make and could avoid packet-based filters that are placed as a countermeasure. Table 5.6 shows the distribution of source port usage in attacks, and we see a large part is implementing the frames using a static port per honeypot. While randomizing source ports is only slightly more complicated in implementation, most adversaries do not seem to choose this route, even though it can be more robust against packet level filtering [2]. Only in NTP and DNS a part of attacks is performed while randomizing ports across the entire port range.

This however raises another important operational aspect. An amplification attack towards a single port could be trivially blocked by packet filtering unless the filtering happens on the victim's premises. The flood is sufficient to congest the connection from the victim to the Internet. However, if the attack targets a service already running on the victim, the attack might be hard to distinguish from legitimate traffic as found by [25], who identify that most of the attacks are aimed at ports connected to online gaming. We do not find many attacks targeted at gaming ports and instead find that adversaries fail to take into account that protocols such as HTTP would run on *TCP* port 80 or 443 and not *UDP* port 80, which would not be whitelisted in a firewall.[1] Thus, the deliberate targeting of select ports would not have any more benefit than a packet flood towards a random destination port. From the 650 single port attacks, 364 send their attack towards port 80 on the victim, while another 20 target port 443. The only attacks aimed at a UDP service were 16 attacks on DNS port 53.

Attack pulses maximize the attack efficiency while minimizing the cost for the attacker. After a packet flood has disrupted a service, it might take a moment after the end of the attack for services to recover and clients to reconnect. This means that even after the flood itself has stopped, the effects might still be ongoing which can be used by an adversary to minimize the bandwidth required for a successful attack. We observed that several adversaries do not launch their attacks as continuous floods but instead adopt a pulsing behavior of floods followed by brief inactive periods. This would have the advantage of reducing the risk of "burning" the amplifiers due to continuous usage while still meeting the objective. Figure 5.12 shows this behavior for one such attack, with multiple waves of flooding. When we observe pulses, their behavior was highly coordinated, as we observe all honeypots used in the attack simultaneously receiving traffic and simultaneously stop receiving requests. We would expect sophisticated actors to perform these types of attacks, and we indeed find for all pulse attacks that they are exclusively

[1] A major service running on UDP would be QUIC on UDP port 80, however none of the hosts attacked on this port ran this protocol.

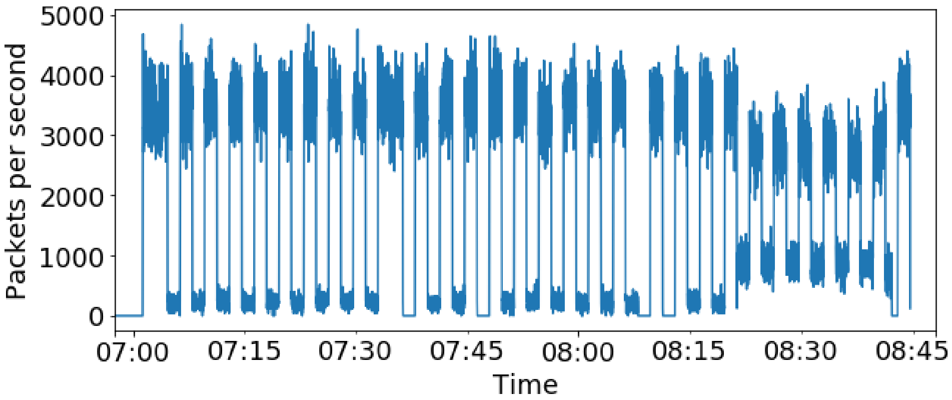


Figure 5.12: A pulsing attack using 40 honeypots.

conducted using only high amplification devices.

5.6. DDoS ATTACK CAMPAIGNS

One of the most important arguments for investigating attacks along the DDoS attack chain presented in this paper is that adversaries will likely reuse components from a previous attack. Instead of finding infrastructure, testing it, and weaponizing every time a victim is being targeted, attackers would instead use the same set of servers, attack packets, etc., to perform multiple attacks.

Actors have a geographical focus. When multiple attacks use the same servers in our system, the probability of this occurring due to random selection of two different entities is negligible given the number of active honeypots. We use this as the first feature to cluster attacks, where we cluster attacks if these are using the same set of our amplifiers. We do not link attack instances if they have all honeypots of a particular kind in common – such as all NTP high amplification services –, as specific preferences of adversaries might result in multiple actors sharing these edge cases. From the 720,995 attack flows, we obtain 749 clusters of attacks, of which 351 attacks more than once. To verify if the same actor indeed performs the attacks inside a cluster, we use three criteria: (1) The attacks are performed using the same request packet. (2) The attacks use the same strategy in picking the source port. (3) The flows inside the cluster have the same characteristics in, for example, pulsing and intensity. Amazingly, all of the 351 clusters attacking more than once match these criteria and will therefore be considered in this section as being valid clusters. Figure 5.13 shows the distribution of targeted countries by the clusters attacking more than 100 separate IP addresses. We see actors show a significant degree of geographical specialization. While previous work locates the bulk of DDoS activity to the US and China [25], we find it much more nuanced from an actor perspective, which focuses and specializes on victims in a handful of countries. This shows that attacks are not randomly carried out but are aimed at an objective from the adversary; for example, NTP clusters 4, 5, and 6, which solely target Turkey identify that

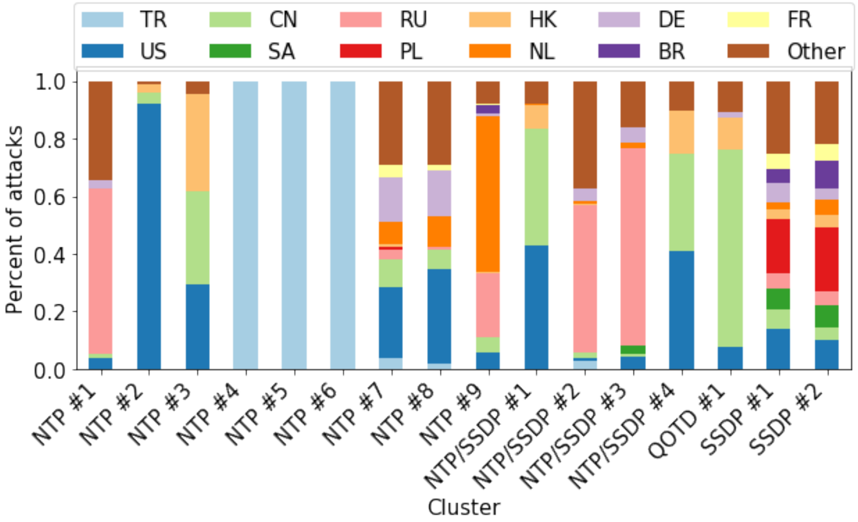


Figure 5.13: Attacks from clusters performing more than 100 attacks during our study.

the objective of these adversaries is different from other clusters. As discussed in section 5.5.3 attacks are often performed during the afternoon or evening at the location of the victim. As most attackers are biased towards certain countries, we would expect these actors to have distinct attack timings as well. From the 86 (25%) clusters performing attacks over multiple days, we indeed observe 68 (19%) showing a diurnal pattern, where the number of attacks drops significantly during certain hours. The remaining 18 (5%) clusters are not significantly lowering their activity in a day/night rhythm and might be an indication of automated booter services that are used by people in multiple countries.

Sophisticated actors reuse the same servers often. As all the individual attacks within a cluster use the same attack packet, with identical traffic dynamics and from the same (randomly chosen) source port, they can be attributed to the same setup by the adversary. Within clusters, all attacks are performed using the same command. For an attacker, this makes sense, as the attacker knows that the request sent to the server is amplified based on previous scan results. When we look at the volume of the attacks originating from a cluster and the way they select our honeypot infrastructure, we find an astonishing rich spectrum of behaviors. Figure 5.14 shows for each of the clusters the volume of attacks through the size of the market and on the x- and the y-axis the percentage of fake and high amplification honeypots that were used during the attacks. As clusters also showed the behavior of selecting the same set of amplification servers, these data points are not averages but static over time. Much of the clusters congregate in the middle of the graph, distributed around the expected values for the ratio of fake servers and high amplification devices an attacker would use based on a random selection, such as an indiscriminate scan of the Internet. Deviations to the left mean that the adversary is performing some post-processing to weed out suspicious devices, shifting

Table 5.7: Groups of attacks modeled on DDoS phases and activities.

	Capability Development	**Infrastructure Reconnaissance**		**Target Reconnaissance**		**Weaponization**		**Testing**		**Execution**	
	Protocol	*Sources*	*Countries*	*Targets*	*Countries*	*Packet Content*	*Amp.*	*Fake serv.*	*Curation*	*Attacks*	*Max Duration*
1	CharGen	19	9	6	5	a	17.97	Yes	Within a day of every attack	9	7,891 seconds
2	DNS	2	2	2	2	Query: ufpa.br	11.60	No	Attacks within 2 hours of scan	2	283 seconds
3	NTP	5	2	8	8	monlist	15.82	No	Only initial scan	10	628 seconds
4	NTP	4	3	3	3	monlist + 10 bytes	5.40	No	Scan every attack	7	20 seconds
5	QOTD	2	2	5	3	single byte	16.12	Yes	One scan 2 days before attacks	5	3,606 seconds
6	QOTD	2	1	24	6	"Bigbo"	14.77	Yes	One scan, attacks for 10 days	26	3,640 seconds
7	QOTD	9	3	23	8	"getstatus"	17,34	No	Continuous scan	27	7,265 seconds

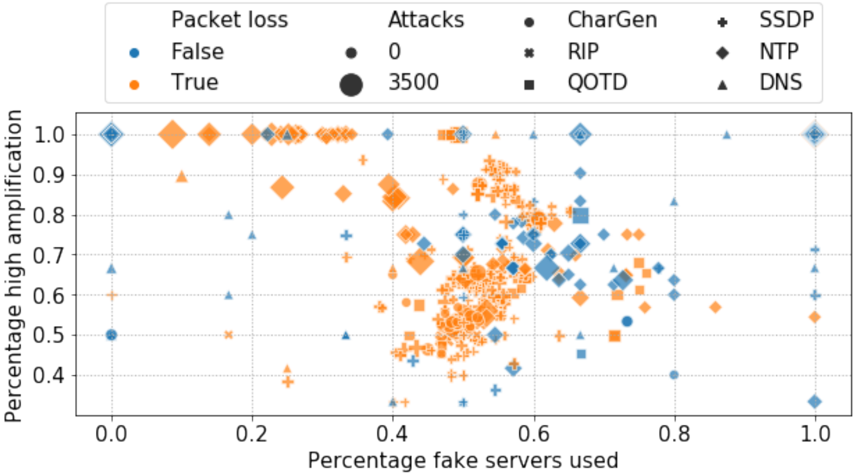


Figure 5.14: Scatterplot of attack clusters.

to the top an active selection to maximize the attack volume. As we see in the graph, the more attacks an attacker performs, the stronger it will pre-select for high amplification servers and discard low-utility devices. The more attacks an attacker performs, the higher the likelihood to identify and discard unusually behaving servers from the attack list. Thus, the most extensive campaigns tend to be run by the more sophisticated adversaries. We indeed find a significant positive correlation ($p < .05$) between the number of attacks in a cluster and the amplification ratio, as well as a significant negative correlation ($p < .01$) with the number of fake servers used. There are however groups performing a large number of attacks using only fake high amplification servers. On inspection of these clusters, we find that these attackers are not using the correct protocol specification and do not receive replies from the real servers. However, the adversaries do select the servers that provide the highest amplification. Additionally, we find a significant negative correlation ($p < .01$) between the fake servers used in an attack and the amplification ratio, hinting that advanced adversaries are actively filtering out these obvious honeypots. This behavior is curiously only present in NTP and SSDP abuse, while adversaries targeting DNS, RIP, or QOTD do not seem to care how they achieve their objective. Surprisingly, the adversaries that perform highly targeted scans towards single cloud zones are not the sophisticated attackers as the adversaries actively removing poorly behaving amplification servers are also evaluating the entire ecosystem and use amplification servers hosted in multiple cloud locations.

Sophisticated attackers do not show advanced behavior across the entire spectrum but innovate only selectively. To better understand how adversaries work, the steps they take can be separated into different phases. Table 5.7 shows a mapping for seven clusters performing attacks using our amplification honeypots on the model shown in figure 5.1. Looking at the entire chain of activities rather than only the separate phases, we find that clusters performing longer-lasting attacks are more likely to use servers with a higher amplification factor, indicating that attackers capable of performing long-lasting DDoS attacks are making more effort to ensure their operation is successful. But there is no relation between the duration or amplification factors in attacks and whether or not the adversaries use our obvious honeypots. However, we find evidence that adversaries capable of performing large attacks are curating the set of amplifiers, whereas adversaries that only perform lower volume and duration attacks are using their amplifier lists for weeks without checking whether these systems are still online. Clusters performing long-lasting attacks with high amplification such as cluster #1 and #7, which perform attacks lasting over 2 hours, are continuously curating and updating their amplifier lists by continuously scanning the servers used in the attacks and are thus not active anymore after the active phase of our experiment concludes, as their continuous curation prevents them from using a non-responsive server in an attack. While we have seen many attacks being performed weeks after the infrastructure did not respond to attacks anymore, these attacks are almost exclusively under 10 minutes and do not care about the amplification ratio provided by a server. The actors capable of performing high-volume and high-duration attacks are thus test servers for their amplification ratio and weed out unresponsive servers before performing an attack.

5.7. Discussion and Limitations

Investigating the Tactics, Techniques, and Procedures used in DDoS attacks may change the way we organize our defenses. By looking at the entire DDoS attack chain with the model presented in this paper, we show that the ecosystem of DDoS amplification is more than a collection of individual attacks but that it is possible to correlate and link them based on solid behavioral features. The results show the presence of hundreds of actors with different sophistication, preferences for attack vectors and victims, and unique modus operandi. This suggests that instead of merely enduring and mitigating DDoS attacks, some degree of attribution for attacks is possible.

Although a significant share of DDoS appears as ‘dumb”, ‘script-kiddie”-driven activity, our study reveals the presence of sophisticated actors who investigate, inspect and measure services on the Internet and perform active selection to maximize their return-on-investment. This means that future research on DDoS and the Internet threat landscape has to account for such adversarial behavior and needs to advance in terms of techniques, as obvious decoys or servers not participating in test attacks that are currently being used in research would be discarded by sophisticated actors. This would result in a drastically biased perspective on the DDoS ecosystem. By using actual services and through momentary but ethical participation when rallied, we were the first to show this unexplored dimension of the threat landscape and demonstrate that ethical investigation of this behavior is feasible.

In this study, we deployed an order of magnitude more honeypots than any previous works, first to allow for systematic and statistically significant testing of different scenarios and configurations, but second because the heterogeneity of the ecosystem would otherwise lead to biased results from the overrepresentation of select actor groups. We also identify the need for rigid methods in separating malicious scanning activity from research scan activity to avoid large measurement biases. Using the technique from [2], we can show that 3-10 times as many honeypots are necessary to capture the richness of the threat landscape as were deployed in previous work, as shown in figure 5.4. To deal with such temporal biases and ecosystem heterogeneity, we advocate that future studies should deploy honeypots in the hundreds, not dozens, to be effective.

Accomplishing this deployment scale also has drawbacks, and in this study, we limit our experiments for monetary reasons to a significantly shorter online duration. While smaller honeypot studies operated for several months or even years [2, 3, 6], our system was deployed for three months, from which actively responding for three weeks. This limited-time might influence the number of adversaries using our systems and would not show those actors who enter the ecosystem only at longer spaced intervals. Finally, our study deployed honeypots within the IP ranges of cloud providers. Although many organizations are now moving their infrastructure to the cloud and adversaries should hence look for abusable systems there, it is possible that some actor groups specifically focus on network ranges owned by enterprises, which we would miss in such a study.

5.8. Future Work

Despite the heterogeneity of the overall ecosystem in the spectrum from script-kiddie to sophisticated actor, our results show that each attack – even though it might hit hundreds of amplifiers – is remarkably similar. This is understandable as the perpetrators are optimizing for the economies of scale, but it also opens up angles for mitigation. By identifying, automatically recognizing common tactics and techniques, and distributing them to defenders for detection [32], much of current distributed amplification DDoS could be mitigated. Our findings show that recent proposals such as BGP flowspec could be highly effective in practice by filtering incoming amplification requests and not only merely the resulting packet floods towards the victim. Given this similarity and the fact that essentially every perpetrator in our study horizontally scaled out to abuse many honeypots in the same attack, also collaborative identification of malicious flows – either at the level of networks or resolvers – could provide an angle towards reducing this attack vector if servers providing amplification would run a service that could not be disabled otherwise. Such an approach creates again interesting but solvable research problems to remediate the resulting privacy concerns. However, it could be a viable and scalable solution as incentives and costs are aligned for the operators of the abused services.

In this study, we focused on attacks measured by our deployed honeypots and cannot measure the attacks at the victim's side. To understand the behavior of the victims and the actual attack sizes, future works should focus on a collaboration with victims, cloud service providers, or DDoS defense companies as an additional vantage point. This would allow for further analysis of attack "pulses" and give insights into the total number of amplifiers used in attacks.

5.9. Conclusions

We have analyzed adversarial techniques for amplification DDoS attacks using 549 honeypots running six amplification protocols that were rallied to 13,479 attacks over three weeks. We show that adversaries tend to select the servers with the highest amplification potential and that adversaries create tests to identify the servers that would create the largest impact. Additionally, we show the existence of "memory" of amplification services, which are abused long after the service is taken down. While the bulk of adversaries pursue simple techniques at a high level, we show the presence of highly advanced attacker groups, selectively picking servers and tactics to perform their attacks with the most firepower.

REFERENCES

[1] V. Paxson, *An analysis of using reflectors for distributed denial-of-service attacks,* ACM SIGCOMM Computer Communication Review **31** (2001).

[2] C. Rossow, *Amplification Hell: Revisiting Network Protocols for DDoS Abuse,* in *NDSS* (2014).

[3] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, K. Yoshioka, and C. Rossow, *Amppot: Monitoring and defending against amplification DDoS attacks,* in *International Symposium on Recent Advances in Intrusion Detection* (Springer, 2015).

[4] A. Büscher and T. Holz, *Tracking DDoS attacks: Insights into the business of disrupting the web,* in *5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 12)* (2012).

[5] Https://cloud.google.com/blog/products/identity-security/identifying-and-protecting-against-the-largest-ddos-attacks, accessed at 2021-05-05.

[6] D. R. Thomas, R. Clayton, and A. R. Beresford, *1000 days of UDP amplification DDoS attacks,* in *2017 APWG Symposium on Electronic Crime Research (eCrime)* (IEEE, 2017).

[7] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, *Exit from Hell? Reducing the Impact of Amplification DDoS Attacks,* in *23rd USENIX Security Symposium (USENIX Security 14)* (2014).

[8] C. Fachkha, E. Bou-Harb, and M. Debbabi, *Fingerprinting internet DNS amplification DDoS activities,* in *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)* (IEEE, 2014).

[9] D. C. MacFarland, C. A. Shue, and A. J. Kalafut, *Characterizing optimal DNS amplification attacks and effective mitigation,* in *International Conference on Passive and Active Network Measurement* (Springer, 2015).

[10] D. C. MacFarland, C. A. Shue, and A. J. Kalafut, *The best bang for the byte: Characterizing the potential of DNS amplification attacks,* Computer Networks (2017).

[11] Z. Sun, B. Liu, and C. Hu, *Method for effectively detecting and defending domain name server (DNS) amplification attacks,* (2011).

[12] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, *DNS amplification attack revisited,* Computers & Security **39** (2013).

[13] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, *Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks,* in *Proceedings of the 2014 Conference on Internet Measurement Conference* (2014).

[14] L. Rudman and B. Irwin, *Characterization and analysis of NTP amplification based DDoS attacks,* in *2015 Information Security for South Africa (ISSA)* (IEEE, 2015).

[15] M. Prince, *Technical details behind a 400Gbps NTP amplification DDoS attack,* Cloudflare, Inc **13** (2014).

[16] B. A. Sassani, C. Abarro, I. Pitton, C. Young, and F. Mehdipour, *Analysis of NTP DRDoS attacks' performance effects and mitigation techniques,* in *2016 14th Annual Conference on Privacy, Security and Trust (PST)* (IEEE, 2016).

[17] J. Krupp, M. Backes, and C. Rossow, *Identifying the scan and attack infrastructures behind amplification DDoS attacks,* in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016).

[18] J. Krupp, M. Karami, C. Rossow, D. McCoy, and M. Backes, *Linking amplification ddos attacks to booter services,* in *International Symposium on Research in Attacks, Intrusions, and Defenses* (Springer, 2017) pp. 427–449.

[19] J. J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Z. Granville, and A. Pras, *Booters—An analysis of DDoS-as-a-service attacks,* in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (IEEE, 2015).

[20] M. Karami, Y. Park, and D. McCoy, *Stress testing the booters: Understanding and undermining the business of DDoS services,* in *Proceedings of the 25th International Conference on World Wide Web* (2016).

[21] A. Hutchings and R. Clayton, *Exploring the provision of online booter services,* Deviant Behavior **37** (2016).

[22] M. Karami and D. McCoy, *Rent to pwn: Analyzing commodity booter ddos services,* Usenix login **38**, 20 (2013).

[23] D. Kopp, M. Wichtlhuber, I. Poese, J. Santanna, O. Hohlfeld, and C. Dietzel, *DDoS Hide & Seek: On the Effectiveness of a Booter Services Takedown,* in *Proceedings of the Internet Measurement Conference* (2019).

[24] A. Vetterl and R. Clayton, *Bitter harvest: Systematically fingerprinting low-and medium-interaction honeypots at internet scale,* in *12th USENIX Workshop on Offensive Technologies (WOOT 18)* (2018).

[25] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, *Millions of targets under attack: a macroscopic characterization of the DoS ecosystem,* in *Proceedings of the 2017 Internet Measurement Conference* (2017).

[26] *Global internet speed index,* Https://www.speedtest.net/global-index, accessed at 2021-05-05.

[27] J. Matherly, *Complete guide to shodan,* Shodan, LLC (2016-02-25) **1** (2015).

[28] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, *A search engine backed by internet-wide scanning,* in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015) pp. 542–553.

[29] P. Richter and A. Berger, *Scanning the scanners: Sensing the Internet from a massively distributed network telescope,* in *Proceedings of the Internet Measurement Conference* (2019).

[30] N. Blenn, V. Ghiëtte, and C. Doerr, *Quantifying the spectrum of denial-of-service attacks through internet backscatter,* in *Proceedings of the 12th International Conference on Availability, Reliability and Security* (2017).

[31] H. Griffioen and C. Doerr, *Quantifying TCP SYN DDoS Resilience: A Longitudinal Study of Internet Services,* in *IFIP Networking* (2020).

[32] H. Griffioen, T. M. Booij, and C. Doerr, *Quality evaluation of cyber threat intelligence feeds,* in *International Conference on Applied Cryptography and Network Security (ACNS)* (2020).

6

UNDERSTANDING THE EVOLUTION OF ACTORS

Using hundreds of thousands of compromised IoT devices, the Mirai botnet emerged in late 2016 as a game-changing threat actor, capable of temporarily taking down major Internet service providers and Internet infrastructure. Since then, dozens of IoT-based botnets have sprung up, and in today's Internet, distributed denial-of-service attacks from IoT devices have become a major attack vector. This proliferation was significantly driven by the public distribution of the Mirai source code, which other actors used to create their own customized version of the original Mirai botnet.

This paper provides a comprehensive view into the ongoing battle over the Internet of Things fought by Mirai and its many siblings. Using 7,500 IoT honeypots, we show that we can use 300,000,000 compromisation attempts from infected IoT devices as well as a design flaw in Mirai's random number generator to obtain insights into Mirai infections worldwide. We find that networks and the particular malware strains that plague them are tightly connected, and malware authors take over strategies from their competitors over time. The most surprising finding is that IoT botnets are not self-sustaining epidemiologically, were it not for continuous pushes from bootstrapping, Mirai and its variants would die out.

This chapter has been published as *"Examining Mirai's Battle over the Internet of Things"* by **Griffioen, H.**, Doerr, C. in the ACM SIGSAC Conference on Computer and Communications Security (2020).

6.1. Introduction

The emergence of the Mirai botnet in late 2016 fundamentally changed the Internet threat landscape. Although the risk of insufficiently protected Internet-of-Things (IoT) devices was long known, the problem made the front pages when its initial distributed denial-of-service (DDoS) attacks exceeded volumes of 600 Gbps, crippling major Internet infrastructure and service providers such as OVH [1] or Dyn [2]. The first attack by an IoT botnet doubled previous attack volumes from the get-go, and in the coming weeks, would continue to increase to break the 1 Tbps threshold [3].

This captured the attention of the general public and defenders and the imagination of perpetrators. Soon after, the source code of Mirai was published online [4], leading copycats to release clones of Mirai-based IoT malware. Named “MIORI”, “JOSHO”, or “MASUTA”, these variants are directly derived from the Mirai source code, copying the core framework and essential routines for scanning, infection, and communication, but also feature actor-specific modifications to passwords, the way the malware identifies itself, and the C&C servers bots report back to.

Trivial access of the source and the abundance of vulnerable devices have led to many Mirai variants, which are fighting over control over the millions of IoT devices deployed worldwide. In this paper, we are examining this battle between Mirai and its siblings that have emerged since. To do this, we leverage the fact that the Mirai botnet behaves like a worm, using an infected IoT device to scan the Internet for other devices it could potentially compromise. With an installation of 7,500 IoT honeypots, we are collecting these infection attempts and can turn 300,000,000 received compromisation attempts into a real-time view of which devices around the world are infected at a given moment with a particular strain.

One of the core features across Mirai, and all its variants, is how it searches for victims. The malware selects targets based on randomly generated destination IP addresses to avoid sequential port scanning that is trivially detected. The authors designed their own random number generator (RNG). As we will show, this design has severe flaws, allowing us to break the seed based on information from a *single* incoming packet. This insight can then be used to learn about the lifetime of the infection on the device itself. With this work, we are making the following contributions:

- We are the first to provide a comprehensive view into IoT devices’ infection and reinfection behavior. We demonstrate that it is possible to exploit structural weaknesses in Mirai’s RNG to extract the precise moment of compromisation, which we use to understand the lifetime of infections.

- We show that IoT devices are under intense attack by a plethora of actors. We follow the activities of 39 variants and analyze how they infect and retain control over some 200,000 unsecured devices. We find that compromised hosts frequently change “ownership” through reboots and reinfections and based on hostile takeovers by other actors.

- We show that IoT malware crashes comparatively quickly on routers, and there are clear differences between particular malware strains and the network devices are placed in.

- We provide the first epidemiological quantification of Mirai and show that epidemiologically speaking, Mirai is not a self-sustaining worm and would die out if it would not be for the bootstrapping infrastructure that constantly spreads reinfections to vulnerable devices.
- We find evidence of specialization of malware authors towards specific victim groups. Strains use information about devices in particular networks to gain a competitive advantage, yet malware owners observe their competition and adapt their password lists with information from their rivals.
- We demonstrate that adversaries are making deliberate efforts to evade network ranges and thus avoid detection by common measurement initiatives.

6.2. The Mirai Botnet

While attacks on IoT devices have become commonplace, the advent of Mirai as the first major IoT malware was a milestone in Internet security. Japanese for “future”, the IoT malware became front-page news when its attacks severely impacted major Internet infrastructure and service providers in DDoS attacks. To understand how Mirai operates, we begin with an overview of the system infrastructure and how it targets victims.

6.2.1. System Infrastructure

The ecosystem of the Mirai botnet consists of three main components as shown in figure 6.1: a loader to bootstrap the botnet, the compromised routers themselves, and the command-and-control (C&C) server to which bots report back and obtain instructions from. Connections to the C&C are made using a plain TCP socket on an address and port hardcoded in the source. The control protocol is straightforward, with instructions in binary status codes. The core of the botnet is the compromised IoT devices, which are responsible for spreading the infection further. To this end, they independently scan the Internet for devices responding on specific ports. When configured to search for telnet, they target ports 23 and 2323.

As adversaries would avoid doing the first initial scan and compromisation activities from their IP address, the Mirai toolchain includes a loader mechanism, which is provided with a list of devices, for example, powerful hosts with a high-bandwidth network connection. These devices are then used to start the scanning and infection activities, thereby bootstrapping the Mirai installation.

6.2.2. Mirai Behavior

Certain implementation choices of Mirai create unique behavioral characteristics that we can efficiently fingerprint. At a high level, the Mirai IoT malware pursues three different objectives after the infection of a device: first, it closes the entry point (typically telnet on TCP port 23, which is also the focus of this paper) to gain exclusivity over the compromised device. Second, the malware will start a continuous scan for other devices, and if it identifies an open telnet port, it begins to brute force based on a built-in credential list. And third, it establishes a connection with the C&C server hardcoded in the source and executes commands it receives. Figure 6.2 shows an excerpt of the Mirai

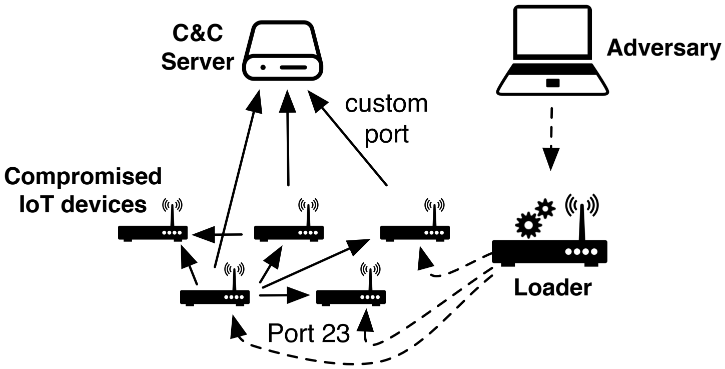


Figure 6.1: Adversaries bootstrap using loaders, targeting a predefined list of devices. Compromised hosts report to a C&C and randomly target IP addresses to proliferate.

source, which we will discuss in the following, with these three tasks spawned in `main()` (see marker ① in figure 6.2).

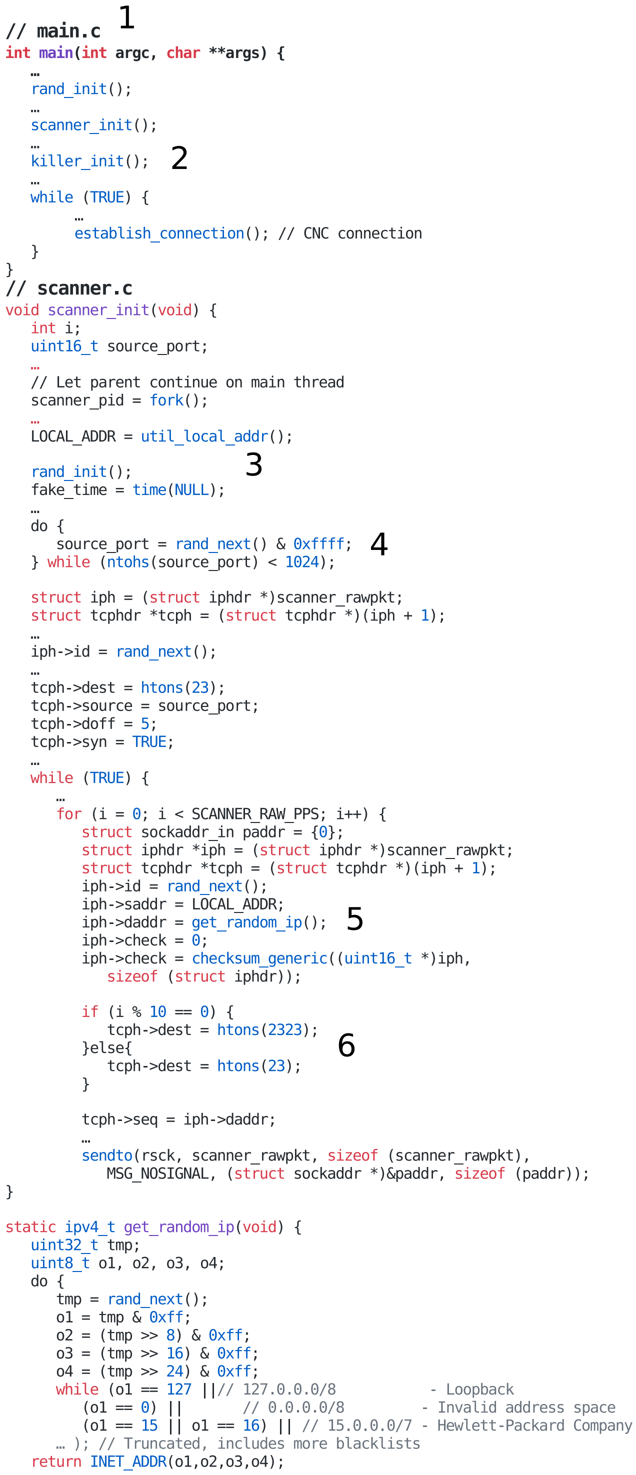
When targeting telnet, Mirai will attempt to bind to port 23. If this is not possible, it tries to shut down the other telnet process through which other logins could occur. In `killer_init()`, it identifies and kills the process that has bound to this port ②, thereby locking the router down for the duration of the infection. However, Mirai and its descendants are non-persistent, so after a reboot, the device is restored to its pre-infected, vulnerable state. It will spin up its original telnet process and is ready for reinfection.

As one of its first actions, the Mirai malware forks off a separate thread that performs an Internet-wide scan for other IoT devices. As shown in `scanner_init()`, it begins by re-initializing the random number generator (RNG) ③ and choosing a random source port in the range of [1024, 65535] ④. It then prepares the IP and TCP header it will use for subsequent scan packets. In an infinite loop, it chooses random IP addresses as target (`get_random_ip()`) ⑤, sets a random IP ID for the packet and chooses port 23 or 2323 as port ⑥. Every IP is probed 10 times before continuing. In case of a response, Mirai will start to brute force the login using a local dictionary.

TARGET SELECTION AND RANDOMNESS GENERATION

Several packet-specific information in the TCP SYN scans sent by Mirai are populated based on output from the RNG, most importantly the destination IP address, source port, and IP ID. This RNG is seeded at startup and again at the fork of the thread with the current time in epochs, the process IDs of the `main()` and the scanning thread, and the number of clock ticks since the current process has been started (which resets in the scanning thread due to the fork). These functions return 32, 15, 15, and 32 bit values respectively, and are used to initialize register variables x, y, z and w used in each iteration. Although this would naively imply 94 bits of entropy in the RNG seed, in practice, most bits are entirely predictable.

The RNG follows a linear shift feedback register design (LFSR), with four XOR operations and three-bit shift operations. The inner workings are depicted in figure 6.3, which shows the state initialization and the generation routine. After the initialization



```
// main.c
int main(int argc, char **args) {
    …
    rand_init();
    …
    scanner_init();
    …
    killer_init();
    …
    while (TRUE) {
           …
           establish_connection(); // CNC connection
    }
}
// scanner.c
void scanner_init(void) {
    int i;
    uint16_t source_port;
    …
    // Let parent continue on main thread
    scanner_pid = fork();
    …
    LOCAL_ADDR = util_local_addr();

    rand_init();
    fake_time = time(NULL);
    …
    do {
        source_port = rand_next() & 0xffff;
    } while (ntohs(source_port) < 1024);

    struct iph = (struct iphdr *)scanner_rawpkt;
    struct tcphdr *tcph = (struct tcphdr *)(iph + 1);
    …
    iph->id = rand_next();
    …
    tcph->dest = htons(23);
    tcph->source = source_port;
    tcph->doff = 5;
    tcph->syn = TRUE;
    …
    while (TRUE) {
        …
        for (i = 0; i < SCANNER_RAW_PPS; i++) {
            struct sockaddr_in paddr = {0};
            struct iphdr *iph = (struct iphdr *)scanner_rawpkt;
            struct tcphdr *tcph = (struct tcphdr *)(iph + 1);
            iph->id = rand_next();
            iph->saddr = LOCAL_ADDR;
            iph->daddr = get_random_ip();
            iph->check = 0;
            iph->check = checksum_generic((uint16_t *)iph,
                sizeof (struct iphdr));

            if (i % 10 == 0) {
                tcph->dest = htons(2323);
            }else{
                tcph->dest = htons(23);
            }

            tcph->seq = iph->daddr;
            …
            sendto(rsck, scanner_rawpkt, sizeof (scanner_rawpkt),
                MSG_NOSIGNAL, (struct sockaddr *)&paddr, sizeof (paddr));
}

static ipv4_t get_random_ip(void) {
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;
    do {
        tmp = rand_next();
        o1 = tmp & 0xff;
        o2 = (tmp >> 8) & 0xff;
        o3 = (tmp >> 16) & 0xff;
        o4 = (tmp >> 24) & 0xff;
    while (o1 == 127 ||// 127.0.0.0/8           - Loopback
        (o1 == 0) ||       // 0.0.0.0/8         - Invalid address space
        (o1 == 15 || o1 == 16) || // 15.0.0.0/7 - Hewlett-Packard Company
    … ); // Truncated, includes more blacklists
    return INET_ADDR(o1,o2,o3,o4);
```

Figure 6.2: Selection of the Mirai source code, responsible for the creation of scanning packets [5]

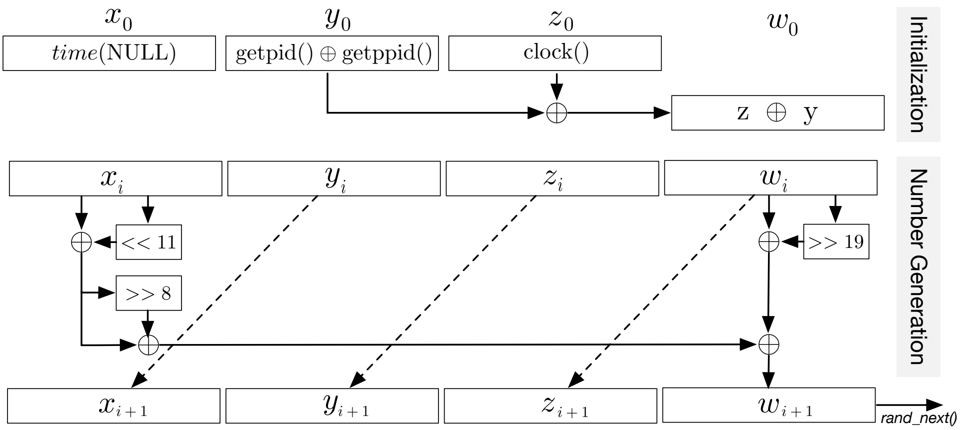


Figure 6.3: RNG initialization and generation.

of (x, y, z, w) with environment variables, in each iteration, the left and the rightmost internal state variables are repeatedly combined with bit-shifts and XOR operations to compute the value of the internal state variable w, which is at the same time the generated random number. Note that the generated randomness is directly taken from the internal state without any output transformation and thus leaks the state information to the outside. The content of the remaining three state variables is shifted by one field to the left, thus the next random number directly depends on the last random number generated and the one from three iterations ago.

While `rand_next()` generates a 32-bit value, its output is truncated when used for the source port and IP ID to the lower 16 bits. Mirai also rejects values lower than 1024 when setting the source port to avoid privileged ports, as well as when the generated target is part of specific IP ranges the malware author chose to skip. The excerpt shows that a new value is drawn if an IP address on the local loopback address (127.0.0.0/8), the 15.0.0.0/8, and 16.0.0.0/8 netblocks. These lists are hardcoded in the source, figure 6.2 shows a truncated list for readability.

6.3. Related work

Over the past two decades, security researchers have tried to identify, analyze and mitigate botnets. One of the first of its kind was Cooke et al. [6], who demonstrated that the activity of botnets might be captured and analyzed by deploying honeypots. While research has studied botnet identification and mitigation, limited focus has been put into analyzing why some botnets are more successful than others and how the increase in botnets has prompted botmasters to compete with each other for control of devices. The relative ease of building botnets and their disruptive power rallied researchers to create detection and mitigation methods for this threat. Although many methods have been proposed [7–12], the continuous evolution of botnets remains a constant challenge for the disruption of these networks [13–17].

One such evolution is the use of poorly configured Internet-of-Things (IoT) devices to build a botnet, which has since gained a lot of attention from the security community [18–23]. When the source code of the IoT-based botnet Mirai was publicly shared [4], new actors could bootstrap their botnet with ease. Antonakakis et al. [3] gave the first comprehensive understanding of this botnet and the effect of the shared source code, describing that new, specialized variants spawned off from Mirai. We build on this large foundation of studies to show these evolutions' impact and how botmasters evolve to trump their competition.

While the source code leak of Mirai enabled many actors to join the IoT game, the raised awareness of these new threats prompted device owners to protect their devices [24]. As a result, the share of devices available to actors has decreased. As the scale of a botnet is important for its monetization [25], by for example using the computation power of infected devices for cryptocurrency mining [26], botmasters have found an increasing number of ways to infect devices. While new infection vectors have been added to also the descendants of Mirai [3], the most commonly probed service is still the telnet port used in the original version of Mirai [27]. The impact of these evolutions on botnets' size and geographic distribution has yet to be evaluated in the scientific community.

Several works have analyzed the structure and infrastructure behind the original Mirai IoT botnet. [23] show the communication pattern between Mirai bots and loader, and show how Hajime has evolved beyond the original Mirai. [28] performed a forensic analysis of the original Mirai botnet, among others, on encoded user credentials. The authors call upon further research to focus on the customizations of Mirai to find these artifacts in them as well. [22] has provided an overview of a handful of Mirai variants, among them Hajime, Persirai, and Brickerbot, show how these variants are used and how they have changed their infection behavior from the original Mirai botnet. Furthermore, [29] found there to be many different password combinations in use by various IoT malware, [30] showed a link between password lists and used tooling during brute-forcing. Similar to the other related work, these papers have also not identified the differences in success between botnets.

Other IoT botnets have also been examined by the security community, with [31] analyzing the Hajime botnet in-depth, [32] surveying the BlackIoT botnet and many reports from industry [33–36]. Similar to the works on the original Mirai botnet, these papers focus on one botnet and do not analyze the effect of multiple botnets competing for the same IoT devices.

In this work, we go beyond state of the art by answering two major questions that were put up but remained unanswered in previous work. First, what is the interaction and competitive behavior of the plethora of IoT and specifically Mirai malware strains in the wild? And second, which factors determine how successful a particular malware can spread?

We extend the work of Antonakakis et al. [3] and in this paper additionally investigate the evolution of Mirai-based botnets and how these adaptations relate to the regions they infect. We do this by introducing a novel way to identify the exact moment a device was compromised, a technique that we also use to track the lifetime of an infection, which allows us to trace the infection characteristics of Mirai and its descendants accurately.

Dataset	Size (Jan-Mar 18)	Purpose
Telescope	1.2 TB	Infected devices, RNG analysis
Honeypots	213 GB	Variant+behavior identification, credentials, staging servers
Netflows	569 GB	Verification and coverage analysis, blacklisting analysis

Table 6.1: Datasets used in this study.

6.4. DATASET

The analysis in this paper is made possible through the combination of three datasets listed in table 6.1, each capturing a different aspect of the worldwide phenomenon of an IoT botnet. First, we use a large network telescope to capture the scanning behavior at the scale of infected Mirai instances looking for other vulnerable hosts. Second, we operate an installation of 7,500 honeypots to determine which particular strain an IoT device is infected with. And third, we use operator netflows to assess the behavior of the botnet at large. We will describe each of these datasets in more detail in the following.

Telescope. As soon as a Mirai variant infects a device, it immediately scans the Internet for other vulnerable hosts. This scanning routine is based on specifically crafted and injected packets and thus displays some implementation particularities. For instance, the malware will always set the TCP sequence number to the destination IP. On startup, a Mirai instance picks random values for window size and source port it will use for every packet until the infection is removed. This allows us to link incoming packets to those potentially sent by Mirai.

To confidently arrive at an estimate of how many and which devices are potentially infected by Mirai, we use a large network telescope consisting of three partially populated /16 networks with ~65,000 IP addresses. These dark IP addresses will only receive scans and Internet backscatter, which can be easily separated based on the TCP SYN or SYN+ACK flags [37]. As Mirai's target selection is random, the large size of the telescope – we monitor about one in 65,000 IP addresses on the Internet – will allow us to quantify and track how long infections are active.

The Honeytrack IoT system. While the telescope provides a basis for counting and tracking a particular infection based on header values, it does not reveal which malware strain has currently infected a device. For this, we developed a distributed honeypot system consisting of an endpoint agent and a backend environment visualized in figure 6.4. The agent listens on TCP port 23 and transparently tunnels incoming packets to an environment where IoT devices are emulated. Incoming requests are randomly allocated to one of eight system images (based on the most common banners in a Censys telnet survey), exposing a fully functional (high engagement) but virtualized IoT device. Once a connection is made, the same IP will always be connected to the same device again, which is subsequently serialized to preserve state if the attacker would be coming back.

For 6 weeks, we deployed 7,500 honeypots in 3 home ISP networks, 2 network ranges of a public cloud, and three network ranges of an enterprise network where they were interleaved on unused IPs between active systems so that the honeypots would blend

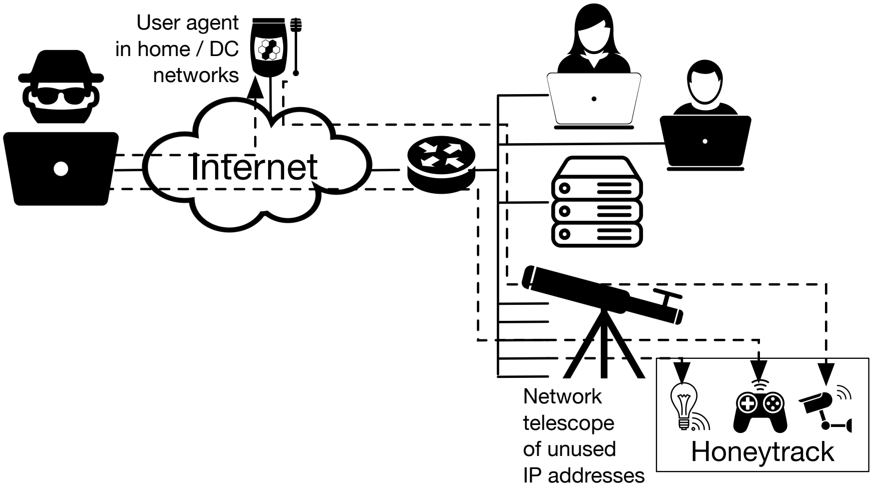


Figure 6.4: The Honeytrack system routes compromization requests into a separate, virtualized environment.

in. For this, we sought permission from the network owners via the established approval mechanisms. The installation recorded a total of 300,000,000 login attempts from 203,920 unique sources, which during the brute-forcing would reveal their hard-coded password list as well as the malware strain. This is possible as the Mirai botnet does not immediately test whether it has successfully logged in with one of the passwords it has sent. Instead, the malware sends a username/password combination followed by a set of commands, the last one being */bin/busybox MIRAI* to the tested device. This has two purposes. If the username/password combination was correct, the command string gets executed. If none of the credentials worked, the busybox command is just another username. If the credentials were correct and the command executed, a router running busybox would respond with “MIRAI: applet not found”, while a host without would throw an error. Mirai thus knows from the response whether a credential was correct, and the system runs busybox. When later botmasters recycled Mirai’s source code to change their own IoT variants, they changed the busybox string. By providing a login and a working shell environment, we can thus track with Honeytrack which variant attacks and whether it was modified to contain different or new credentials.

To assess the role of Mirai and its variants in the entire spectrum of actors targeting telnet and the IoT, we tracked all source IPs that scanned or attempted login over the whole observation period. 87% of all source IPs exhibited Mirai’s special way of crafting packets, thus making this particular IoT malware family the dominant player and most relevant phenomena in the IoT malware arena. For this reason, we have made Mirai and its variants the focus of the discussion in this paper.

Netflows. The above two datasets show which malware strains are currently deployed at which IoT device around the world. To also understand what these devices are doing and how the support infrastructure works, we also use netflows from a Tier 1 network operator. To preserve the privacy of the end users, the source and destination

IP addresses are masked using the prefix-preserving anonymization scheme presented in [38], proven to be semantically secure. This AES masking key is only known to the operator, which does provide us with a mapping of IP addresses that attacked us to their anonymized counterparts. The data access procedure was cleared with the corresponding protection authority of the operator. This setup protects the identity of particular clients, but at the same time, does allow us to understand in which networks and countries particular variants are raging.

6.5. Attacking a Low Entropy PRNG

Although the PRNG of Mirai is relatively simple, it creates good quality output. The cycle length has a period of $2^{96}-1$, and the output passes the Dieharder test suite [39, 40]. Still, LFSRs are not equivalent to cryptographic RNGs and are inherently vulnerable, and the lack of a suitable output transformation effectively leaks the internal state via the generated randomness. If enough consecutive bits become observable, it allows the prediction of future results.

The LFSR used within Mirai depends on four internal state-keeping variables, which means that if we were to learn four consecutive random values, we could predict all future ones. Observing four complete consecutive values is, however, impossible since the `rand_next()` function is used not only in the packet generation but also elsewhere, and the 32-bit output is truncated to 16-bits when setting the IP ID or source port. If we were to obtain two Mirai scan packets back-to-back, we would thus learn twice 32 bit RNG output from the destination IP and twice 16 bits from the IP ID. However, as IP addresses are hit randomly, it is unlikely that two consecutive packets hit our telescope or honeypots. Still, as two subsequent calls to `rand_next()` are used in the creation of a single packet, we can efficiently confirm a correct state guess as the likelihood to obtain a 32-bit, and a 16-bit match is vanishingly low.

Attacking the seed. Any PRNG is only as good as the entropy it has been seeded with. Even a cryptographically strong PRNG can be broken when initialized with (largely) predictable data. Such insufficient entropy is another problem in Mirai's home-grown design. Upon startup of the thread, the router fills the internal state variables with four values: (1) the epoch in seconds, (2) the process ID, (3) the parent process ID, and (4) the number of clock ticks since the launch of the program. In theory, combining these four values could provide 94 bits of entropy on a 32 bit system. In practice, however, the entropy of the random numbers is lower due to several conceptual and implementation mistakes:

- `time(NULL)` returns the time in seconds since January 1, 1970. As Mirai immediately starts to send packets at a high rate, the startup time will be very close to the time of the first arriving packets, especially in a telescope. Even if we very conservatively assume that the device has been infected for 24h before we see the first packet, this would imply a mere 16 bits of entropy. Our measurements have shown this to be much shorter in practice, with the first attack packets reach us on average within 15 minutes after startup.

- Before the RNG is initialized, the process is forked to create a dedicated thread for scanning. As a fork is essentially the start of a new program, the number of clock ticks elapsed starts at 0 for the child process. By the time the RNG is seeded, only four instructions have been executed. This reduces 32 bits of potential entropy to just a handful of bits. In glibc prior to version 2.18, which is frequently used in IoT devices, the resolution of `clock()` was limited to a granularity of 10,000 ticks. Thus, at this stage in the program, possible values are either 0 or 10,000, reducing the entropy to just 1 bit.

- The state y is set to the process ID XORed with the parent process ID. While both process IDs are 15 bits, the combination of both also has 15 bits of entropy.

Due to these issues, the 94-bit seed has in practice only 32 bits of entropy. We can analytically derive which bits in the four registers contain entropy. However, given that LFSRs can be efficiently computed, it is trivial to brute force the remaining 32 bits and evaluate the sequence to find a value pair where the 16 bits used in the source port and window size match the output.

As the source port and window size only provide the last 16 bits generated, multiple seeds in our search space will generate these numbers in correct succession. To verify which of the matching combinations is the actual seed, we can generate the sequence of numbers and verify if this matches packets coming into the telescope. In these sequences, we can identify whether an IP ID and destination IP address are generated consecutively and repeat the process until we have only one candidate seed left, the right seed. In most cases, we only have to generate the sequence until the first IP address to find the correct seed. Generating the sequence until the first observed IP address is a process that will terminate for the correct seed, but for incorrect seeds, it will run in the worst case indefinitely. The number of steps we have to take depends on the statistical likelihood to be hit by a Mirai infected device, and as our telescope contains more than 2^{16} IP addresses, a fully random scan would target one of our IPs after, on average 2^{15} steps. For 99% certainty that a seed is not correct, we can stop our verification if it does not generate the first observed combination from the telescope within 2^{25} steps. Therefore, program execution can be halted soon without potentially losing candidate solutions and yields the seed within 100 milliseconds.

6.6. Mirai's Battle over the IoT

The attack on the random number generator and the ability to efficiently compute the seed value allows us to characterize the current state and behavior of Mirai infections worldwide. When receiving scan and brute-forcing attempts into our telescope and honeypots, we can determine the exact moment when a particular IoT device was infected. Since packets have the same random source port and window size, we can relate all subsequent interactions to a specific infection of which we know the variant type from the honeypot interactions. Finally, as soon as no further connection attempts appear within the expected inter-arrival time, we can tag the device as cleaned again due to a reboot or being patched by the owner. We will use this method to look at the lifetime of infections, first the infection characteristics in general, and second operational aspects such as the regional biases and other modifications that have been introduced into the source code by later botmasters.

	Variant	Hosts		Variant	Hosts
1	MIORI	75,249	6	MASUTA	5,338
2	MIRAI	62,235	7	NGRLS	5,113
3	JOSHO	23,487	8	SORA	4,631
4	daddyl33t	12,583	9	RBGLZ	4,076
5	Cult	5,621	10	OWARI	2,201

Table 6.2: Unique hosts for the top 10 advertised botnets.

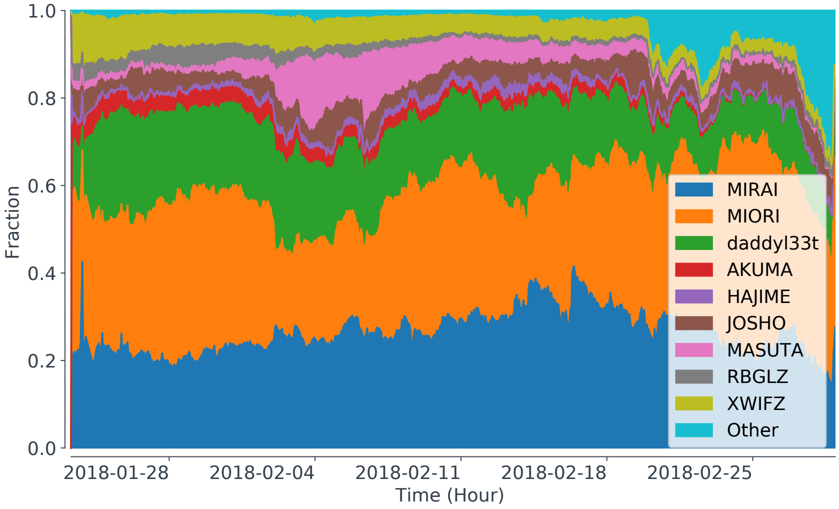


Figure 6.5: Marketshare of advertised variants.

In total, we received scanning and credential brute-forcing from 203,920 hosts that matched the Mirai fingerprint. This number is significantly lower than the peak size of Mirai in 2017 [3], but we find a similar total number as the 200,000 - 300,000 reported by [3]. While our honeypots would welcome any telnet traffic, we see that with 87% of all telnet compromisation attempts, Mirai and its siblings are the dominant players in the IoT malware arena. Table 6.2 lists the top variants and how many hosts were observed in our study, the largest one infecting more than 75,000 hosts. We have identified 39 variants in total, with only the top 10 being advertised from more than 2,000 hosts during our study. The complete list of variants covered in this paper is in the appendix.

6.6.1. INFECTION CHARACTERISTICS

As Mirai spreads like a worm, the growth of the infections is in principle exponential but is naturally bounded by the total number of vulnerable machines. After the rapid expansion, the infection will reach a steady-state or die out depending on the parameters of injection and curing. Indeed, Antonakakis et al. [3] show that Mirai rapidly grew in size at the beginning, but after a while, the total number of infected devices fluctuated between 200,000 and 300,000. When we measure Mirai a year later, we still find the ecosys-

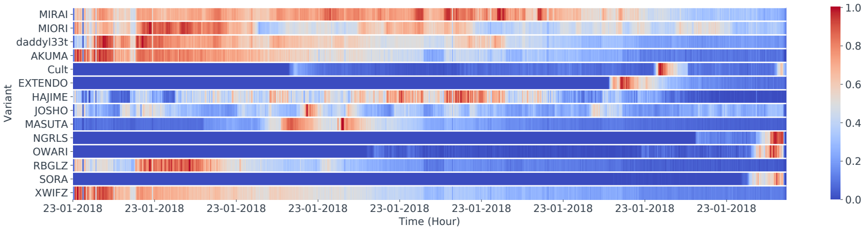


Figure 6.6: Lifetime distribution of variants over time, normalized per variant. Red shows peak per hour, blue shows little hosts being infected by the variant.

tem of Mirai-infected IoT devices in a steady-state, except the infected population, has significantly decreased.

Epidemiologically speaking, the memory-bound infection follows a mixed SIS (Susceptible-Infected-Susceptible) or SIR (Susceptible-Infected-Recovered) process [41]. When we measure the infection rate, we see that, on average, 27,182 devices are newly infected each day with a standard deviation of 18,381. With the curing rate being almost identical – 26,757 devices are cleaned up every day with 19,153 as standard deviation –, it is clear why the ecosystem is in a steady state. We empirically determine a basic reproduction number of $R_0 = 1.0033$, which is surprisingly low given the aggressive scanning and infection behavior and barely enough for the worm to sustain itself. As Mirai targets IPs randomly, it is highly unlikely for pockets of undiscovered, vulnerable devices to still exist, hence new infections can only come from newly introduced vulnerable devices or from taking over devices from someone else.

Indeed, we find that the battle over the Internet of Things is largely a zero-sum game, with significantly fluctuating market shares between variants. Figure 6.5 shows the distribution of infected devices over the largest botnets, which surprisingly shows that the original Mirai is today only one player among many, and variants that evolved from it have gained significant market share. Surprisingly so, this competition between Mirai variants has received only little attention to date. When we count the number of devices infected by a particular strain at any given moment, we see momentary explosions in activity, where a lot of hosts suddenly get infected by a new variant. As shown in figure 6.6, this is especially true for smaller strains such as Cult, MASUTA, or OWARI, but even the main players in the ecosystem experience frequent jolts in installation size. These spikes are counterintuitive, as brute-forcing from thousands of infected hosts towards random IPs should lead to a continuous influx of new infections. We can explain the loaders and reboots as we show later.

TRANSITIONS BETWEEN BOTNETS

While new IoT devices are constantly being added to the Internet, awareness and improved security (such as omitting open admin interfaces or default credentials) should reduce the attack surface over time. This would mean that to increase their botnet, botmasters could, in the long run, only grow by compromising existing devices, potentially already under the control of another botmaster. Indeed, we already observe this, with infections rarely occurring on "fresh" devices and 88.5% of all new infections on systems that have been exploited previously.

Botmasters do not want to share an infected device with other botnets, so when a botnet infects a device, one of its first actions is to kill processes such as telnet and ssh that can be used to infect this device. While this ensures that the device cannot be accessed after infection, once the infection is removed from the device – typically after a reboot – it will be vulnerable again. Such reinfections happen comparatively fast, the average duration to be reinfected is 1 day and 9 hours, with a standard deviation of 4 days and 7 hours. We determined the type of device based on daily IPv4 banner grabs from Shodan and Censys. Surprisingly, we see that routers frequently reset in practice, and remain infected for only 12 hours (standard deviation 89 hours). Still, takeovers can happen if the telnet port was not exposed to TCP port 23 but to port 2323. While Mirai brute forces on both ports, it only eliminates programs listening on port 23. These devices can hence be “stolen” from other adversaries.

Figure 6.7 shows such takeover behavior for one example device. The device is reset on multiple occasions, indicated by the red triangles, but immediately reinfected. From the figure we see that this particular device is mostly reinfected with the same malware, but frequently another variant takes over. In the time between reboot and reinfection, IoT devices are up for grabs. Figure 6.9 shows the transition behavior between infections for all devices after a reboot for four of the major variants encountered. It shows that botnets do at a global scale lose some IoT devices towards other major players, however, in most cases, the reinfections happen with the same malware strain. This is made possible as successful login attempts are reported to the C&C server, thus providing the botmaster with a list of good credentials and facilitating easy reinfection. This feature was already part of the original Mirai. As the reinfections by the same malware are so successful, it might be the case that botmasters actually use the alive pings sent to the C&C server to monitor whether a device is still active or has dropped off infect the device again. By monitoring and reinfecting devices in this manner, they could preempt another botnet from taking over the device. While we do find indications for this hypothesis given that the average time for a device to be reinfected with the same malware is only 1.5 hours instead of the average infection time of 1 day and 9 hours, it is impossible to prove this by merely observing the ecosystem.

Figure 6.9 however, also shows that a large portion of devices is never infected again after cleanup, denoted by an edge going to the “end” node. As it is unlikely for a device not to be invaded or reinfected, the device is most likely secured by its owner. In total, 175k devices are infected by only one variant throughout this study, whereas 28k devices were taken over at least once.

Without information about “infectable” devices, the only way for botnets to locate victims is based on port scanning. As Mirai performs scanning based on randomly chosen candidate IPs, the largest botnets should be most likely to find these restarted devices to add them to their network and would therefore need the least time to find and infect a device that another malware has just cleaned up. Figure 6.10 shows the probability density function for different variants and shows that the larger the variant, the quicker it takes over a device. Size does matter: we find a strong negative correlation between the original size of a botnet and the time it takes to accumulate new devices and grow, that with a Pearson coefficient of -0.501 ($p < 0.01$) can explain half of the difference in

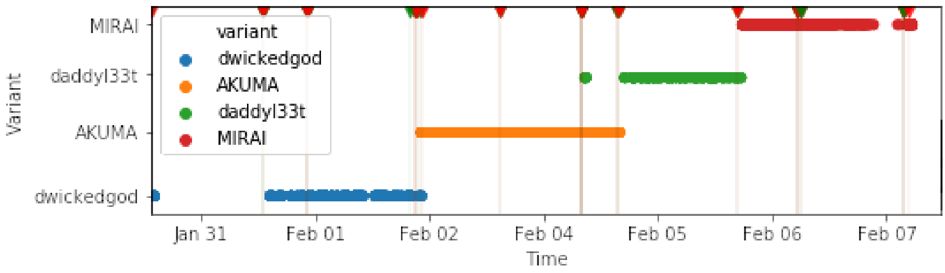


Figure 6.7: Devices get cleaned up and get reinfected by the same malware variant, until another variant takes over on the restart of a device.

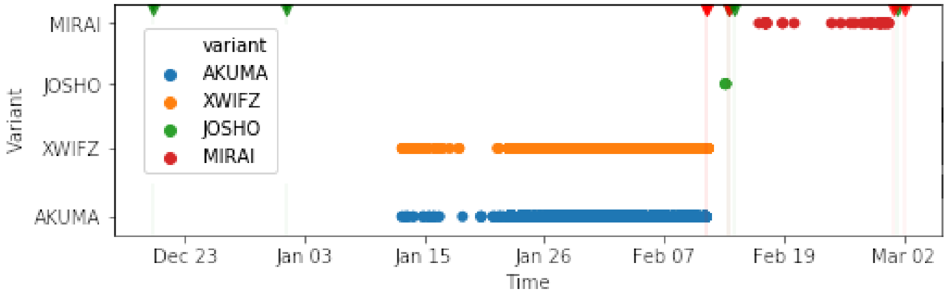


Figure 6.8: Concurrent infections on 1 IP. The first started 10 days before the second, in mid Jan our setup launched and registered both variants.

infection behavior between strains.

CONCURRENT INFECTIONS

To gain exclusivity over a device, the Mirai malware shuts down all processes running on ports 22, 23 and 80, and binds itself to these ports to prevent other processes from doing so. While this is an effective way of keeping others out, it does not fully eliminate the possibility of other bots attacking certain devices. As telnet is a protocol that receives much-unsolicited traffic [42], network operators sometimes bind their telnet not to port 23 but port 2323. The original Mirai takes this into account by scanning port 2323 in every 10th scanning packet. While the scanning part takes this into account, the port killing part of the source code does not. Therefore, devices using port 2323 as their telnet port will continue to present an open port even after infection. Figure 6.8 shows a host being infected with two variants at the same time, XWIFZ and AKUMA, before restarting and getting infected afterward with other variants. In total, we observe 8.9% of the total infected devices being infected by multiple variants at one time. Note that this behavior results from a co-infection and not of two devices behind a NAT, when the packet generation of both stops simultaneously. In [43], we further elaborate on the issue of NATs and show how the faulty RNG can be used to identify NATs, and thus can be used to quantify IP churn across the Internet.

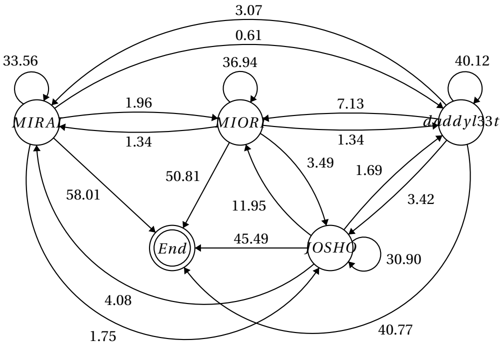


Figure 6.9: Device transitions between botnets in percentage of out-degree over the entire data collection period.

While this oversight in the Mirai source code transfers to most of its descendants, several malware authors tried to understand the inner workings and have removed this oversight from their codebase. When looking at the variants present on concurrently infected devices, we can identify several variants that successfully kill their competitors and lock them out. While MIORI and JOSHO are present on 5680 and 3726 concurrently infected devices, we find no rivals when MASUTA and SORA are running, showing them more effective in stopping competition.

Compatibility of Malware

While botnet size significantly influences its ability to infect new victims, we also see major differences in infection characteristics between different countries and autonomous systems (ASes). Figure 6.11 shows the duration a particular strain has control over a given device until it reboots, for a selection of ASes which more than 1,000 compromised devices. While the average duration of infection is relatively short, there are major outliers where a large share of IoT malware can hold on to devices for up to a week or longer. As can be seen on the box plots, average values of infection times are not a good measure to understand the behavior of IoT malware, given the sheer number of these outliers and their deviation from the mean. Not only are the distributions highly heterogeneous, but we also see the emergence of clusters, indicating that there are groups with similar behavior.

We can clarify this behavior better if we look at infection characteristics not only from the perspective of the victim device but in symbiosis with a particular malware strain. Figure 6.12 breaks down the infection duration for one of the ASes with large outliers. Here, we see that the time a particular malware strain holds a device captive is vastly different. In AS9121, MIORI and MASUTA are largely unsuccessful in maintaining a foothold, and devices reboot on average after 106 and 32 minutes. The fact that in 99% of cases neither malware runs longer than 239 minutes on vulnerable IoT devices within this AS before the device resets and is later infected by something else, suggests some incompatibility of this malware or the way it is used with a particular type of host pre-

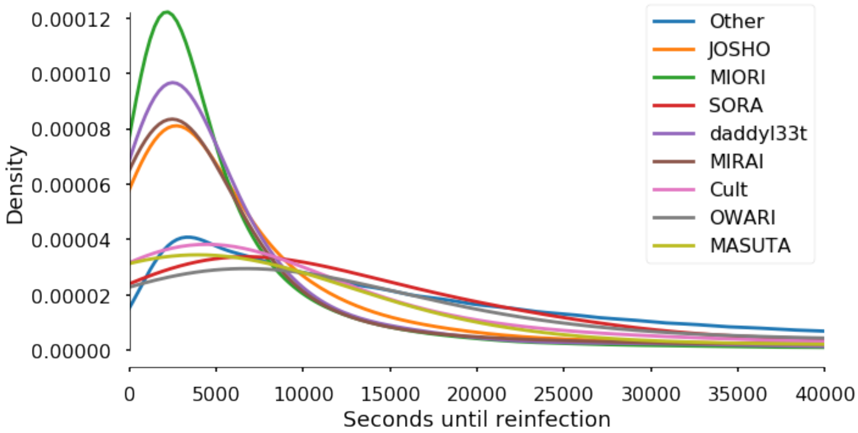


Figure 6.10: Probability density function of the time for a variant to take over a device previously "owned" by another.

dominantly used in this network. We can rule out that this behavior results from coordinated cleanup activities or some centralized reboots due to a power outage, as devices eventually and at different times fall victim again. As we see from the graph, it appears that JOSHO is a more suitable IoT malware for this AS, while the best performance is achieved by Hajime with an average infection duration 9 times larger than the average for this AS, and even the shorter Hajime infections outperforming the best performers of the other IoT malware by far. We find similar heterogeneity for many ASes and malware types. It seems that malware frequently causes issues, and reboots of IoT devices can be attributed not just to external influences (user reboots, power outages, updates, etc.) but to a significant degree to the malware itself.

6.6.2. CUSTOMIZATION AND EVOLUTION

Different types of malware seem to work better or worse depending on the AS devices are located in. Once an IoT device reboots, it is just a matter of time before it is rediscovered and reinfected. As shown previously, the larger a botnet is, the faster it can locate routers to potentially infect. This makes it hard for new botnets to enter the "market" and compete successfully for vulnerable devices. In response to this, it would make sense for botmasters to customize their strategy where to look for victims and localize productive niches to exploit [26]. On the one hand, such customization could be done by only targeting the address space of particular ASes, or on the other hand, be done based on curating credential lists. Using open-source intelligence, actors could identify default username/password combinations for devices that occur frequently but are not so mainstream that other botmasters might target them.

REGIONAL BIASES

One way of creating a niche in which a botnet can thrive is by targeting devices, not in demand by others. In the case of Mirai, which sole attack vector is the use of weak cre-

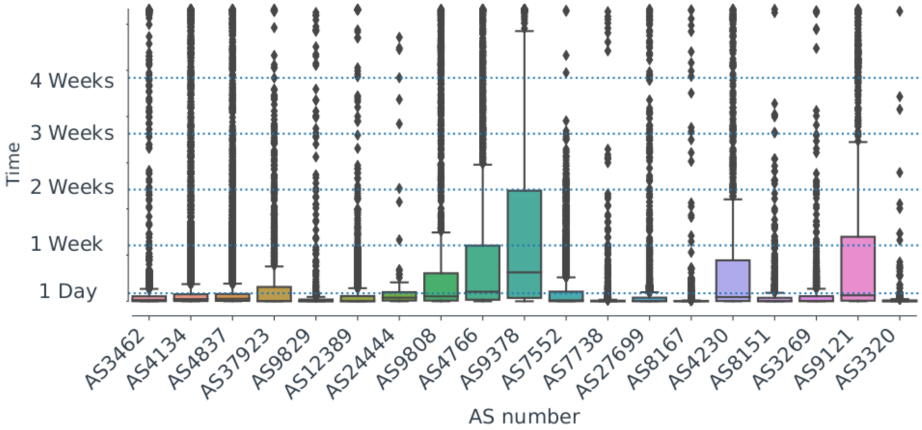


Figure 6.11: Infection times per AS with more than 1,000 infections, showing large differences between the infection times of different ASes.

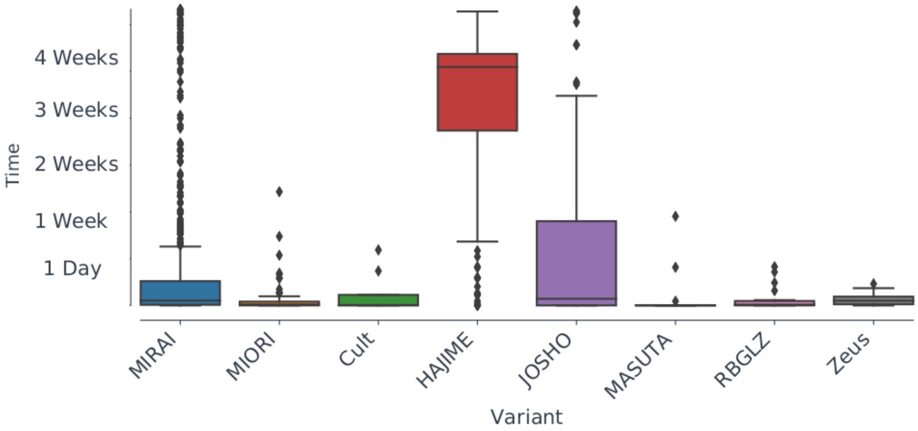


Figure 6.12: Infection times of different variants on AS9121, showing large differences of variants within ASes.

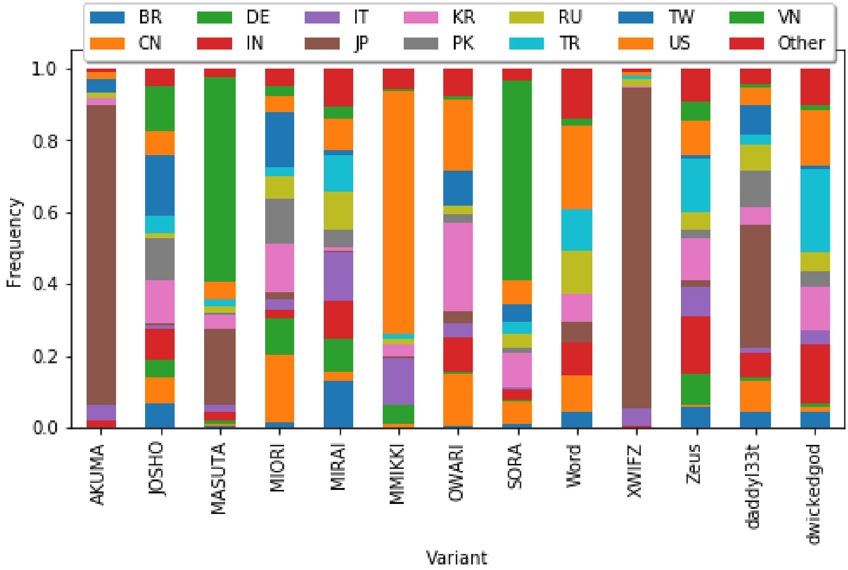


Figure 6.13: Regional infections per variant, normalized over the total number of seen IoT devices in a country. Several show heavy biases, mainly due to targeted password lists.

dentials, the way to target new devices would be to add a new username and password combinations that no other variant uses.

An effective approach would be to include credentials used by a significant number of devices as their default. With devices such as routers deployed in bulk by telecom operators to their customers, the location of these devices can be heavily biased to a region. When a variant targets such a device, it would grow its influence in a specific geographic location. In figure 6.13, the distribution per country is plotted for different Mirai variants. The biggest botnets, MIORI and Mirai, occur widely. Others, however, such as AKUMA and MASUTA, are heavily biased toward Japan and Vietnam, respectively. This imbalance stems from the credentials: AKUMA includes 17 unique username and password combinations not seen in other variants, including the combination "admin,oelinux123". This combination is targeted at the EE 4GEE HH70 ROUTER, a mobile WiFi router especially popular in Japan. Masuta, on the other hand, has only one unique combination: "root,00000", the default credentials used to log in to an old DVR. After adding this, the MASUTA botnet took over 2645 hosts in Vietnam in one single day, growing their botnet by a factor 4 from 810 infected to 3455 hosts.

Increasing your market share

Specializing on ASes pays off when one can find networks with a large number of vulnerable devices. After the initial foothold, variants seem to effectively maintain the majority share of an AS, making the initial foothold even more important. Botmasters can optimize their botnet for certain ASes by changing the credential list and removing those that will not be successful as bots execute a limited number of login attempts. By doing

Name	R	p
MASUTA	-0.064	<0.1
Cult	-0.086	<0.05
OWARI	-0.120	<0.001
daddyl33t	-0.124	<0.001
XWIFZ	-0.140	<0.001
dwickedgod	-0.170	<0.001
MIORI	-0.172	<0.001
MIRAI	-0.179	<0.001
HAJIME	-0.188	<0.001
JOSHO	-0.206	<0.001
OBJPRN	-0.663	<0.001

Table 6.3: Correlation between botnet size and its growth.

AS	R	p
Frontier Communications of America Inc.	-0.519	< 0.001
asn for Heilongjiang Provincial Net of CT	-0.511	< 0.001
Ratt Internet Kapacitet i Sverige AB	-0.501	< 0.001
Bredband2 AB	-0.484	< 0.001
Bredbandsson AB	-0.475	< 0.001
Viettel Group	-0.129	< 0.001
OPTAGE Inc.	-0.127	< 0.001
Jupiter Telecommunications Co. Ltd.	-0.123	< 0.001
Jupiter Telecommunication Co. Ltd	-0.119	< 0.001
NTT Communications Corporation	-0.104	< 0.01

Table 6.4: Correlation between the size of an AS and its growth, ordered by coefficient for the top and bottom 5 ASes.

so, the effectiveness in other ASes might be impacted due to the deletion of important credentials. For MIORI, Mirai and JOSHO, we have observed infected devices belonging to the same variant from several ASes brute-forcing our honeypots using different password lists, showing that malware authors diversify and launch different versions to maximize effectiveness.

Another way to grow a botnet is by better targeting the scanning activity towards vulnerable devices. While Mirai chose to send packets to either port 23, or with 10% probability to 2323, we observe botmasters removing either one of the two to make the scan more focused towards one port. Removing 2323 does, in theory, make sense, as Shodan reports there are more than 20 times as many devices on port 23 than on 2323. We classify a host actively avoiding a port when we have observed 44 packets but none on the considered port, as this gives us a 99% chance the port should have been observed, and find that mainly variants of Mirai and JOSHO avoiding port 2323 with 7484 and 4558 hosts only scanning 23. On the other hand, we find 105 hosts belonging to MASUTA only targeting port 2323. For hosts avoiding port 2323, we find no difference in distribution over ASes.

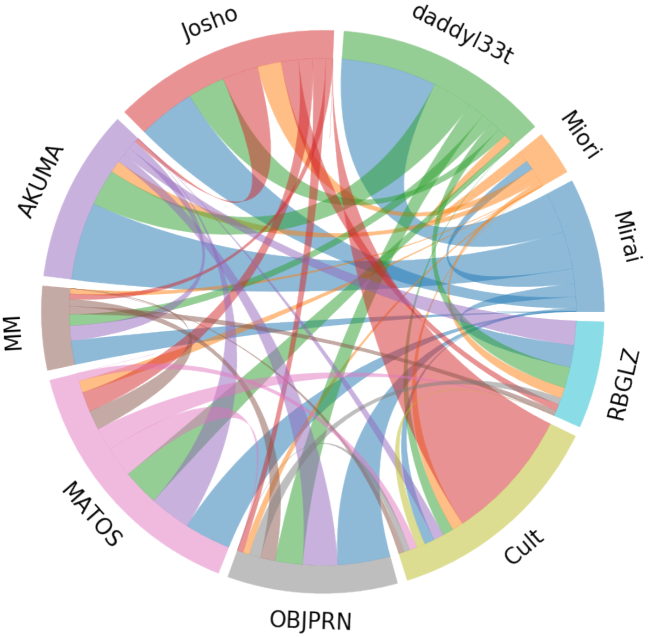


Figure 6.14: Passwords adoption between variants. Outgoing edges denote a credential adoption, edges are colored in the color of the variant they came from.

WATCHING AND LEARNING FROM YOUR COMPETITORS

If such strategies are fruitful and actors are aware of the success, we would assume those with a competitive advantage to proliferate. As unique passwords are one success factor, we tracked how Mirai variants introduced new credentials during our study and whether others would adopt these combinations to their codebase.

Figure 6.14 shows password adoption behavior of the largest Mirai variants and the smaller but particularly noteworthy strains MM and OBJORN. The size of the connections shows the number of passwords that get transferred from one to the other, with the color of the edge denoting the variant that pioneered the password. What immediately springs out is that all variants except Mirai engage in password adoption but that the two largest botnets predominantly operate on credentials they have pioneered themselves. Mirai does not introduce external credentials, and the other large player, MIORI, borrows a minor share only from Mirai. Overall, smaller botnets are generally much more likely to pick up passwords (MATOS/MM/OBJPRN) rather than being innovative on their own. Even more surprisingly, we see that copying behavior follows some form of hierarchy. The large botnets only feed on other large botnets but not smaller ones. There are also clear preferred relationships, for instance, passwords used in the variant Cult predominantly originate from JOSHO.

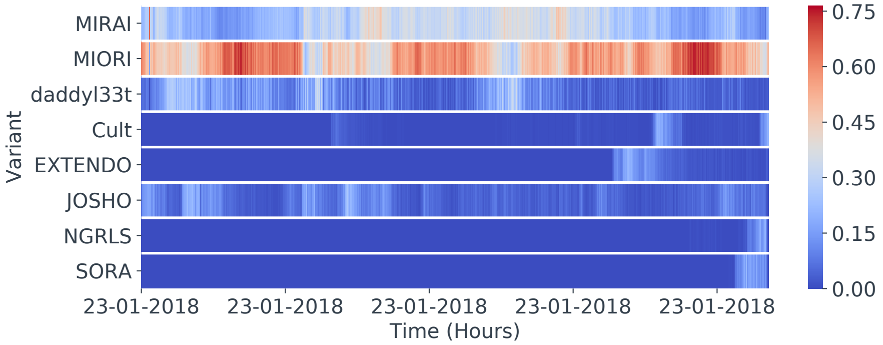


Figure 6.15: Infection distribution for AS4134 per hour, in percentage of infected devices per variant. The AS is highly dominated by MIORI, others only hold a small percentage.

KEEPING THE NET ALIVE

If a large botnet is amassed, the worm-like structure of the botnet should ideally be able to sustain the loss of devices by growing at least as fast as they decay. The authors in [3] refer to this as the stable state and find that at the end of their measurement period, the overall size of the Mirai botnet was shrinking. When looking at the different strains, we also observe all variants shrink over time, and there is an overall negative correlation between the size of a botnet and its growth ($r = -0.16$, $p < 0.01$). Table 6.3 shows these correlations per variant and shows large differences between the variants, where large ones such as Mirai, MIORI, and JOSHO are below average, but others such as OBJPRN are much less likely to survive. The survival rate of a botnet seems to be dependent on the time it takes for bots to find a victim to brute force, giving a correlation of $r = -0.516$ with $p = .02$ between the correlation scores in Table 6.3 and the average time between brute force attempts on our honeypots from a single infected host.

Table 6.4 shows the growth rates of the top and bottom performing ASes in our study. We confirm the findings by [3], who showed a decline in the number of bots on certain ASes. We have not observed ASes with a positive correlation between the number of infected devices and the growth of this number. This shows that the botnets are slowly losing their grip on IoT devices.

LOADING INFRASTRUCTURE

Devices infected by Mirai are used to scan the Internet to help infect new devices by brute-forcing passwords. After the correct credentials are found, and the device has logged in, the credentials are sent to a server responsible for the infection of new devices [3, 23]. Figure 6.16 shows this process. Making a centralized server responsible for the infection makes it easier to update the software loaded onto new devices or customize infections if new devices are discovered.

Identifying these loading infrastructures is trivial, as they always immediately provide the correct credentials when they log in to our honeypots and never input false credentials. If infections occurred from the infected devices themselves, these would

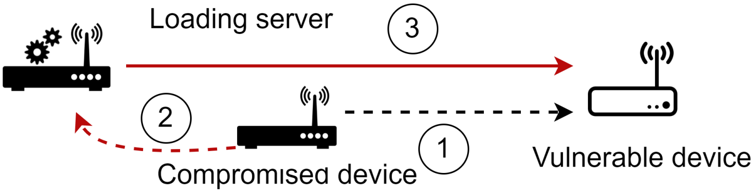


Figure 6.16: In Mirai's loading infrastructure, bots identify devices where they can log in (1). These are then reported to a loading server (2), which then performs the infection (3).

need at least several tries to guess the correct combination before loading the malware onto the device. By identifying the bot that has successfully brute-forced the password that is later sent by the loading infrastructure, we can identify which variant connects to which loading IP address. Figure 6.17 shows IP addresses used for loading the malware on devices and the bot variant responsible for letting the server know the correct password. Each colored circle represents a loading server, where the color indicates the variant it spreads and the size of the number of installations it does, respectively. Individual dots mark the IP address of an infected device, the line to a circle which loading server it provided the credentials to. The color of the line represents the malware variant the infected IP had at that moment. We find that XWIFZ and AKUMA both use many loading servers relative to other malware strains. Even more surprisingly, we see that a large portion of this infrastructure base is shared, as either a bot of AKUMA or XWIFZ has brute-forced a password correctly before one of these loaders logged in. For MIRAI we can observe a few small clusters, using the variant name MIRAI as its loader, while in the original source code [5] the loader for MIRAI identified itself with the ECCHI string, which means that actors also adapt the software in the backend.

Almost all variants use centralized loading servers and are prone to takedowns and IP-based blocking of enabling infrastructure. To limit the risk of a takedown or block, botmasters might opt to change their loading infrastructure by regularly using DNS. In practice, however, we do not see this behavior in any centralized variant, as identified loading servers have been active over the entire period our honeypots were active. Only two variants have significantly changed their loading infrastructure and use the bots to scan and perform the infection, making these botnets significantly harder to block. One of these botnets, identified by eight random characters as variant names, spreads its loading servers throughout the network and cycles them around, making it hard to identify and block them.

Mirai's source code included the original loader used by Mirai, which consequently was copied by its descendants. As with the password lists, the loading code has been altered by some of the variants to remove indicators from the issued commands [44]. Alterations include the original ECCHI variant string, file names created on the system, and the entire loading procedure. These alterations extend into the commands issued by the bots after a compromisation, where some variants already request more data from the device such as the "/proc/mounts" file. Table 6.5 shows the changes made by different variant authors in these criteria and shows that full customization is rare. Additionally, the last two loader services are shown in figure 6.17 change their command

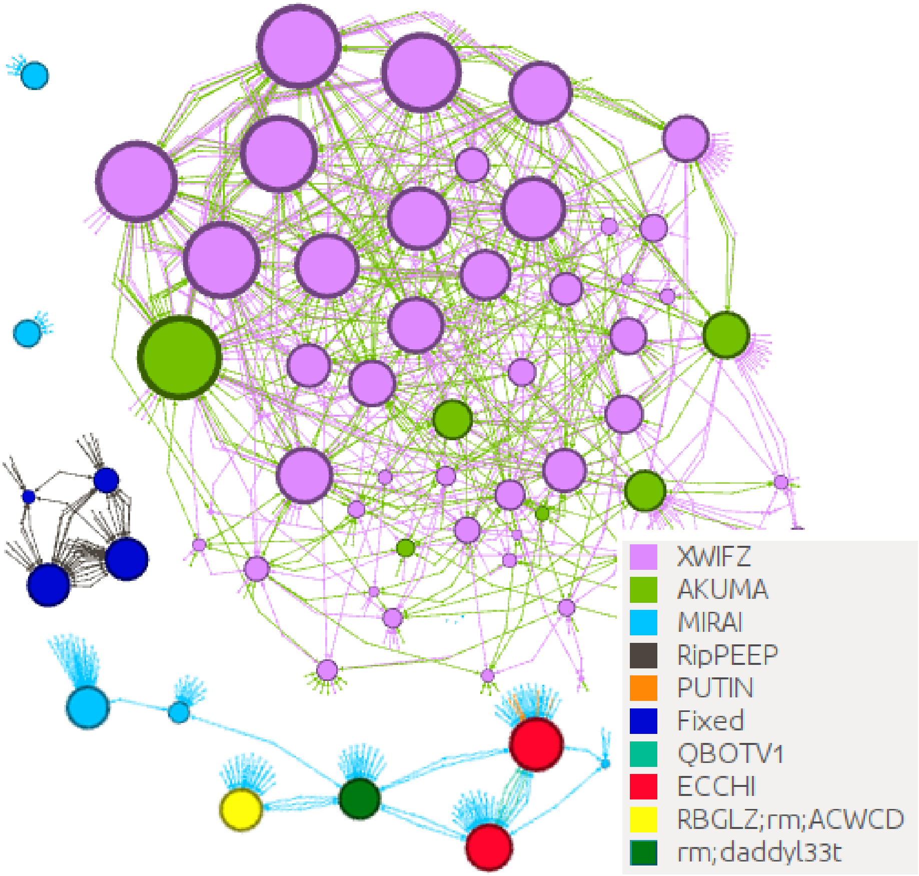


Figure 6.17: Identified loading servers for several variants. Edges denote successful brute force attempts and are colored by the variant of the brute-forcing bot.

structure over time. We first identify the server “RBGLZ;rm;ACWCD” as variant ACWCD, but the server changes the loader code and its identifier over time. The loader server for “rm;daddyl33t” behaves the same, first identifying with daddyl33t, and later changing the commands sent from the infrastructure. Curiously, both modified the identification string to include commands.

HONEYPOT EVASION

In our discussion of the source code, we have pointed out that Mirai skipped over IP addresses it randomly generated if they would fall within a particular range. The original Mirai blocked probes to multicast ranges or invalid addresses, such as `0.0.0.0/8`, as well as selected large organizational networks such as the Department of Defense and the US Postal Service.

To remain undetected and speed up the scanning process by skipping known IP ad-

Name	Loader name	Load via bots	Loader commands	Bot shell entries
XWIFZ	✓			
RipPEEP	✓		✓	✓
PUTIN				✓
HAJIME	✓	✓	✓	✓
daddyl33t				✓
8-characters	✓	✓	✓	

Table 6.5: Changes in bot and loader code across variants from the original Mirai source code.

	Network	**Akiru**	**AKUMA**	**JOSHO**	**MIRAI**	**MIORI**	**OWARI**	**RBGLZ**	**SORA**
Enter.	authors1 /16		✓	✓	✓	✓			
	authors2 /16		✓	✓	✓	✓			
	authors3 /16		✓	✓	✓	✓	✓	✓	✓
Cloud	165.227/16	✓	✓	✓	✓	✓	✓	✓	✓
	167.99/16	✓	✓	✓	✓	✓	✓	✓	✓
	172.31/16	✓	✓	✓	✓	✓	✓		✓
Home	80.114/16		✓	✓	✓	✓	✓		✓
	83.172/16	✓	✓	✓	✓	✓	✓		✓

Table 6.6: Honeypot locations hit by variants of Mirai.

dresses that are not of interest, adversaries can update the blocklist to include these IP ranges. To quantify to which extent malware variants update and utilize blocklisting, we distributed our honeypot agents in 8 different /16 networks assigned to Internet Service Providers, public clouds, and one enterprise network. By comparing incoming probes across the different ranges coming from the same source during the same infection period, we can quantify which Mirai variants update and customize their blocklists. Table 6.6 lists a selection of variants and the ranges we have observed them at. We see that the largest malware MIRAI, MIORI, or JOSHO, show no discriminative behavior but that smaller variants such as OWARI selectively exclude the enterprise range. Cloud providers are attractive across all variants, even though one would not expect to find many IoT devices in these networks. The ranges of DigitalOcean (165.227/16 and 169.99/16) are, for example, not excluded by any single variant.

This blocklisting behavior indicates that adversaries are conscious about where to find victims or where their activities might be monitored. Multiple variants perform evasion, and the ranges they evade are similar, which could mean that all of these actors either research where infrastructure is located or that locations are shared among actors. While the first reason is hard to verify, we have found evidence of these lists being shared online [45], labeled with the exact institutions or organizations these ranges belong to. Additionally, a plethora of blocklists that can be readily put into source code exists online, which block major Internet ranges.

Verifying Blocklists. We verify whether a variant blacklists additional IP addresses in two ways: First, as Mirai's target selection merely skips generated IP addresses if the target appears on the list, we can use this feature and our ability to efficiently bruteforce the seed to selectively test which blacklist an actor is using. Whenever IP addresses are skipped, the state of the RNG is advanced, and we can now test whether we can break the seed and reproduce the sequence of probing packets given one of the lists we could locate online. Second, we use netflows from a Tier-1 operator to verify that the bots do indeed never probe the particular IP ranges.

We find that some botmasters go the extra mile of updating this part of the Mirai source code. Akiru, OWARI, RBGLZ, and SORA have adopted customized blocklists, whereas MIORI, AKUMA, and JOSHO run with the original Mirai algorithm. Especially larger variants are not homogeneous, and there exists not just one version that identifies itself as Mirai. 93% of all hosts infected by Mirai is running the source, but 7% have adapted lists. Even within these, we find differences in behavior, with some of them blocklisting some of the ranges we monitor but not others, while some versions block more extensively.

6.7. LIMITATIONS

Infected devices had to connect to either our telescope or one of our 7,500 honeypots to be detected by us. As discussed above, Mirai and its variants employ a blocklist, and there may be variants that do not connect to any of the eight /16 networks we were present in. To assess potential blind spots, the Tier1 operator collected the set of IP addresses that showed brute-forcing behavior on telnet on a given day, which we compared against the list of hosts that brute-forced us that day. This revealed 2.9% additional IP addresses that were targeting hosts on the Internet via telnet but not present in our analysis, thus our study provides an almost complete picture of telnet brute-forcing, with Mirai accounting for 87% also a solid assessment of the Mirai ecosystem.

For this study, we define malware as Mirai-based given distinct features in the packet generation process, such as a TCP sequence number - IP match and the fixed window size and IP ID fixed for the session. While later botmasters copied the original Mirai source code, they also introduced modifications. These were functional in nature, to introduce new features or avoid certain ranges. Some may modify the packet generation itself, such as randomizing every header value with an RNG unrelated to the original one. These variants are not part of our analysis as they would not stand out based on structural features. Given that the netflows reveal only very few IP addresses going after telnet but not in our honeypot dataset indicates that such deep modification would rarely occur if at all.

6.8. CONCLUSION

After the source code of the Mirai botnet was shared, many new actors have sprung up to take advantage of misconfigured IoT devices. In this work, we exploited a flaw in the design and entropy of Mirai's RNG, allowing us to track the exact infection time of a host and track how infections are evolving. We observe a continuous battle over the Internet of Things among different strains. We find that these IoT botnets on their own are not self-sustaining. The success of malware variants depends on their installation size and how well they seem adapted to occupy specific niches of vulnerable devices.

REFERENCES

[1] *Ddos attack on ovh,* https://www.ovh.com/world/news/articles/a2367.the-ddos-that-didnt-break-the-camels-vac.

[2] *Ddos attack on dyn dns,* https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, *Understanding the mirai botnet,* in *26th USENIX security symposium (USENIX Security 17)* (2017) pp. 1093–1110.

[4] *Mirai source code shared,* (), https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/.

[5] *Mirai scanner source code,* (), https://github.com/jgamblin/Mirai-Source-Code/blob/3273043e1ef9c0bb41bd9fcdc5317f7b797a2a94/mirai/bot/scanner.c.

[6] E. Cooke, F. Jahanian, and D. McPherson, *The zombie roundup: Understanding, detecting, and disrupting botnets.* SRUTI **5**, 6 (2005).

[7] A. Karasaridis, B. Rexroad, D. A. Hoeflin, *et al.*, *Wide-scale botnet detection and characterization.* HotBots **7**, 7 (2007).

[8] G. Gu, R. Perdisci, J. Zhang, and W. Lee, *Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection,* (2008).

[9] J. Wang and I. C. Paschalidis, *Botnet detection based on anomaly and community detection,* IEEE Transactions on Control of Network Systems **4**, 392 (2016).

[10] C. Mazzariello, *Irc traffic analysis for botnet detection,* in *The Fourth International Conference on Information Assurance and Security* (2008).

[11] J. R. Binkley and S. Singh, *An algorithm for anomaly-based botnet detection.* SRUTI **6**, 7 (2006).

[12] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, *Botnet detection based on network behavior,* in *Botnet detection* (Springer, 2008) pp. 1–24.

[13] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, *From throw-away traffic to bots: detecting the rise of dga-based malware,* in *Presented as part of the 21st USENIX Security Symposium* (2012) pp. 491–506.

[14] F. Haddadi and A. N. Zincir-Heywood, *Botnet detection system analysis on the effect of botnet evolution and feature representation,* in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation* (2015) pp. 893–900.

[15] C. Li, W. Jiang, and X. Zou, *Botnet: Survey and case study,* in *Fourth International Conference on Innovative Computing, Information and Control (ICICIC)* (2009).

[16] M. R. Rostami, M. Eslahi, B. Shanmugam, and Z. Ismail, *Botnet evolution: Network traffic indicators,* in *International Symposium on Biometrics and Security Technologies (ISBAST)* (2014).

[17] S. Pletinckx, C. Trap, and C. Doerr, *Malware coordination using the blockchain: An analysis of the cerber ransomware,* in *2018 IEEE Conference on Communications and Network Security (CNS)* (IEEE, 2018) pp. 1–9.

[18] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, *Iotpot: analysing the rise of iot compromises,* in *9th USENIX Workshop on Offensive Technologies (WOOT 15)* (2015).

[19] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, *Iot security: ongoing challenges and research opportunities,* in *IEEE 7th international conference on service-oriented computing and applications* (2014).

[20] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, *Iotpot: A novel honeypot for revealing current iot threats,* Journal of Information Processing **24**, 522 (2016).

[21] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim, and J. N. Kim, *An in-depth analysis of the mirai botnet,* in *International Conference on Software Security and Assurance (ICSSA)* (2017).

[22] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, *Ddos in the iot: Mirai and other botnets,* Computer **50**, 80 (2017).

[23] G. Kambourakis, C. Kolias, and A. Stavrou, *The mirai botnet and the iot zombie armies,* in *IEEE Military Communications Conference (MILCOM)* (2017).

[24] O. Çetin, C. Ganán, L. Altena, T. Kasama, D. Inoue, K. Tamiya, Y. Tie, K. Yoshioka, and M. van Eeten, *Cleaning up the internet of evil things: Real-world evidence on isp and consumer efforts to remove mirai.* in *NDSS* (2019).

[25] G. Bottazzi and G. Me, *The botnet revenue model,* in *Proceedings of the 7th International Conference on Security of Information and Networks* (2014).

[26] H. L. Bijmans, T. M. Booij, and C. Doerr, *Just the tip of the iceberg: Internet-scale exploitation of routers for cryptojacking,* in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019) pp. 449–464.

[27] H. Heo and S. Shin, *Who is knocking on the telnet port: A large-scale empirical study of network scanning,* in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (2018) pp. 625–636.

[28] X. Zhang, O. Upton, N. L. Beebe, and K.-K. R. Choo, *Iot botnet forensics: A comprehensive digital forensic case study on mirai botnet servers,* .

[29] A. Costin and J. Zaddach, *Iot malware: Comprehensive survey, analysis framework and case studies,* BlackHat USA (2018).

[30] V. Ghiette, H. Griffioen, and C. Doerr, *Fingerprinting tooling used for ssh compromisation attempts,* in *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)* (2019).

[31] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, *Measurement and analysis of hajime, a peer-to-peer iot botnet,* in *Network and Distributed Systems Security (NDSS) Symposium* (2019).

[32] S. Soltan, P. Mittal, and H. V. Poor, *Blackiot: Iot botnet of high wattage devices can disrupt the power grid,* in *27th USENIX Security Symposium* (2018) pp. 15–32.

[33] T. Micro, *Persirai: New internet of things (iot) botnet targets ip cameras,* URL: https://blog. trendmicro. com/trendlabs-security-intelligence/persirai-new-internetthings-iot-botnet-targets-ip-cameras (2017).

[34] D. Goodin, *Brickerbot, the permanent denial-of-service botnet, is back with a vengeance,* Ars Technica (2017).

[35] T. Seals, *Bricker bot follows mirai tactics to permanently dos iot devices, apr. 7 2017,* .

[36] C. Reasearch, *Iotroop botnet: The full investigation, oct. 29 2017,* .

[37] N. Blenn, V. Ghiëtte, and C. Doerr, *Quantifying the spectrum of denial-of-service attacks through internet backscatter,* in *Proceedings of the 12th International Conference on Availability, Reliability and Security* (2017).

[38] J. Xu, J. Fan, M. Ammar, and S. B. Moon, *On the design and performance of prefix-preserving ip traffic trace anonymization,* in *ACM SIGCOMM Workshop on Internet Measurement*, IMW '01 (2001).

[39] R. G. Brown, D. Eddelbuettel, and D. Bauer, *Dieharder: A random number test suite,* Open Source software library, under development, URL http://www. phy. duke. edu/˜ rgb/General/dieharder. php (2013).

[40] *Tracking mirai: An in-depth analysis of an iot botnet,* Master thesis, Pennsylvania State University, 2017.

[41] W. O. Kermack and A. G. McKendrick, *A contribution to the mathematical theory of epidemics,* Proceedings of the Royal Society A **115** (1927).

[42] Z. Durumeric, M. Bailey, and J. A. Halderman, *An internet-wide view of internet-wide scanning,* in *23rd USENIX Security Symposium* (2014) pp. 65–78.

[43] H. Griffioen and C. Doerr, *Quantifying autonomous system ip churn using attack traffic of botnets,* in *Proceedings of the 15th International Conference on Availability, Reliability and Security* (2020) pp. 1–10.

[44] *Hajime botnet analysis,* https://www.infopoint-security.de/media/Botnet_Hajime_Radware_Analyse.pdf.

[45] *Shared network ranges to blacklist,* https://sciencesmaths.skyrock.com/323627548-very-famous-ip-s-try-2-hack-them-&-they-will-f-u.html.

7

Taking down malicious activity

Botnets often spread through massive Internet-wide scanning, identifying and infecting vulnerable Internet-facing devices to grow their network. Taking down these networks is often hard for law enforcement, and some people have proposed tarpits as a defensive method because it does not require seizing infrastructure or rely on device owners to make sure their devices are well-configured and protected. These tarpits are network services that aim to keep a malware-infected device busy and slow down or eradicate the malicious behavior.

This paper identifies a network-based tarpit vulnerability in stateless-scanning malware and develops a tarpitting exploit. We apply this technique against malware based on the Mirai scanning routine to identify whether tarpitting at scale is effective against containing the spread of self-propagating malware. We demonstrate that we can effectively trap thousands of devices even in a single tarpit and that this significantly slows down botnet spreading across the Internet and provide a simulation framework to simulate malware spreading under various network conditions to apriori evaluate the effect of tarpits on a particular malware. We show that the self-propagating malware could be contained with the help of a few thousand tarpits without any measurable adverse impact on the compromise routers or Internet Service Providers, and we release our tarpitting solution as an open platform to the community to realize this.

Parts of this chapter will be published as *"Could you clean up the Internet with a Pit of Tar? Investigating tarpit feasibility on the Mirai malware"* by **Griffioen, H.**, Doerr, C. in the IEEE Symposium on Security and Privacy (2023).

7.1. INTRODUCTION

When connecting a host to the open Internet, it does not take long before the first connection requests arrive. Probing for services such as SSH, telnet, DNS, or SIP, these are part of continuous scanning activity from various actors, testing which IPs are active on the Internet and which hosts may be compromised, used as amplifiers for attacks or stolen from. If a computer responds to these requests, the machine may be inspected further, either by the scanner itself or a second-stage device trying to establish a foothold on the machine.

The bulk of the scanning and initial compromise activity comes from automated tooling and scripts [1] that either rely on common passwords [2], attack services based on pre-defined signatures, or blindly launch exploits directed at common and frequently unpatched vulnerabilities. Given the plethora of Internet devices and generally low-security awareness, it is only a matter of time before one of these attackers “gets lucky”. As Internet-connected devices are often set up using weak, common, and insecure passwords [3] or are not updated to address vulnerabilities [4], there are enough victims to make automated scanning and exploitation a profitable activity. This explains figures such as [5] which report that 63% of Internet scans target port 23 and [6] that the portion of malicious Internet traffic is heavily increasing.

As many insecure devices expose themselves on the Internet and automatic scanning is both low effort and lucrative, neither problem will go away by itself on its own. Today, botnets compromising of hundreds of thousands of low-powered devices [7] are together capable of launching major distributed denial-of-service attacks [8], originate SPAM [9], or participate in cryptocurrency fraud [10]. This poses the question of whether it is possible to prevent large-scale exploitations with all their severe outcomes by addressing the link between perpetrator and victim: can we somehow influence that these scripts do not efficiently find victim devices to exploit? As we cannot and do not want to interfere with the network itself, we advertise decoy devices so that adversaries concentrate the bulk of their malicious activities on them and have, in turn, fewer resources to go after real victims.

One of the key engineering principles to make high-speed scanning and exploitation possible is that devices do not keep track of past and ongoing connection requests they have sent [11]. As the book-keeping from making contact to an IP address and retrying on a non-response would be way too slow to cover significant parts of the Internet, modern scanning tools blast out connection requests via raw sockets without actually going through the operating system to establish a connection. If an answer comes back, the tool checks based on header field values whether this response could be related to the previous scan and hones in on those IP addresses that have responded.

In this paper, we exploit this design, and trick scanners to believe they have received an answer from a device they have not even contacted. We utilize this flaw to collect connection attempts from infected devices and actively respond to all these requests from a decoy victim using these connection details. The decoy then traps the device at the transport and application layer, and due to the single-threaded nature of these malwares, brute-forcing does not continue elsewhere on the Internet. The malware will thus be hindered in propagating further and ultimately decline. As we show in section 7.5, the trapping of the malware does not impair the devices or the Internet connection of the in-

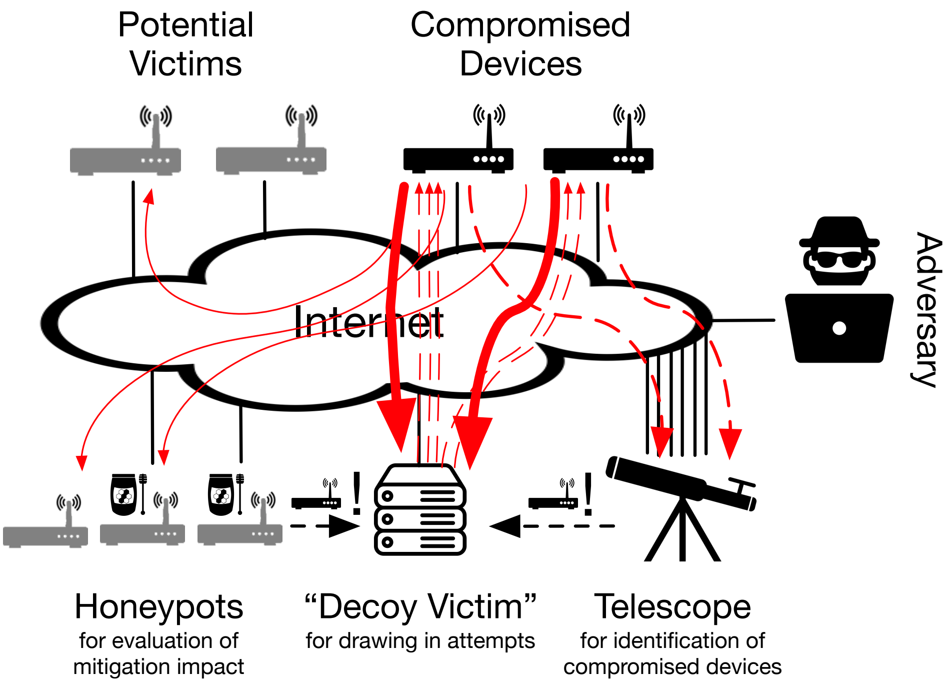


Figure 7.1: A network telescope detects infected devices and informs a tarpit to offer itself as a decoy victim. Honeypots measure the effect from reduced compromization activity.

fected users, and only adds insignificant additional traffic to the Internet. Through our work, we make the following contributions:

- We are the first to evaluate the merits of tarpitting at scale against a real-world botnet and introduce a method to "trick" a stateless scanner into making connections to a tarpit and evaluate how to traverse a NAT.
- We set up an operation to tarpit botnet connections that scan the Internet using stateless-scanning techniques. We show that using a *single* decoy server, we can tie up 48% of all Mirai-infected devices from further spreading the malware at the cost of 35 Euro/month. With 45 Mbps sent from a single decoy, we cut brute-forcing attempts on the Internet in half.
- We demonstrate that full containment of self-propagating malware is feasible in practice based on experimental measurements and agent-based simulations and open-source a multi-protocol tarpit allowing volunteers to join the containment of botnets.

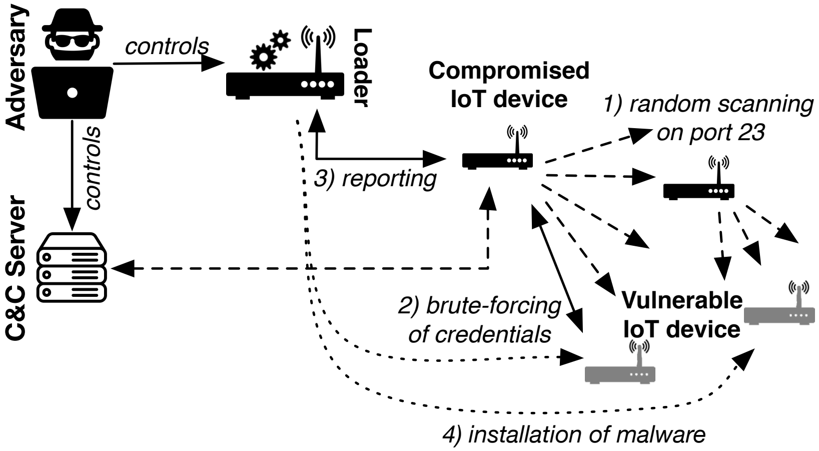


Figure 7.2: Devices infected with Mirai scan the Internet for vulnerable hosts, and report their findings to a loader.

7.2. BACKGROUND

The work put forward in this paper is applicable to the containment of any self-propagating malware. In the following, we focus on the Mirai IoT malware as one of the most significant Internet threats [12]. To understand why trapping would be effective in today's high-performance scanning routines, we first need to introduce some background about the Mirai ecosystem and the way it selects its targets.

The Mirai Ecosystem - The Mirai IoT botnet is the result of three infrastructure components: compromised IoT devices, the malware loader and a command & control server. As shown in Figure 7.2, the core of the botnet consists of vulnerable IoT devices, which are recruited to launch attacks on victims and play a key role in spreading the infection further: Compromised hosts independently scan the Internet on TCP ports 23 and 2323 (1), and once they discover a host with an open telnet port, they start brute-forcing it based on a fixed list of credentials (2). If some username/password combination has provided access, it is reported to a loader server (3), which then installs the Mirai malware on the vulnerable device (4). Mirai is a volatile malware; in other words, the infection only lasts until the device is rebooted or crashes. The device is then able to be infected again. Infected IoT devices call back to a central command & control (C&C) server for commands.

Scanning and Brute-Forcing Processes - Although the malware loader technically makes the infection, Mirai behaves like a worm as the compromised devices are responsible for discovering and breaking into additional victims. In order to spread effectively, Mirai uses some architectural designs commonly found in high-performance port scanning software such as ZMap or Masscan. As opening TCP sockets in the operating system comes with significant overhead and the OS limits open connections, consecutively making connections would be too inefficient. Instead, Mirai performs scanning based

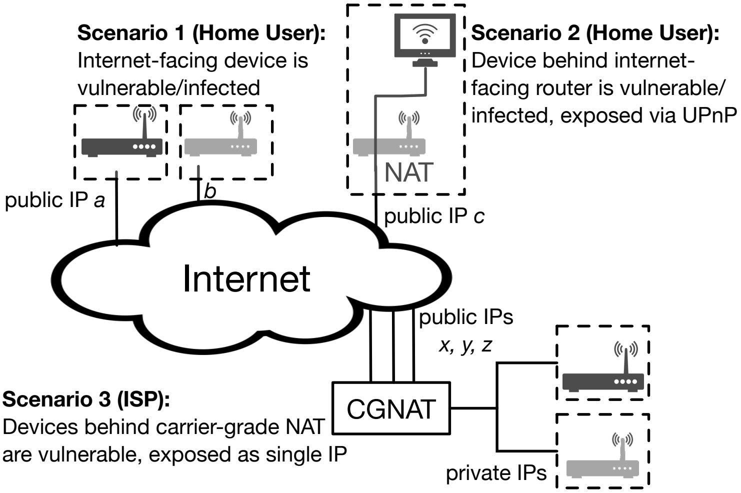

Scenario 1 (Home User):
Internet-facing device is vulnerable/infected
Scenario 2 (Home User):
Device behind internet-facing router is vulnerable/infected, exposed via UPnP
NAT
public IP a
b
public IP c
Internet
public IPs
x, y, z
Scenario 3 (ISP):
Devices behind carrier-grade NAT are vulnerable, exposed as single IP
CGNAT
private IPs

Figure 7.3: Vulnerable devices could be exposed directly to the Internet or via UPnP at public IPs, or hidden behind NATs.

on RAW packet injection: the malware crafts TCP SYN packets to randomly generated destination IPs and directly injects them into the network. If the destination IP is a device listening on TCP port 23/2323, it will answer with a TCP SYN+ACK. Once a SYN+ACK is received, the malware opens a "real" network socket to the destination and begins brute-forcing. By avoiding opening connections for scanning, Mirai does not have to save any internal state and therefore progresses fast; in our experiments, we saw IoT devices routinely scanning at speeds exceeding 8500 packets/second.

Internally, this design is realized by two separate processes, where the first one is injecting SYNs and the second one is initiating brute-forcing upon receipt of SYN+ACKs. The malware opens a maximum of 128 brute-forcing connections which are fed by a queue with observed SYN+ACKs. SYN+ACKs that arrive when this queue is full get dropped, and we use this fact in our work to eliminate the spreading of the worm: by tying up as many of these 128 connections as possible, the bot has no resources to brute-force newly discovered devices which means the bot is therefore not able to find the correct credentials and inform the loader about a victim. The botnet propagation thus slows down, potentially leading to the containment of the malware.

NATs and IP Churn - Mirai infections are not limited to end users' Internet routers, any device opening a telnet port with weak login credentials is in principle susceptible. This means that for dealing with Mirai infections we have to consider three scenarios depicted in figure 7.3:

Scenario 1: Infected Internet-facing device. In the simplest case, the device is exposed via a public IP address to the Internet and has opened port 23. The situation is typically the case for home users where the router or Internet modem is vulnerable. However, it may also occur in smart devices such as TVs, printers, sensors, or enterprise environments assigned to publicly routable IP addresses. Scans and brute-forcing attempts will thus originate from the public IP address *a* and *b*. Counting the number of scanning/brute-forcing public IPs will reveal the number of infected devices on the Internet.

Scenario 2: Infected device behind a router. In some cases, the infection is not borne by the device connecting to the Internet, but by a device behind the router. In these situations, a device has requested the router to open the telnet port and forward incoming packets internally (e.g., via UPnP) and got compromised as a result. As all telnet connections terminate at the same device, any compromisation attempt from the outside will end up at the same device. Thus, a quantification based on public IPs will tally the number of infections.

Scenario 3: Carrier-grade NAT. As publicly routable IP addresses are in short supply, many ISPs do not hand out individual public IPs to each customer. Instead, the routers of home users are given a local IP address only and connect to the Internet via a carrier-grade NAT (CGNAT). Thus, an arbitrarily large number of home installations would be exposed via the same IP address to the Internet. The CGNAT tracks the state and matches incoming responses from the Internet to an earlier request to deliver it to the correct local client. For tracking Mirai infections, this creates the complication that more than one device might be infected behind a public IP. Occasionally, CGNATs are assigned a pool of public IPs (x, y, and z in figure 7.3), and may re-assign a particular home user to a different public IP in regular time intervals, creating so-called IP churn. This creates another complication for tracking, as the same infection would be recorded multiple times over its lifetime if only counting scanning IP addresses. IP churn is known as a major factor in over- and under-estimating infections [13].

The particular setup of Mirai's port scanning routines conveniently addresses both issues. As upon startup, the malware picks a random source port and window size, this 32-bit value is fixed during the entire infection period. Suppose scanning packets originate from the public IP address x with two different sourcePort/windowSize combinations. In that case, we are confident that two devices behind that IP address have to be infected, as Mirai binds to port 23 and thus prevents two concurrent infections from being active on port 23 [14]. Similarly, if scanning traffic stops on IP x and shortly after commences on IP y with the same sourcePort/windowSize combination, there is only a one in four billion chance that this is not the same infection. We can thus also reliably track and count infections behind carrier-grade NATs.

7.3. RELATED WORK

The default action against botnets in the recent past has not been based on tarpits but rather through *sinkholing*, where the Command and Control (C&C) infrastructure is taken down by researchers or law enforcement [15, 16]. The coordinated takedown of a C&C is very labor-intensive and often escapes long-term effects as malware authors will simply restart their activities [17]. Major botnets such as ZeuS or CodeRed could be reactivated in a matter of days after such takedown [17, 18]. Furthermore, over the past decade, malware has evolved to use peer-to-peer-based overlays [13], fast fluxing [19], proxy layers [20], or bitcoin-based signaling [21], making it increasingly hard to find and take down a C&C server.

Some studies have shown that so-called *tarpits* can be used to reduce impact of DDoS attacks [22], SPAM [23] and Web Crawlers [24] by occupying infected devices.

While researchers have focused on creating and evaluating tarpits, no work has evaluated the effect of a tarpit on large malware installations. We are, however, interested in how this approach would work in practice with a sufficiently large monitoring infrastructure, and whether a large tarpit infrastructure would be capable of slowing down the spread of a botnet.

Tarpits aiming to trap malicious devices have been around for a long time. The first tarpit was called “LaBrea” and was introduced by Tom Liston [25] in response to the CodeRed worm [26] that exploited a vulnerability in Microsoft IIS web servers to spread through the Internet. LaBrea monitors TCP traffic towards a host and replies with a SYN+ACK that completes the TCP handshake and makes the malicious program wait for new replies that are never sent. Since its inception, other research has developed tools to reduce network attacks based on tarpits. Borders et al. [27] created OpenFire, which fills unused address space with honeypots and makes the entire port space appear to be open to an attacker. Modern scanning software and malware have both implemented multi-threaded infrastructures and network timeouts, limiting the impact that these tarpits will make nowadays.

When a worm connects to a tarpit that attempts to keep the malware occupied on the network layer, the worm does not receive the expected application layer data, and the malware author can use a timeout to detect the trap. Walla and Rossow [28] propose an automated method to identify implementation mistakes in the malware that can be used to halt the program execution remotely. Other researchers have devised methods to scale tarpits to IPv6 [29], but only evaluated the method on a real-world /24 network as the IPv6 network did not receive packets. Furthermore, the authors only report on general trapping durations and not the effects on a worm.

Given the increased difficulty and diminishing returns of takedowns, tarpits have been recently rediscovered and suggested as an alternative to contain malware [28], but no works have identified whether this would work at-scale. As replacing a single C&C is easy, taking away the foundation of a botnet is more challenging to replace and might offer longer-lasting results. From epidemiological evaluations of the Mirai malware, this might very well work, as researchers show that the Mirai-malware, which is a large botnet, has a low reproduction number $R_0 = 1.0033$ and is barely self-sustaining [1]. If it would be possible to slow down the infection process with a tarpit significantly, it might lead to a collapse of the botnet. Tarpits exist for many different botnets and the epedimiological effects of the tarpit on the malware spread would be similar between these. For this study, we will evaluate the effect on a botnet in-the-wild, for which the epedimiological results will be transferable to other botnets as well. We evaluate whether we can contain botnets based on Mirai's scanning routines [2] and significantly disrupt the spread and impact of these botnets through a tarpit.

7.4. METHODOLOGY

We deploy a two-stage methodology to trap hosts of botnets that implement a form of stateless-scanning. First, we identify infected hosts by monitoring scanning traffic. Then, we forge response packets from a centralized host and use these packets to “trick” an infected device into connecting to our tarpit.

Table 7.1: Datasets used during this study. Data generated by the tarpit and the cloud honeypots will be shared with the community.

Dataset	**Date (2021)**	**Description**
Network telescope	Feb 19 - Apr 16	Set of 65.000 unused IP addresses used to identify infections and required packet header fields
Cloud honeypots	Feb 19 - Apr 16	100 honeypots used for: (1) collecting application-level traces on port 23, 1023 and 2323. (2) Identifying new infections and required packet header fields
Main tarpit	Mar 3 - Apr 8	Machine used to trap infections identified by honeypots and telescope
Extra tarpit	Apr 2 - Apr 8	Machine used to trap infections identified by honeypots and telescope
Tarpit code	-	Source code of a tarpit responding to scanning probes
Tarpitting simulator	-	Source code of agent-based simulator simulating malware infection

7.4.1. DATA COLLECTION

To identify devices infected by a variant of Mirai, we use two complementary datasets that are also shown in Figure 7.1:

1. Network telescope - We use a large network telescope of approximately 65,000 IP addresses in an enterprise environment to record scanning attempts. As these IP addresses never respond, they will only receive Internet scans and backscatter. Scattered throughout actively used enterprise IP address space, we find them not to be on the blocklists of Mirai malware [30].

2. Cloud-based honeypots - While the telescope provides an overview of the infections on the Internet, it does not actively respond and can therefore not tell which devices would scan and which would try to brute-force credentials. To differentiate between these two, we deploy 100 honeypots with a major cloud provider to monitor traffic on all ports and respond to all telnet traffic on ports 23, 1023, and 2323. These instances complement the data collection in the network telescope and report infected IP addresses to the tarpit.

In addition to the sightings at telescope and honeypots, the tarpit collects a log of all IP addresses using a scanning routine based on the Mirai source code. Furthermore, it records connection attempts and raw application-level traces of tarpitted devices. As stated in section 7.2, Mirai scans devices using TCP SYNs from a session-static source port and recognizes the return packets by embedding the destination IP address as the TCP initial sequence number. Distinguishing Mirai-based scanners from other scanning traffic can thus be easily achieved. In total, we collected 103 billion connections from 423,810 IP addresses over 35 days in the tarpit and almost two months in the monitors, providing a total of 28 TB of traffic across all measurement points. To identify whether a single tarpit is enough to handle this many concurrent connections, a second tarpit was briefly activated and collected 4 billion connection traces. Most data collected during this study will be made available to the community. Table 7.1 lists the datasets used for this study, dataset sizes, and in what form it will be released after publication of this paper.

7.4.2. SYSTEM ARCHITECTURE

The setup of the entire system is shown in figure 7.1. Our network telescope and honeypots forward infected IP addresses and the corresponding source and destination ports towards our tarpit, which in turn starts to forge artificial reply packets and maintains connections to compromised devices. The tarpit is designed to send out 100 artificial replies (6.4 Kb) per second to infected devices (the motivation for this will be shown in section 7.6). It will timeout after the device has not responded for one minute so that the system will send at most 384 Kb towards devices cleaned up after a reboot.

As the tarpit will send 100 packets per second (pps) to infected hosts and respond to all incoming connections from infected devices, the system needs to send large amounts of packets if we aim to address Mirai infections worldwide from a single device. Raw Ethernet sockets would therefore be too expensive, as it uses the `sendto` system call, which performs a context switch and transfers packets through kernel space. Similar to [31] we use the PF_RING ZC interface, which works as a kernel bypass and allows user-space code to access the NIC directly, eliminating the need for context switches [32]. To support this, we implemented a simplified TCP stack in Golang to construct valid responses. The machine running the tarpit was comparatively moderate, with a quad-core *Intel Xeon E3-1225* from 2011 at a clock speed of *3.20 GHz*, a *1 Gbps* uplink and *32 Gb* of memory.

7.4.3. TARPITTING STRATEGIES

After an infected device opens a connection to the tarpit due to the crafted SYN+ACK, the goal of the tarpit is to prolong the connection. Tarpits can work on different layers in the network stack, for example, layer 4 (Transport) and layer 7 (Application). While layer 4 tarpits keep a network socket open, applications will not receive any responses and might therefore timeout. Layer 7 tarpits could trap a connection on the application layer but require more book-keeping to save the state of connections. To figure out what type of tarpit is effective in practice, we cover the possible tarpit strategies, we deploy five response methods that try to trap a device in different ways:

1. Only complete the handshake. Evaluating every incoming request in the tarpit takes up processing power. A socket will be tied up until a timeout occurs by only completing the handshake, limiting the strain on the tarpit.

2. Send TCP keepalives. The tarpit prevents the OS to timeout the connection with TCP keepalive packets. This means that a timeout occurs only at the malware itself.

3. Reply with random data. This method slowly sends an endless stream of characters over the connection avoiding special characters such as *newlines*, *null bytes* and *EOF*. An application reading until such a special character will be continuously stuck if no timeout is specified in the application.

4. (Telnet only) Emulate a telnet connection. As a baseline, we measure how long connections last on a telnet connection. The first reply in the connection will ask for a username: *Sun OS 5.8 Login:* . After receiving a reply, the tarpit will send a password prompt and accept every password to access a fake shell. The connection remains open as long as the device desires.

5. (Telnet only) Emulate a telnet connection and reply with delays. The tarpit again provides an emulated prompt but delays all responses, including the shell, by 5 seconds.

7.4.4. Limitations

In normal TCP operation, SYN+ACK packets are only used in the handshake to respond to a SYN packet. The unsolicited scan replies sent by the tarpit are therefore not part of the TCP specification and might be dropped by some firewalls. Additionally, infected devices might be located behind a NAT that is dropping the unsolicited SYN+ACK packets. In these cases, the tarpit will be unable to reach infected devices. We will revisit these operational issues in sections 7.7 and 7.8.

7.5. Verification and Ethics

After the introduction of the concept and implementation strategy, we must however first validate and verify before proceeding that (1) the proposed tarpitting approach is in practice actually limiting the propagation activity of infected hosts, and (2) that our methodology does not negatively impair the proper functioning of infected devices or Internet Service Providers in general, in other words that we are not launching a denial-of-service on the involuntary perpetrators. These two aspects are the focus of this section.

7.5.1. Verification of the trapping of infected hosts

The telescope and honeypots provide a real-time view of infections, but they only serve as monitors. To trap the malware, we (ab)use the scanning protocol of Mirai and craft answer packets such that (1) the expected *acknowledgement number* in a reply packet is the *Destination IP address* + *1* and (2) the packet returns the correct, current session port. Because the malware does not keep a list of scanned IPs, any SYN+ACK satisfying these two requirements will convince the infected device that it has found a new potential victim.

The scanning packets captured in the telescope and honeypots allow us to find the IP addresses of infected devices and their randomized source ports on which they are listening. We can then send a SYN+ACK from any IP towards the infected device, leading it to believe a new victim is located at this IP address and open a regular socket connecting to it. The original Mirai malware (and all variants we observed) allows for a maximum of 128 concurrent sockets, meaning that every device can simultaneously brute-force or exploit 128 devices. A regular tarpit would only block one of these sockets at a time, as it traps a single socket of the infected device when found in a scan. By proactively sending out many SYN+ACKs, we can tarpit all sockets of an infected device using a single host.

To verify this concept, we infected an IoT device in a controlled environment with the original Mirai malware [30], and measured its scanning and brute-forcing activity before and after we initiate our trapping routine. Figure 7.4 shows the number of brute-forcing connections open towards a vulnerable device located in a network of 20 devices. This means the malware has a 5% chance of discovering the vulnerable device every time it sends out a scanning probe, which is magnitudes larger than the probability of finding a vulnerable device during an Internet-wide scan. The orange line shows the amount of connections that are active towards a vulnerable device that is identified by the malware. The blue line shows the amount of connections towards the tarpit, that is not identified by scanning but only advertises itself by forging 100 SYN+ACKs per second.

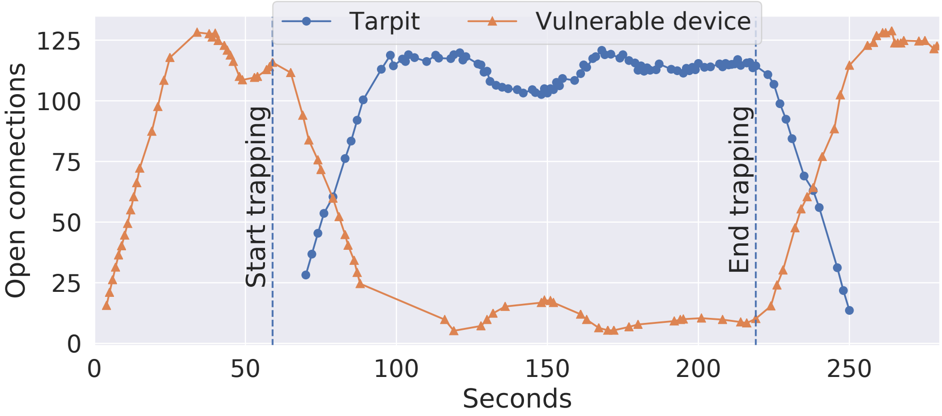


Figure 7.4: Connection trapping against a base version of Mirai.

In this extreme example where the malware has a 5% probability of finding a vulnerable device with each probe and without any connection slow-down, the trapping can reduce the amount of brute-forcing towards the vulnerable device by a factor of 20.

7.5.2. Ethics: Is our tarpitting impairing users or the network?

While the trapping works as intended, it is essential to verify that this procedure does not impair the stability or performance of the device to the end user, and instead only prevents the spread of the botnet. To test this, we created the verification setup as shown in figure 7.5, in which we benchmark the behavior of four Internet routers, ranging from a $20 budget device to mid-range home routers.

In our environment, these routers are setup to connect an end-user to the Internet either via a 1 Gbps Ethernet cable or 100 Mbps WiFi connection and operate the router at its maximum transmission rate. We connect a high-end router on the Internet uplink of the infected IoT device that drops the outgoing scan probes to prevent pollution and links the setup to our tarpit. We then measure with iperf the performance of a bandwidth test between the end user machine and the router's uplink in terms of packet less, latency and throughput, and at the same time monitor the "health" of the router by monitoring its CPU load. This allows us to quantify whether either the end user experience is impaired or the device itself has to process an abnormal workload.

Each router is tested for 300 seconds in the following scenario. We first establish the iperf connection measuring the performance of the end user's connection. We infect the router, which triggers the device to begin scanning the Internet for other devices. We record this baseline activity for 100 seconds. We then activate our tarpit to trap the device at the intensity of 100 SYN+ACKs per second, identical to the speed we will use throughout our experiments. This trapping continues for 100 seconds. At second 200 into the experiment, the tarpit shuts down so that we can monitor for the remaining 100 seconds the rebound behavior of the router. We perform this experiment for a variety of routers, ranging from a $20 MikroTik HAP Lite budget router with only 32 Megabyte

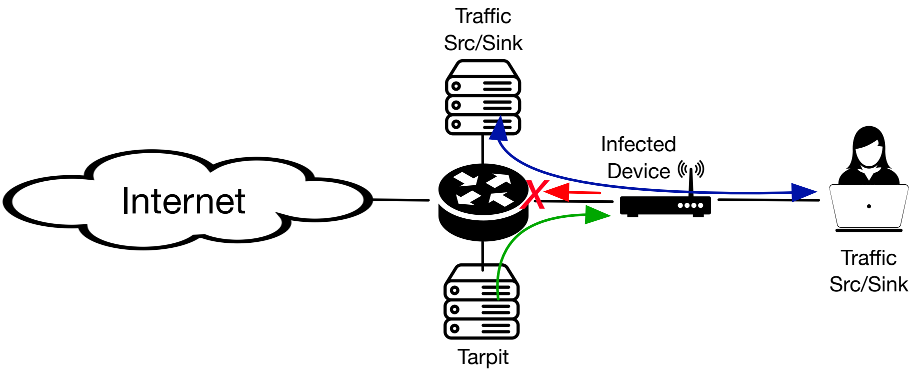


Figure 7.5: Experimental setup to test for potential impairment of the router in forwarding capacity, delay and CPU utilization.

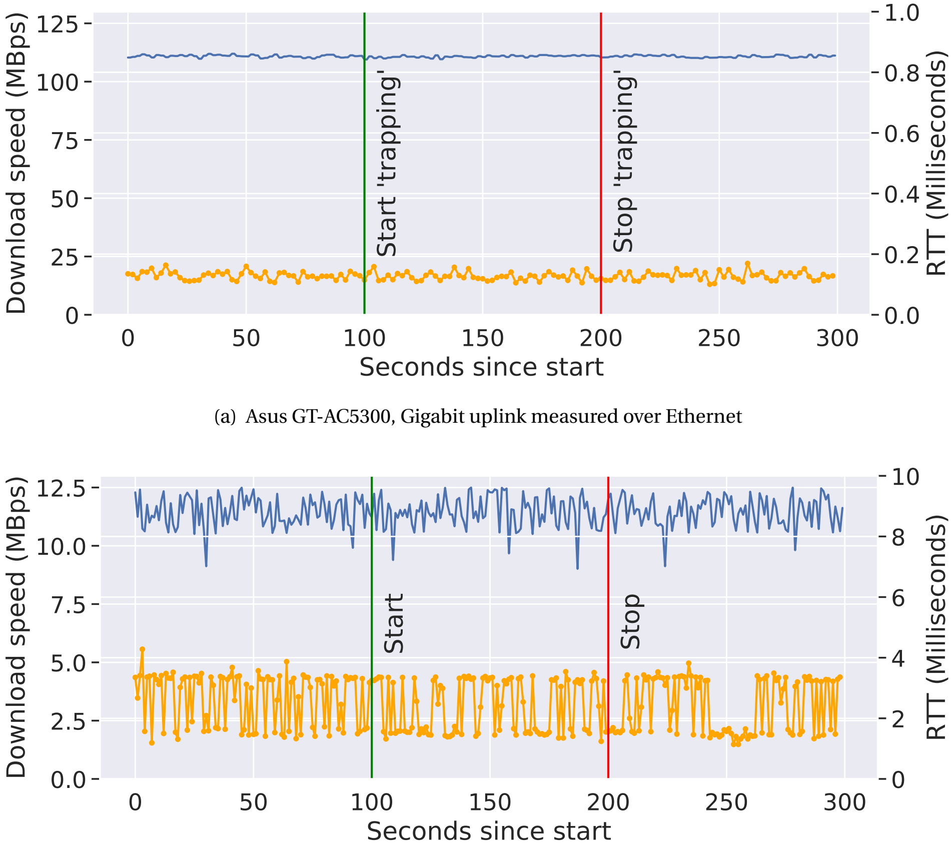
RAM, two mid-range routers Zyxel VMG8825-T50, a TP-Link AD7200 to a high-end Asus GT-AC5300. The device only has 100 Mbps Ethernet ports, the all others feature a Gigabit Ethernet interface.

Figure 7.6 shows exemplarily the download speed and the round-trip time (RTT) latency of the Asus GT-AC5300 and the Zyxel VMG8825-T50 during the course of the experiments. The graphs are depicted separately, as the Asus can forward at Gigabit, while the Zyxel has only 100 Mbps interfaces. As we can already see visually, the tarpitting at 100 SYN+ACKs per second has no impact on the forwarding performance of the router, also the CPU load does not measurably increase after the tarpit starts and during the time it is running. We performed a Kolmogorov-Smirnov test on the individual measurement samples, and the test confirms that the router behavior is not impaired by our trap. While only two examples are shown, all routers behave identically.

Finally, we have to ascertain a potential impact on the network in which the devices are located, as they have to forward our tarpit responses to the infected devices. To estimate the impact, we take a worse-case scenario approach to obtain an upper bound on the amount of traffic that is forwarded through a network. Let us consider the most infected AS in the world, AS4134, in which we have seen *a total* of 125,000 IPs being infected historically. If we suppose that we would trap all 125,000 devices ever seen concurrently at the tarpitting speed we utilize in this work, we would send a total of 12 Mbps towards this AS. This is approximately half of the bandwidth requirement that would be generated by a single customer watching a 4K video from a streaming platform. We hence can consider the additional traffic on ISPs and the Internet in general to be negligible, especially as by trapping the infected devices we would also prevent the response traffic from brute-forcing to be ever generated.

7.6. TRAPPING MIRAI IOT INFECTIONS

During the 35 days in which we operated our tarpit, we observed a total of 423,810 distinct IP addresses scanning and brute-forcing the Internet using the Mirai fingerprint.



(a) Asus GT-AC5300, Gigabit uplink measured over Ethernet

(b) Zyxel VMG8825-T50, 100 Mbps uplink measured over WiFi

Figure 7.6: Impact of our tarpit on forwarding speed and latency. The blue line at the top of the graphs shows the download speed per second and the orange line at the bottom of the graphs shows the RTT of a packet.

These devices originated from 178 countries and 3,150 Autonomous Systems. Since its inception, Mirai's principles have been adopted by other malware, which also diversified in terms of targets: we found that in addition to telnet, the diversified Mirai ecosystem now also targeted 17 other ports. As all of these malwares rely on Mirai's scanning routines, our tarpitting is effective against all of them. During this time, we successfully trapped 202,003 (48%) of these IP addresses in our tarpit.

Although we observed hundreds of thousands of infected IP addresses, the *momentary* amount of infections is significantly lower: Mirai-infected devices frequently crash and become available for reinfection [1], furthermore Internet Service Provider often dynamically reassign IP addresses regularly – so-called IP churn –, which means that a unique infection might reappear later from a different IP. We account for IP churn using the methodology in [14] and find on average 13,298 unique infections on a particular day. We designed and executed a series of experiments to trap devices and understand the dynamicity of the environment and the resulting requirements on a tarpit infrastructure. In this section, we report on the success of trapping these infections with respect to the following questions:

- Steady-state: When the tarpit reaches a steady state, what is the overall impact on the ecosystem?
- System startup: As devices are trapped when stumbling upon a tarpit, how long before effects are visible?
- System retention: As IoT devices are continuously introduced, restart or churn away, how much agility is needed in a trapping infrastructure?

7.6.1. The impact of a tarpit in steady state

During the entire experiment, we encountered Mirai infections at 423,810 IPs, of which we were able to trap a total of 202,003. As discussed above, the number of momentary infections is much lower, and of the 13,298 addresses infected on average per day, the hosts were able to trap on average 6,326 (47.5%). In this subsection, we report on the effectiveness of the infrastructure once the tarpit was fully active, namely when the telescope and honeypots (see figure 7.1) feed their Mirai sightings to the tarpit, enticing hacking attempts. In subsections 7.6.2 and 7.6.3, we report how long it took to get there and the performance degradation if we would not keep track of new infections.

Even a single tarpit can tie up the brute-forcing of thousands of devices. As discussed in section 7.2, Mirai uses separate routines for finding victims and exploiting them. By allegedly responding to its scanning probes, our theory was to tie up a device's brute-forcing resources so that other concurrently identified victims could not be compromised. In practice, this turned out to be remarkably effective: The original Mirai source code supports 128 concurrent brute-forcing sockets, none of the variants we observed have ever increased this limit. To measure the effectiveness of our infrastructure, we count the number of brute-forcing connections we can concurrently trap, figure 7.7 shows a CDF of the average new connections per minute across all IP addresses. On average, we see 356 new sockets per minute, 2.8 times the maximum number of connections Mirai can maintain at a single point in time. As the sockets timeout on either the OS or in the application itself, the malware immediately creates a new socket towards the tarpit. This means that when the duration-before-connections timeout is longer, the number of sockets created is lower. We find that the duration it takes for a socket to timeout is indeed inversely proportional to the number of new sockets created over time. When we combine our measurements, we see that we trap on average 126.2 threads per infected IP and thus absorb nearly the entire brute-forcing capability of Mirai devices through our tarpit.

All tarpitting strategies are equally effective. As described in Section 7.4, tarpitting could take place at layer 4 or layer 7 using different strategies. Our tarpit supports three application protocol-agnostic connection handling strategies and two additional ones geared towards telnet. We find that the duration for which a tarpit connection lasts differs greatly by protocol. For example, connections from malware targeting port 23 usually last longer than half a minute, while connections towards port 445 are closed within 10 seconds regardless of what type of reply is sent.

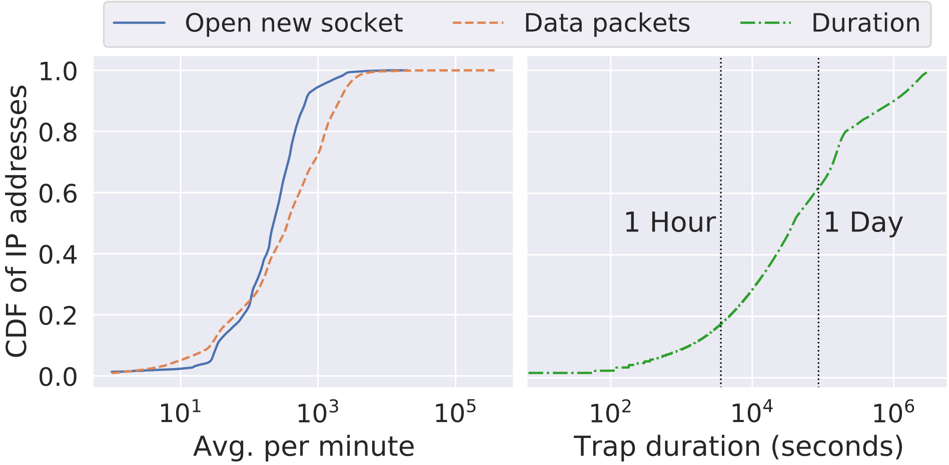


Figure 7.7: CDF of average number of connections per minute for trapped IPs and the total duration an IP address is trapped.

Different reply strategies generally do not increase the time a socket is kept open. This means that tarpits could be designed much simpler: after completing handshakes with a SYN+ACK, we can forget about the connection, limiting the strain on the tarpit itself. For port 80, only completing the handshake even traps the median connection longer than other strategies as the client waits for data and hangs up on unexpected data. A notable exception is port 5555 (Android Debug Bridge), where sending random data increases significantly the duration of connections to the tarpit.

Devices can be trapped for a long duration until the network interferes. The success of the tarpit largely depends on the duration for which it can trap devices, as during this time the device will connect to less vulnerable devices. Figure 7.7 shows a CDF of the *total duration* successfully trapped devices spend contacting the tarpit, where 60% of the IP addresses are only trapped for less than a day and 19% for even less than one hour. IP-based measurements are often biased due to ISP's IP churn [33], where networks forcibly disconnect customers' Internet connections and let them rejoin with a new IP address [34]. The infection would however persist and reappear at the new IP address. The authors in [14, 34] show that the bulk of ISPs reset in intervals between 4 and 24 hours, which would heavily skew the distribution as many churning IP addresses have a short lifetime and can therefore not be trapped for an extended period. Using the methodology of [14], we can link Mirai infections before and after a churn event together and thus estimate the total trapping duration, which allows us to estimate the effect of IP churn for non-Mirai infected devices as well. We find that infected devices are trapped for on average 6 days when accounting for IP churn. This 6-day timeframe is the average lifetime of Mirai infections we see active on a particular day. In other words, if an infected device contacts us and responds to the tarpitting event, we find that we can keep it trapped in 100% of the cases until the end of the infection. Once devices fall off our trap, none of the honeypots ever record any scanning traffic afterwards coming

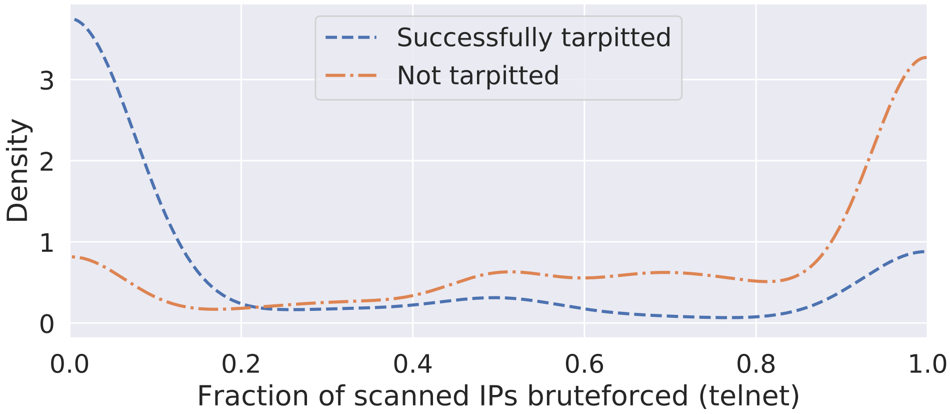


Figure 7.8: PDF of the brute-force attempts on port 23, 2323 and 1023 after a device is found after a scan. Devices that responded to tarpits show significantly reduced brute-forcing.

from such IP addresses, which means they can hold onto it entirely and thus neutralize the infection impact.

Clean-up success differs by country, due to ISP architecture. Internet Service Providers do not only periodically reassign IP addresses, but they also aggregate multiple customers behind a network address translation (NAT). NATs keep track of outgoing connections to match incoming responses and forward them to the correct user. As our tarpit claims to have been contacted by an infected IoT device, the NAT would have no record of an outgoing connection and could not forward our invitation to be brute-forced to the infected device. This means that a centralized tarpit could not trap NATted devices, coming sections revisit this issue.

Due to the differences in how NATs are used and configured in different countries and Autonomous Systems [33], the effect of our tarpit largely varies between geographical regions. For example, in the two countries with the largest IP space [35], the US and China, the share of trapped devices are respectively 30 and 95%. This is beneficial as Mirai infections are far more prevalent in China than in the US [1]. The reverse effect of network policies is also observed, as some networks have much higher churn rates than other networks. When a device is allocated a new IP address due to network churn, the tarpit must again learn of its existence. This means that in networks with higher churn, infected devices are temporarily disappearing from the tarpit and are thus able to perform their regular operation. In the US, we find that the successfully trapped devices are much less likely to churn than devices located in China. The tarpit traps the US-based devices for more extended periods without any downtimes when the device is unknown. The effect of this IP churn downtime is however partially mitigated because of the brute-forcing queue of Mirai, where it remembers the tarpit IP address as a vulnerable device even after the infected device is allocated another IP address. The infected device will then connect to the tarpit from the new IP address, revealing its presence.

Tarpitting drastically reduces Mirai's ability to exploit others. The main goal of the

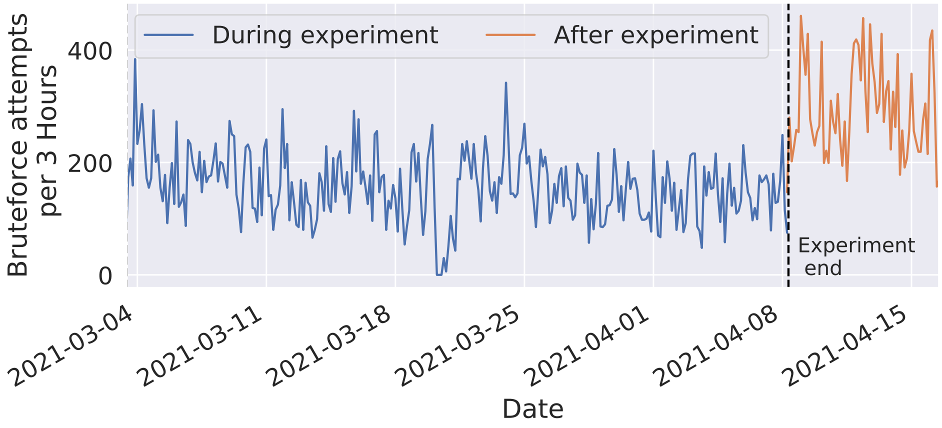


Figure 7.9: Bruteforce attempts on port 23, 2323 and 1023 by 2,150 long-term infections, during and after tarpitting.

tarpit is to exhaust the brute-forcing capabilities of infected devices. To understand whether the tarpit is successful, we can utilize the fact that we cannot impair the infected devices located behind a NAT and compare their activity to those we trap. For this comparison, we utilize the 100 cloud honeypots as a passive listening post to quantify the telnet brute-forcing attempts from infections.

When the first scan packet from an infected device arrives, and the tarpit entices it to brute-force, the infected IoT devices will keep scanning the Internet and eventually reach our cloud honeypots. We analyze what percentage of the infected devices attempt to brute-force the cloud honeypot after it was discovered. Figure 7.8 shows the Probability Density Function (PDF) of the brute-forcing attempts made after a scanning packet is observed in a honeypot, in blue those that responded to the tarpit, in orange those that ignored our tarpit requests. To account for the case that both tarpit and honeypot were concurrently discovered and the brute-forcing was already about to happen before the tarpit reached the IoT device, the first 60 sec at the on-set of the tarpit activity are excluded. We see that the devices trapped in the tarpit are much less likely to brute-force other devices on the Internet after they discover potential victims. While for non-tarpitted devices, on average, 84% of the discovered hosts are subsequently brute-forced, for those trapped by our tarpit, the average is only 17%.

To demonstrate the difference tarpitting makes, let us zoom into those devices continuously infected during the entire experiment, which means that we can exclude confounding factors such as IP churn. Figure 7.9 shows the number of recorded brute-forcing attempts in 3-hour intervals for these 2,150 long-running infections, both outside and inside of a NAT. In blue while we tarpitted the devices and in red after we released them and continued to monitor for another week. After the devices were released, the average number of connections to the cloud honeypots almost immediately doubles. With about half of the devices shielded behind a NAT and half affected by the tarpit, we see a total reduction by half in brute-forcing.

Table 7.2: Effect of the tarpitting experiment on scanning speed.

Port	**# IP**	**PPS (normal)**	**PPS (tarpit)**
Telnet: 23	150,661	8,498	7,617
RSFTP: 26	3,171	9,545	4,280
HTTP: 80	1,299	8,007	2,755
Telnet - alt: 1023	3,413	1,573	543
Telnet - alt: 2323	26,463	4,706	3,884
ADB Worm: 5555	10,347	2,872	1,516
Overall	174,167	7,822	7,105

While during the tarpitting, the brute-forcing capability of IoT devices significantly reduces, it does not entirely drop to zero. The reason for such non-zero activity is the concurrency between Mirai's scanning and brute-forcing threads: while we fill up the brute-forcing queue with our honeypot as decoy victims, we compete for slots in this queue with Mirai's scanning routine. Whenever the saturated list drops one entry, it is a matter of chance whether our tarpit's response or a newly discovered victim enters its place. When we refer back to figure 7.4, we already see this phenomenon in the controlled experiment of our validation. The solution to this problem is simple though: always have a packet waiting in line as the brute-forcing queue clears a spot, which is essentially a question of sending tarpit requests fast enough. By using the same setup as in figure 7.4, we find that with 10 times the intensity, we can reduce the number of requests that slip through by 92% to 127.8 out of 128. While we could thus eliminate the problem almost entirely, this creates a strain on the devices already victimized by the malware. In this work, we limit our tarpit requests to the mentioned 6.4 Kb / second, or 0.09% of the average worldwide Internet connection speed [36], which, as we saw above, already occupies 126.2 threads.

Tarpitting the brute-forcing threads also reduces scanning speed. The main goal of our tarpit is to prevent the infection of others by filling up the queues of the Mirai malware, with a volume we specifically designed to be negligible for an end user's Internet uplink. A bit unexpected, though, was that devices in practice also effectively scanned slower by tying up the brute-forcing functionality. This reduced scanning across the Internet also implies a reduced probability of identifying vulnerable devices, which further slowed the speed of infection. Figure 7.10 shows the average scanning speed per IP address in pps before, during, and after the experiment for port 23 and port 5555. For port 5555, the average scanning speed nearly halves from an average of 2,872 pps to 1,516 pps, for port 23 it declines by 10% from 8,498 to 7,617 pps. Table 7.2 lists the measured effects on the scanning speed across the ecosystem during the tarpitting experiment. This decrease in scanning speed results in vulnerable devices being less likely to be identified, slowing down malware spread.

One tarpit slows world-wide Mirai infections by 21%. As discussed above, the tarpit proved highly effective in trapping Mirai instances when they were accessible from the Internet. While effectivity could be further increased by sending more requests from the tarpit, we could significantly decrease both scanning and brute-forcing speeds of infected IoT devices, which means that it will take more time for the malware to find potential victims and infect them. Although only 48% of all infected devices responded

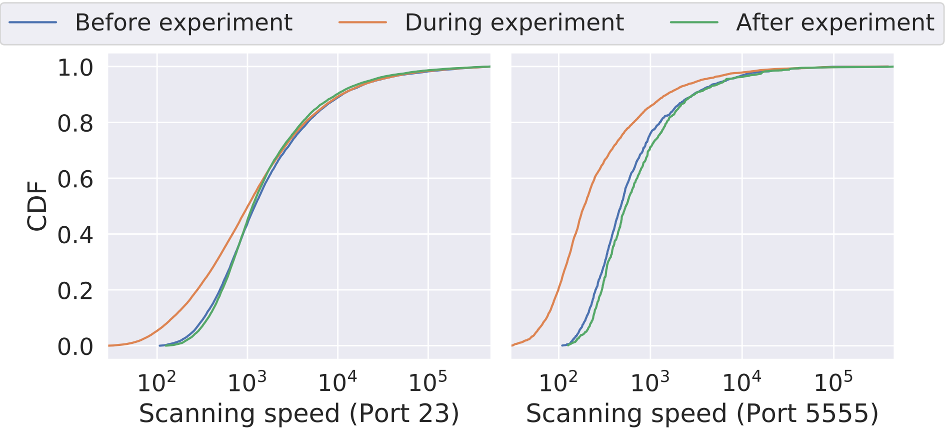


Figure 7.10: CDF of (estimated) scanning speeds in packets per second before, during and after our experiment for two ports.

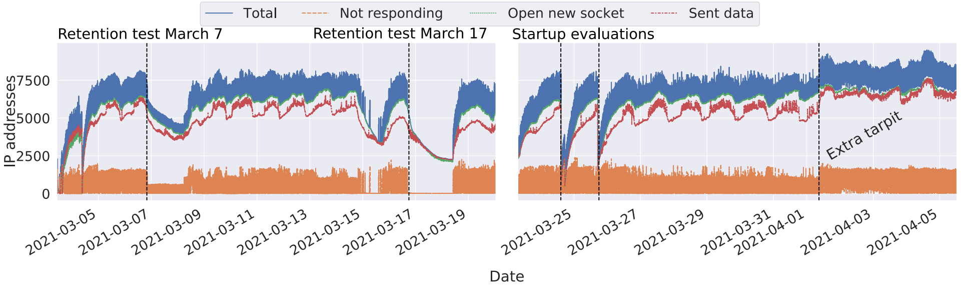


Figure 7.11: Timeline of the experiments running from March 3 - March 20 and March 23 - April 8 showing the trapped IP addresses per minute. A portion of the IPs that were sent SYN+ACK never respond. The vertical lines indicate when we tested the retention, startup time, and tested for network congestion through a second tarpit.

to our requests, the tarpitting still significantly affected the worldwide Mirai ecosystem. When tracking the number of infected devices and the average time between infections (as IoT devices fairly often crash and are typically reinfected only shortly later [1]), we see that during our tarpitting, the average re-infection interval increased from 12.1 hours to 14.6 hours. Even when reaching only half of all Mirai instances worldwide, a single tarpit would thus slow down the malware spread by 21% after locking up almost half of the brute-forcing capabilities in the ecosystem. The re-infection rate drops less than the brute-forcing capacity due to the saturation of the ecosystem where most vulnerable devices are already infected and the brute-forcing power of the network, thus being higher than the total power needed to infect every other device.

7.6.2. SYSTEM STARTUP: HOW LONG BEFORE WE ACHIEVE THESE EFFECTS?

To measure the time it takes the tarpit to reach a steady-state and the time it takes for the measurement infrastructure to capture devices vulnerable to connection trapping, we completely disable and start up the tarpit three times starting from March 23. On average it takes the tarpit about a day to start up and reach the steady-state, where we have seen all Mirai infections at least once. To identify the effect that the size of the measurement infrastructure has on the performance of the tarpit, we take random samples of our network telescope and average over 20 samples. We find that the decrease in devices trapped in the tarpit is not proportional to the size of the measurement infrastructure, as 2,500 honeypots identify almost 60% of the devices observed by 65,000 honeypots in the same time span. The difference in trapped devices also fades over time as a large network observes all infected devices, and smaller networks are still identifying new devices. While a large network would thus be faster in identifying the infections, a smaller network would still be suitable for identifying and trapping a botnet.

7.6.3. SYSTEM RETENTION AND SCALING CONSTRAINTS: HOW MUCH AGILITY IS NEEDED IN PRACTICE?

As discussed above, the Mirai ecosystem is highly volatile, and although there are a plethora of potentially vulnerable devices, only a small percentage is infected at any given point in time. This poses the obvious question of whether this high volatility impacts the performance of the tarpitting. If we would, for instance, not continuously feed updates into the tarpit, how much would the trapping degrade?

Our experiment uses the input from a network telescope with approximately 65,000 IP addresses to discover infected devices. When the tarpit reaches a steady-state, where on average 6,326 hosts are trapped, this large collection system might not be needed, and the tarpit can instead rely on a small set of monitoring points to remain in a steady state. To identify whether this retention exists, we conduct two separate experiments where we eliminate a large portion of monitoring IP addresses. In the first test on March 7, only 1,000 IP addresses instead of 65,000 were used to monitor for newly infected devices. As shown in Figure 7.11 the number of trapped devices gradually decreases, mainly due to IP churn where devices are still infected but moved to another IP address. In the steady-state of this degradation , the tarpit trapped 4,613 devices, 27% less than the tarpit monitoring 65,000 IP addresses. Thus it would, in principle, be realistic to jumpstart a tarpit with a lot of IP addresses and maintain a reasonable performance with a much smaller installation. In our second test on March 17, only 100 cloud instances were used for monitoring. While still some new devices were identified, a large portion of devices disappeared from the system. The steady-state of the tarpit using only 100 monitoring instances lies at 2,318 trapped devices. The system's retention is mainly influenced by IP churn, where devices leave the trap.

Finally, as a single tarpit reduces Mirai by 21% as 103 billion Mirai connections hit it, is the containment of Mirai just a question of scaling out? While our four-core server on a Gigabit uplink is never saturated on CPU load or uplink capacity, it might be that the large volume of packets leads to packet drops further upstream. To learn about potential upstream limitations, we added a second tarpit to load-balance trapping activity. As shown in the last part of figure 7.11, a doubling of the infrastructure increased the

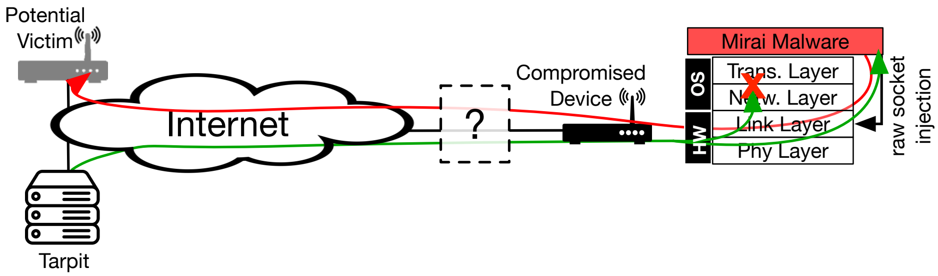


Figure 7.12: Mirai sends and receives with a raw socket, which means that return packets are processed by OS and malware.

number of trapped devices by 5%. The tarpits received more response packets from successfully trapped devices, where more devices send application layer data. There is a slight but statistically significant increase in the number of devices trapped per minute (R=1.02, p<.05). While the second tarpit makes the system more stable, the effect on the containment of Mirai as a whole is however marginal, thus the single tarpit was not held back.

7.7. REACHING THROUGH NATS AND FIREWALLS

The problem we faced with trapping infected devices behind NATs results from the hasty design of the Mirai malware. As the author intended to scan the Internet as fast as possible, the author did not use regular network sockets of the operating system to establish connections but directly crafted requests and injected TCP/IP frames into the network utilizing raw sockets. This allows the malware to scan the Internet at speeds otherwise impossible. This bypassing of the regular OS networking stack for outgoing packets means that return packets are received by both the malware and the networking stack of the OS. To see how this creates an issue for a NAT, let us first consider a Mirai instance connected directly to the Internet.

As shown in figure 7.12, Mirai would scan the Internet with TCP SYN packets via a raw socket to find victim devices. When a response arrives from either an identified victim or our tarpit, the TCP SYN+ACK response (in green) would be received by Mirai as well as the operating system. Mirai adds the destination to its brute-forcing queue, but the OS would not know how to treat this SYN+ACK, having never sent a SYN request to the destination. The TCP protocol mandates the OS to respond with a TCP RST [37], asking the remote party to tear down this connection. When infected IoT devices are directly connected to the Internet, this does not pose any problem, as Mirai would initiate several new brute-forcing connections shortly afterwards. When our tarpit floods Mirai with alleged SYN+ACKs, these connections would be unknown to the OS, but Mirai still takes note of them.

An issue however occurs when Mirai is located behind a network appliance such as a NAT or firewall. These devices usually keep state, and when a NAT or firewall receives the TCP RST from the host OS, it will take note of this unwanted traffic and in the future block

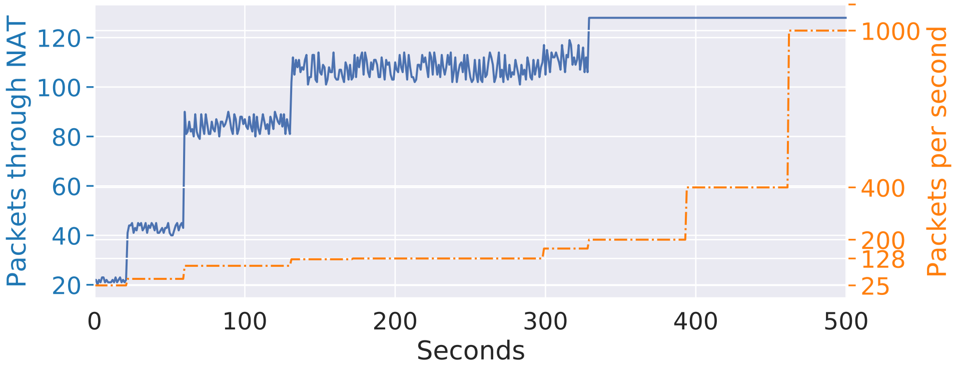


Figure 7.13: Packets received by a device located behind a NAT that has sent a SYN to a tarpit. Until the TCP RST is processed, the tarpit is able to send through multiple SYN+ACKs.

packets from this IP address to the IoT device. For regular Mirai operation, this is not a problem because soon Mirai's brute-forcing threads open new connections, and the NAT will again allow this traffic through. However, this is an issue for tarpitting because we can only get tarpitting requests into the brute-forcing queue until the IoT's OS has returned a TCP RST and the network device shuts off the connection. While tarpitting in principle also works for NATted hosts, the window for trapping is limited.

To investigate this window, we experimentally test how many packets will be allowed through three different NATs before they fall shut: two standard consumer home routers with NAT functionality, (1) a Zyxel VMG8825-T50 router and (2) a Fritzbox 7590, on both devices UPnP was disabled, as well as (3) a Carrier-grade NAT of a large telecom provider for mobile Internet. A SYN packet is sent out from a device located behind the NAT to a tarpit in the experiment. The tarpit then responds with 128 SYN+ACKs, varying the speed between 25 and 1000 packets per second. This allows us to measure the typical processing time window of these devices.

Surprisingly, the first NAT on the Zyxel router lets all packets pass as it ignores the device's RSTs. This means that in such non-compliant TCP implementations, the NAT would remain open indefinitely. A tarpit can trap a device even behind such NAT by continuously sending trapping packets. The NAT on a Fritzbox 7590 correctly closes when observing a RST packet, leaving only a tiny window to send packets. Figure 7.13 shows the number of packets received before the NAT closes for various tarpit speeds. We empirically find that the NAT closes after around 850 milliseconds of receiving the first SYN+ACK. While we can still trap all brute-forcing threads in a device behind this NAT, this tarpitting is only active for a limited time depending on our speed of SYN+ACKs and how large the brute-forcing queue of the infected device is. Also, the carrier-grade NAT we tested ignored the returning TCP RST and allowed arbitrary amounts of traffic to go through. While we would expect large Carrier-grade NATs to understand regular TCP operation, the lack of this checking also allows us to trap devices permanently by constantly sending packets and tying up the threads in the malware.

7.8. Can distributed tarpits eradicate botnets from the Internet?

Middleboxes such as NATs and firewalls not only shut off a connection if the internal device responds with a TCP RST, they also keep a state of the outgoing and incoming connections. This is necessary because an Internet Service Provider might connect hundreds of customers using a single public IP to the Internet. When the customer connects to the outside, the Carrier-Grade NAT (CGNAT) records this request and later matches the response to forward it to the correct host. This is however a significant problem for tarpit operation: as we would have learned about an infected device from a honeypot, the brute-forcing invites from the tarpit towards the Mirai infection could not be matched in the CGNAT to a previous flow and thus be discarded. For this reason, about half of the Mirai infections we observed in our experiments never ended up brute-forcing. In a sense, Mirai infections behind a NAT are therefore immune. However, the solution is trivial: do not only brute-force from a central location, but also respond to brute-forcing attempts from other honeypot locations when they are contacted. As each of these return flows can now be matched, every honeypot increases our leverage on Mirai infections and slightly reduces the spread of the malware.

To evaluate the feasibility of this approach and evaluate the number of instances required for potential eradication, we implemented a discrete-event simulator, which we calibrated based on the earlier experimental results. The framework simulates and predicts the scenarios presented before and will be shared with the community after publication.

7.8.1. Simulator design

Mirai is a collection of independently acting endpoints. The most logical approach for a simulation is an agent-based, discrete event simulation where all infected devices are autonomous, independent entities. Previous work has used similar modeling to test defense methods against, for example, botnet-based DDoS attacks [38], and a similar method can be used to simulate the spread of a botnet. We assume a Susceptible-Infected-Susceptible (SIS) model, where devices return to the pool of vulnerable devices after they have been rebooted or cleaned up. For Mirai, this is a valid assumption, as we observe devices to become reinfected frequently.

In each time step, every agent generates a number of scanning probes and infects newly found vulnerable devices. We randomize the scanned destinations using the randomization method of the Mirai source code. Behind an IP address, a device can locate other agents, honeypots, or vulnerable devices. If an agent encounters a honeypot, it will enter a “trapped” state that reduces the likelihood of infecting other devices. If it finds a vulnerable device, the agent will try to infect it, succeeding with some likelihood. Additionally, the agent is blocked for several time-steps to simulate the brute-forcing.

In a tarpit scenario, there are two ways for an infected device to leave the “trapped” state: the device can be (1) cleaned up, or (2) IP churn changes the IP address of the device, which makes it unreachable for the tarpit so that the device needs to be detected again by a honeypot. A percentage of devices can also be placed behind a NAT, where adversaries can infect these devices, but it is not possible for the tarpit to address these

Table 7.3: Parameters of the agent-based discrete-event simulation

Parameter Name	Description	Default (7.6)
1. Vulnerable devices	Total number of devices on the Internet vulnerable to infections	13,298
2. Honeypots	Number of honeypots on the Internet monitoring and trapping infected devices	65,000
3. Infected devices	Initial amount of infected devices at the start of the simulation	100%
4. NATted devices	Number of the vulnerable devices behind a NAT and thus “immune”	40%
5. Average scan rate	Number of packets sent per average per tick, proportional to the spreading rate	15,000
6. Churn rate	Probability that IP churn occurs on an IP per simulation step	0.05%
7. Cleanup rate	Probability that an infected device is cleaned up and becomes “vulnerable” again	0.02%
8. Blocked time	Number of ticks an agent is blocked having identified a vulnerable device	1
9. Infection probability	Likelihood that a brute-forcing is successful	90%
10. Time steps	Number of ticks for which the simulation is run	4,000
11. Trapping start	Time step at which the trapping starts, giving the infection time to spread	0

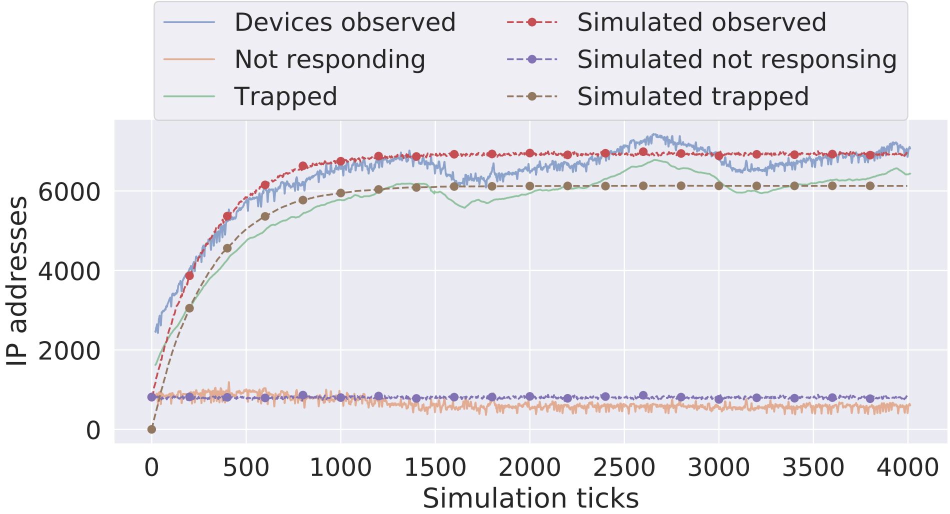


Figure 7.14: Simulation of the system on the experiment.

devices and subsequently trap them. The simulator supports all three aspects, and we will evaluate these variables in the following. Table 7.3 lists all parameters used for the simulation, as well the experimentally measured value from section 7.6.

7.8.2. Simulation validation

To validate the simulation framework, we perform an agent-based simulation of the Mirai ecosystem with one tarpit, based on parameter values that we experimentally measured in section 7.6 and previous work. We are particularly interested in two aspects for the validation: first, does the agent-based simulation reproduce the same characteristics we experimentally measured in the steady-state in section 7.6.1, and second, does it also match the significantly more sensitive transient state when the system is initialized and scales up in section 7.6.2?

Figure 7.14 shows an excerpt of our simulation result together with the measured experimental results from initialization until steady state. The experimental measurements are smoothed using a rolling average of 20 data points to allow for easier readability. The figure shows similar proportions of devices we can trap and those that evade tarpitting, and we see especially in the dynamic scaling a close match. A two-sample Kolmogorov-Smirnov test confirms the similarity as we cannot reject the null hypothesis with $p > 0.1$.

7.8.3. What is needed to eradicate a botnet?

Given this simulator, we can now evaluate the sensitivity of the Mirai botnet to those parameters that we can not change in the real world through our experiments, namely the prevalence of NATs and IP churn, in other words the percentage of devices hidden behind address translation and the turnover of IP addresses which influences how long devices are turned loose before a tarpit captures them again.

"Immunity" through NATs. As seen above, devices can be immune to a centralized tarpit behind a NAT if it drops unsolicited SYN/ACKs. These infected devices will however remain in operation and keep infecting others. In the experiment, we capture all devices observed that are not behind a NAT, but the system remains steady as there are too many immune systems. Running a tarpit directly on a honeypot would however solve this issue: packets will go through the NAT and reach the device because the connection was instantiated from the device behind the NAT. When we simulate a scenario where there are thus no devices immune to the tarpit, and all attacking threads are captured after a device is identified. When honeypots are traversing the NAT themselves after observing a scanning packet from a device, we find that tarpits consisting of only 1,000 devices can effectively mitigate botnets consisting of 100,000 devices.

When devices are placed behind a NAT and honeypots cannot reach them, we find that the system will not trap every device that is located outside a NAT. Depending on the number of devices that "break free" due to IP churn or device cleanups, there is an equilibrium where as many devices are *trapped* every tick as there are devices that are leaving this state. An example of this is shown in Figure 7.14, where 60% of the devices are not behind a NAT and could therefore be trapped, but only 49% is trapped in the tarpit. Immunity to the tarpit from some parts of the ecosystem thus also affects the other devices in the ecosystem, which will not all be trapped.

Permanent tarpit vulnerabilities. Our system needs to continuously contact an infected device to work. Trapped devices can thus break free when a network operator churns the IP addresses in the network. As other research has shown, there are also tarpit vulnerabilities in malware itself that can permanently block a thread of the malware [28]. When a permanent vulnerability is found and exploited to block all malware threads using the method shared in this work, a device contacting a single honeypot can be permanently trapped. Our simulations show that large botnet installations could be completely eradicated using a small number of honeypots. We perform different simulation experiments using various honeynet sizes based on a scenario where tarpit vulnerabilities are permanent and also lock up a device behind a NAT. While these simulations are performed using 50,000 vulnerable devices, this amount does not influence the simulations much, as an increase in infections will also increase overall Internet scanning, making it more likely for the honeypots to find an infected device. Overall it takes a honeynet consisting of only 1,000 devices 3 days to trap 80% of all infections.

Reducing the speed of infection spread. Walla and Rossow note that a tarpit can be very effective in slowing down infection spread from a device [28]. While one device can be significantly impaired and therefore be slowed down, the effects of such slowdowns on the entire ecosystem are not yet known. We simulate a botnet scenario where 1% of the vulnerable population is infected at the start, and a tarpit is active, aiming to disrupt the botnet from spreading through the network. We find that in practice, the effect of this slowdown is marginal as a botnet grows exponentially, and a device has to be observed by the tarpit network first before the slowdown occurs. Suppose the number of vulnerable devices is larger than the size of the tarpit. In that case, infected devices have a higher chance of finding and infecting a new device than ending up in the tarpit, keeping the infection spread alive. In our simulations, we find that when using a tarpit network of 100 devices, the infections reach their highest point after 1,424 ticks, where a tarpit of 1,000 devices would slow the infection spread such that the highest point is reached after 1,577 ticks. Only a tarpit consisting of more honeypots than the number of vulnerable devices would slow the tarpit down enough to remove the infection from the Internet immediately.

Cleanup rate. If a device is cleaned up by an operator or loses the infection due to a device crash or reboot, it will become vulnerable again if no additional action is taken to secure the device. While it is a good thing that infections are removed from the ecosystem, the cleanup rate of systems counter-intuitively slows down the tarpitting efforts significantly as devices can be cleaned up when they are in the "trapped" state and become vulnerable again to malware infection. In our simulations, we have used an average cleanup rate of 7 days. However, when decreasing this rate to once every day, the system cannot trap all devices anymore depending on the scanning speed of the malware, the size of the honeynet, and the amount of vulnerable devices. Thus, in this case, the tarpit would only be effective in slowing down the spread of the infections. When operators however secure their systems after cleanup, a tarpit network contains the spread of the malware and will eventually eradicate the malware.

7.9. DISCUSSION

Can we clean up the Internet with a pit of tar? Although it sounds a bit counter-intuitive to take on botnets bottom-up by spinning up decoy victims, in practice, this turns out to be an effective strategy against self-propagating malware, especially as C&C takedowns are getting increasingly tricky and provide diminishing downtimes. Tarpits have been known for a while but mainly were overlooked to address the threat of botnets, and to this date, there has surprisingly not been a study that evaluated their merits at scale.

Even though NATs are somewhat of a deterrent in theory, their impact is much less pronounced in practice. First, we can easily scale out so that the individual honeypots not only monitor but respond. As SYN+ACK flows of 6.4 Kbps already trapped 98.5% of Mirai's brute-forcing threads, such a scale-out would require trivial resources and not impair the Internet connectivity of organizations or volunteers operating such a device. Second, even if we ignore immune devices, those trapped are missing in the further propagation. At a reproductive rate $R_0 = 1.0033$, this is already enough to contain Mirai, even though we cannot fully eradicate it, as infected pockets behind NATs continue to jump-start the infection. Lastly, we have seen that implementations can be faulty and allow us to punch through with many packets, not just single ones. Together with the stateless scanning now prevalent in port and vulnerability scanning, this gives an exploitable edge even in single tarpit scenarios. As we can frequently get at least one packet through a NAT, it would be possible to turn the tables entirely. Walla and Rossow [28] developed an automatic toolkit to discover vulnerabilities in malware that will cause the software to lock up. Combined with our tarpit at scale, such vulnerabilities would further amplify our impact. Given a permanent vulnerability, even a tarpit infrastructure of trivial size could scale a botnet down.

The easiest route forward – and the one that malware authors could not adapt to – would be to democratize botnet cleanup. Given that enough devices participate – where the required number is low enough that this could be done as a community project – , it will be possible to contain and eliminate this threat platform that has significantly heightened the Internet threat landscape in the past years. We release a flexible, turnkey honeypot/tarpit solution as open-source to the solution to leverage this insight. Organizations and individual volunteers can run their part of a botnet sinkhole on spare devices, requiring minimal resources from the hardware and the network. We aim to set up this platform, and we invite others to join this effort. Given a low entry barrier and a clear path towards mitigation, it will be possible to accomplish this jointly. The software and related documentation is provided at *removed for review*.

7.9.1. APPLICATION TOWARDS OTHER BOTNETS

Throughout this work, we mainly focused on the IoT botnet Mirai, as this malware are still largest class of botnets when looking at network telescope data [39]. Since the inception of Mirai, stateless scanning has been adopted by a plethora of other botnets and malware. For instance, Brickerbot and VPNFilter [40, 41] rely on stateless scanning, as does the ADB Miner worm that exploits Android televisions and set-top boxes [39]. Aside from malware variants that are direct branches off the original Mirai source code, of which now several dozens exist that are vulnerable to our trapping [42], malware authors have over the past years included parts of the Mirai source code into their own designs and thus carry this vulnerability over. As an example, the Gafgyt botnet uses the Mirai scanning routine on the ports it targeted [43], and therefore inherits Mirai's susceptibility to our tarpit. Together, Gafgyt and Mirai account for more than half of DDoS attacks in 2021 [44]. Due to its clear performance advantages, the use of stateless scanning is increasing, Hiesgen et al. find that today 13% of all scanning activity is stateless and tries to connect in a second stage after the initial scan [45]. This number will only increase over time. While we focus on Mirai as the largest class of botnets in this paper, the presented method is widely applicable beyond that.

7.10. CONCLUSION

Taking down a botnet is generally done by removing or taking over the C&C server a botnet is contacting or by arresting the perpetrators behind the botnet. These measures become increasingly complex as botnets use sophisticated means to hide or make their infrastructure more resilient. In this paper, we evaluate a possible alternative whether a network of *tarpits* can be used to contain malware, and we find that even with one device of minimal requirements on CPU and bandwidth, the propagation can be significantly slowed down, and complete containment of these threats is within reach.

REFERENCES

[1] H. Griffioen and C. Doerr, *Examining mirai's battle over the internet of things,* in *ACM Conference on Computer and Communications Security (CCS)* (2020).

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, *Understanding the mirai botnet,* in *26th USENIX security symposium (USENIX Security 17)* (2017) pp. 1093–1110.

[3] T. Alladi, V. Chamola, B. Sikdar, and K.-K. R. Choo, *Consumer iot: Security vulnerability case studies and solutions,* IEEE Consumer Electronics Magazine **9**, 17 (2020).

[4] K. Vaniea and Y. Rashidi, *Tales of software updates: The process of updating software,* in *Proceedings of the 2016 chi conference on human factors in computing systems* (2016) pp. 3215–3226.

[5] H. Heo and S. Shin, *Who is knocking on the telnet port: A large-scale empirical study of network scanning,* in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (2018) pp. 625–636.

[6] Z. Durumeric, M. Bailey, and J. A. Halderman, *An internet-wide view of internet-wide scanning,* in *23rd USENIX Security Symposium (USENIX Security 14)* (2014) pp. 65–78.

[7] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, *Ddos in the iot: Mirai and other botnets,* Computer **50**, 80 (2017).

[8] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, *Botnet in ddos attacks: trends and challenges,* IEEE Communications Surveys & Tutorials **17**, 2242 (2015).

[9] A. Pathak, F. Qian, Y. C. Hu, Z. M. Mao, and S. Ranjan, *Botnet spam campaigns can be long lasting: evidence, implications, and analysis,* ACM SIGMETRICS Performance Evaluation Review **37**, 13 (2009).

[10] H. L. Bijmans, T. M. Booij, and C. Doerr, *Just the tip of the iceberg: Internet-scale exploitation of routers for cryptojacking,* in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019) pp. 449–464.

[11] Z. Durumeric, E. Wustrow, and J. A. Halderman, *Zmap: Fast internet-wide scanning and its security applications,* in *22nd USENIX Security Symposium (USENIX Security 13)* (2013) pp. 605–620.

[12] M. Anisetti, C. Ardagna, M. Cremonini, E. Damiani, J. Sessa, and L. Costa, *Security threat landscape,* .

[13] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, *Measurement and analysis of hajime, a peer-to-peer iot botnet,* in *Network and Distributed Systems Security (NDSS) Symposium* (2019).

[14] H. Griffioen and C. Doerr, *Quantifying autonomous system ip churn using attack traffic of botnets,* in *Proceedings of the 15th International Conference on Availability, Reliability and Security* (2020) pp. 1–10.

[15] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee, *Beheading hydras: performing effective botnet takedowns,* in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013) pp. 121–132.

[16] D. Dittrich, *So you want to take over a botnet...* in *5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 12)* (2012).

[17] C. Gañán, O. Cetin, and M. van Eeten, *An empirical analysis of zeus c&c lifetime,* in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security* (2015) pp. 97–108.

[18] J. Healey, N. Jenkins, and J. Work, *Defenders disrupting adversaries: framework, dataset, and case studies of disruptive counter-cyber operations,* in *2020 12th International Conference on Cyber Conflict (CyCon)*, Vol. 1300 (IEEE, 2020) pp. 251–274.

[19] J. Nazario and T. Holz, *As the net churns: Fast-flux botnet observations,* in *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)* (IEEE, 2008) pp. 24–31.

[20] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, *Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus,* in *2013 8th International Conference on Malicious and Unwanted Software:" The Americas"(MALWARE)* (IEEE, 2013) pp. 116–123.

[21] S. Pletinckx, C. Trap, and C. Doerr, *Malware coordination using the blockchain: An analysis of the cerber ransomware,* in *2018 IEEE Conference on Communications and Network Security (CNS)* (IEEE, 2018) pp. 1–9.

[22] J. Lathrop and J. B. O'Kane, *A case study in opportunity reduction: Mitigating the dirt jumper drive-smart attack,* in *2014 IEEE Joint Intelligence and Security Informatics Conference* (IEEE, 2014) pp. 224–227.

[23] A. A. Antonopoulos, K. G. Stefanidis, and A. G. Voyiatzis, *Fighting spammers with spam,* in *2009 International Symposium on Autonomous Decentralized Systems* (IEEE, 2009) pp. 1–5.

[24] A. H. UW, Y.-K. Liu, and A. R. UW, *Counter-attacks for cybersecurity threats,* (2005).

[25] Https://labrea.sourceforge.io/Intro-History.html.

[26] D. Moore, C. Shannon, and K. Claffy, *Code-red: a case study on the spread and victims of an internet worm,* in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2002) pp. 273–284.

[27] K. Borders, L. Falk, and A. Prakash, *Openfire: Using deception to reduce network attacks,* in *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007* (IEEE, 2007) pp. 224–233.

[28] S. Walla and C. Rossow, *Malpity: Automatic identification and exploitation of tarpit vulnerabilities in malware,* in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)* (IEEE, 2019) pp. 590–605.

[29] K. G. Hunter, *Scaling Network Tarpits For IPV6*, Tech. Rep. (NAVAL POSTGRADUATE SCHOOL MONTEREY CA MONTEREY United States, 2018).

[30] *Mirai source code,* https://github.com/jgamblin/Mirai-Source-Code.

[31] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, *Zippier zmap: Internet-wide scanning at 10 gbps,* in *8th USENIX Workshop on Offensive Technologies (WOOT 14)* (USENIX Association, San Diego, CA, 2014).

[32] Https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/.

[33] G. C. Moura, C. Ganán, Q. Lone, P. Poursaied, H. Asghari, and M. van Eeten, *How dynamic is the isps address space? towards internet-wide dhcp churn estimation,* in *2015 IFIP Networking Conference (IFIP Networking)* (IEEE, 2015) pp. 1–9.

[34] R. Padmanabhan, A. Dhamdhere, E. Aben, K. Claffy, and N. Spring, *Reasons dynamic addresses change,* in *Proceedings of the 2016 Internet Measurement Conference* (2016) pp. 183–198.

[35] *Ip addresses by country,* https://worldpopulationreview.com/country-rankings/ip-address-by-country.

[36] *Internet speed by country,* https://www.fastmetrics.com/internet-connection-speed-by-country.php.

[37] P. John, *Transmission control protocol,* RFC 793 (1981).

[38] I. Kotenko, A. Konovalov, and A. Shorov, *Simulation of botnets: Agent-based approach,* in *Intelligent Distributed Computing IV* (Springer, 2010) pp. 247–252.

[39] L. Gioacchini, L. Vassio, M. Mellia, I. Drago, Z. B. Houidi, and D. Rossi, *Darkvec: automatic analysis of darknet traffic with word embeddings,* in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies* (2021) pp. 76–89.

[40] S.-Y. Hwang and J.-N. Kim, *A malware distribution simulator for the verification of network threat prevention tools,* Sensors **21**, 6983 (2021).

[41] B. Vignau, R. Khoury, and S. Hallé, *10 years of iot malware: A feature-based taxonomy,* in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (IEEE, 2019) pp. 458–465.

[42] Y. Liu and H. Wang, *Tracking mirai variants,* Virus Bulletin , 1 (2018).

[43] *Mirai code re-use in gafgyt,* https://www.uptycs.com/blog/mirai-code-re-use-in-gafgyt (2021).

[44] *Botnet statistics 2021,* https://www.comparitech.com/blog/information-security/botnet-statistics/ (2022).

[45] R. Hiesgen, M. Nawrocki, A. King, A. Dainotti, T. C. Schmidt, and M. Wählisch, *Spoki: Unveiling a new wave of scanners through a reactive network telescope,* arXiv preprint arXiv:2110.05160 (2021).

8

Discussion

The number of cyber security incidents increases each year rapidly. To defend against the increasing number of threats, cyber threat intelligence (CTI) is relied upon by many organizations and their Security Operation Center (SOC) to identify threats, calculate risk profiles, and even make business decisions. However, it is largely unknown what the quality of this CTI information is. As relying on CTI information that is outdated or inaccurate would not lead to the desired result, it is paramount that this information is of good quality. With the contributions in this thesis, we improve the current state-of-the-art in CTI research. We propose and evaluate quality metrics for CTI feeds, introduce new ways to estimate measurement biases in network data, add to the understanding of threats and actors, and evaluate a method to fight back against botnets, which have been a significant security threat for many years.

In this chapter, we present our findings and discuss their limitations. We will address some of the challenges listed in Section 1.2 and the research questions introduced in Section 1.3.

8.1. Quality of CTI feeds

In this section, we explain our contributions to the first subquestion:

RQ: *How can we measure the quality of Cyber Threat Intelligence feeds?*

To answer this question and thus better understand the quality of CTI information that is shared in public feeds, this thesis proposes and evaluates several quality metrics in Chapter 2. The metrics defined in this thesis and discussed in this section can be used to measure the quality of CTI feeds.

8.1.1. Quality metrics of CTI feeds

Threat Intelligence feeds are crucial to many defensive infrastructures as they provide organizations with a proactive way to mitigate previously unseen threats. While these feeds are a popular tool in the defensive arsenal, their quality is widely unknown. This section discusses the four metrics we use to measure and compare CTI feeds.

Timeliness - In Chapter 2 we utilize the netflows of a Tier-1 operator to identify when malicious activity was first seen coming from an IP address. We compare this with the listing time in a feed to identify the time lag between the start of malicious activity and the successful detection in a feed. Other studies have tried to estimate this *timeliness* by identifying the first feed in which an indicator appears and counting latency for the other feeds from this time on [1]. This method does not correctly estimate the *timeliness* as the first listing did not necessarily show when the malicious activity started. Furthermore, the lack of overlap between the various lists makes for only a small sample size of indicators where this supposed *timeliness* can be measured. By using measurements of activity from an IP address, we can more accurately estimate the *timeliness* for all the indicators listed on feeds and find that, on average, it takes 21 days for malicious activity to appear in a feed. There are, however, limitations to these measurements as well. Firstly, the netflows from the Tier-1 operator are sampled such that only see 1 in 8192 packets are translated to flows. When malicious activity is minimal, this can lead to underestimating the total time for which malicious activity originates from an IP address before being listed in a feed. Secondly, while the network of the Tier-1 operator is extensive and contains netflows from several routers distributed globally, there is still a geographical bias to the netflow measurements. Indicators might not show up in the netflow measurements because their traffic is never routed through the operator's routers. Thirdly, the constant flux of IP addresses known as IP churn might bias our measurements as "new" malicious activity might instead be going on for a while on another IP address [2].

Sensitivity - The sensitivity of a feed relies on the measurement points that gather the indicators listed on the feed. When an organization is attacked before the measurement infrastructure of a feed, there is no prior knowledge for the organization, and therefore the activity is not proactively blocked. To our knowledge, no other studies have looked at the measurement infrastructure from which feeds are generated and the bias that this creates. While we do not know where the measurement infrastructure of these feeds is located, we find that almost all investigated feeds are highly biased towards certain geographical regions. With the Internet's global connectivity, this is curious, as it shows that these feeds observe various localized attacks. We have only measured open source feeds, while a major reason that people opt to pay high fees for commercial threat intelligence feeds is that the measurement infrastructure of the feed providers is much larger and therefore contains many more threats. The main limitation in the evaluation of this metric is, therefore, the comparison between the open-source and commercial feeds for their *sensitivity* and evaluating whether the commercial feeds thus show a broader threat landscape.

Originality - When feeds measuring the same threat have almost no overlap at all, none of these feeds are showing a complete picture of the threat. If feeds would instead show a complete picture and therefore list all indicators belonging to a threat, all indicators for this threat would overlap for the feeds. A considerable overlap between feeds measuring the same threats would signal that either the feeds are near complete or the feeds have the same bias and look at the same sub-portion of the threat. Low overlap between feeds would signify that the feeds are not nearly capturing the entire threat. When looking at overlap between feeds, we find that the originality is lacking and therefore conclude that single feeds are not detecting a large part of the indicators. Similar research identifies the same lack of overlap between feeds [1, 3]. To measure the value of a single feed, Li et al. [1] split *originality* into two parts that identify the added benefit of a feed as a whole. In contrast, our methodology only identifies the state of the feed landscape. Like with *sensitivity*, it can be reasoned that commercial feeds will perform well in their coverage of a threat because of their more extensive measurement infrastructure. However, in follow-up research that looks at some large commercial threat intelligence providers, it is shown that even for these large providers, the *originality* is low when measuring the same threats [4].

Impact - Automatically blocking an IP address or domain name based on an indicator in a feed should solely remove the malicious activity. In practice, however, some indicators shared through feeds point towards shared hosting providers or websites that should not be blocked. *Impact* measures how much “collateral damage” will result from blocking all indicators in a feed. The majority of feeds we investigate perform well in this category, with some notable exceptions. In one case, blocking an indicator in a feed would deny access to almost a million domains. Without ground truth, it is impossible to identify the actual false-positive rate of a feed, and we provide only a lower bound. Other researchers also provide a lower bound on feed accuracy by manually identifying items that should not be in a feed [1].

8.1.2. Limitations

We gather data from a highly diverse number of sources but note that many of the numbers that we can compute based on the data are either upper- or lower bounds. The netflow data from a Tier-1 operator covers a broad spectrum of the Internet but is still limited to certain geographical regions where data has to pass through to be included in these flows. Together with a sampling rate of 1 in 8192 and network artifacts such as IP churn, measurements on *timeliness* can only be provided as a lower bound.

With the lack of ground truth about malicious activity, the absolute value of a single feed is hard to identify. While we estimate to some extent the metrics *originality* and *impact*, we cannot identify the absolute value of CTI feeds. While the ground truth of all malicious indicators will likely never be fully known, our metrics would benefit from a more robust method to identify false-positive indicators.

Some CTI information is not included in our evaluation as commercial CTI feeds are very expensive or prohibit publication or measurements of the indicators shared on these feeds. While we show the state of open CTI feeds in our work, the state of CTI, in general, is also dependent on the large commercial providers. Follow-up work has partially investigated our metrics on the feeds of two large commercial vendors [4], but does not identify *timeliness* or *sensitivity*.

8.1.3. Reflection

The metrics defined in this thesis can be used to measure the quality of CTI feeds and provide a means to identify the current state of CTI feeds as a whole. While the metrics are thoroughly evaluated, the usefulness of these metrics for organizations might be limited. For an organization, it is essential to evaluate the value of a feed before integrating it, especially when considering the sometimes hefty price of a feed. However, the quality of a CTI feed, which can be measured using the metrics introduced in this work, will not always translate to the value of the feed for an organization. To measure value, it is important to understand the area in which the organization operates and the services that the organization uses. A feed listing all known SSH scanners does not help an organization that does not use SSH. Furthermore, many feeds have a sizeable geographical bias, indicating that attacks can be biased towards a measurement infrastructure. Similarly, we can expect this attacker bias for organizations for the same reason, reducing the value of some feeds for this particular organization because their biases do not match. While we thus measure the quality of CTI feeds, we do not measure the *value* of these feeds for an organization, which should be a focus of future work.

Typical network artifacts such as IP Churn make static indicators very unreliable. In generating CTI, these are not taken into account. This could partially explain why indicator lists score poorly on quality criteria. However, we have not investigated the precise effect of, for example, IP churn with respect to these feeds. Conclusions are drawn about, for example, emerging threats that are important to consider in CTI could be biased when counting, for example, a high number of infected IP addresses while the exact number of infections are much lower. So due to these effects, even if the measurement infrastructure used is sound, the data collected might not provide the most accurate CTI. In Chapter 3 we address IP churn and carrier-grade NATs and show that this can indeed lead to large over-or underestimations of a threat. With the ability to better measure IP churn in networks, we can now identify the effect of churn on indicator lists and how this affects the false positives, providing a better understanding of a lists' quality. However, during our study on these metrics, we do not take these artifacts into account.

Overall we find that the quality of CTI feeds, measured using our metrics, is poor. The indicators shared in automated forms are often not practical due to a lack of sensitivity, timeliness, too much originality, and too much collateral damage. These shortcomings are a result of how threat intelligence is collected. The collection is "reaction-based"; we first need to see something before we can list it in a feed. This means that a small change from an adversary will make their method undetected again. To know someone is "out there", it is necessary to observe them once. However, with current CTI, we have to "rediscover" them repeatedly when the adversary changes some basic indicators such as an IP address. Instead, we should create more "active" approaches, where we adopt intelligence that better describes how the actor behaves so that it is much harder to evade detection once an actor is discovered.

8.2. TOOLS, TTPS AND ACTOR EVOLUTION

In this section, we discuss the second question of this thesis:

> ***RQ:*** *How do actors differ in their TTPs, and how can we better measure these TTPs?*

We answer this question using three case studies. While this limits the scope, it does allow for a better investigation of specific TTPs and adversarial strategies for the discussed cases. We address the limitations of this approach in section 8.2.4.

8.2.1. Detecting SIP scanning tools

In Chapter 4 we investigate the fingerprinting of SIP scanning tools and the evolution of actors that perform SIP scans. The contribution of this chapter is two-fold. First, we provide a way to fingerprint SIP scanners by observing a single packet from a source, enabling defenders to pre-emptively identify and block emerging SIP scanners on a higher indicator level of the pyramid of pain introduced in section 1.1. Second, using the fingerprints, we can perform campaign analysis on these SIP scanners and follow the evolution of these scanners over time. While some tools are easily detectable because they set SIP header values identifying themselves as scanners, other tools try to stay undetected by using randomized values instead. We find a wide variety of scanning tools and find that most actors stick to the same tool even over five years. In terms of TTPs, we do not find significant evolution in tool usage, "stealthiness", or large scanning infrastructures.

8.2.2. TTPs in DDoS attacks

We measure DDoS attacks and identify TTPs using an extensive network of honeypots in Chapter 5. The difference between our honeynet and similar projects is the size and the ability to provide adversaries with different configurations of honeypots, which allows us to test what adversaries are looking for when selecting a server that they will abuse in their attack.

Chapter 5 has two main contributions. First, we identify a "kill chain" model for DDoS attacks. This is an important step for the identification of TTPs and campaign analysis, as the regular kill-chain illustrated in section 1.1 does not fit this type of attack. Second, we analyze the different adversaries that perform these attacks using the model. Most importantly, we identify whether these adversaries are actively trying to increase their attack power by testing the servers that they abuse in an attack. The goal is to identify how "sophisticated" adversaries behave when performing these attacks.

We find a large diversity of attackers, where there are actors that actively select the servers that are hitting the hardest and attackers that do not care which servers they use as long as the server responds. We do not observe these differences in single attacks, but they also become apparent when looking at attacker campaigns. For the most part, attackers who operate attack campaigns scrutinize the servers they use more than the attackers who perform a single attack. In other words, attackers performing multiple attacks are more likely to ensure that their attack has the highest chance of success. Between campaigns, there are also considerable differences. For example, some actors identify whether a server is an obvious honeypot or is severely rate-limited and do not include these servers in their attack. Other actors only care about the response size that a server provides and do not look at any other information. Using our proposed model, we can classify these actors and identify the strategy in each attack preparation stage.

8.2.3. Evolution of botnet actors

Chapter 6 measures brute-forcing attempts from the Mirai botnet and its copycats. After the Mirai source-code leak, many botnets emerged using the same codebase and exploitation methods. This ecosystem of Mirai-based botnets provides us a unique opportunity to learn how adversaries adapt to beat the competition and obtain a sizeable share of vulnerable devices. By tracking these botnets using 7,500 honeypots for three months, we visualize the constant “battle” over IoT devices and provide an epidemiological quantification of the sustainability of Mirai.

Mirai-based malwares compete over the same devices as they utilize the same weak password brute-forcing to exploit a device. The botnet operators have two ways to grow their network, (1) they can “steal” devices from other botnets, and (2) they can try to find a new vulnerable population by adding more credentials to log in. We observe both measures in our data and notice that actors try to prevent others from gaining an advantage. Most of the re-infections that we observe are from the botnet that already had control over the device because actors take them over again right away if they stop responding, and actors quickly pick up on new login credentials used by others.

Epidemiologically, Mirai is only barely self-sustaining. With the vulnerable population shrinking and the competition in this space, it is hard for botnet operators to keep their botnet alive. We see that the first-mover advantage is a key factor in the success of a botnet, where a botnet takes over many devices in a particular autonomous system (AS) and remains the dominant infection in this AS for a long time. The relation between infections and ASes is also impacted by the stability of malware variants for different router models, as some variants perform much better in certain ASes.

From the findings in Chapter 6 we conclude that actors tailor their malware towards certain ASes by selecting login credentials that match and keep learning from their competition to grow their botnets.

8.2.4. LIMITATIONS

In all three case studies, we utilize measurement infrastructures that rely on the ability of an adversary to find the probe. With many mass-scanning tools and the growing popularity of stateless scanning, the probability of adversaries hitting our infrastructure are high. Nonetheless, some of our infrastructure is located in a known network telescope that adversaries can circumvent to thwart our measurement efforts, for which we have found evidence in Chapter 6. In the analysis of Chapter 6 we estimate that we miss 2.9% of the IP addresses that scan for telnet, based on netflows of a Tier-1 operator. While this number is seemingly small, it might consist of the most sophisticated adversaries introducing a bias in the data.

We deploy sizeable honeypot infrastructures and show that we need many more honeypots than used in comparable studies to provide an accurate picture of the threat landscape. Performing such large-scale studies comes with high costs and maintenance time, limiting our studies’ duration. It would be advisable to investigate threats with an extensive monitoring infrastructure that is active for a more extended period.

TTPs are a broad term, encompassing the strategy of the adversary, the way in which the adversary conducts attacks, and the way in which the adversary approaches different challenges. TTPs are inherently “fuzzy” as it is hard to measure such abstract principles. In this chapter, we identify TTPs manually using campaign analysis, limiting the scope of this study to the three subjects of the case studies. Ideally, there would be an ontology of TTPs and a method to describe these to broaden TTP-related studies’ scope automatically.

8.2.5. REFLECTION

Our case studies find a clear separation between adversaries actively trying to stay undetected by distributing their scanning efforts and not using obvious honeypots and adversaries that do not care about being detected. The same goes for how "sophisticated" adversaries behave, with some adversaries testing out new credential combinations to find new vulnerable devices, and others copy what others are doing. In all three case studies, we find a surprising lack of sophistication and evolution for the bulk of adversaries. These actors are still successful, which points to our inability as defenders to detect and mitigate emerging threats quickly. If the timeliness with which we identify and list malicious indicators on a feed increases, these unsophisticated adversaries will slowly be prices out of the market. However, we also observe a small subset of actors that evolve rapidly in response to countermeasures. While these actors would not be caught by blocking simple indicators, the blocks can result in the removal of many "script kiddies", stopping their traffic from reaching an organization. This means that analysts have to spend less time with alerts originating from unsophisticated attackers and more time focusing on the alerts from skilled adversaries.

We have shown in Chapter 4 that it is possible to identify SIP scanning tools by observing as little as one network packet targeting our network telescope. The methodology used in this chapter could also be scaled to other protocols. Similar methodologies could be used in defensive infrastructures as a form of deep packet inspection, which is already present in modern intrusion detection systems [5]. The success of such measures in a real-world network used in practice is unknown, and using these fingerprints might be prone to false positives. The downside from using fingerprints or signatures to identify malicious activity is that adversaries might adapt their fingerprints to evade detection. However, the lack of evolution of adversaries and tools that we observe in our studies indicates that this might not be a significant issue. Most attackers would use the same tool for a long time regardless of these countermeasures.

In Chapter 5 we find that adversaries can be highly selective of the services that they target or abuse. This selectiveness has implications for studies measuring malicious activity using honeypots or attacks on servers, as adversaries might not be interested in the specific configuration of the monitoring device, or they are looking at specific geographical regions. Systems should be scattered and use different configurations to understand the natural threat landscape better. However, the number of configurations and locations are endless, and only part of this space can be covered. An experimentally sound setup is thus essential, as the setup of the systems can influence the type of adversary that is observed within this system. Works performing such empirical research should therefore be cautious about their experimental setup and be clear about the biases that the configuration of the monitoring services could introduce.

USING CTI TO FIGHT BACK

In this section, we discuss the third research question:

RQ: *What is the feasibility of using information about adversarial tactics to disrupt malicious activities?*

To answer this question, we try to disrupt the Mirai botnet using the information gathered in Chapter 6 that Mirai is only barely self-sustaining with a reproduction rate just above 1. If this reproduction rate dropped below 1, the botnet would shrink over time and ultimately be removed from the Internet.

8.2.6. TARPITTING THE MIRAI BOTNET

Chapter 7 identifies a tarpit vulnerability in the scanning process of Mirai. This vulnerability stems from Mirai implementing stateless scanning to spread its infections. The main reason for this scanning strategy is speed, and similar methods are used in high-performance scanning tools such as ZMap [6]. Because the scanner does not keep a 'state' of which scans have been sent out, we can impersonate a scanned device and trick the scanner into believing that we should be brute-forced. If we trick an infected device enough times, it will spend all of its brute-forcing power on our impersonated device and cannot identify real vulnerable devices.

We validate our system on local instances of Mirai and make sure that it does not affect the routers of end-users. During this validation, we find that we can reduce scanning rates of infected devices by a factor of 20 without affecting the standard functionality of an infected device. The reduction in scanning rate would translate to a reduction in infection spread because it takes longer before vulnerable devices are found.

After verifying the method, we deploy a server in the wild that actively tarpits the stateless-scanning methods used by the Mirai botnet. Using a single decoy server, we can tie up 48% of Mirai-infected devices and cut brute-forcing across the Internet in half. Our results show that we can indeed use adversarial strategies to disrupt malicious activity.

8.2.7. LIMITATIONS

We only evaluate our method to tarpit stateless-scanning malware for Mirai. Different stateless-scanning malware that does not embed secret information into the header fields of a packet should be equally vulnerable, but we have not tested this in practice.

Using our method, we cannot address devices behind a NAT or restrictive firewall that does not allow TCP sessions without a prior handshake. We are therefore unable to reach and subsequently tarpit these devices. To obtain a more accurate estimation of the impact of tarpits on a botnet, we need a different measurement approach that handles these shortcomings. The number of devices we can tarpit and the speed reduction of the scanning rate of the botnet is therefore only a lower bound.

8.2.8. REFLECTION

We investigate a vulnerability of stateless scanners that are used to speed up scanning efforts. These scanners are increasing [7], and we also observe this technique being used in botnets [8]. While investigating a particular tarpit vulnerability, we also evaluate the merits of tarpitting at scale. Our evaluation finds that tarpitting can be an effective method if it is implemented on a large enough infrastructure. While taking down botnets is a challenging task [9], our analysis shows that targeting the techniques of the malware authors can lead to better defenses against these botnets.

A downside of our tarpit strategy is that it also generates traffic on intermediate infrastructures. So while we prove in this thesis that we can significantly slow down botnet spread (Chapter 7), there are ethical questions to address on whether we can introduce more unsolicited network traffic in the process if the community starts to do this on a massive scale. However, scaling up other tarpit vulnerabilities that are less intrusive [10] can provide a way to defend better against emerging threats without introducing much traffic in regular infrastructures.

In Chapter 7 we investigate defenses against stateless-scanning. Adversaries could circumvent these defenses by reverting to stateful-scanning techniques to spread the infections, but by doing so, the scanning speed would be lowered [6]. This tarpit study is an example of how we can leverage information about adversarial techniques, and we can find new ways to stop these emerging threats and force adversaries to change their techniques to other techniques that perform worse.

8.3. CONCLUSION

The main question in this thesis, as introduced in section 1.4 is the following:

RQ: *To what extent can we increase the quality of CTI by learning about adversaries, and how can we use this information to create better defenses?*

To address this question, we first measure the quality of CTI feeds in the form of simple indicator lists and find that these feeds lack every quality criteria that we identify. While the value of these feeds for a specific organization can differ from the quality because it may suit the threat landscape of this organization very well, the lack of timeliness within these feeds makes them mostly unusable even when it is a perfect match. It is thus clear that the quality of CTI is lacking and that simple indicator lists do not measure up to the increasing number of cyberattacks. Therefore, we explore whether we can identify and fingerprint specific tools and provide frameworks that we can use to describe the TTPs of adversaries better. We show that we can identify adversarial campaigns and track them over time to measure their evolution using these new frameworks. While measuring the TTPs of adversaries, we identify a flaw in a significant innovation of scanning techniques used by malicious actors and show that we can impact the growth of a real-world botnet when exploiting this flaw. To increase the quality of CTI, we can thus leverage methods to measure the TTPs of actors and identify weak points in these strategies. By defending against malicious activity by exploiting the weak points of the actor we can ultimately forcing adversaries to redesign their operations and increase the cost for these actors drastically. Moreover, by fingerprinting tools, we are able to better identify emerging threats, a vital part of CTI, but also a part which the current state of CTI and threat intelligence feeds does not sufficiently cover.

REFERENCES

[1] *Reading the tea leaves: A comparative analysis of threat intelligence,* in *28th USENIX Security Symposium (USENIX Security 19)* (USENIX Association, Santa Clara, CA, 2019).

[2] L. Vu, D. Turaga, and S. Parthasarathy, *Impact of dhcp churn on network characterization,* ACM SIGMETRICS Performance Evaluation Review **42**, 587 (2014).

[3] M. Kührer, C. Rossow, and T. Holz, *Paint it black: Evaluating the effectiveness of malware blacklists,* in *International Workshop on Recent Advances in Intrusion Detection* (Springer, 2014) pp. 1–21.

[4] X. Bouwman, H. Griffioen, J. Egbers, C. Doerr, B. Klievink, and M. Van Eeten, *A different cup of ti? the added value of commercial threat intelligence,* in *29th USENIX Security Symposium (USENIX Security 20)* (2020) pp. 433–450.

[5] R. T. El-Maghraby, N. M. Abd Elazim, and A. M. Bahaa-Eldin, *A survey on deep packet inspection,* in *2017 12th International Conference on Computer Engineering and Systems (ICCES)* (IEEE, 2017) pp. 188–197.

[6] Z. Durumeric, E. Wustrow, and J. A. Halderman, *Zmap: Fast internet-wide scanning and its security applications,* in *22nd USENIX Security Symposium (USENIX Security 13)* (2013) pp. 605–620.

[7] R. Hiesgen, M. Nawrocki, A. King, A. Dainotti, T. C. Schmidt, and M. Wählisch, *Spoki: Unveiling a new wave of scanners through a reactive network telescope,* arXiv preprint arXiv:2110.05160 (2021).

[8] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, *Understanding the mirai botnet,* in *26th USENIX security symposium (USENIX Security 17)* (2017) pp. 1093–1110.

[9] C. Gañán, O. Cetin, and M. van Eeten, *An empirical analysis of zeus c&c lifetime,* in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security* (2015) pp. 97–108.

[10] S. Walla and C. Rossow, *Malpity: Automatic identification and exploitation of tarpit vulnerabilities in malware,* in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)* (IEEE, 2019) pp. 590–605.

Acknowledgements

At the end of this dissertation I would like to look back on my PhD journey and express my gratitude to the people who made the past years much more enjoyable. With Covid-19 and conferences moving online, it was sometimes hard to keep enjoying the work. Luckily I have many great people in my life who made this journey enjoyable and kept me going through the difficult times. I know that it is a gift to have the skills and opportunity to do a PhD, and I am grateful that I have received this gift.

First and foremost I would like to thank my mom, without whom I would never have made it to work in time. While I was not always grateful at the time, thanks for kicking me out of bed almost every morning. When I moved out, Rienke took over this job, and I am even more grateful to her for not dragging me out of bed to go to work. I am so grateful that she puts up with me, and I hope she forever will.

I also thank both my promotors, Inald and Christian. Inald, I have enjoyed our discussions and count myself lucky to have worked with you. I am deeply inspired by your professionalism and appreciate the support you have provided throughout my PhD. Christian, I would like to deeply thank you for the discussions, ideas, trips, late-night working sessions, and fun we had together. I would not be at the point I am today without your guidance. I apologize for not attending your lectures during the Master program, but I am happy you were able to look past that when hiring my as a PhD student ;). I wish you the best in the future at the HPI and hope to read the papers that your group will inevitably publish. From the bottom of my heart, thank you.

I also extend my gratitude to the committee members Michel, Herbert, Frans, Matthias, and Christoph for taking the time to review my thesis, providing me with valuable feedback, and being part of the defense ceremony.

During my PhD, I have met many wonderful colleagues. I am grateful for all of them, for being people i can look up to, people that challenge and inspire me, and people to have a great time with. I will be forever grateful to Sandra for reminding me to keep hydrated by dragging me to the coffee machine 30 times a day. The lack of productivity from standing around at the coffee machine was always negated by the overdose of caffeine resulting from these trips. I will always remember the nice atmosphere you created and I hope that you will never change.

I have shared my office with many different people, some in Delft and some in Potsdam. Mark, you were the doyen and I hope you will finish your PhD one day. Vincent, I have had the closest contacts with you over the years and have enjoyed our talks at the morning meetings and our trips to Japan. I won't ever forget the shisha-lounge we ended up in in Tokyo. Watson and Afshin, we have not seen each other often, I hope you have a wonderful time during the PhD and find it as valuable as I do.

Kris, while your PhD was part-time we still have had a lot of time together. You are one of the hardest working people I know, and I admire how you have juggled your job, the PhD and your personal life in this time. I am happy that you can openly share your experiences with me and appreciate that I can do the same with you. The discussions we have had did not only shape me professionally, but also made me reflect on myself as a person and I cannot thank you enough.

During my time in Delft I have also been constantly confronted with the boring office at the other side of the building. I have taken it upon myself to bring some life and joy to the people in this office during my time here. Gamze, thanks for showing me how delicious Turkish dishes are and inviting me to your birthday parties. I wish you had graduated later so that we could have more time together. You are an amazing person. Ozzy, thanks for being my *BFFF* and for all your jokes. I know you always wanted to make it to the morning coffee, and its a shame that you never made it to the office that early. Azqa, we started at the same time in the 6-year PhD contract, sorry for bailing early. I admire your love for teaching and the effort you put into creating very good teaching material. Miray, please learn how to ride a bike, I know you can do it. Marina, we haven't spoken that often, but thanks for bringing some cool-vibes to the otherwise boring office and for taking Ozzy snowboarding.

Zeki, thanks for all the fun times at the coffee machine, the abundance of cake, and the arm-wrestling competitions. I only let you win is because I know you would be devastated when losing. Stjepan, thanks for inspiring me with the absolute massive amounts of papers you write. Although I always say I don't like you, I secretly think you are quite awesome ;). Mauro, thank you for trying to convince me to stay in academia. I will keep it in mind, let's see where life brings me.

Outside of work I realize that I also have amazing friends. I want to thank everyone from my volleyball team for the great time and for keeping me moving. Without you I would be glued to the couch. For the time that I was glued on the couch I want to thank Max, my cat, for keeping me company during all the home-working due to COVID. Hugo, Tim and Lars, I really like playing Pandemic with you guys and enjoy the good food you cook for when we are playing. But when you visit my house I will still probably just get some frozen pizzas. Everyone from TiCTaC, I really like to have a group to have good conversations with. Thanks for letting Rienke and me join the Christmas dinner three years ago.

For having a loving family I am forever grateful. It is easy for me to take this for granted, but I realize it is amazing to have a family with so much warmth. For a large part they have shaped me into the person I am, and I know they are proud of me. The thing I am most grateful for towards my parents is that they have taught me personal responsibility. Life is a matter of making choices, and where you are in life depends a great deal on the choices you make. Looking back, I believe I have made the right choices. But even if I wouldn't have, I know my family would still give me their unwavering support, which is the greatest gift of all.

Curriculum Vitæ

Harm Jonathan Griffioen

Harm Griffioen was born in Gouda, the Netherlands on March 2, 1995. He obtained his bachelor's degree Computer Science with distinction from the Technical University of Delft in 2016. During his master's, he specialized in Cyber Security and obtained this degree with distinction from the Technical University of Delft in 2018.

In 2018, he started as a Ph.D. student at the TU Delft under supervision of prof. dr. ir. R.L. Lagendijk and prof. dr. Christian Doerr. In the first two years of the Ph.D., he spent 40% of his time in teaching, obtaining part of the University Teaching Qualification. He has been involved in teaching AI, Cyber Security and Computer Networking concepts as a responsible lecturer. He also supervised multiple master students and bachelor student groups during their final projects. From April 2020, he worked at the Hasso-Plattner-Institute in Potsdam, Germany alongside the TU Delft.

During his PhD, he worked on improving the field of Cyber Threat Intelligence. For this, he has visited the Fujitsu lab in Tokyo, Japan and presented his work at multiple international conferences.

List of Publications

12. **Griffioen, H.**, Doerr, C., *Could you clean up the Internet with a Pit of Tar? Investigating tarpit feasibility on the Mirai malware*, IEEE Symposium on Security and Privacy (IEEE S&P 2023).

11. **Griffioen, H.**, Oosthoek, K., van der Knaap, P., Doerr, C., *Scan, Test, Execute: Adversarial Tactics in Amplification DDoS Attacks*, ACM SIGSAC Conference on Computer and Communications Security (ACM CCS 2021).

10. **Griffioen, H.**, Hu, H., Doerr, C., *SIP Bruteforcing in the Wild-An Assessment of Adversaries, Techniques and Tools*, IFIP Networking Conference (IFIP Networking 2021).

9. Taniguchi, T., **Griffioen, H.**, Doerr, C., *Analysis and Takeover of the Bitcoin-Coordinated Pony Malware*, ACM Asia Conference on Computer and Communications Security (ACM AsiaCCS 2021).

8. **Griffioen, H.**, Doerr, C., *Quantifying autonomous system IP churn using attack traffic of botnets*, 15th International Conference on Availability, Reliability and Security (ARES 2020).

7. **Griffioen, H.**, Doerr, C., *Examining mirai's battle over the internet of things*, ACM SIGSAC Conference on Computer and Communications Security (ACM CCS 2020).

6. **Griffioen, H.**, Doerr, C., *Quantifying TCP SYN DDoS resilience: A longitudinal study of Internet services*, IFIP Networking Conference (IFIP Networking 2020).

5. **Griffioen, H.**, Doerr, C., *Discovering Collaboration: Unveiling Slow, Distributed Scanners based on Common Header Field Patterns*, IEEE/IFIP Network Operations and Management Symposium (NOMS 2020).

4. Bouwman, X., **Griffioen, H.**, Egbers, J., Doerr, C., Klievink, B., van Eeten, M., *A different cup of TI? The added value of commercial threat intelligence*, 29th USENIX Security Symposium (USENIX Security 2020).

3. Ghiette, V., **Griffioen, H.**, Doerr, C., *Fingerprinting tooling used for SSH compromisation attempts*, 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019).

2. **Griffioen, H.**, Booij, T., Doerr, C., *Quality Evaluation of Cyber Threat Intelligence Feeds*, International Conference on Applied Cryptography and Network Security (ACNS 2019).

1. **Griffioen, H.**, Doerr, C., *Taxonomy and adversarial strategies of random subdomain attacks*, 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS 2019).