Threat Intelligence in the Early Stages of Cyberattacks

Ghiette, V.D.H.

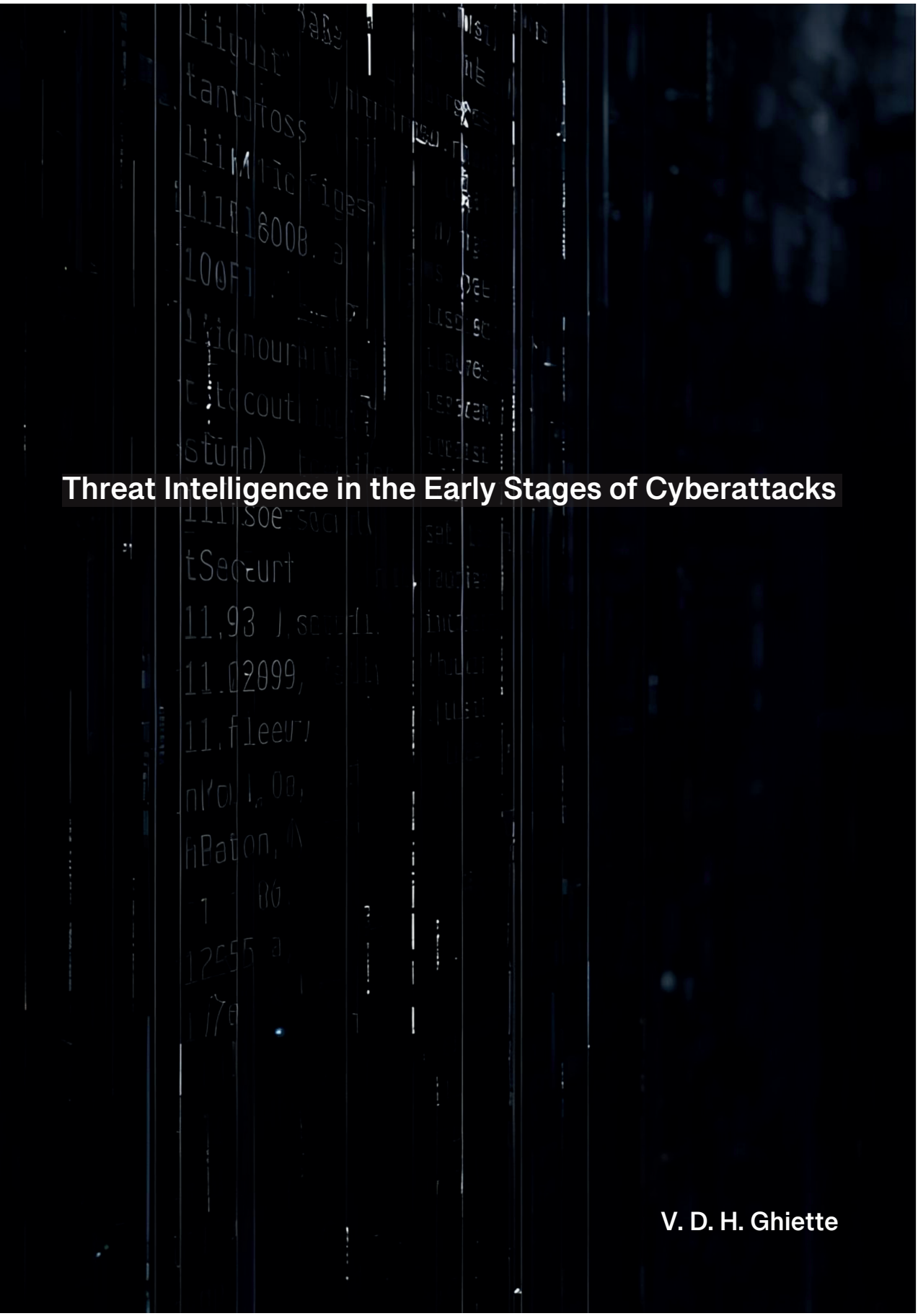
DOI
10.4233/uuid:2c570939-a884-4568-bcf4-d37942476cac

Publication date
2025

Document Version
Final published version

Citation (APA)
Ghiette, V. D. H. (2025). *Threat Intelligence in the Early Stages of Cyberattacks*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:2c570939-a884-4568-bcf4-d37942476cac

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.



Threat Intelligence in the Early Stages of Cyberattacks

V. D. H. Ghiette

Threat Intelligence in the Early Stages of Cyberattacks

THREAT INTELLIGENCE IN THE EARLY STAGES OF CYBERATTACKS

Dissertation

for the purpose of obtaining the degree of doctor,
at Delft University of Technology,
by the authority of the rector Magnificus Prof. dr. ir. T.H.J.J van der Hagen,
chair of the Board of Doctorates,
to be defended publicly on Friday 10 October 2025 at 12:30

by

Vincent GHIETTE

Master of Science in Computer Science, Delft University of Technology,
The Netherlands
born in Etterbeek, Belgium

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	Chairperson
Em. prof. dr. ir. J. van den Berg,	Delft University of Technology, *promotor*
Prof. dr. C. Doerr,	TU Delft / University of Potsdam, Germany, *promotor*
Independent members:	
Prof. dr. G. Smaragdakis,	Delft University of Technology
Prof. dr. F. A. Kuipers,	Delft University of Technology
Prof. dr. A. Sperotto,	University of Twente
Prof. dr.-ing. M. Fischer,	University of Hamburg
Dr. E. Bou-Harb,	Louisiana State University

Keywords: Cyber Threat Intelligence, Scanning, Botnets, Clustering

An electronic copy of this dissertation is available at
https://repository.tudelft.nl/.

Contents

SUMMARY

In the past years, cybercrime has become an increasing burden on society. Criminals have a rising amount of available cheap attacking resources. Additionally, we observe a growing amount of Internet-connected devices containing lots of (new) vulnerabilities and Internet services full of threats for end-users. In response, defending parties have developed cyber threat intelligence (CTI). CTI focuses on collecting, processing, and analyzing data of criminal activities to understand the actor's motives, targets, and behavior, allowing defending parties to understand the cybercrime landscape, and adapt their defense strategies accordingly.

Although CTI has been developed and generated for the past decades many challenges remain to be solved. Most of the defense systems designed using CTI utilize simple indicators such as hash values and IP addresses, and are therefore easily circumvented by cybercriminals. In addition, the cyber landscape is in continuous motion, causing intelligence to become stale, and decreasing the effectiveness of related defense mechanisms. In this dissertation, we contribute to the state of the art by investigating whether it is possible to, in the early stages of attacks, extract information characterizing the criminals toolchain and identify behavioral traits. Generating CTI at the early stage of attacks provides the support for defending parties to develop a defense mechanism able to stop criminals in the early stage of attacks preventing them from causing further damage. We focus on gathering CTI related to the tools, tactics, techniques, and procedures attackers use, as they provide the valuable information for developing defense systems.

We observe that the scan landscape has significantly changed in the past nine years and that defense mechanisms based on low-value CTI provide insufficient protection. The observed changes and analyses confirm the necessity to continuously monitor the current threat landscape and adapt the defense mechanisms accordingly. Following, we propose a clustering-based methodology matching randomized scan probe payloads allowing for the reconstruction of templates used by attackers. Using reconstructed templates we can detect large-scale scanning campaigns, and the convergence of cybercriminals using similar toolchains over time increasing the difficulty of distinguishing between criminal operations. To differentiate similar toolchains we analyzed the fingerprinting of SSH handshakes and reveal that the libraries and their versions used to compile said toolchains can be identified allowing further differentiation between campaigns and thus the behavioral analysis of cybercriminals. We showcase that attackers employing distinct toolchains at the early attack stages will behave differently at later attack stages. Finally, we analyze Mirai brute-forcing attacks identifying the different botnet configurations cybercriminals use. We simulate competing botnets configured with the observed settings and conclude that the implemented changes improve the botnet's success rate, indicating that criminals are constantly refining their operations.

In summary, we contribute to the state of the art by generating CTI characterizing the

tools, tactics, techniques, and procedures attackers use. The generated CTI spans multiple layers of the toolchains attackers use, providing actionable intelligence at the early attack stages. The generated intelligence can, therefore, contribute to the design of defense in depth systems aimed at mitigating cyberattacks.

SAMENVATTING

De afgelopen jaren is cybercriminaliteit een steeds grotere last geworden voor de maatschappij. Criminelen hebben een groeiend, steeds goedkoper wordend aantal aanvalsmiddelen tot hun beschikking. In aanvulling daarop zien we dat het aantal met Internet verbonden toestellen, met (nieuwe) beveiligingslekken en kwetsbare diensten voor de eindgebruikers, drastisch toeneemt. Als antwoord op deze ontwikkeling hebben verdedigende partijen cyber threat intelligence (CTI) ontwikkeld. CTI focust zich op het verzamelen, verwerken en analyseren van data afkomstig van criminele activiteiten en vormt zich zodoende een beeld van hun motieven, doelwitten en gedrag. Deze informatie kan vervolgens gebruikt worden door verdedigende partijen om de evoluties in het cyberlandschap te begrijpen en om adequate verdedigingsstrategieën te ontwikkelen.

Niettegenstaande CTI al decennialang ontwikkeld en gegenereerd wordt, vragen nog veel uitdagingen om een oplossing. De meeste verdedigingsmechanismen die gebaseerd zijn op CTI gebruiken simpele indicatoren zoals hash-values en IP adressen en zijn daardoor voor cybercriminelen makkelijk te omzeilen. Bovendien is het cyberlandschap continu in beweging, waardoor ingewonnen informatie snel relevantie verliest en dus de effectiviteit, van de op deze gegevens gebaseerde defensiesystemen, snel afneemt. In dit proefschrift dragen we bij aan de huidige stand van zaken door te onderzoeken of het mogelijk is om in de beginstadia van cyberaanvallen informatie te onttrekken die de toolchain van de criminelen karakteriseert en helpt de gedragskenmerken te identificeren. Het vergaren van CTI in de beginstadia van aanvallen draagt bij aan het ontwikkelen van defensie mechanismen die aanvallen in een prematuur stadium kunnen ontdekken, waardoor er snel ingegrepen kan worden en verdere schade beperkt wordt. We concentreren ons op het verzamelen van CTI met betrekking tot de tools, tactieken, technieken en procedures die criminelen gebruiken omdat ze waardevolle informatie opleveren die gebruikt kan worden voor het ontwikkelen van verdedigingssystemen en strategieën.

We observeren dat het scan-landschap de afgelopen negen jaar sterk veranderd is en dat verdedigingsmechanismen gebaseerd op simpele indicatoren geen adequate dekking bieden. De geobserveerde veranderingen, tezamen met onze uitgevoerde analyses, bevestigen de noodzaak om het cyberlandschap continu te monitoren en het belang om verdedigingsmechanismen conform aan te passen. Vervolgens stellen wij een op clustering gebaseerde methodologie voor, die gerandomiseerde scan payloads matcht, waardoor de templates die door de aanvallers worden gebruikt kunnen worden gereconstrueerd. Deze gereproduceerde templates helpen ons bij de detectie van grootschalige scan-campagnes en de convergentie van cybercriminelen die vergelijkbare toolchains gebruiken waardoor het steeds moeilijker wordt om onderscheid te maken tussen de criminele campagnes. Om alsnog onderscheid te kunnen maken tussen diverse campagnes analyseren we de vingerafdrukken van verscheidene SSH handshakes. We tonen aan dat de verschillende handshakes gekoppeld kunnen worden aan specifieke versies van

libraries die criminelen gebruiken om hun toolchains te compileren, waardoor we weer onderscheid kunnen maken tussen meerdere campagnes. Door onderscheid te maken tussen de uiteenlopende campagnes laten we zien dat criminelen die in de beginstadia van een aanval andere toolchains gebruiken, verschillend gedrag vertonen en zich in latere stadia ook anders gedragen. Tot slot analyseren we Mirai brute-force aanvallen en identificeren we variërende botnet-configuraties die criminelen gebruiken. We simuleren concurrerende botnets, met de geïdentificeerde configuratie, die het tegen elkaar opnemen en concluderen dat de gemaakte aanpassingen aan de configuratie bijdraagt aan de effectiviteit van de botnets. Dit geeft aan dat criminelen constant bezig zijn met het verbeteren van hun operaties.

Samenvattend, dragen we bij aan de huidige stand van zaken door CTI te genereren die de tools, tactieken, technieken en procedures van criminelen karakteriseert. De gegenereerde CTI beslaat meerdere lagen van de toolchains die criminelen gebruiken en biedt informatie die gebruikt kan worden in de vroege stadia van aanvallen. Tevens kan de gewonnen informatie bijdragen aan het ontwikkelen van defense-in-depth strategieën die gericht zijn op het beperken van cyberaanvallen.

1

INTRODUCTION

One of the earliest acts of cybercrime, a simple case of password theft [1], dates back to 1962 when computers were not a widespread commodity. During the 90s, we saw a leap in the adoption of home computers accompanied by the fast development of the Internet. This new era unlocked fresh opportunities for cybercriminals, and in 1994 Vladimir Levin performed the first online bank robbery siphoning a total of 10 million dollars from multiple bank accounts [2]. To date, cybercriminal activities have continued exploding. In the past decade, a three-fold increase in cybercriminal activities has been observed in the United States [3], with sources estimating the global damages exceeded 3 trillion dollars in 2021 [4]. Part of the problem is caused by the growing amount of potential resources cybercriminals have at their disposal for attack purposes. For instance, the number of Internet of Things devices has increased from 16 to 18 billion in the past years [5]. Similarly, the number of recorded vulnerabilities (CVEs) has increased from 7,928 in 2009 to 34,357 in 2024 [6], making more services vulnerable to attacks and potential abuse by cybercriminals. The increased resources available to cybercriminals and resulting damage pose extensive concerns for governments, businesses, and individuals seeking protection.

Unsurprisingly, security firms, universities, and nations responded to increased cybercrime by developing and contributing to the field of cyber threat intelligence (CTI) and translating the gained knowledge into defense systems to handle cybercrime. This dissertation contributes to the field of CTI, mainly by analyzing the early attack stages and generating actionable intelligence. In the following chapters, we present our contributions. In this chapter, we elaborate on the concept of CTI and the importance of generating actionable intelligence at the early attack stages in section 1.1. Furthermore, we provide background information on the treated topics by covering related work in section 1.2 and introduce the research question in section 1.3. Following, in section 1.4, we provide an overview of the methodology used to generate CTI. Finally, section 1.5 presents an overview of this dissertation's structure and contributions.

1

1.1. CTI AND ITS IMPORTANCE IN THE EARLY ATTACK STAGES

1.1.1. CYBER THREAT INTELLIGENCE

CTI is based on data that is collected, processed, and analyzed in order to understand a threat actor's motives, targets, and attack behaviors [7]. Stakeholders, such as defending parties, can leverage this information to implement defense systems or design infrastructure capable of resisting currently known attack vectors. CTI can be classified into three types [8]:

Strategic threat intelligence focuses on providing an overview of the threat landscape identifying threat actors, their capabilities, and their motives. This type of intelligence allows defending parties to make informed decisions while managing risks and is often used by higher management layers for developing cyber security road maps.

Tactical threat intelligence: focusses on analyzing malware, uncovering indicators of compromise, and detecting the tools, tactics, and procedures (TTP) used by attackers. This type of intelligence is used by cybersecurity teams to develop and deploy new defense strategies and systems.

Operational threat intelligence: focusses on real-time monitoring of cyber attacks and identifying the "When", Where", and "How".

To summarize, CTI provides defending parties with an overview of the threat landscape. Furthermore, CTI provides information revealing the motives of an adversary, allows for the identification of the tools tactics, and procedures (TTPs) used, and sheds light on the actor's decision-making process allowing for the implementation of defense mechanisms and other appropriate actions. Although all types of intelligence are equally valuable, in this dissertation we focus on generating tactical threat intelligence, specifically extracting actionable intelligence from attacks.

1.1.2. ACTIONABLE INTELLIGENCE

Actionable intelligence consists of specific details characterizing cybercriminals' attacks and behavior [8]. This type of intelligence is part of tactical intelligence. It comes in the form of indicators of compromise such as hash values, IP addresses, and the TTPs used by criminals during attacks. Although each form of intelligence is valuable, some have a greater impact on cybercriminals than others when translated into defense mechanisms. For instance, defending parties can use IP addresses to identify an attacker's infrastructure. However, attackers can easily circumvent such a defense strategy by deploying attacks using different IP addresses. Alternatively, using a defense mechanism able to identify the tools used by the attackers will oblige the criminal to change or even redesign the used tools to remain undetected, which is a costly endeavor. The pyramid of pain shown in figure 1.2 is often used to assess the value of gathered intelligence. The less valuable intelligence is located at the bottom while the more valuable ones are located at the top. As mentioned before, hash values and IP addresses are at the bottom due to the ease with which attackers can change them during attacks, while the TTPs and Tools are located at the top as they are more costly to replace. In this dissertation, we focus on gathering intelligence regarding the TTPs and tools used by attackers as this type of intelligence is more valuable than other types located at the bottom of the pyramid.

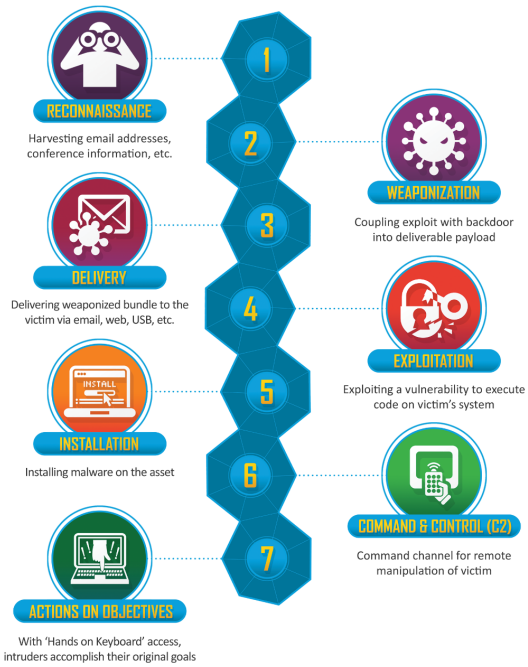


Figure 1.1: The cyber kill chain, image sourced from Lockheed Martin[9]

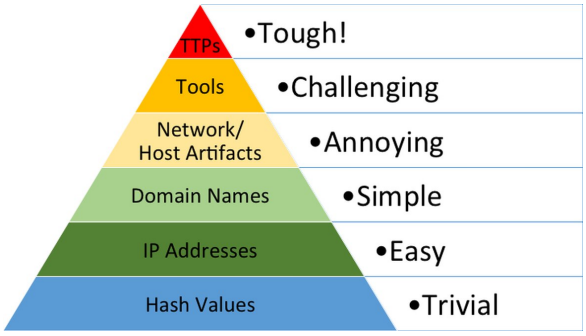


Figure 1.2: The pyramid of pain, image sourced from David Bianco[10]

1.1.3. THE IMPORTANCE OF THE EARLY ATTACK STAGES

During cyber attacks, criminals have an advantage with regard to the defending party they are targeting. Since the criminal orchestrates the attack they are in control of the sequence of events and the defending party must anticipate and react to the decisions made by the criminals. One strategy defending parties can adopt to increase the likelihood of fending off attacks is deploying defense in depth security measures [11]. Defense in depth is a strategy in which multiple defense measures are stacked, such as including a lock and deadbolts on a front door. This strategy can be applied to cybersecurity by deploying multiple defense systems aimed at detecting cybercriminal activities at different stages of the attack. Although cybercriminal attacks vastly differ, for example, a DDoS attack differs from a brute forcing attack, they can be generalized into multiple stages. The cyber kill chain developed by Lockheed Martin [9], and the more detailed unified kill chain developed by Paul Pols [12] are such abstractions that can be used to analyze the stages involved in a given attack. For the purpose of this introduction, the simpler cyber kill chain suffices, as this dissertation focuses on the first attack stages which are shared between the two models. Therefore, in the remainder, we will use its abstraction of cyber attacks shown in figure 1.1. The cyber kill chain shows 7 steps summarizing cyber attacks. An attacker starts by performing a reconnaissance, which can be scanning the Internet for potential targets, and ends with the actions and objective which could be, in the case of corporate espionage, extracting information. With a defense-in-depth strategy defending parties would set up defense systems detecting attacks at each attack stage increasing the chance of identifying and mitigating an attack as missing the detection at one stage would only result in the attack being detected in a later one. Although deploying multiple defense layers is considered a best practice, it is beneficial for defending parties to detect ongoing attacks at the early stages. When an attack is detected in the early stages it can be stopped before the attacker causes damage. The early stages such as reconnaissance and weaponization are part of the attack but do not immediately harm the defending parties as the attacker has not gained access or control of the attacked system. In this dissertation, we focus on gathering intelligence at the early stages of the attack providing actionable intelligence as detecting attacks early on is the most advantageous.

1.2. BACKGROUND INFORMATION

In the previous subsection, we argued for collecting valuable intelligence at the early stages of attacks. In this subsection, we provide background information related to the topics treated in this dissertation.

1.2.1. SCANNING

The first action taken by criminals setting up an attack is reconnaissance. During this phase, criminals often search the Internet for potential targets that they can attack or abuse during attacks. For instance, criminals wanting to gain control over devices might search for devices running the secure shell service (SSH) while attackers wanting to conduct DDoS attacks might look for domain name servers (DNS). To find these targets criminals scan the Internet. Scanning is often done by sending packets to IP addresses on the

Internet and listening for response indicating the presence of a potential target. Scanning the Internet is not novel and many researchers have analyzed the behavior of scanners revealing their behaviors. Many studies analyze the strategies used by scanning criminals, summarizing the techniques employed to find vulnerable devices on the Internet [13–15]. Besides the technique used by cyber criminals, researchers analyze the contents of the scan probes, focussing on the header values to distinguish scan probes from regular traffic and to identify different tools employed by cybercriminals [16, 17]. Other research [18] reveals that the scan probe's payload can be used to identify the tools while Bou-Harb et al. [19] use the inter-arrival times of scan probes to identify the used scan software. Besides tool identification, sudden shifts in targeted ports are recorded suggesting a changing scan landscape [20]. Next to scanning different destination ports, cybercriminals perform large distributed scan campaigns, leveraging multiple IP addresses to send scan probes from. Researchers studied scan probes' header values [21] or the overlap between scanned destinations [22] identifying distributed scan campaigns. Finally, studies find criminals, launch their attacks from specific locations. Multiple works [16, 20] show criminals launching scans from particular countries. Other studies and reports identify autonomous systems (ASs) [23] and even internet service providers (ISPs) [24] from which scans are commonly launched.

The previously discussed works all contribute to advancing the state of the art of CTI by generating intelligence concerning criminals scanning the Internet. However, due to the ever-changing cyber threat landscape, many new challenges emerge. We will further elaborate on these challenges in section 1.3.

1.2.2. BRUTE-FORCING

Once attackers have found potential targets through an Internet scan they can continue with their attack and move to the weaponization and delivery phase. In this dissertation, we analyze the behavior of attacker brute-forcing devices running the telnet or secure shell service. Brute forcing these devices is often done in two steps, first establishing a connection during which the attacker and target exchange parameters ensure proper communication. Second, the attacker repeatedly sends login credentials attempting to guess the correct one to gain access.

Attackers employ common brute-forcing strategies to avoid detection such as randomizing the credentials, randomizing the targets, and lowering the number of brute-forcing attempts per device. Besides these well-studied techniques, researchers [25, 26] have studied the credentials used during the attacks showing that different attackers share the same brute forcing dictionaries indicative of shared credential lists. Other studies show the opposite, in which attackers adopt new passwords in an attempt to stay ahead of the competition [27, 28]. Other works have utilized the characteristics of the connection such as source ports and data flows to identify brute forcing attacks [29–33] allowing for the detection of brute forcing campaings.

The previously discussed works all contribute to advancing the state of the art of CTI by generating intelligence concerning criminals brute-forcing vulnerable device the Internet. However, due to the ever-changing cyber threat landscape, many new challenges emerge. We will further elaborate on these challenges in the following section.

1

1.3. PROBLEM STATEMENT

In the previous section, we provided an introduction to the concept of CTI, the importance of valuable intelligence, and the importance of detecting attacks at the early stages. Furthermore, we supplied background information concerning the early attack stages and covered relevant works contributing to the generation of CTI. Despite the many efforts, there are still opportunities to improve the current state of the arts as most of the intelligence is based on low-value indicators such as hash values and IP addresses [34]. In addition, the response times to network breaches are long, with an average response and containment times of nine months [35], leaving criminals ample time to conduct their activities. This dissertation aims to contribute to the field of CTI by analyzing attackers' behavior at the early stages of attacks. The scope of this work is deliberately limited to the first stage of attacks, as gathering intelligence early in the lifespan of an attack can contribute to the early detection and profiling of attacks. Furthermore, this work commits to identifying valuable information, such as TTPs and behavioral characteristics. Extracting valuable information at the early stages of attacks is challenging due to the small amount of available information inherent to the beginning of an attack. More so, in recent years, criminals have adopted detection evasion strategies such as domain name generation algorithms [36], fast flux networks [37], and encryption [38] increasing the difficulty of detecting cyber criminal activities. In addition, criminals have converged to using similar tools [28], increasing the difficulty of identifying different campaigns due to the similarity of cybercriminals' set-ups. This dissertation contributes to the state of the art by answering the following research question:

Can we, in the early stages of attacks, extract information characterizing the criminals toolchain and identify behavioral traits?

1.4. METHODOLOGY

This dissertation presents multiple works analyzing scans and brute-forcing attempts. This section provides an overview of the methodology used to extract actionable intelligence from the monitored attacks.

1.4.1. METHODOLOGY RELATED TO SCANS

Scanning is often the first step attacks take in preparation for an attack. To analyze these attacks and extract actionable intelligence, we must first capture these scans. Therefore, we set up a network telescope scattered across three /16 subnets containing over 60,000 IP addresses. A network telescope uses a large space of unused routable IP addresses. To find targets attackers send scan probes to IP addresses. Due to the nature of the telescope, we are able to collect the scan probes sent by the attackers. Once collected, these scan probes are parsed and the header fields, arrival times, and payloads can be analyzed. As we are interested in finding behavioral traits of attackers we first analyze the properties of scan probes sent from a single source address. We record the speed with which probes are sent, the number of addresses they send probes to, whether they sent multiple problems to the same destination, which destination ports they target, and other behavioral characteristics. These characteristics, when collected during prolonged periods,

can indicate general trends in the scan landscape such as criminals slowing down their speeds to stay undetected, or criminals shifting from scanning for one protocol to another. These trends provide defending parties with actionable intelligence for detecting scans or provide valuable insight into which services are commonly targeted by criminals. Furthermore, clustering can be applied based on the witnessed behavior identifying large scanning campaigns. In addition, the data can be enriched by cross-referencing with other datasets. For instance, the source addresses can be looked up in IP address location databases revealing whether criminals tend to launch scans from the same geographical regions or the same regions on the Internet such as ISPs or ASs. Besides manual analysis, clustering algorithms can also be leveraged to identify behavioral aspects. Attackers are known to use templates to generate scan probe's payloads. Grouping source addresses using similar payloads can reveal additional behavioral characteristics reflecting criminals' attack strategies such as the use of large-scale scan campaigns or payload randomization.

1.4.2. METHODOLOGY RELATED TO BRUTE-FORCING ATTEMPTS

Brute forcing is a common means for attackers to gain access to and control over devices. To study the behavioral characteristics of and the tools used by attackers we must capture these brute forcing attempts. Therefore, we deployed over 4,500 SSH and Telnet honeypots. Honeypots are, in our case, virtual devices simulating commonly found devices on the Internet running a particular service. When an attacker connects to such a device all the interactions are logged and stored for further analysis.

Once collected, the connection settings are parsed, and the information exchanged between the attacker and the honeypot are saved. Attackers use different libraries to compile their brute-forcing tools. These libraries dictate the settings used while establishing a connection. By analyzing the used settings in attacks fingerprints of different toolchains can be made. These settings can then be used to identify large-scale brute forcing campaigns and can later be correlated with the used brute forcing credential to distinguish between the different behaviors of attackers providing actionable intelligence. Furthermore, the exchange settings can be linked to different libraries and event versions providing an overview of the tools used by attackers to compile their toolchains.

Analyzing the credentials used while brute forcing can also provide actionable intelligence. Monitoring the used credentials over a period of time can reveal that criminals are trying out new credentials most likely in an attempt to gain control over new devices. Other metrics can be collected concerning the credential usage such as the frequency with and the order in which credentials are used. This information itself can be added to rule-based intrusion detection systems. Furthermore, these metrics can also be used, together with time correlation, to identify large brute forcing campaigns contributing to the study of criminals' behavior.

1.5. CONTRIBUTIONS AND OUTLINE

Cybercriminals constantly adopt new techniques and methods to evade detection and reach their goals. In the past decade, we saw the rise of multiple botnets such as Mirai,

Emotet, and Mantis, all evolving and adopting new attack vectors exploiting newly discovered vulnerabilities [27, 39]. Besides botmasters, other cybercriminals adopt new attack methods, such as using newly discovered vulnerabilities to install ransomware or new forms of spam campaigns [40, 41]. The evolution conducted by cybercriminals drives the necessity to continuously update CTI to keep up with the changing landscape. New strategies capable of responding to these new threats must be developed to counter them. The first step toward building defenses is extracting valuable intelligence at the early stages of attacks, as explained in section 1.1, as it provides the required information to build defense mechanisms. This dissertation contributes to filling the CTI gap created by the continuous evolution of cybercrime. More specifically, intelligence is gathered by analyzing the behavioral differences and TTPs used at the early stages of attacks. Actual attack data is analyzed, showing that attackers utilize different setups when conducting similar attacks. These behavioral differences allow the characterization and differentiation of attacks. Chapters 2, 3 and 4 contain studies analyzing the output of attackers' toolchains. Chapter 5 reveals that criminals use different libraries in their toolchains. Chapter 6 shows that a distinction between identical malware attacks can be made due to criminals using different configurations. The different chapters in this dissertation show that analyzing data from the early stages of attacks can reveal the changes attackers make at different levels of their toolchains. Identifying these changes allows for discerning different attacks, contributing to better behavioral analysis. The following paragraph enumerates the individual contributions made in each chapter.

Chapter 2 examines the behavioral changes of UDP scanners over the past nine years. Scanning is often the first step toward an attack and is well-studied. However, most studies specifically analyze TCP scanning as UDP scans only account for a small part of the scan traffic. Nevertheless, protocols running on top of the UDP stack are also vulnerable [42] and are an excellent choice for setting up DDoS attacks due to their high amplification factor and their connectionless nature [43]. Monitoring incoming scan traffic can reveal whether attackers are taking advantage of the ever-increasing broadband speeds or, on the contrary, are slowing down their scan speed to avoid detection. Furthermore, monitoring the scanned destination ports can reveal a shift in the targetted protocols over recent years.

Chapter 3 continues the study of UDP scan traffic. Scan traffic around the time of the Memcached vulnerability is studied. Studying the changes surrounding a disclosed vulnerability can reveal how attackers react to new attack vectors. Scan probes criminals use are linked to published proofs of concepts revealing whether criminals develop their tools or copy ones found online.

Chapter 4 presents a methodology to reconstruct attackers' UDP scan probe production process. Using the proposed approach on scan traffic reveals that attackers use randomization to evade detection. Furthermore, applying the algorithm to real scan traffic allows the identification of large scan campaigns.

Chapter 5 focuses on extracting the libraries used by attackers using little information at the start of attacks. Using the SSH handshake of brute-forcing attempts, the libraries used by criminals to build their tools can be identified. Differentiating between used libraries allows the identification of specific toolchains, which are then linked to large-scale brute-forcing campaigns.

Chapter 6 goes one layer deeper into the toolchains used by cybercriminals and studies attackers using identical tools, with slight modifications. Login attempts of Mirai botnets are studied, showing criminals make modifications to existing software to suit their needs. These modifications can be discerned at the early stages of the attack prior to the targetted device being compromised. In addition, the identified changes are assessed by simulation competing botnets, revealing the advantage of certain configurations over others.

Finally chapter 7 summarizes the results of the chapters, answers the research question, and provides an insight into potential future work.

References

[1] R. Mcmillan. 'The World's First Computer Password? It Was Useless Too'. In: *Wired* (27th Jan. 2012). URL: https://www.wired.com/2012/01/computer-password/.

[2] FBI. *A Byte Out of History*. 31st Jan. 2014. URL: https://www.fbi.gov/news/stories/a-byte-out-of-history-10-million-hack.

[3] Beyond Identity. *How Has a Decade of Cybercrime Impacted the United States?* 30th Aug. 2021. URL: https://www.beyondidentity.com/blog/rise-cybercrime-study.

[4] S. Morgan. 'Global Cybercrime Damages Predicted To Reach $6 Trillion Annually By 2021'. In: *Cybersecurity Ventures Official Annual Cybercrime Report* (26th Oct. 2020). URL: https://cybersecurityventures.com/annual-cybercrime-report-2020.

[5] S. Sinha. *State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally*. URL: https://iot-analytics.com/number-connected-iot-devices/.

[6] T. M. Corporation. *Published CVE Records*. URL: https://www.cve.org/about/Metrics.

[7] K. Baker. *What is Cyber Threat Intelligence?* URL: https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/.

[8] Paloalto Networks. *What are the Types of Cyberthreat Intelligence (CTI)?* URL: https://www.paloaltonetworks.com/cyberpedia/types-of-cyberthreat-intelligence.

[9] Lockheed Martin. *Cyber Kill Chain*. 26th Oct. 2020. URL: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.

[10] D. J. Bianco. 'The Pyramid of Pain'. In: *SANS* (17th Jan. 2014). URL: http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html.

[11] Fortinet. *What Is Defense In Depth?* URL: https://www.fortinet.com/resources/cyberglossary/defense-in-depth.

[12] P. Pols and J. van den Berg. 'The unified kill chain'. In: *CSA Thesis, Hague* (2017).

[13] M. de Vivo, E. Carrasco, G. Isern and G. O. de Vivo. 'A Review of Port Scanning Techniques'. In: *ACM SIGCOMM Computer Communication Review* (1999). DOI: 10.1145/505733.505737.

[14] M. H. Bhuyan, D. Bhattacharyya and J. Kalita. 'Surveying Port Scans and Their Detection Methodologies'. In: *The Computer Journal* (2011). DOI: 10.1093/comjnl/bxr035.

[15] E. Bou-Harb, M. Debbabi and C. Assi. 'Cyber Scanning: A Comprehensive Survey'. In: *IEEE Communications Surveys and Tutorials* (2014). DOI: 10.1109/SURV.2013.102913.00020.

[16] Z. Durumeric, M. Bailey and J. A. Halderman. 'An Internet-Wide View of Internet-Wide Scanning'. In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. USA, 2014.

[17] V. Ghiette, N. Blenn and C. Doerr. 'Remote Identification of Port Scan Toolchains'. In: *8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 2016. DOI: 10.1109/NTMS.2016.7792471.

[18] H. Griffioen, H. Hu and C. Doerr. 'SIP Bruteforcing in the Wild - An Assessment of Adversaries, Techniques and Tools'. In: *IFIP Networking Conference (IFIP Networking)*. 2021. DOI: 10.23919/IFIPNetworking52078.2021.9472857.

[19] E. Bou-Harb, M. Debbabi and C. Assi. 'A systematic approach for detecting and clustering distributed cyber scanning'. In: *Computer Networks* (2013). DOI: https://doi.org/10.1016/j.comnet.2013.09.008.

[20] 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. DOI: 10.1145/3196494.3196537.

[21] H. Griffioen and C. Doerr. 'Discovering Collaboration: Unveiling Slow, Distributed Scanners based on Common Header Field Patterns'. In: *IEEE/IFIP Network Operations and Management Symposium*. 2020. DOI: 10.1109/NOMS47738.2020.9110444.

[22] C. Gates. 'Coordinated Scan Detection.' In: *NDSS*. 2009.

[23] C. A. Shue, A. J. Kalafut and M. Gupta. 'Abnormally Malicious Autonomous Systems and Their Internet Connectivity'. In: *IEEE/ACM Transactions on Networking* (2012). DOI: 10.1109/TNET.2011.2157699.

[24] M. Goncharov. 'Criminal hideouts for lease: Bulletproof hosting services'. In: *Forward-Looking Threat Research (FTR) Team, A TrendLabsSM Research Paper* (2015).

[25] T. Barron and N. Nikiforakis. 'Picky attackers: Quantifying the role of system properties on intruder behavior'. In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM. 2017, pp. 387–398.

[26] S. Udhani, A. Withers and M. Bashir. 'Human vs Bots: Detecting Human Attacks in a Honeypot Environment'. In: *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*. 2019, pp. 1–6.

[27] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, Z. Durumeric, J. Cochran, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.* 'Understanding the Mirai botnet'. In: *Proceedings of the 26th USENIX Conference on Security Symposium*. 2017.

[28] H. Griffioen and C. Doerr. 'Examining Mirai's battle over the Internet of things'. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020.

[29] K. Hynek, T. Bene, T. ejka and H. Kubátová. 'Refined detection of SSH brute-force attackers using machine learning'. In: *ICT Systems Security and Privacy Protection*. 2020.

[30] S. K. Wanjau, G. M. Wambugu and G. N. Kamau. 'SSH-brute force attack detection model based on deep learning'. In: *International Journal of Computer Applications Technology and Research* (2021).

[31] M. D. Hossain, H. Ochiai, F. Doudou and Y. Kadobayashi. 'SSH and FTP brute-force attacks detection in computer networks: LSTM and machine learning approaches'. In: *2020 5th international conference on computer and communication systems (ICCCS)*. 2020.

[32] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre and A. Pras. 'SSHCure: a flow-based SSH intrusion detection system'. In: *Conference on Autonomous Infrastructure, Management and Security*. 2012.

[33] M. M. Najafabadi, T. M. Khoshgoftaar, C. Kemp, N. Seliya and R. Zuech. 'Machine learning for detecting brute force attacks at the network level'. In: *International Conference on Bioinformatics and Bioengineering*. 2014.

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

[35] Ponemon Institute and IBM Security. *What is the Cost of a Data Breach in 2022?* 1st Jan. 2023. URL: https://www.ibm.com/reports/data-breach.

[36] H. Suryotrisongko, Y. Musashi, A. Tsuneda and K. Sugitani. 'Robust botnet DGA detection: Blending XAI and OSINT for cyber threat intelligence sharing'. In: *IEEE Access* (2022).

[37] A. Al-Nawasrah, A. A. Almomani, S. Atawneh and M. Alauthman. 'A survey of fast flux botnet detection with fast flux cloud computing'. In: *International Journal of Cloud Applications and Computing (IJCAC)* (2020).

[38] K. Keshkeh, A. Jantan, K. Alieyan and U. M. Gana. 'A review on TLS encryption malware detection: TLS features, machine learning usage, and future directions'. In: *Advances in Cyber Security: Third International Conference, ACeS 2021, Penang, Malaysia, August 24–25, 2021, Revised Selected Papers 3*. Springer. 2021.

[39] T. B. R. I. Team. *Emotet Returns With New Methods of Evasion*. 23rd Jan. 2023. URL: https://blogs.blackberry.com/en/2023/01/emotet-returns-with-new-methods-of-evasion.

[40] S. Gatlan. 'Microsoft patches Windows zero-day used to drop ransomware'. In: *Bleeping Computer* (14th Dec. 2022). URL: https://www.bleepingcomputer.com/news/security/microsoft-patches-windows-zero-day-used-to-drop-ransomware/.

[41] M. T. I. Center. *New sophisticated email-based attack from NOBELIUM*. 27th May 2021. URL: https://www.microsoft.com/en-us/security/blog/2021/05/27/new-sophisticated-email-based-attack-from-nobelium/.

[42] MITRE. *CVE-2022-26143*. 30th Nov. 2022. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-26143.

[43] C. Rossow. 'Amplification Hell: Revisiting Network Protocols for DDoS Abuse'. In: *NDSS*. 2014.

2

UDP SCANNING AND THE LIMITATIONS OF BEHAVIORAL AND BLOCKLIST BASED COUNTERMEASURES

In this dissertation's introductory chapter, we argue for the acquisition of cyber threat intelligence characterizing cybercriminal activities at the early stages of attacks. We emphasize the nature of the gathered intelligence, namely, the importance of belonging to the top of the pyramid of pain, maximizing its potential effectiveness when translated into defense mechanisms. The following chapters of this dissertation explore the possibility of gathering information characterizing a criminal's toolchain and identifying behavioral traits in the early stages of the attack.

In this chapter, we present work studying the UDP scan landscape. Scanning is often the first step cybercriminals take when setting their attack. Criminals perform scans to find devices that can be directly attacked, such as devices with standard login credentials. Alternatively, criminals scan for devices that they leverage during attacks such as DNS servers. Studying scans reveals how criminals prepare for their attacks. In the following work, we study the behavioral characteristics of scanners and assess the effectiveness of commonly used detection strategies. Our work shows that scan characteristics belonging to the bottom of the pyramid of pain are ineffective in detecting scans when translated into defense mechanisms. For instance, we analyze the effectiveness of several blocklist types and conclude that they do not provide adequate protection against scans and that their effectiveness varies. We also show that the behavior of scanners has changed in recent years, rendering several detection methods based on probe counting ineffective. The findings presented in the work support the notion that using intelligence belonging to the bottom of the pyramid of pain is ineffective and reveals the necessity of collecting intelligence belonging to the top of the pyramid. The subsequent chapters focus on identifying and extracting additional behavioral characteristics of cybercriminals.

2.1. INTRODUCTION

Network scanning is often the first step taken by cybercriminals when setting up an attack. Scanning the Internet allows malicious actors to locate vulnerable devices that can

be compromised or abused. For example, criminals can discover devices such as routers vulnerable to remote code execution or servers running services, which can facilitate DDoS amplification attacks. Therefore, network operators benefit from detecting or, better yet, preventing scan probes from reaching their networks, reducing the probability of attackers compromising or abusing their devices.

Many solutions exist to detect and block scan traffic. Simple tools such as IP blocklists or similar forms of cyber intelligence allow network operators to identify and block scan traffic based on information provided by third parties. Similarly, intrusion detection systems allow network operators to define rules labeling network flows as potentially malicious, hence dropping incoming scan traffic. Other more complex solutions exist in which the behavior of the scanners or their generated traffic is analyzed, allowing monitoring parties to detect ongoing scans.

In the past years, new tools have emerged allowing criminals to scan the Internet within an hour. Tools also allow the distribution of scans, sending probes from different addresses on the Internet. Botnets changed the scan landscape, allowing criminals to launch large, coordinated, distributed scan campaigns. Alternatively, each bot in a botnet acts as an independent entity scanning the Internet, as seen with the Mirai botnet. Therefore, the developed scan detection solutions' effectiveness must be assessed against the changing scan landscape.

Previous work has primarily focussed on detecting and developing new methods for identifying TCP scan traffic [1, 2]. Although TCP scans generate the most traffic, UDP scans can offer valuable information to cybercriminals. Many UDP services can be abused to set up DDoS attacks, and services running on top of UDP can be triggered to execute arbitrary code using a single packet [3]. With most of the state-of-the-art focusing on developing new scan detection strategies, we assess whether simple solutions are still effective in detecting UDP scan traffic. Using a dataset spanning eight years, we analyze the scan traffic, studying the changes in the scan landscape and the effectiveness of scan detection mechanisms.

In this work, we present the following contributions:

- We study the effectiveness of IP address based blocklists and show that despite being 77.17% effective in blocking scan traffic, the effectiveness varies depending on factors such as time and destination ports scanned.
- We study the effectiveness of payload-based blocklists and show that, in general, they are 94.92% effective but likewise suffer from varying degrees of effectiveness due to factors such as the targeted destination port.
- We analyze the behavior of scanners and show a trend of scans becoming longer, sending fewer probes, and increasing the distance between scanned destinations, making it harder for simple rules used by IDSs to identify scans.
- We study the origin of scans, showing that the locations from which scans are launched can contribute to the detection of scan traffic.

2.2. RELATED WORK

Scanning the Internet is often the first step cybercriminals take when setting up an attack [4]. Criminals commonly search the Internet for vulnerable devices that they can take over [5, 6] or for servers they can abuse when launching different attacks [7, 8]. To avoid falling victim to such attacks, defending parties, research institutions, and security companies have studied the behavior exhibited by scanning cybercriminals to improve the identification of scans and update their defense mechanisms.

Many studies analyze the strategies used by scanning criminals, summarizing the techniques employed to find vulnerable devices on the Internet [9–11]. Besides the technique used by cyber criminals, researchers analyze the contents of the scan probes, focusing on the header values to distinguish scan probes from regular traffic and to identify different tools employed by cybercriminals. Durumeric et al. [12] find that the header values of incoming packets can be leveraged to identify scan probes. Ghiette et al. [13] show that the header values can be used to differentiate between the tools used by criminals to send probes. Other research [14] reveals that the scan probe's payload can be used to identify the tools while Bou-Harb et al. [1] use the inter arrival times of scan probes to identify the used scan software.

Besides tool identification, sudden shifts in targeted ports are recorded suggesting a changing scan landscape. Ghiette et al. [8] showcase an increase in scanning activity towards the Memcached standard port after the disclosure of a vulnerability concerning the protocol. Similarly, Heo and Shin [15] note a sudden increase in Telnet scan activity.

Next to scanning different destination ports, cybercriminals perform large distributed scan campaigns, leveraging multiple IP addresses to send scan probes from. Researchers studied scan probes' header values [16] or the overlap between scanned destinations [17] identifying distributed scan campaigns. Alternatively, Ghiette and Doerr [18] use the payloads of UDP scan probes to identify distributed scan campaigns.

Finally studies find criminals, launch their attacks from specific locations. Multiple works [12, 15] show criminals launching scans from particular countries. Other studies and reports identify autonomous systems [19] and even ISPs [20] from which scans are commonly launched.

The referenced works above show that header values, payloads, and behavioral characteristics can be used to identify scan probes. The works also suggest that the scan landscape is continuously evolving due to criminals targeting different ports, launching attacks from different locations and employing, and employing diverging scan strategies. The presented works often use small datasets containing short captures of scan traffic sent to C class networks. Therefore, in the following work we analyze the scan traffic sent towards three class B network for the duration of 8 years. Our dataset allows the analysis of commonly used defense strategies and their resilience against the changing scan landscape.

2.3. CAPTURING UDP SCANS

This work focuses on assessing the effectiveness of scan detection methods and monitoring the behavioral changes of cybercriminals' scan behavior. Therefore, we must capture the probes sent by criminals scanning the Internet. When scanners probe the Internet,

they do not know which IP addresses are in use and will send scan probes to IP addresses to validate whether a machine is assigned to the address and is running a specific service. A network telescope collects all incoming traffic destined towards unassigned IP addresses and can, therefore, gather scan traffic. We collect UDP traffic using a network telescope [21] spanning three partially used /16 subnets containing over 60,000 unallocated IP addresses daily. Traffic is collected for one month each year, starting in 2015 and ending in 2022.

Although network telescopes are commonly used to collect scan traffic, they also collect non-scan traffic. Traffic from misconfigured machines or prematurely closed connections can end up in the dataset gathered by the telescope. For instance, an administrator might fill in the wrong IP address for a DNS server, causing devices to send DNS requests to an unused IP address in the telescope range. Another example is a user in the telescope range closing his laptop while using a torrent client, causing the connected peers to keep sending packets to a now unallocated IP address in the telescope's range. Therefore, we must sanitize the dataset collected by the telescope by removing all non-scanning traffic from it.

Unlike TCP, where each connection starts with a TCP SYN packet, services running on top of the UDP protocol have the liberty to specify the structure and contents of the initial packet sent by the client. With 5599 registered [22] services running on top of UDP with their respective implementation, UDP scan probes can take many forms. In addition, some protocols, such as Chargen and DNS, allow the initial packet to contain arbitrary payloads. The Chargen protocol responds to any UDP packet regardless of its content, while the DNS protocol will respond to a query containing any valid domain name. The connectionless nature of the UDP protocol, combined with the different implementations of the initial packet requirements of the services, makes it impossible to enumerate all valid scan probes. Therefore, identifying scan probes based on their content is unfeasible for all services running on top of UDP.

Our proposed solution identifies scan probes based on the properties of data streams. We identify scan probes as belonging to a stream of packets sent from a single IP address toward at least ten destination IPs in the telescope. Setting a threshold of ten destinations contacted ensures that no streams caused by misconfigured machines or abruptly closed connections are identified as scans. In addition, we define a scan as terminated when no more packets are received within six hours of the last one.

Applying our proposed methodology to identify scans on the collected data, we identify 2,777,634 scans launched from 857,707 source IP addresses sending 3,469,807,527 packets. Table 2.1 shows statistics of the collected data presenting simple insights into the behavioral evolution of scanners. For instance, the fluctuation in received scan traffic per destination port and the diminishing percentage of source IP addresses used to scan the same ports.

2.4. MEASURING BLOCKLISTS' EFFECTIVENESS

Blocklists are low-effort instruments defending parties use to configure intrusion detection systems and firewalls. They allow the identification and blocking of malicious traffic, preventing the compromization of devices. Due to their simplicity, they are commonly

Table 2.1: Table summarizing the characteristics of scan traffic received in the past eight years

		2015	2016	2017	2018	2019	2020	2021	2022
Packets per day		6,887,994	11,445,190	13,351,562	8,893,830	12,363,338	22,429,773	32,240,995	19,403,393
Scans per day		3,065	9,294	4,369	1,024	3,415	26,333	31,464	19,480
Source IP addresses per day		2,014	8,263	3,616	474	938	20,340	22,784	10,882
Top ports by packets	1	24209 (19.07%)	53413 (28.93%)	1900 (41.64%)	5060 (41.68%)	5060 (22.62%)	5060 (15.58%)	123 (43.43%)	5060 (17.34%)
	2	123 (16.39%)	5060 (19.89%)	5060 (18.65%)	1900 (12.91%)	123 (7.02%)	123 (10.39%)	5060 (12.42%)	123 (10.49%)
	3	5060 (13.12%)	123 (8.22%)	53413 (9.24%)	123 (9.90%)	1900 (5.12%)	1900 (7.25%)	389 (5.68%)	53 (5.60%)
	4	1900 (12.57%)	53 (4.25%)	123 (6.33%)	389 (4.66%)	389 (4.85%)	389 (5.12%)	1900 (4.02%)	1900 (4.53%)
	5	53 (7.64%)	1900 (3.37%)	53 (2.72%)	53 (3.18%)	53 (4.43%)	53413 (3.64%)	53 (3.09%)	389 (2.95%)
Top ports by source IP addresses	1	80 (78.97%)	53413 (97.34%)	53413 (45.55%)	3544 (21.76%)	5353 (28.95%)	59929 (8.57%)	49296 (14.26%)	1900 (8.59%)
	2	30840 (40.90%)	3544 (1.22%)	1900 (42.28%)	123 (19.77%)	53 (24.95%)	52383 (8.57%)	64250 (7.78%)	59800 (8.22%)
	3	9999 (5.00%)	123 (0.30%)	3544 (5.89%)	53 (15.38%)	5060 (24.54%)	53547 (8.50%)	64484 (7.63%)	5353 (8.14%)
	4	443 (3.95%)	9999 (0.26%)	53 (1.53%)	1900 (14.55%)	1900 (21.41%)	53014 (8.44%)	49510 (7.62%)	57503 (8.14%)
	5	7777 (3.50%)	47808 (0.19%)	123 (1.51%)	5060 (10.85%)	11211 (21.18%)	62914 (8.39%)	51854 (7.61%)	5060 (8.13%)
Top ports by scans	1	80 (40.13%)	53413 (85.76%)	1900 (38.76%)	123 (14.70%)	53 (6.31%)	59929 (2.69%)	49296 (4.48%)	1900 (3.98%)
	2	30840 (20.45%)	123 (1.35%)	53413 (33.75%)	53 (6.24%)	5353 (5.88%)	52383 (2.69%)	5060 (2.55%)	5060 (3.98%)
	3	123 (9.41%)	3544 (1.09%)	123 (6.56%)	5060 (5.43%)	123 (3.72%)	53547 (2.66%)	5353 (2.35%)	5353 (3.90%)
	4	47808 (7.85%)	47808 (0.60%)	47808 (5.64%)	1900 (5.22%)	5060 (3.69%)	53014 (2.64%)	64250 (2.30%)	6881 (3.44%)
	5	443 (2.51%)	53 (0.35%)	3544 (2.88%)	5353 (3.98%)	5632 (2.75%)	61212 (2.63%)	64484 (2.26%)	11211 (3.34%)

deployed and their usage is considered a best practice. Therefore, in the following section, we analyze the effectiveness of both IP address and payload-based blocklists.

2.4.1. IP ADDRESS BLOCKLISTS

IP address blocklists contain IP addresses associated with malicious activities. When traffic is received from a registered IP address, firewalls or intrusion detection systems can drop the received traffic, stopping the attack. Many commercial threat intelligence providers promote blocklists, and public providers such as FireHol agglomerate 217 lists from various sources [23]. Although blocklist usage is common, their effectiveness is debated and dictated by several factors and a combination thereof. IP address re-usage by criminals scanning the Internet will influence the effectiveness of blocklists. Similarly, the time between consecutive scans launched from the same source IP addresses influences the blocklist's effectiveness. Furthermore, the effectiveness of blocklists is influenced by the speed with which the lists are updated. Other concerns are the varying effectiveness of blocklists, their tendency to grow in size, and potential staleness. The following section analyzes blocklists' effectiveness and addresses the aforementioned concerns.

BLOCKLIST EFFECTIVENESS

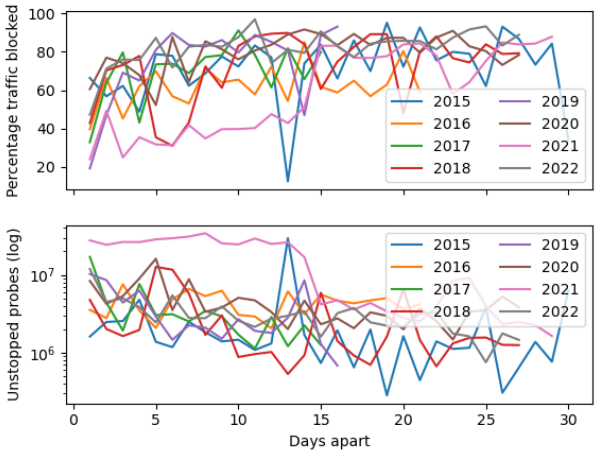


Figure 2.1: Effectiveness of an IP address blocklist

We can measure the effectiveness of defense systems using IP blocklists by blocking traffic sent from IP addresses previously recorded in the telescope dataset. For this analysis, we block traffic from a source IP one day after its first appearance in the dataset. A one-day delay seems short given Griffioen et al. [24] show that the fastest inclusion time of malicious IP addresses is two days. However, later in this section, we address delaying the addition of IP addresses to blocklists and analyze its effect on efficiency.

Figure 2.1 shows the percentage of stopped scan traffic and the number of scan probes unaffected by the blocklist. On average, between 59.45% and 82.64% of all traffic is blocked each year. The figure illustrates the variation of the blocklists' effectiveness between different years. More so, the varying effectiveness between days falling within the same month is evident, with the standard deviation of daily blocked traffic varying between 9.66% and 22.38%. Later in this section, we show that a heterogenous scan landscape causes the variation.

Although the effectiveness of defense mechanisms using blocklists varies, we see a trend in the data showing that blocklists perform better over time. Figure 2.1 shows a steady increase in the percentage of stopped traffic. On average, after the first day, 41.59% of the traffic is blocked, whereas at the end of the collection period, this percentage has risen to 79.83%. The rise in the effectiveness of blocklists is expected, as scan traffic from criminals performing multiples using the same infrastructure is blocked after the first scan.

Using blocklists can be an effective method to block scan traffic, stopping on average 77.17% of all received scan probes. However, figure 2.1 shows that on average, on a daily basis, 4,326,893 scan probes are unaffected by the blocklist. We show in section 2.5.1 that 95% of the scan traffic originating from a single source IP is directed towards unique destination addresses. If we extrapolate these numbers, 72 scans covering our entire telescope will remain undetected daily despite using blocklists. Therefore, blocklists should not be considered as an effective means to protect devices in a network against scan traffic.

IP ADDRESS RE-USAGE AND SENT TRAFFIC

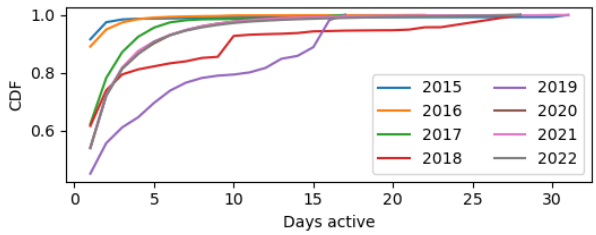


Figure 2.2: CDF representing the number of days a source IP address is used for sending scan traffic

The effectiveness of IP address blocklists depends on scanners reusing IP addresses when launching multiple scans and on the number of probes sent from addresses used during multiple days. Therefore, to explain blocklists' effectiveness, we first analyze the number of days an IP address is used to launch scans from toward our telescope. Figure 2.2 shows the cumulative distribution function of the number of days an IP address was used to send scan probes to the telescope. The figure clearly shows that most of the IP addresses used to launch scans, on average 63.89%, are only active for one day. The high amount of blocked scan traffic combined with the low amount of multi-day active addresses indicates that scanners reusing addresses send more traffic per source address

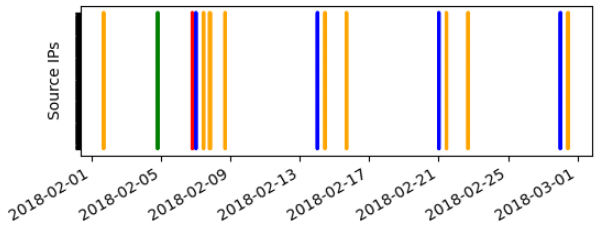


Figure 2.3: Example of coordinated distributed scan targeting several ports over the span of a month

than others. Besides most IP addresses being used for a single day, the percentage varies from year to year, fluctuating between 45.01% and 91.57%. The fluctuating amount of addresses active for a single day partially explains the varying effectiveness of IP address-based blocklists measured in the previous section.

The second factor influencing IP address-based blocklists is the percentage of scan traffic sent from single and multi-day active addresses. In 2015, 30.26% of scan traffic was sent from 90.57% of the IP addresses that were active for only a single day. In 2018, these percentages were 16.10% and 61.55%, respectively. The amount of traffic sent from single-day active IP addresses varies between 14.16% and 35.26% for the different measured years. Furthermore, there is no strong correlation between the percentage of traffic sent by single-day active addresses and the percentage of single-day active IP addresses, with a Pearson correlation coefficient of 0.36. The varying amount of traffic sent from single-day active addresses further adds to the varying effectiveness of blocklists. The fluctuating amount of traffic sent from addresses active for multiple days combined with the weak correlation between traffic sent and time active contributes to the varying effectiveness of blocklists.

Despite stopping an average of 77,17% of the received scan probes, blocklists' effectiveness depends on the tactics used by the scanning entities. A blocklist will not stop a scanner from launching a scan from a single address for a single day. On the other hand, repeated scan campaigns can increase a blocklist's effectiveness. Figure 2.3 is a scatterplot visualizing a distributed scan campaign using 244 IP addresses in 2018. The y-axis denotes the source IP addresses from which scan probes are sent. The x-axis represents the time a scan took place and the color of the scanned destination port. The visualization shows that 15 distributed scans were launched during ten days. Such scan campaigns can influence the effectiveness of blocklists, such as seen in figure 2.1, where the blocklist's effectiveness varies between days in 2018. The sensitivity of blocklists varies according to the tactics used by scanners and the heterogenous scan landscape resulting in inconsistent effectiveness of blocklists on both a yearly and daily scale. Therefore, we can conclude that the effectiveness of blocklists is not stable and varies with time, offering different degrees of protection according to the current scan landscape.

VARYING EFFECTIVENESS

We demonstrated that blocklists are 77% effective with a considerable amount of variation on both a daily and yearly basis. The varying effectiveness raises the question of whether blocklists' effectiveness varies for different types of scan traffic. In the following subsection, we analyze the blocklists' varying effectiveness for traffic targeting different destination ports and traffic with different countries of origin.

Table 2.2: Percentage of traffic blocked per destination port

	2015	2016	2017	2018	2019	2020	2021	2022
123	80.29	69.45	66.65	61.21	59.00	65.87	17.94	61.05
5060	67.17	75.08	79.79	50.54	78.66	77.78	81.81	86.16
1900	71.22	64.82	55.83	72.56	62.74	68.29	67.43	78.00
53413	86.48	28.06	48.96	68.62	46.08	87.86	94.44	40.66
53	70.24	72.76	63.57	75.67	55.20	61.80	71.02	73.94

Destination ports Network operators using blocklists to protect machines running certain services benefit from knowing whether blocklists' effectiveness varies per destination port, as services often run on designated ports. Therefore, we analyze blocklists' effectiveness for the five ports receiving the most traffic in the past eight years. Table 2.2 shows the yearly percentage of blocked traffic for those ports. The table shows that the percentage of blocked traffic is not homogenous between destination ports. In 2015 19.31% more traffic was blocked towards port 53413 than port 5060, and in 2019 the difference was 32,58%. Clearly, blocklists have varying degrees of effectiveness in blocking traffic toward different ports. The variation of blocklist effectiveness also varies for the same port in different years. In 2015 86.48% of traffic towards port 53413 was blocked, which is reduced to 28.06% in 2016. The standard deviation of blocked traffic towards a specific port varies between 2.44 and 25.04, showing that blocklists' efficiency varies yearly for the same destination ports. Therefore, blocklists do not guarantee the same effectiveness for network operators offering different services, nor do they guarantee constant effectiveness.

Table 2.3: Percentage of traffic blocked per country of origin

	2015	2016	2017	2018	2019	2020	2021	2022
US	76.83	80.12	81.59	81.34	73.73	80.55	78.08	85.53
KR	25.23	60.85	55.48	69.37	13.27	10.79	9.63	20.23
NL	53.03	53.33	88.16	73.48	76.04	91.13	83.11	91.29
CN	26.94	23.42	69.58	72.24	64.90	58.51	44.45	58.17
FR	75.97	72.70	60.09	48.36	72.81	80.03	77.34	88.06

Scan traffic origin Wan et al. [25] show that launching scans from select geographical regions can improve the accuracy of a scan. Furthermore, Durumeric et al. [12] show

that most scan traffic originates from a few countries. Although the motives behind criminals selecting one country over another remain unclear, blocklist effectiveness should remain constant despite the origin of the scan traffic. Table 2.3 shows the percentage of scan traffic identified by blocklists for the top 5 countries from which the most scan probes were sent. The table shows a difference between the percentages of blocked traffic for countries. The most notable is South Korea (KR), for which blocklists are ineffective, blocking on average only 33.10% of the scan probes sent from that country. Similarly, scan traffic originating from China, the country responsible for most of the scan traffic [12], also has a lower chance of being detected by blocklists. For other countries such as the Netherlands and France, scan traffic is more likely to be detected by blocklists. The amount of stopped traffic using blocklists varies depending on the country of origin, showing that blocklists have an inconsistent degree of effectiveness depending on the origin of the scan traffic. Therefore, we can conclude that the effectiveness of blocklists is, yet again, affected by additional parameters of the scanning infrastructure, leading to unpredictable effectiveness in the field.

TIME BETWEEN IP ADDRESS RE-USAGE

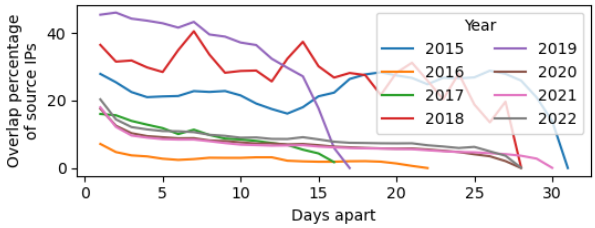


Figure 2.4: Percentage of recurring IP addresses after n days

The time between IP address re-usage can influence the effectiveness of blocklists. A blocklist's effectiveness is greatly reduced if the time between successive scans is larger than the time needed to update blocklists. Therefore, in this subsection, we analyze the time between scans launched from the same source address to provide insight into the speed with which blocklists should be updated. Figure 2.4 shows the average percentage of overlapping source IP addresses between two days. The x-axis indicates the number of days between the measured overlap, and the colors indicate the used dataset's year. The figure shows a trend of a decreasing address overlap over time, demonstrating the necessity to update blocklists promptly when new malicious addresses are identified. For instance, in 2021, on average, 17.99% of the IP addresses were reused the next day, while only 2.78% were reused 30 days later. We also note large fluctuations between the overlaps, which are caused by the heterogeneous scan landscape shown in the previous subsection. Besides IP blocklists requiring prompt updates, the lists also become stale over longer periods of time. When analyzing the re-usage of IP addresses over multiple years, we see that, on average, 2.11% of the identified IP addresses are reused the next year, decreasing to 0.07% after seven years. The analysis of the time between IP address re-usage shows the necessity to promptly update blocklists after identifying addresses

used to send scan traffic.

ASSESSING THE TIMELINESS OF ADDING ADDRESSES TO THE BLOCKLIST

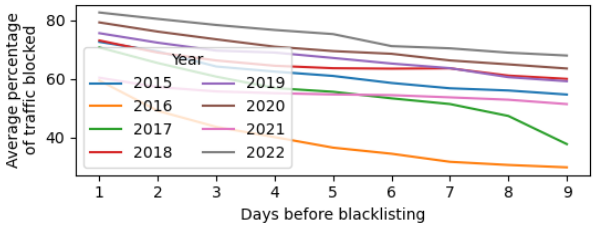


Figure 2.5: Effectiveness of an IP address blocklist with delayed entries

The demonstrated effectiveness of blocklists earlier in this section relies on the assumed response time of one day. Griffioen et al. [24] show that the average inclusion time of IP addresses into a blocklist is 21 days after the first malicious activity has been recorded, with a few of the fastest feeds including them after two days. The previous subsection shows that most of the recurring scans launched from the same IP address occur within a few days. Therefore, our demonstrated effectiveness at the beginning of this section can be considered optimistic and should be analyzed using different timeframes. Figure 2.5 shows the average percentage of traffic blocked when increasing the time it takes for the defending party to add the IP address to the blocklist. As expected, the percentages of blocked traffic decrease as the response times increase. On average, we measure an 18.66% drop in blocked traffic when increasing the reaction time from 1 to 9 days. This shows the sensitivity of blocklists towards update times and, therefore, the necessity to promptly update blocklists, as their effectiveness relies on timely information.

PURGING BLOCKLISTS

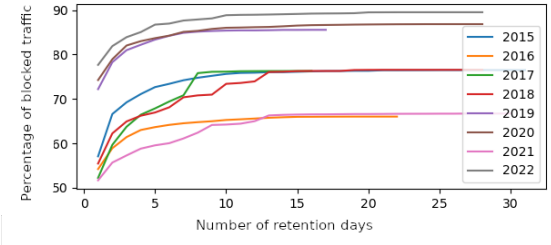


Figure 2.6: Influence of IP address retention in blocklists

On average, our datasets contain 111,145 unique source IP addresses per month. Blocklists must be trimmed over time as they can grow large, requiring an ever-increasing amount of resources for the systems using them. Figure 2.6 shows the average percentage of blocked traffic against the number of days a source IP is kept on a blocklist after its last appearance. The figure shows that keeping addresses in the list has diminishing returns after a certain period of time. For instance, in 2022, 11,17% more traffic is blocked when increasing the retention time from one to ten days, whereas increasing from ten to twenty days yields an increase of 0,46%. This trend is similar for all recorded years. A retention period of ten days yields, on average, a 15.01% increase in blocked traffic while prolonging the retention to the end of the collection period results in an additional 1% of blocked traffic. The data clearly shows that IP addresses on blocklists can be purged to free up system resources after ten days, causing a minimal effect on the amount of blocked scan traffic.

2.4.2. PAYLOAD BLOCKLISTS

Payload-based blocklists are a standard tool used by IDSs, such as Bro [26] and Snort [27], to identify malicious traffic and consequently block the identified packets from entering the network. These blocklists are used to identify and drop incoming scan traffic, preventing scanning criminals from gaining information about devices running specific services in the network. Although payload blocklist mechanisms can be effective, criminals can circumvent them in two manners. Criminals can evade detection by sending arbitrary payloads unrelated to the scanned protocol, relying on received ICMP responses indicating that the destination address is not in use or that the port is closed. Criminals can also evade detection by crafting a custom payload for each scanned destination IP address, making it difficult for defending parties to identify scan traffic, as adding all unique payloads on a blocklist is unfeasible. In the following section, we assess whether a scanning technique relying on ICMP responses [11] effectively finds devices running a specific service on top of the UDP protocol. Afterward, we assess whether criminals use the same payload or generate different payloads for each targeted destination address to circumvent blocklists. Next, we evaluate the frequency with which scan probes payloads are re-used. Finally, we evaluate the effectiveness of payload-based blocklists using our datasets.

EFFECTIVENESS OF ICMP STYLE SCANNING

Table 2.4: ICMP and protocol-appropriate responses to ten million scan probes sent to four different ports

Port	ICMP	Responses
123	383,731	2,841
389	452,516	10
5060	436,625	20,220
53	367,837	18,467

When scanning for devices running a service, an attacker can send arbitrary UDP

probes toward a destination IP address and port combination. The attacker waits for incoming ICMP responses, informing him that the address is not in use or that the port is closed. Using a process of elimination, an attacker can derive which IP addresses are allocated to a device and which devices have an open port, indicating that the devices are running a service. The advantage of relying on ICMP responses is that an attacker can use the same scanning setup to scan the Internet for devices running different services. In addition, the content of the UDP probes does not matter. Therefore, attackers can randomize the scan probes' content, evading detection by IDS using payload blocklists. The drawback of relying on ICMP responses is that network operators can disable them. In this subsection, we evaluate the effectiveness of scans relying on ICMP responses and show that this scanning method does not yield accurate results.

To assess the effectiveness of the previously described scanning strategy, we probed 10 million random destination addresses targeting four commonly scanned destination ports and recorded the ICMP responses. For our scan probe payloads, we used protocol-appropriate messages for the targeted ports, such as sending a DNS query requesting the A record of *amazon.com* for destination port 53. Sending protocol-appropriate payloads does not interfere with the results, as scanners only consider ICMP responses sent when the IP address is not allocated or the port on the device is closed. Table 2.4 shows the received responses for each scan campaign. From the 10 million scanned destinations, we only received an average of 410,177 ICMP responses, with less than 20,220 servers responding. Our test shows that using a scanning technique relying on ICMP responses is inaccurate as more than 95% of the probed destinations do not respond with an ICMP message. The scan results show that criminals are incentivized to use appropriate scan payloads for the targeted protocols, suggesting that the use of payload-based blocklists can be an effective means to detect and drop scan traffic.

STATIC OR GENERATED PAYLOADS

Table 2.5: Scan traffic per payload generation type

	% IPs		**% Packets**	
Year	Static	Generated	Static	Generated
2015	9.91	94.70	57.85	42.15
2016	98.09	2.04	53.01	46.99
2017	85.37	16.87	64.26	35.74
2018	52.40	53.37	47.76	52.24
2019	63.95	52.26	50.22	49.78
2020	2.01	98.97	68.40	31.60
2021	5.10	95.67	71.28	28.72
2022	3.55	97.90	59.51	40.49

We previously showed that criminals cannot rely on ICMP response messages to determine which IP addresses are assigned to machines and, more importantly, which services are running on them. Therefore, scanners must send valid UDP payloads to which services respond. Criminals can use popular scan tools such as Nmap and Zmap to send

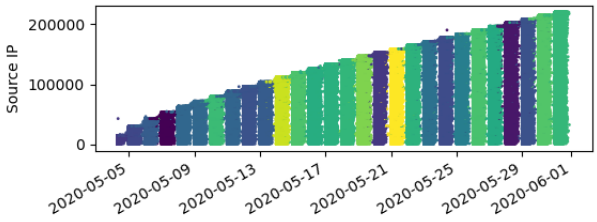


Figure 2.7: Large scan campaing in 2020

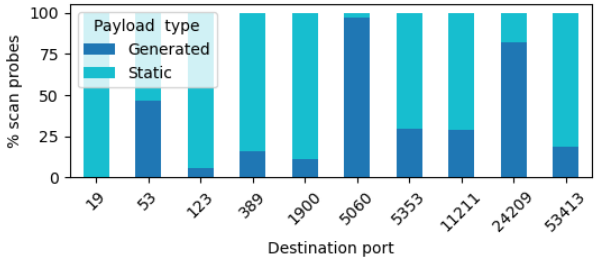


Figure 2.8: Percentage of traffic received for the top destination ports according to payload generation method

protocol-specific probes, as these tools come preconfigured with UDP payloads for common protocols. However, these payloads are well known and can be added to blocklists, allowing IDS to identify and drop scan traffic. Criminals can also set up their infrastructure using different payloads than the ones used by standard scan tools to evade detection. However, once payloads are associated with scan traffic, they can be added to the blocklist, requiring criminals to adapt their payloads to remain undetected. Criminals can opt for a third option: generating an individual scan probe for each scanned destination. For instance, criminals scanning for DNS servers can send scan probes containing queries for arbitrary subdomains, making each scan probe's payload unique and, therefore, undesirable to integrate into a blocklist, causing lengthy blocklists potentially slowing down the IDSs. The following subsection analyzes whether the recorded scans use a static or generated payload approach. This analysis allows us to assess whether criminals have changed their payload strategies over the past nine years and allows us to assess the potential effectiveness of payload-based blocklists.

To identify the payload generation method used by scanners, we record each payload sent during a scan launched from a source IP address. When a single payload is identified during a scan, we label it as static; otherwise, it is labeled as generated, meaning that the criminal uses different payloads per scanned destination. Table 2.5 shows the percentage of source IP addresses from which static or generated scans are launched and the percentage of traffic received for static or generated scans.

The metrics displayed in table 2.5 show significant fluctuation in the percentage of source addresses from which scans using a generated payload are performed, varying

between 2.01% and 98.09%. However, the percentage of traffic received follows a different trend. On average, 40.96% of the received traffic originated from scans using generated payloads, with a standard deviation of 8.51%. We attribute the measured fluctuation to different scan tactics, such as distributed scan campaigns launched from multiple source IP addresses. Figure 2.7 shows a scatterplot representation of a distributed scan campaign conducted in 2020 involving a total of 220,415 source IP addresses scanning a different destination port each day, sending an average of 14.27 scan probes with generated payloads. Each dot of the scatterplot represents a scan launched from a source IP address, which is numbered on the Y axis. The different colors indicate one of the 28 different destination ports targeted. Although large scan campaigns can influence the percentage of source IP addresses used to perform scans using generated payloads, it does not influence the percentage of the received traffic originating from scans using a static payload generation. On average, 59.04% of the received scan probes have a static payload, suggesting that intrusion detection systems relying on payload blocklists can effectively identify more than half of the incoming scan traffic.

Table 2.5 showcases a second trait of the scan landscape characterized by an overlap between the percentage of IP addresses from which scans using generated and static payloads are launched. In 2018 52,40% and 53.37% of the recorded IP addresses were used to launch static and generated payload scans, indicating that 2.88% of the recorded IP addresses were used to send both generated and static payloads, suggesting that criminals launching multiple scans from the same source IP addresses use a different approach to craft their scan probes. Further analysis of this subset shows that 84.07% of the IP addresses are used to scan multiple destination ports, revealing that criminals are changing their payload generation strategy based on the targeted protocol. We confirm that criminals generally adapt their payload generation strategy according to the probed destination port. Figure 2.8 shows the percentage of traffic received according to the payload generation method used by scanners for the top ten ports across multiple years. The figure shows that 97.23% of the scan traffic captured towards destination port 5060 has a generated payload. We note large differences in the percentage of generated probes for different destination ports. Criminals changing their payload generation strategy according to the targeted ports indicates that using a blocklist will not be equally effective in identifying scan traffic destined for different ports.

PAYLOAD RE-USAGE

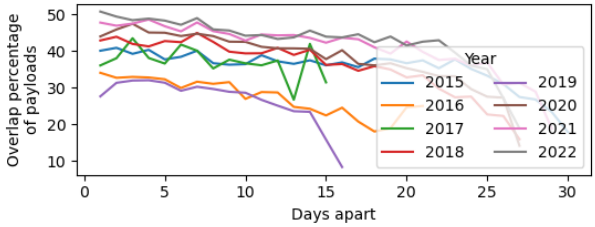


Figure 2.9: Percentage of recurring payloads after n days

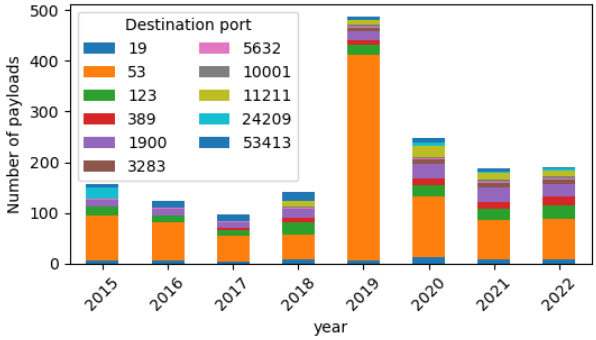


Figure 2.10: Number of payloads per destination port per year

Griffioen et al. [24] show that the fastest update time for blocklists is two days. Therefore, payload blocklists can only detect scans using identical payloads launched on different days. In the following subsection, we evaluate the re-usage of observed static payloads, indicating whether blocklist can be effectively used to detect scan traffic.

Figure 2.9 shows the percentage of shared payloads between two days. The figure clearly shows a considerable overlap between used payloads. In 2018 42.87% of the payloads were used the next day, decreasing to 22.30% after 26 days. This trend of decreasing payload overlap between days is shared among the different years in our dataset. When analyzing the overlap between years, we see a steady decline from 21.23% after one year to 7.80% after seven years. The data shows that scanners are re-using or using the same payloads when scanning. In addition, the data shows that blocklists need to be promptly updated with newly discovered payloads as the payload re-usage declines with time.

Further analysis studying the payloads per destination ports reveals that the targeted protocols by scanners dictate the effectiveness of payload blocklists. Figure 2.10 shows the number of unique recorded payloads per destination port per year. The figure shows that in 2019, 406 unique static payloads were sent toward port 53. On average 31.55% of all 2,472 recorded payloads are sent towards port 53, indicating that scanners use a wider variety of payloads towards that port. It is unsurprising that we record a high number of different payloads towards port 53 as it is the standard port assigned to the DNS protocol, which allows a high degree of freedom in scan probe customization. The freedom of payload customization offered by some protocols, such as allowed by DNS, can also influence the efficiency of blocklists. Services offering a wider range of customization allow scanners to switch payloads, lowering the effectiveness of payload-based blocklists for those particular services.

Both results support the necessity to promptly update blocklists when new payloads are detected as scanners tend to alter their payloads on a regular basis.

BLOCKLIST EFFECTIVENESS

Throughout this section, we have shown that 59,04% of the received scan traffic is created using a static payload generation approach, that payloads are less prone to be re-

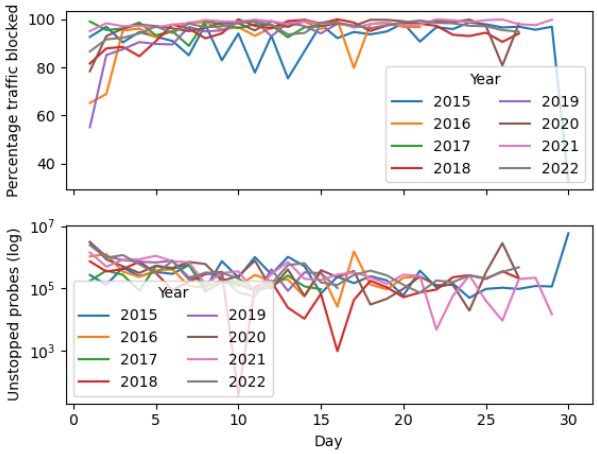


Figure 2.11: Effectiveness of an payload blocking mechanism

used when time passes, and that blocklists' effectiveness will vary depending on the scanned ports. In this final subsection, we evaluate the overall effectiveness of blocklists using our dataset.

We can measure the effectiveness of defense systems using payload blocklists by blocking traffic using payloads previously recorded in the telescope dataset. We apply our blocklisting only on scans using a static payload generation approach and later generalize the obtained results. For this analysis, we block traffic containing a specific payload one day after its first appearance in the dataset, creating a best-case scenario [24]. Figure 2.11 shows the percentage of stopped scan traffic and the number of scan probes unaffected by the blocklist. On average, 94.92% of the scan probes are blocked, suggesting that payload-based blocklists effectively detect scan traffic. However, we have shown that IP-based blocklists' efficiency declines when the lists are not promptly updated, and the payload re-usage section above shows that payload re-usage, similarly to IP address re-usage, decreases with time. Therefore, the results depict a best-case scenario, and payload-based blocklists will lose their effectiveness when the time taken to add payloads to the blocklists increases.

The 94.92% effectiveness is only applicable for 59.04% of all scan traffic, as the remaining traffic originates from scans using a generated payload approach. Ghiette et al. [18] propose a methodology that can generate regular expressions covering 96% of all scan traffic. Popular IDSs can use regular expressions to identify payloads and effectively function as entries in a payload blocklist. When projecting the measured effectiveness on the entire dataset, IDSs using payload-based blocklists would detect 92,40% of the scan probes. The percentage of identified scan traffic can be classified as high. However, on a daily basis, our telescope receives in the order of 14 million scan probes. We show in section 2.5.1 that 95% of the scan traffic originating from a single source IP is directed towards unique destination addresses. If we extrapolate these numbers, 244 scans covering our entire telescope will remain undetected daily despite using blocklists which are

92,40% effective. Therefore, we can conclude, similarly to IP blocklists, that payload-based blocklists should not be considered an effective means to protect devices in a network against scan traffic.

2.5. BEHAVIORAL ASPECTS OF SCANS

In section 2.4, we analyzed the effectiveness of blocklists as a means to detect ongoing scans and block incoming scan probes. The effectiveness of these defense mechanisms varies and relies on attackers reusing the same infrastructure and depends on the frequency and speed with which these lists are updated. An alternative method to identify scans and thereby provide an opportunity to block their probes is behavior. Therefore, in this section, we analyze the behavior of scanners targeting our telescope. The first behavior analyzed is the number of probes sent per scan toward a particular destination. The second behavior is speed, which translates to the number of scan probes sent during a scan, given the scan's duration. Both behavioral aspects can be translated into rules IDS uses to identify and block scans. Therefore, in the following section, we analyze the aforementioned behavioral aspects and asses whether these have changed in the past years, providing an insight into whether the rules are effective and need to be adapted over time. We continue our behavioral analysis and show that in past years, scanners have increased the distance between successively scanned addresses and reduced the number of probes sent per source IP address, increasing the difficulty of identifying scans. Finally, we show that smaller network operators are less likely to identify scans than larger ones.

2.5.1. SENDING MULTIPLE PROBES TO THE SAME DESTINATION

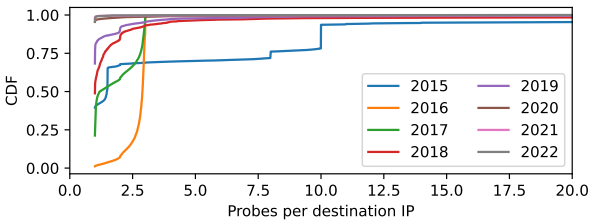


Figure 2.12: Packet sent per destination IP address in the telescope

Criminals searching for vulnerable devices benefit from accurate scan techniques that identify devices running desired services. Packet loss can influence the results obtained from a scan. For instance, missing responses to scan probes can result in false negatives, leading the attacker to believe that the probed devices do not run the desired service. Attackers can limit the effect of packet loss by sending multiple scan probes to the same destination, lowering the chances of a false negative. Although this technique increases accuracy, it also allows defending parties to detect scans based on the number of probes received per destination. Intrusion detection systems can identify scans that target the same destination multiple times by keeping a simple counter. Therefore, in the following subsection, we analyze whether criminals adopted such behavior and whether

it has changed in the past years, providing insight into the requirements of such detection rules.

Figure 2.12 shows the cumulative distribution function of the packets sent per destination IP address and port combination in our telescope during a scan launched from a single source IP address. The figure shows an evident change in behavior from 2020 onward. After 2020, on average, 95.51% of recorded scans only sent one scan probe per destination IP address. The high percentage indicates that using the number of probes sent per destination has lost its effectiveness in identifying scans in recent years. Before 2020, the number of probes sent per destination fluctuated, with a notable change in 2016, where 98.8% of the scans were recorded sending more than one probe per destination. In 2016, 79.24% of the launched scans were directed towards port 53413, sending between 2.5 and 3 probes per target using the same payload. The behavior suggests the mass adoption of a scan tool similar to the findings of Ghiette and Doerr [8], resulting in criminals sharing similar behavior.

The presented results show that scan detection rules based on repeated probing are ineffective in detecting ongoing scans and, therefore, cannot effectively block scan traffic.

2.5.2. SCAN SPEED

Scanners can adopt different strategies with regard to the speed with which they send scan probes. Modern tools such as Nmap and Zmap allow scanners to probe the entire Internet in hours. These tools can also throttle the scan speed, allowing users to fine-tune their output rate. Griffioen et al. [24] show that the fastest update time for IP blocklist is two days, revealing an incentive for cybercriminals to perform fast scans as they would circumvent their used IP addresses being added to such lists when probing the Internet within that time frame. However, when performing fast scans, criminals risk their scan campaigns being detected and halted by IDS using a scan probe rate threshold. Bro and Snort allow for such detection strategies based on rules detecting scans as source IP addresses sending a defined amount of packets per time frame toward the monitored network. Therefore, criminals may slow their scan rate so as not to trigger these defense mechanisms. In the following section, we analyze the probing speed of identified scans, providing insight into the scan behavior of criminals targeting our networks over the past years.

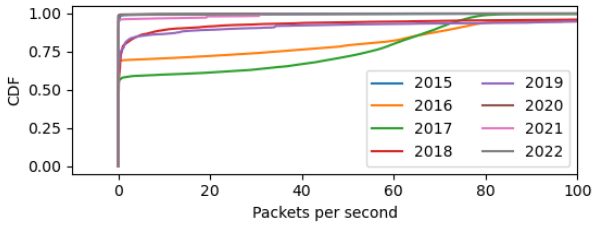


Figure 2.13: CDF of scan speeds

Figure 2.13 shows a cumulative distribution function of the packets per second with which scanners send scan probes from a particular source IP address to our telescope

per year. The figure reveals that the scan speeds have decreased over the last few years. For instance, in 2017, 57.93% of the scans sent more than one probe per second, while after 2020, on average, 98.04% of the recorded scans sent less than one probe per second. The data shows that scanners have generally slowed down their operations.

Scanners slowing down their operations seems counterintuitive when IP blocklists are, in the best case, only updated within two days after scan detection [24]. However, on average, 83.07% of the recorded scans terminate within two days, showing that most scans would not be detected by IDS using blocklists if criminals performing successive scans would cycle through source IP addresses.

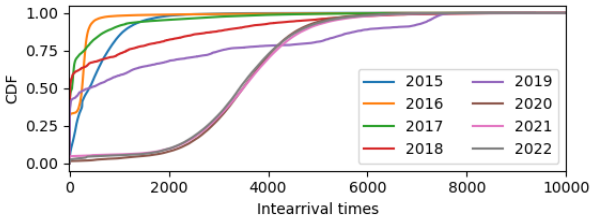


Figure 2.14: CDF of scan probes interarrival times

With criminals slowing down their operations, defending parties must adjust their detection rules accordingly. Figure 2.14 shows the cumulative distribution function of the average scan probe interarrival times for different years. The data reveals that the time between packets in 2020 drastically increased as scan speeds decreased compared to prior years. After 2020, the average interarrival time is 3411.53 seconds, equivalent to 56.85 minutes, while in 2016, that was 251.87 seconds or 4 minutes. The data clearly shows that in order to detect scans based on their speed, the time windows for counting received packets need to be increased in recent years, moving from several minutes to hours when requiring more than three packets.

2.5.3. REDUCING PROBES SENT PER SCAN

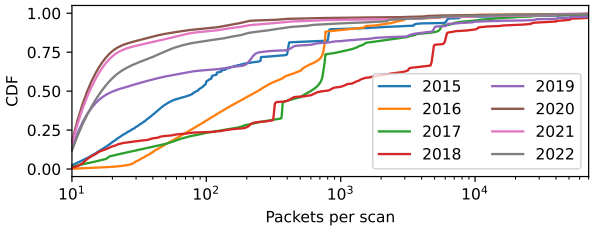


Figure 2.15: CDF of packets per scan

Throughout this work, we show two examples of distributed scans, one involving 244 source IP addresses and another involving over 220,000 IP addresses. Popular scanning

tools allow criminals to use multiple source IPs to distribute the scan, lowering the number of probes sent from each address. Lowering the number of probes sent per IP address increases the chance of evading detection by IDSs using a threshold of probes received per source IP address. Network operators often use a minimum packet threshold to reduce the number of false positives caused by Internet backscatter. In the following subsection, we analyze the number of probes sent per source IP address in the past years to determine whether criminals are lowering this to evade detection.

Figure 2.15 shows the CDFs of the number of packets sent per scan for the past eight years. The figure shows a trend, notably that scanners send fewer packets per scan in the past years. Prior to 2020, on average, 33.59% of the effectuated scans sent less than 100 packets; after 2020, this percentage rose to 81.11%. The reduction in packets sent per scan indicates that network operators managing the IDS rules should revise their settings to accommodate for the change in packets sent per scan.

2.5.4. RANDOMIZING SCANNED DESTINATIONS

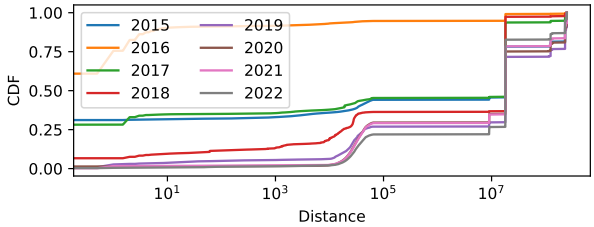


Figure 2.16: CDF of median distance between successive destinations scanned

Popular scan tools allow attackers to randomize the order in which destination addresses are scanned. Furthermore, botnets scanning for vulnerable devices like Mirai have adopted a random selection process. Randomizing the destination scanned lowers the chance of IDSs detecting attacks. Namely, it increases the distance between scanned destinations, enlarging the time between packets sent to the same network range, improving the odds of evading detection by IDSs. Furthermore, the previous section shows that criminals send fewer packets per scan. Therefore, randomizing the scanned destination will result in subnets receiving fewer scan probes, allowing criminals to evade IDSs using packet number thresholds. This section analyzes whether criminals have adopted a randomized scan approach to increase the distance between successively scanned destinations.

Figure 2.16 shows the CDF of the median of the distance between the successive destinations probed in scans for the past eight years. The figure shows that criminals have augmented the distance between successively probed destination addresses. Before 2018, over a quarter of the scans sent successive probes to neighboring IP addresses. In 2019, this number was 6.64% and gradually reduced to 0.049% in 2022. Criminals scanning the Internet have moved towards a scanning strategy that increases the distance between successfully scanned destinations, most likely in an attempt to evade detection by IDSs.

Therefore, IDS rules based on detecting scan traffic using the destinations of successive scan probes can be considered ineffective.

2.5.5. SMALLER NETWORKS AT A DISADVANTAGE

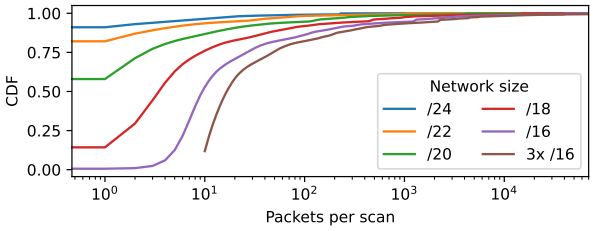


Figure 2.17: CDF of received scan probes according to network size

Previously, this section showed that scans have become slower, generate less traffic, and increase the distance between successively targeted destination addresses. These behavioral changes can hamper small network operators' ability to detect scans. The following section analyzes whether smaller networks are indeed unable to detect scans due to the behavioral changes mentioned above.

To assess the ability of smaller network operators to detect scans, we measure the number of packets received per scan for different network sizes in our 2022 dataset. Figure 2.17 shows the CDF of the number of packets received per scan in our dataset for various network sizes. The figure shows that a smaller network operator cannot detect all scans identified in our telescope. The/24 network does not register a single probe sent from 82.58% of the scans. This is expected as low amounts of probes are sent during a scan from a single source address, and the targeted destinations are far apart, making it unlikely that a small network will receive probes from a launched scan. Therefore, the problem small network operators face is not the absence of scan traffic but rather the presence of small amounts of scan traffic. Figure 2.17 shows that an operator of a /24 network are unable to detect 13.97% of the scans reaching his network if a limit is set to 10 received probes per scan. Similarly, operators monitoring a /20 and /16 networks will not detect 55.39% and 52.28% of the scans probing their networks when setting a minimum probe threshold to 10. Clearly, the monitored network size affects the ability of operators to detect scans based on the number of probes received. A direct consequence of receiving fewer probes per scan is the inability to identify scans using behavioral characteristics requiring a minimum amount of scan traffic. The inability to detect scans results in smaller network operators being unable to maintain their own IP address blocklists. Therefore, smaller network operators are at a disadvantage and are forced to use external resource for identifying scans such as publicly available blocklists. The combination of long blocklists update times [24] and the importance of promptly updating blocklists, shown in section 2.4, suggests that smaller network operators using the studied defense mechanism will not be able to detect and defend against incoming scans.

2.6. ORIGIN OF LAUNCHED SCANS

Cybercriminals can launch their attacks from various locations in the world [12]. Some may prefer launching scan campaigns from particular countries due to relaxed or unexisting legislation against cybercrime [28]. Other factors, such as cost and infrastructure, may also play a role. Criminals can also select more precise locations on the Internet to launch their scan campaigns. Some autonomous systems have the reputation of allowing malicious behavior [19], enticing cybercriminals to launch attacks from within their allocated IP ranges. Reports [20] even show that criminals use specific ISPs, better known as bulletproof providers, to conduct their business. With criminals preferring specific locations on the Internet to conduct their operations from, the question arises whether using a reputation system based on the location of IP addresses can benefit the detection of scan campaigns. The following subsection analyzes the countries and ISPs from the identified scans in the past years, revealing whether criminals commonly launch their operations from specific places on the Internet, foreshadowing the effectiveness of location in identifying criminal activities.

2.6.1. SCANS ORIGINATING FROM SPECIFIC COUNTRIES

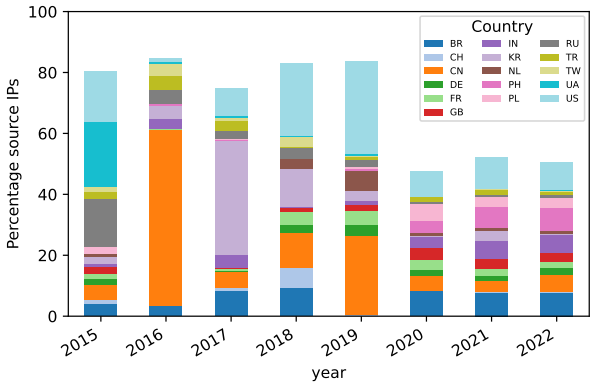


Figure 2.18: Percentage of IP addresses from top 5 countries

We use the ge-location databases of MaxMind [29] to link the source IP addresses from our datasets to their corresponding countries. First, we analyze the number of source IP addresses per country from which scans have been launched. The number of addresses would reflect whether criminals launch campaigns from a particular country or set of countries. Figure 2.18 shows a barplot with the percentage of source IP addresses for the top five countries per year from which scans have been launched. From the legend in the plot, we can already see that the top five countries fluctuate per year, as there are 16 countries shown in the figure representing the top 5 countries for the past eight years. Furthermore, the percentage of source IP addresses used to launch scans fluctuates in different years. For instance, in 2016, 57.51% of the used IP addresses originated from China, while in other years, the average was only 8.92%. Similar outbursts occur for the

percentage of source IP addresses from other countries. We can use the rank biased overlap (RBO) method, introduced by Webber et al. [30] to measure overlap between the top n countries from which scanning source IP addresses originate. We use RBO as it can compare ranked lists containing different elements, and the method's overlap factor is influenced by the elements ranking. The method outputs a number between 0 and 1, with 0 indicating no overlap and 1 indicating identical lists. The average overlap for the top 5, 10, 20, and 50 countries in the past eight years is 0.48, 0.47, 0.48, and 0.49, indicating no substantial overlap between the top countries. From the data shown in figure 2.18 and the calculated RBO values, we can conclude that criminals launch their scans from different countries each year. Therefore, using an IP address's country of origin to detect scan traffic will not be effective when using data from previous years.

We continue the analysis by monitoring whether criminals launch scans from varying countries within the same month. Therefore, we calculate the RBO values for the top 50 countries from which scans are launched daily. Calculating the RBO values for consecutive days in different months results in an average similarity metric of 0.77. The calculated values show that criminals launch their scan campaigns from the same countries within the same month. As an example, in 2017, each day, most of the source IP addresses used to launch scans were located in Korea. The overlap of countries from which scans are launched within the same month shows that the countries of origin could be used on a monthly basis to aid in the detection of scans. Thus, defending parties could use the country of origin of traffic to aid in the identification of scan probes.

2.6.2. SCANS ORIGINATING FROM SPECIFIC ISPS

Reports show that criminals tend to use specific ISPs to conduct their activities from [20]. In the following subsection, we analyze whether ISPs can be used to aid in the detection of scan traffic. If criminals often use the same ISP network, network operators can use a ranking system to assess the incoming traffic's probability of being malicious. In addition, operators can block the entire range of ISPs to protect their networks. To assess whether criminals reuse the same ISP to launch their operations, we analyze the number of offending source IP addresses per ISP. We continue our study by analyzing the scan traffic sent per ISP, showing that small ISPs are more likely to send scan traffic. Finally, we reveal that small ISPs are more likely to be active for a short duration, suggesting that blocking traffic from ISPs is only effective when having short response times.

TOP ISPS

Reports show that criminals use certain ISPs to launch their scans from [20]. In the following subsection, we analyze whether criminals use the same ISPs in consecutive years to conduct their activities or migrate to different ones. If criminals use the same ISPs they can be used to aid in the detection of scan traffic.

We use the ISP databases of Maxmind to link the IP addresses to their corresponding ISP. Figure 2.19 shows the top 5 ISPs per year ranked by the percentage of source IP addresses in our dataset. Similar to the findings of the previous subsection, the ISPs hosting the most source IP addresses from which scans are launched vary every year. We identify 23 different ISPs hosting the most IP addresses from which scans have been launched in

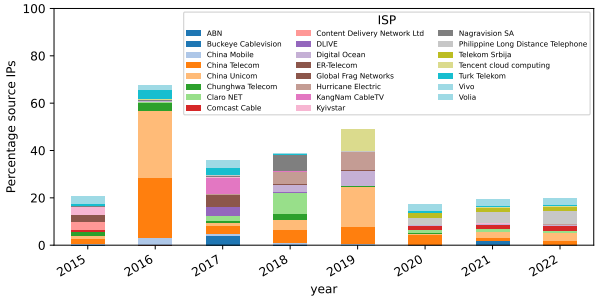


Figure 2.19: Percentage of IP addresses from top 5 ISPs

the past eight years. Each year, the ISPs change, and the calculated RBO values between years vary between 0.22 for the top 10 ISP and 0.23 for the top 100 ISPs. The calculated RBO values show that scans are launched annually from different ISPs. Analyzing the number of years ISPs are in our datasets explains the low RBO values. 47.17% of the ISPs in our dataset are used to launch scans in only one year, 22.82% in two years, and only 0.64% of the ISPs are used in all recorded years. The fluctuation of ISPs hosting the most IPs from which scans are launched indicates that using a reputation system based on the number of scanning IPs from ISPs is ineffective throughout different years.

Although criminals change ISPs on a yearly basis, they tend to use the same ones within a month. Calculating the RBO values for the top 10 and 100 ISPs used daily results in a similarity value of 0.63 for both measures. The daily overlap of ISPs used to launch scans indicates that the origin of scanning IP addresses can be used to facilitate the detection of scans when the frequency with wich the offending ISP lists are refreshed is high.

CHARACTERIZING OFFENDING ISPS

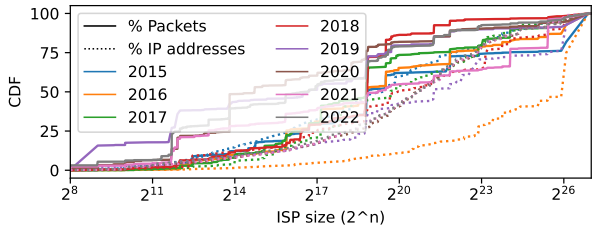


Figure 2.20: CDF of percentage of IP addresses and sent traffic from ISPs according to size

We show that criminals use the same ISPs to launch their scan campaigns within a monthly period. In this subsection, we look closer at the ISPs from which large amounts of traffic are sent. If criminals tend to send large amounts of scan traffic from select ISPs,

defending parties can block incoming traffic from them, stopping scanners from gaining information about the targeted network.

We first identify the top 20 ISPs from which most scan traffic is sent for each year. The results show that each year, between 70.59% and 84.01% of scan probes are sent from those ISPs. Blocking the top 20 ISPs would be an effective means to reduce the amount of scan traffic. However, this solution would significantly impact the reachability of the defending networks. In the top 20 ISPs, we identify large providers such as China Telecom and cloud hosting providers such as Google. These large ISPs host more than 124 and 3 Million IP addresses. Therefore, blocking access to these entire ranges is not feasible for network operators hosting services, which should be accessible to the public.

Within the top 20 ISPs from which we receive the most scan traffic are also ISPs that can be classified as bulletproof hosters. Each year, between 4 and 6 of the top 20 analyzed providers fall within this category. These providers have a commonality: they contain few IP addresses. The observation of smaller ISPs sending large amounts of scan traffic can be generalized. Figure 2.20 shows the CDF of the percentage of scan traffic and the percentage of total source against the ISPs' size. The figure shows that small ISPs send more scan traffic than larger ones. On average ISPs with less than 2^{16} IP addresses are responsible for 38.97% of the sent scan traffic while only hosting 11.73% of the addresses from which scans are launched. Network operators could use the size of ISPs to aid in the detection of scan traffic, as it is more likely to originate from small ISPs.

2.7. CONCLUSION

In this work, we analyzed the behavioral changes of scanners in the past years and the effectiveness of simple scan detection strategies.

We show that, in recent years, scanners have reduced the number of probes sent per scan, prolonged the duration of scans, and increased the distance between scanned addresses. These identified changes demonstrate the necessity for network operators to to adapt the rules used by IDSs to allow for the detection of scans. Furthermore, the resulting changes suggest that smaller network operators depend on intelligence gathered by other parties which should be updated and published more frequently.

We analysed Blocklist-based systems using IP addresses or payloads and show that these defense mechanisms do not suffer from the evolving scan landscape and maintain the same level of effectiveness when it comes to blocking scan traffic throughout the measured years. However, these systems are not robust, as we have shown that they suffer from varying degrees of effectiveness depending on several factors, such as targeted ports and the update frequency of blocklists. Therefore, network operators should deploy other detection systems if possible.

Our location analysis shows that the location from which scans originate does change yearly but remains similar within a month. In addition, smaller networks send disproportional amounts of scan traffic compared to larger ones. Therefore, network operators can use the origin of traffic, when using up to date scan location information, to aid in detecting scan probes when.

In conclusion, our work shows that analyzed basic scan identification and defense mechanisms provide some level of security but suffer from various shortcomings linked

to the dynamic scan landscape. Future work could explore the resilience of advanced scans detection and defence mechanisms against the changing scan landscape.

References

[1] E. Bou-Harb, M. Debbabi and C. Assi. 'A systematic approach for detecting and clustering distributed cyber scanning'. In: *Computer Networks* (2013). DOI: https://doi.org/10.1016/j.comnet.2013.09.008.

[2] S. Torabi, E. Bou-Harb, C. Assi, M. Galluscio, A. Boukhtouta and M. Debbabi. 'Inferring, Characterizing, and Investigating Internet-Scale Malicious IoT Device Activities: A Network Telescope Perspective'. In: *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2018. DOI: 10.1109/DSN.2018.00064.

[3] *CVE-2020-13160*. Available from MITRE. Dec. 2020. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-13160.

[4] Lockheed Martin. *The cyber kill chain*. URL: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.

[5] R. Hiesgen, M. Nawrocki, T. C. Schmidt and M. Wählisch. 'The race to the vulnerable: Measuring the log4j shell incident'. In: *arXiv preprint arXiv* (2022).

[6] H. L. J. 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. DOI: 10.1145/3319535.3354230.

[7] H. Griffioen, K. Oosthoek, P. van der Knaap and C. Doerr. 'Scan, Test, Execute: Adversarial Tactics in Amplification DDoS Attacks'. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021. DOI: 10.1145/3460120.3484747.

[8] V. Ghiette and C. Doerr. 'How Media Reports Trigger Copycats: An Analysis of the Brewing of the Largest Packet Storm to Date'. In: *Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity*. 2018. DOI: 10.1145/3229598.3229606.

[9] M. de Vivo, E. Carrasco, G. Isern and G. O. de Vivo. 'A Review of Port Scanning Techniques'. In: *ACM SIGCOMM Computer Communication Review* (1999). DOI: 10.1145/505733.505737.

[10] M. H. Bhuyan, D. Bhattacharyya and J. Kalita. 'Surveying Port Scans and Their Detection Methodologies'. In: *The Computer Journal* (2011). DOI: 10.1093/comjnl/bxr035.

[11] E. Bou-Harb, M. Debbabi and C. Assi. 'Cyber Scanning: A Comprehensive Survey'. In: *IEEE Communications Surveys and Tutorials* (2014). DOI: 10.1109/SURV.2013.102913.00020.

[12] Z. Durumeric, M. Bailey and J. A. Halderman. 'An Internet-Wide View of Internet-Wide Scanning'. In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. USA, 2014.

[13] V. Ghiette, N. Blenn and C. Doerr. 'Remote Identification of Port Scan Toolchains'. In: *8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 2016. DOI: 10.1109/NTMS.2016.7792471.

[14] H. Griffioen, H. Hu and C. Doerr. 'SIP Bruteforcing in the Wild - An Assessment of Adversaries, Techniques and Tools'. In: *IFIP Networking Conference (IFIP Networking)*. 2021. DOI: 10.23919/IFIPNetworking52078.2021.9472857.

[15] 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. DOI: 10.1145/3196494.3196537.

[16] H. Griffioen and C. Doerr. 'Discovering Collaboration: Unveiling Slow, Distributed Scanners based on Common Header Field Patterns'. In: *IEEE/IFIP Network Operations and Management Symposium*. 2020. DOI: 10.1109/NOMS47738.2020.9110444.

[17] C. Gates. 'Coordinated Scan Detection.' In: *NDSS*. 2009.

[18] V. Ghiette and C. Doerr. 'Clustering Payloads: Grouping Randomized Scan Probes Into Campaign Templates'. In: *IFIP Networking Conference*. 2022. DOI: 10.23919/IFIPNetworking55013.2022.9829757.

[19] C. A. Shue, A. J. Kalafut and M. Gupta. 'Abnormally Malicious Autonomous Systems and Their Internet Connectivity'. In: *IEEE/ACM Transactions on Networking* (2012). DOI: 10.1109/TNET.2011.2157699.

[20] M. Goncharov. 'Criminal hideouts for lease: Bulletproof hosting services'. In: *Forward-Looking Threat Research (FTR) Team, A TrendLabsSM Research Paper* (2015).

[21] D. Moore, C. Shannon, G. Voelker and S. Savage. 'Network Telescopes: Technical Report'. In: (2004).

[22] Internet Assigned Numbers Authority. *Service Name and Transport Protocol Port Number Registry*. Tech. rep. 2023. URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[23] FireHOL. 'All Cybercrime IP Feeds'. In: (2023). URL: https://firehol.org/.

[24] H. Griffioen, T. Booij and C. Doerr. 'Quality Evaluation of Cyber Threat Intelligence Feeds'. In: *Applied Cryptography and Network Security*. Springer International Publishing, 2020.

[25] G. Wan, L. Izhikevich, D. Adrian, K. Yoshioka, R. Holz, C. Rossow and Z. Durumeric. 'On the Origin of Scanning: The Impact of Location on Internet-Wide Scans'. In: *Proceedings of the ACM Internet Measurement Conference*. Association for Computing Machinery, 2020. DOI: 10.1145/3419394.3424214.

[26] R. Larsen. *BRO-an Intrusion Detection System*. 2013.

[27] J. Koziol. *Intrusion detection with Snort*. Sams Publishing, 2003.

[28] United Nations Conference on Trade and Development. *Cybercrime Legislation Worldwide*. 2023. URL: https://unctad.org/page/cybercrime-legislation-worldwide.

[29] L. Maxmind. *GeoIP Country Database*. 2023. URL: https://www.maxmind.com.

[30] W. Webber, A. Moffat and J. Zobel. 'A Similarity Measure for Indefinite Rankings'. In: *ACM Transactions on Information Systems* (2010). DOI: 10.1145/1852102.1852106.

3

How Media Reports Trigger Copycats: An Analysis of the Brewing of the Largest Packet Storm to Date

This dissertation strives to answer whether it is possible to extract information characterizing a criminal's toolchain and identify behavioral traits in the early stages of the attack. In the previous chapter, we analyzed the changes in the scan landscape over the period of nine years. We concluded that the behavior of scanners has changed, rendering some rules-based detection approaches ineffective in identifying scan traffic. In addition, we have shown that common detection methods, such as blocklists, suffer from varying effectiveness, potentially providing a false sense of security to defending parties who use them. This chapter presents a case study analyzing the changes in the scan landscape over a short period of time surrounding the publication of a new vulnerability.

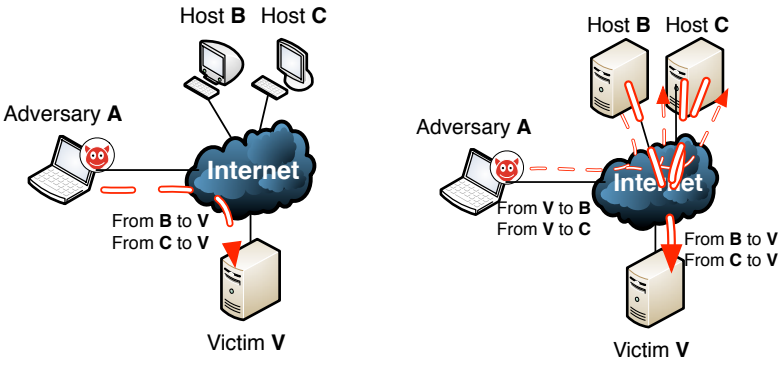
New vulnerabilities with varying degrees of severity are reported and published daily, likely influencing cybercriminals' behavior. In this chapter, we analyze the scan landscape surrounding the release of a proof of concept allowing the identification of vulnerable Memcached servers. Servers running the Memcached service were abused to launch one of the biggest DDoS attacks, 1.3 Terabit per second, against the GitHub code-sharing platform. Analyzing the scans surrounding the publication of the proof of concept allows us to study the short-term behavioral changes of cybercriminals when a new opportunity is presented to them.

In our work, we collect the scan probes criminals send searching for vulnerable servers identifying the different toolchains scanners use. Immediately after the publication of the vulnerability, we see a massive increase in scan traffic, indicative of an expected fast reaction from cybercriminals, showcasing the importance of quickly patching vulnerable systems. Analyzing the scan traffic reveals that the content of scan probes can be used to distinguish between toolchains and thus identify scan campaigns, allowing for the behavioral analysis of scanners. Our analysis also shows a general trend of cybercriminals initially using their own tools and later switching to published proofs of concept. The analysis performed in this chapter showcases the ability to identify toolchains allowing for the characterization of cybercriminals' behavior at the early stage of an attack.

3.1. Introduction

The past twelve months changed the perspective on denial-of-service attacks. Over years, until 2017, the magnitude and method of large scale distributed DoS attacks remained relatively static, with noteable large incidents typically reaching a maximum volume of 300 - 400 Gigabit per seconds, abusing well-known vectors such as DNS or NTP amplification. The proliferation of Internet-of-Things (IoT) devices, which frequently contain improperly secured remote access and administration capabilities, began to change these established practices. The first widely distributed IoT-based attack in August 2017 overshadowed past records by about a factor of two, only to be eclipsed by another IoT-based DDoS a few weeks later allegedly breaking the Terrabit/second mark. In contrast to previous DoS campaigns, these IoT-based attacks leveraged the power of millions of small connected devices that together could generate a large packet flood. In late February 2018, this record was again eclipsed, when various DDoS mitigation providers reported attacks that began to abuse the memcached service to reach combined packet floods of initially 1.3 Tbps, and later 1.7 Tbps.

Denial-of-service attacks may be classified in two types as shown in figure 3.1. In resource depletion attacks, adversaries overwhelm a victim by initiating many connection attempts that will dry up a limited but critical resource for service delivery. This ranges from buffer space in the operating system network stack for IP packet reassembly to a maximum number of available connections, which is exploited in the most common type of resource depletion attacks on the Internet, TCP SYN floods. In the second type, volumetric attacks, the adversary aims to generate a very large traffic volume to saturate the victim's uplink. As this would otherwise come at a great expenditure for the adversary, this is normally done through exploitation of a third-party service, from where he can request some data. By spoofing the source IP address of the request to that of the victim, the response is then delivered to the victim. When exploiting Internet services



(a) Depletion (b) Amplification

Figure 3.1: DoS attacks may be classified in resource depletion and volumetric attacks. For economics, these packet floods are typically realized using amplification attack vectors.

with a high difference between request and response size, these so-called amplification attacks may be generated using comparatively small effort. As the memcached service is meant for caching media files, it features a very high amplification ratio and thus lets adversaries initiate a very effective DDoS using comparatively low effort.

In order to prepare and counteract coming attacks, it is necessary to better understand the ecosystem of actors behind DDoS attacks, assess their capabilities and trace their techniques and tactics. This so-called threat intelligence can hence be used to plan and place DDoS migitation techniques, design more effective countermeasures and predict future adversarial capabilities and techniques. In this context, this paper makes three contributions:

- We find that the memcached amplification DDoS attacks are preceeded by a systematic search for vulnerable servers weeks before the attacks, at least for several actors.
- We show that there are subtle differences in the attack packets in terms of packet header and payload, which can be used to fingerprint actors and trace their evolution over time.
- We demonstrate that the bulk of the hosts engaged in testing for memcached services can be attributed to copycat behavior. Entering the scene after the media coverage of the attacks, they utilize a proof-of-concept implementation to realize their objective.

The remainder of this paper is structured as follows: Section 3.2 provides an overview of previous studies that investigated past DDoS incidents. Section 3.3 describes the purpose and setup of the memcached system, which was abused in this specific incident. Section 3.4 briefly outlines the data set and data acquisition for our investigation. Section 3.5 presents our findings on the dynamics of memcached probing activities. Section 5.7 summarizes our work.

3.2. RELATED WORK

As we have discussed in the introduction, the current memcached attacks are actually the result of a long evolution of DDoS activities. The idea of creating packet floods through amplifying reflectors has actually been known for a long time, and was originally described by Paxson [1] based on the DNS and Gnutella services. While there exist a number of countermeasures such as deactivation of UDP for high amplification services or network wide enforcement of BCP38, misaligned economic incentives and speed requirements have deterred these initiatives in practice.

In 2014 Rossow published [2] a detailed work presenting different DDoS amplification attack vectors. He explains which UDP protocols can be used to perform an attack, as well as the expected amplification factor. At that time, the highest measured amplification was based on the network time protocol (NTP) at an amplification factor of 4670. Like our study, Rossow also leverages two darknets of size /17 and /23, to monitor the scanning activity for vulnerable protocols in the wild. While he identified highly lucrative services for abuse, he did not come across evidence that all of them are actually

being tested for, especially the NTP protocol which has the highest amplification factor. Thomas et al. ran a comparable study [3] and tracked scanning activity for various candidate services on a /28 subnet for a duration of 3 years after the publication of [2]. They do see an uptake of the services Rossow described as attractive targets, and further note that specific services are not continuously targeted but appear in traffic surges.

Until now, very little information is available how actors engaged in DDoS attacks find exploitable targets and use them in an attack. One noteable study is the work of Santanna et al. [4], who have analyzed the ecosystem of booter services. They find that there is cut throat competition between different services, but when they ordered attacks on themselves for investigative purposes, most booters did not exceed an overall volume of 7 Gbps. Santanna et al. report that actors running a booter service need to be highly competitive, and thus would need to take any advantage over other services if they can perform more powerful attacks with less infrastructure. Thus, if a DDoS service can obtain more amplification it would mean that it could make bigger attacks happen or use less resources to perform more "small" attacks.

Recently Krupp et al. [5] presented a honeypot system for amplification attacks. Their amppot service offers protocols which are prone to DDoS amplification attacks, and tracks which actors exploit certain services to get a better understanding of the DDoS dynamic, allowing them to attribute the attacks to booter services.

While the memcached system is neither new nor free of known exploiteable vulnerabilities [6, 7], this service has not been picked up until now by actors for distributed denial-of-service attacks at production scale until the events in Feburary 2018. The following is the first study investigating the behavior of the actors exploiting this new service for amplification attacks.

3.3. The memcached System

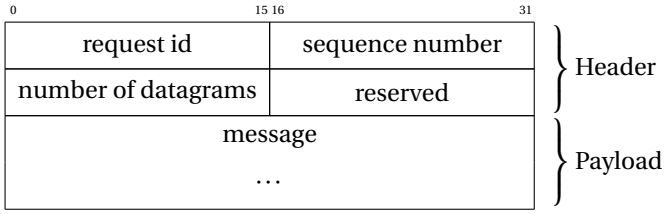
The goal of the memcached software [8] is to speed up data requests a client would normally make to a database. Instead of for example performing a lookup for an image embedded on a web page at an SQL server which would then be fetched from comparatively slow disk, the client could instead request the file from a memcached server which would serve it directly from fast system memory, and only if the file is not available fall back to the slower primary data source. If the available cache at a server runs out, the least recently used item is displaced from memory.

Clients find and request a resource from the memcached server based on a hashed identifier. For better scalability, multiple memcached servers can each be assigned a part of the cached data, but these servers do not communicate with each other, for example to distribute and reallocate shards. Memcached essentially provides an opportunitistic (distributed) key-value store. For security reasons, these setups should be placed within a separated, trusted network, as clients can interact with the caches and obtain data in an unauthorized way, a threat vector which was outlined by [6].

The memcached server fulfills all requirements to be an interesting target for amplification abuse, as it is meant to deliver media files at fast pace to speed up Internet sites. Common to many protocols designed for fast turnaround, requests to a memcached

server can be issued by UDP[1] in addition to TCP, which reduces latency through omission of the transport layer handshake and was suggested by the developers as a preferred mode according to the protocol manual [9] as the number of TCP connections can cause scaling difficulties in large installations. This however allows the service to be misused by an adversary by spoofing the requesting IP address to that of the victim. An abuse of memcached is furthermore very attractive, as the service can feature a high amplification ratio, potentially delivering large files in response to a small query.

While the UDP protocol of memcached is similar to the TCP protocol (the main difference is a simplified header in UDP), we will in the following only briefly outline the UDP version as it is the one most easily abused. Knowledge of the format of memcached requests is important to understand the fingerprinting techniques we will introduce in the section 3.5. The figure below displays the general format of the memcached protocol message [9]:



The header of the protocol consists of four 16-bit words. The request id is an identifier the client uses to match query and responses, which is necessary due to the connectionless nature of UDP. The specification states that the request id should be randomly selected by the client at the beginning of a session and then increased by one for each request. An initial value randomization is however not obliged by the protocol so the client is free to always start at 0. The sequence number is to enable larger responses that exceed the length of a single UDP packet. Stating a number between 0 and $2^{16}-1$, it marks the position of the current UDP in the response or query. The third word lists the overall number of datagrams in the request or response. The reserved field should always be 0.

The header is followed by a payload field, which contains either (part of) the requested resource or a control command sent to the server. The command `status` will return the status of the server, while `version` fetches the memcached server version. The request `get a b c d e f` will retrieve files associated with any of the keys *a*, *b*, *c*, *d*, *e* or *f*. The end of a response or query should always end in a `\r\n`, it is the symbol used to indicate the end of the message.

3.4. DATA COLLECTION

Data source for this study is a network telescope of three partially populated /16 networks. All data directed at the unused IP addresses from these ranges is logged, which

[1] Until version 1.5.6, released in response to the incident described in this paper.

provides an insight into port scanning, probing of available systems, as well as information about current attacks on the Internet from backscatter. As the monitored ranges contain no clients, the log files are entirely void of any user traffic. This means that this monitoring method creates a very clean data set of only adversarial traffic, and simultaneously eliminates possible issues about the privacy of users.

The dataset used for this study spans from 14th of August 2017 until 1st of April 2018. Over this time frame, we have recorded 31,278,705 packets directed to memcached ports at 130,000 monitored IP addresses, which originated from a total of 38,383 IP addresses. In order to track the availability of memcached services and the response of the ecosystem to the attacks, we relied on two UDP 11211 port sweeps by Rapid7 made available through the scans.io platform. As the scans.io data did not extend beyond the initial attacks, we performed one additional port sweep, sending one packet querying the version number to potential memcached servers to verify whether they are still present or a new instance has spun up. For this procedure we tested the IP addresses in random order to minimize the impact on an AS at any given time. While a single packet is unlikely to cause issues and this was a one-time measurement, an opt-out protocol was in place for network owners to deregister from measurements in the future.

3.5. Dynamics of memcached Actors

After the discussion of the memcached protocol and our data collection procedures, this section will describe the behavior of actors searching for and trying to exploit memcached servers for denial-of-service attacks. Being a spoofed amplification vector, the aggregate traffic of the denial-of-service attacks themselves would be visible only at the victims' sites or at a DDoS mitigation provider. [10] provides a brief write-up of the specifics of the attacks.

In order to execute the DDoS, the perpetrators would however first need to know a large number of memcached servers to exploit. Adversaries would typically accomplish this by scanning the Internet for machines responding to memcached's well-known service port TCP/UDP 11211, and in the following we will report about these reconnaissance activities and exploitation attempts across some 130,000 addresses we continuously monitor for this and other attack traffic. As an adversary is trawling through the Internet to find vulnerable services, we can assume the source IP address of the connection request to be authentic, and either belong to the adversary or to a host he has compromised as otherwise the perpetrator would not be able to collect any responses from exploitable targets. This allows us to quantify who and how many actors are performing reconnaissance for this service and the techniques they use during their search.

3.5.1. A DDoS Attack Triggering a Scan for DDoS

While news about these attacks was first covered in the media on Feb 27th, the attacks exploiting memcached servers at a large and systematic scale did not start there. Figure 3.2 shows a historical account of memcached protocol traffic directed towards our monitored IP ranges between August 2017 and April 2018 aggregated in hourly intervals, on the top for the total number of packets, on the bottom the total number of unique IP

addresses involved per hour, the red line in both plots shows the date of the first news release.

As we can clearly see in the graphs, actors have scanned and attempted to exploit memcached servers already for months before the emergence of the incident, by sending requests that would trigger responses with a high amplification factor. Although a similar volume of exploitation attempts was recorded at least twice before, four days before the media coverage the total amount of memcached attack traffic massively increased and remained at that level for the duration of an entire month. When looking at the volume of adversary IP addresses over these 8 months, an entirely different picture emerges. Aside from one peak in activity one month prior to the attacks, few people have ever probed our IP ranges for evidence of unsecured memcached servers. With the public discussion of the attacks this however changes drastically, and the overall volume of attacking IP addresses increased by an order of magnitude and remained a continuous activity until the end of the study.

We see that following the February 27th public news break, a lot of actors jumped on the band wagon. In essence, the media reports thus triggered a massive influx of new and previously unseen actors, copycats who are then scanning the Internet and trying to find memcached servers they can exploit themselves.

3.5.2. RECONNAISSANCE AND ATTACK CAMPAIGNS

From figure 3.2 we see that already long before the February attacks, there were continuous attempts to find exploitable memcached servers. These tests originated from only a handful of IP addresses, what is even more astonishing is that we find that adversaries frequently require extensive trials and iterations to send memcached commands that are syntactically correct and would trigger valid, useful responses for an attack. Few IPs actually start out with the right commands to use.

When comparing the top and bottom plots in figure 3.2, we furthermore see that the reconnaissance campaigns prior to the massive DDoS were quite different in nature. We can observe two intense traffic peaks as early as September and October 2017, similar in size to the overall probing activities after the public reports, which were generated by a single IP address located in China. Based on the test progression through our IP ranges, the two scan campaigns appear to target the entire Internet, and from the beginning show a good understanding of the memcached protocol.

The tallest spike in the bottom plot embodies actually an actor with entirely different characteristics. As we can see in the figure, the surge in unique IP addresses involved in a scan is not followed by an equal surge in attack traffic. In order to stay undetected from intrusion detection systems which typically flag IP addresses as malicious if their traffic exceeds a predefined threshold, this adversary is distributing the scan for memcached instances over a total of 125 IP addresses. In this reconnaissance, each address would only send five probe packets in total, before the next IP address will continue the scan within the next second. The IP addresses are distributed over multiple autonomous systems and countries with tight coordination, which indicates some level of sophistication and determinism on behalf of the attacker. Surprisingly though, this sophistication does not extend to the specification of the memcached protocol, as each of the involved hosts malforms the application protocol packet in exactly the same way. Besides these two

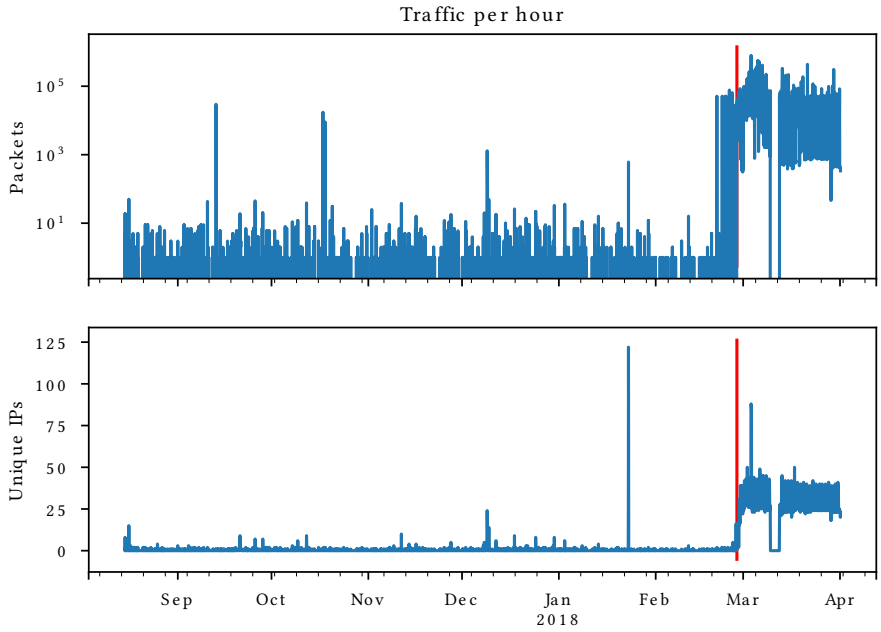


Figure 3.2: Aggregated traffic directed at TCP and UDP ports 11211 between August 2017 and April 2018. The top part shows the number of packets, the bottom part the total number of unique IPs involved in the reconnaissance.

adversaries, most of the pre-incident scanning for memcached remains unsystematic, and the bulk of the activity is only triggered with the influx of additional actors after the media coverage.

3.5.3. Attack Strategies

If an adversary would like to test the presence of a service at a TCP port, he could negotiate a connection and test whether the service would respond to a query as expected. More commonly however, an attacker would only go through the first part of the negotiation and assert the presence having received a response to the initial TCP SYN. For UDP, there is no such negotiation that would provide this information, thus in order to test whether a service in question is actually running at a given port, it is necessary to send a valid application protocol message in UDP.

As adversaries attempt to exploit the UDP version of the memcached protocol as it allows response redirection, our IP ranges hence do not just receive connection attempts, but actual application layer data from the remote parties. We have briefly mentioned above that packets frequently appear malformed and not every type of command is suited to elicit a usable response from an exposed memcached server, these kinds of artifacts allow us to understand the strategy each party pursues.

While the 34,000 IP addresses we have identified as testing for memcached sent a total

of about 16,000 different commands, there are a number of clear favorites people are trying. Overall, 10 commands account for 97% of the total traffic by packets. Figure 3.3 shows an evolution of the activity of these 10 most popular commands over time, aggregated by the total number of unique IP addresses that use them. As we can see in the graph in brown, the command `stats`, which returns a listing of memcached's system status variables, is predominantly used throughout the reconnaissance activities, but only dominates in the wild *after* the reporting of the incident in the media. The other main commands (by volume) originate only from a handful of IP addresses, which again underlines that reconnaissance for memcached is primarily driven by a small subgroup.

When we look at the evolution of the main command `stats`, we immediately notice two main surges in activity. On the first day of the media coverage, somebody implemented and posted a proof of concept (PoC) on pastebin (pastebin.com/GZcekqLz), which tests for the presence of a server using a fixed UDP payload by requesting statistics from the memcached server. The software can further be identified in the data as it scans through IP ranges in a distinctive way. A few days later, while the attacks continued and an increasing number of media outlets were reporting, another proof of concept appeared on pastebin (pastebin.com/ZiUeinae), which used the same UDP payload but has a different target scanning behavior, and was in contrast to the first PoC not suitable for convenient scanning of a large number of IP addresses. Curiously though, when we traced which articles linked to the two pastebin buckets, we found that media articles actually more often referred to the second, less capable PoC.

The publication of both PoC results in clear visible spikes in reconnaissance activity in the Internet. We see that after the wide media coverage, new actors seem to adopt the tool that was linked in the reports, a clear characteristic of copycat or script kiddie behavior. Very shortly afterwards however, much of the second surge disappears and a core group remains that continue their scanning activity until the end of the study one month later.

Horizontally across the graph, we can see a line of dots reappearing in regular intervals. This small group can be attributed to shadowserver by IP ownership and rDNS lookup. This set is characterized by a separate command, as the scanners make a mistake in the memcached payload. As discussed in section 3.3, a memcached command must be terminated by `\r\n`, this entire group however sends an additional `\0`, the null character, at the end of each command. This mistake can most easily happen when the tool scanning for memcached was implemented in C, and designed to append the fixed payload string to the socket data stream. Similar artifacts can however be found across many source IP addresses and many types of implementations leave distinctive fingerprints which allow clustering of sources and partial attribution to specific tools. In figure 3.3, we can identify two faint purple dots in the first and second week as well as a peak in the third week of March. These activities, which again stand out by unique implementation characteristics, point back to the Rapid7 scan for vulnerable memcached instances, which is one of the datasets used for this study.

3.5.4. EVOLUTION OF ADVERSARIES

These mistakes in how to format memcached packets however do give us the opportunity to consolidate individual scans to a likely common origin, especially when the

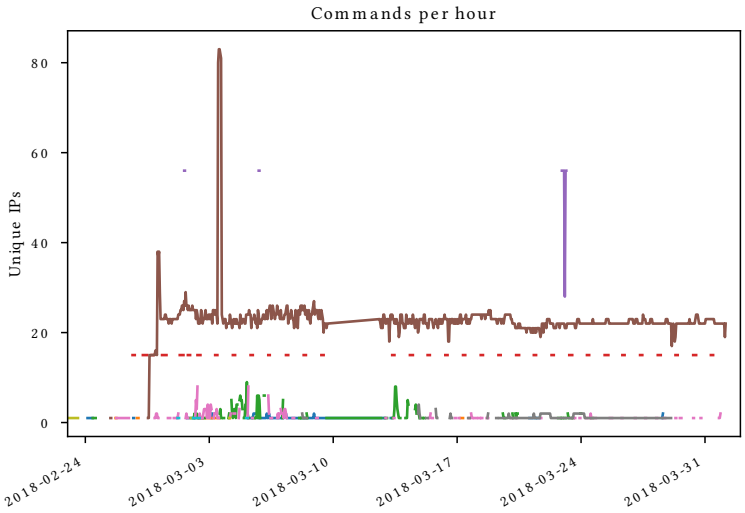


Figure 3.3: Activity of the 10 most popular memcached commands over time by the total number of remote IP addresses.

campaign appears synchronized in time and behavior. It furthermore provides us with a leverage to investigate how actors evolve their understanding of memcached over time and consequently also adapt their tooling.

Indeed, we see many attackers change their techniques during the observation period. Frequently, we detect that the tooling initially contains some mistakes in the protocol, for example header fields are incorrectly set, a wrong command is sent or improperly terminated and formatted. A very noticeable evolution that appeared in late March belongs to an actor who mistakenly sends malformed packets and instead of originating a memcached query from a client, attempts to connect a memcached server to a memcached server. As the protocol is designed as a client-server exchange, this however does not work and the activity soon after stops.

As discussed in section 3.3, there are some header fields which can be arbitrarily set by the client while others require a very specific content. When participating in large scale scanning for vulnerable systems, it is much more economical to inject a pre-made payload towards many destinations than freshly crafting a new one for each trial. In result, we see for example the same request id reappearing from single origins to a large number of IP addresses in our range. Furthermore, some actors pick valid but unusual or invalid content for the remaining fields, which in combination with the commands are so characteristic to the actors that it allows us to fingerprint sources and their evolution over time. While as stated above most actors do not send the correct command at first, we find that they adapt in time and converge to the same set of valid and productive com-

mands. This process is surprisingly quick, most sources manage these adaptation steps with a turnaround time of just a few hours. It is however astonishing to see that a large number of sources perform large scale port sweeps with the wrong command and rescan with each command update, rather than first test locally for the right configuration and then farming the scan out.

Figure 3.4 shows such a command evolution exemplarily for 8 IP addresses, which replace their strategies over the course of several days but ultimately converge to the same behavior. This progression of commands is visualized in figure 3.5 for the entire adversary space, but due to the data volume and tens of thousands of transitions is aggregated to the percentage of overall traffic volume a particular command is used on a given day. The color of each bar denominates a specific command. As we see in the graph, the commands used in the exploitation traffic go through several rounds of evolution, with clear transitioning moments at the public disclosure and the first PoC until the bulk of traffic ultimately arrives at the same solution.

3.5.5. RECONFIGURATION OF MEMCACHED SERVERS

As the abuse of a memcached server also creates an annoyance for the operator of the server, one would expect that after such a major incident and given the expectation that similar attacks are soon going to follow, those who operate an insecure server would change their setup. In fact, with the emergence of the attacks, memcached released version 1.5.6 which by default disables its UDP service, thus removing the easy exploitability of the system.

In order to test this hypothesis, we are comparing two datasets from Rapid7 provided to scans.io, which list IP addresses of open UDP 11211 ports on 1st and 5th of March

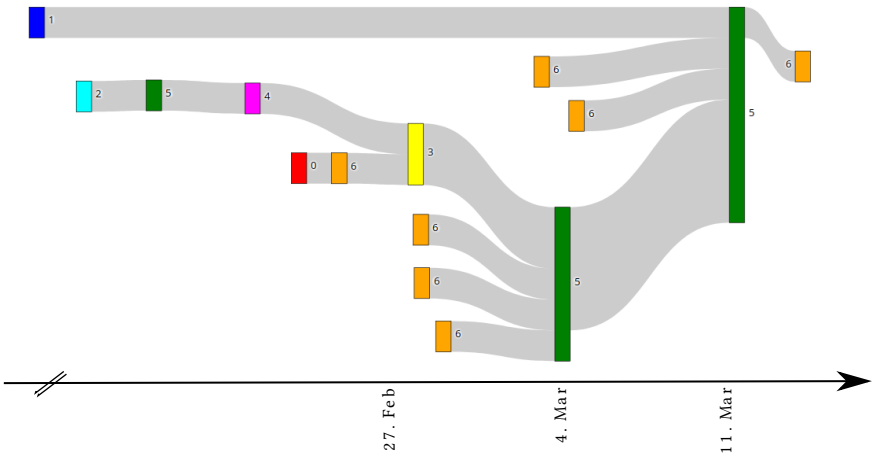


Figure 3.4: Exemplary command conversation of a set of IP addresses over time. Each strain symbolizes the progression of an IP, the color the command in use at a given moment.

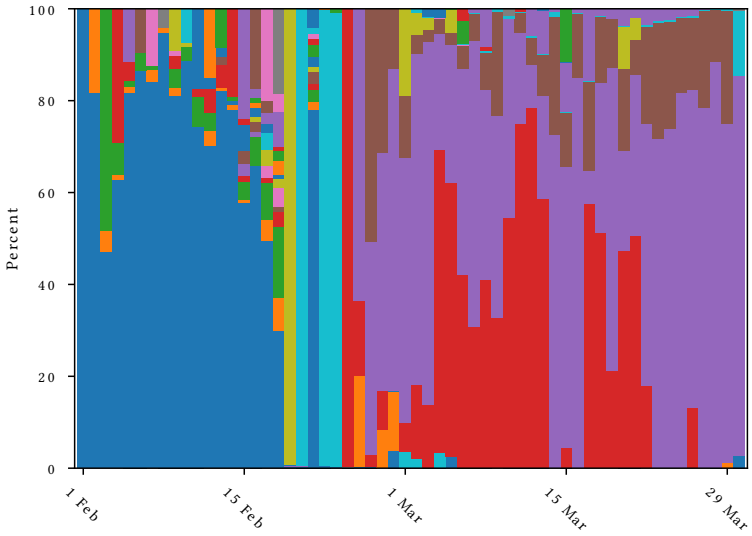


Figure 3.5: Transition and consolidation of used commands during Feburary and March, shown in percentage of packets per day. The color indicates the command in use.

2018. This enumerates the landscape just after the emergence of memcached abuse, once before and once after the surge of activity from the widely available PoC. To track the long term change in either reconfiguring their setups or operators updating to 1.5.6, we redid the scan exactly one month later on the 5th of April.

Figure 3.6 shows how the total volume of memcached servers changes throughout the lifespan of the attack. The two public data sets at the beginning of the DDoS campaigns, one public scan by the security firm Rapid7 indicated in yellow and a list of vulnerable servers provided together on Pastebin with the PoC indicated in blue, list a total of 28120 servers open for exploitation. At a second scan 4 days later, about half of the targets from the yellow group had vanished, while from the pastebin list only one had been closed. Surprisingly, 5128 new exploitable memcached servers appeared on the Internet within this timeframe. Within a month after the incident, 83% of the exposed servers had been closed down, deactivated the UDP socket or had been updated.

Those which remain, seem to be heavily geographically clustered. The top 5 of locations by country is dominated by China, where a whopping 37% of vulnerable instances can be found, followed by 12% in the US, 6% in Russia, 3% in Vietnam, and 2% in Brazil. Also when normalizing this finding by the total number of registered IPs in these countries or the number of inhabitants, this geographic bias remains. While it seems that the IPs originally identified as vulnerable do react within a month's timeframe, it is troublesome that actually additional, new vulnerable hosts are introduced over time. This is

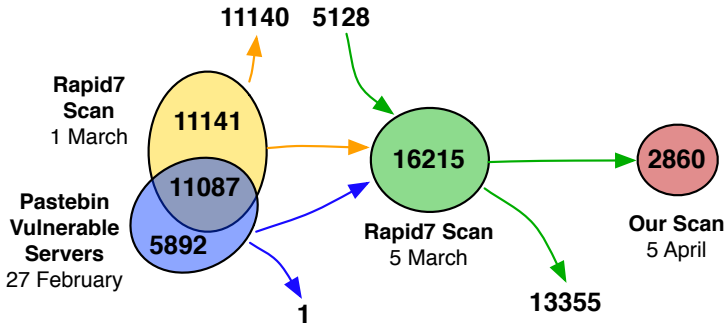


Figure 3.6: Reconfiguration of exploitable memcached servers in public datasets in response to the DDoS attacks.

especially curious as with the memcached version that was released on Feb 27th, the vulnerability is by default switched off, which would indicate either a deliberate reactivation or that new installations draw from outdated software repositories.

3.6. CONCLUSION

Before a service can be abused for amplification attacks, an adversary must first know about the presence and location of such services. In this paper, we have analyzed the reconnaissance activities on the Internet for memcached servers, which were used to launch the largest DDoS attack to date, 7 months prior to the attacks and in the month thereafter. We find that the attacks are preceeded by some systematic reconnaissance before and actors frequently make mistakes in this process and evolve their techniques for the final goal. We also find that the media coverage about the attacks resulted in a major surge in new actors into the ecosystem, copycats or skript kiddies who adopt ready-made tools and attempt to compile lists of vulnerable machines on their own. Within a month of the attacks, we see that the vast majority of originally vulnerable services are removed or reconfigured from the Internet, we unfortunately also find that even after the attacks new exploitable machines are still added to the Internet. As the latest memcached release has removed the default option for UDP, it should thus be a matter of time before the installation base is sufficiently small to deter adversaries from this attack vector.

References

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

[2] C. Rossow. 'Amplification Hell: Revisiting Network Protocols for DDoS Abuse'. In: *NDSS*. 2014.

[3] D. R. Thomas, R. Clayton and A. R. Beresford. '1000 days of UDP amplification DDoS attacks'. In: *Symposium on Electronic Crime Research*. 2017.

[4] 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: *International Symposium on Integrated Network Management (IM)*. 2015.

[5] J. Krupp, M. Karami, C. Rossow, D. McCoy and M. Backes. 'Linking Amplification DDoS Attacks to Booter Services'. In: *RAID*. 2017.

[6] Sensepost. 'Cache on Delivery'. In: *Black Hat*. 2010.

[7] I. Novikov. 'The New Page of Injections Book: Memcached Injections'. In: *Black hat 2014*. 2014.

[8] B. Fitzpatrick. 'Distributed caching with memcached'. In: *Linux Journal*. 2004.

[9] M. protocol. https://github.com/memcached/memcached/blob/master/doc/protocol.txt. 2017.

[10] M. Majkowski. *Memcrashed - Major amplification attacks from UDP port 11211*. Tech. rep. Cloudflare, 2018.

4

Clustering Payloads: Grouping Randomized Scan Probes Into Campaign Templates

This dissertation's goal is to answer whether it is possible to extract information during the early attack stages allowing for the characterization of criminals' toolchains and identifying behavioral traits. In the previous chapter, we presented a case study demonstrating the extraction of information characterizing a criminal's toolchain. In this chapter, we continue the analysis of the first stage of attacks by studying UDP scan probes and present a methodology for clustering probes, allowing the study of cybercriminals' behavior.

While analyzing Memcached-related scan probes, we observed cybercriminals using probes containing different payloads to uncover servers running the same services. In addition, we observe cybercriminals randomizing their probes in a likely attempt to circumvent blocklists and intrusion detection systems using fingerprint-based detection mechanisms. The randomization and use of different probes to identify servers running the same services is a behavior exhibited by cybercriminals targeting various services running on top of the UDP protocol. Cybercriminals customizing their custom templates to craft scan probes allows defending parties to use them to study cybercriminal behavior. Therefore, we developed a protocol-agnostic methodology for clustering scan probes and regenerating the templates used by cybercriminals to craft scan probes. We validate our methodology using scan traffic sent towards commonly scanned ports and generate templates matching 96% of all scan traffic. Our generated templates facilitate the identification of scan campaigns. More so, by analyzing our templates, generated from DNS probes, we show a general trend of cybercriminals replacing one type of scan probe for another. Our proposed methodology allows us to identify and analyze cybercriminals' behavior during the early stages of an attack.

4.1. Introduction

Network scanning is often the precursor to a subsequent attack, and is used by adversaries to discover active hosts, exposed and vulnerable services, and potential weaknesses in a network [1]. As such, network scans provide useful information to defenders as it

can alert them to the type of services and vulnerabilities the adversary is looking for, and provide the opportunity to strengthen and complement defenses long before a compromisation attempt will commence [2].

Two important and diverging developments have drastically changed the scanning landscape over the past decade. On the one hand, high-speed scanning tools such as Masscan or Zmap enable anyone to systematically trawl through the entire Internet from a single PC in a matter of hours and even minutes on a 10 GbE link. While these activities will be immensely noisy and obvious to anyone, they allow adversaries to keep track of hosts and open ports, for example to keep up to date information about DDoS amplifiers [3]. On the other hand, we observe a more widespread application of stealthy scanning, where adversaries are probing the IPv4 address space at a very low rate from thousands and even tens of thousands of vantage points [4, 5]. With inter-arrival times of scan probes in the order of hours or days, it is difficult for organizations to keep track of these scan campaigns and relate those IP addresses participating in one to each other with existing tools such as intrusion detection systems or SIEMs [6].

Previous work has mainly focused on detecting scans and scanning campaigns in TCP, as this transport protocol accounts for the majority of scanning traffic [7, 8]. Although TCP scans are more prevalent, scanning for devices running services on top of the UDP protocol can be lucrative, as these services are on the one hand typically abused in amplification attacks, and on the other hand it is easy to realize high-performance, low latency protocols on UDP, as exemplified by the transition of HTTP3 on UDP. As the user datagram protocol is handshake-less, this offers a different, novel angle to identify and interpret scan campaigns. While in TCP a service (or honeypot) needs to be active to complete the TCP handshake before any actual data is exchanged, in UDP the scanner has to send some payload already in the first probe packet. As a service might not respond at all unless some correctly formatted and valid request is sent, the adversary cannot simply conduct surveys of open ports but necessarily has to reveal what he is looking for, potentially even to the extent of including some specific exploit in the probe packet. This is, for instance, the case for a vulnerability found in Juniper routers [9], depending on the sent UDP payload, an attacker can reveal whether he has the intent to execute code or perform a DoS attack. This allows us to quantify port scanning and characterize scanners based on what they are trying to do and relate them based on common intent.

This angle has received almost no attention to date, and in this paper, we demonstrate that it is possible to automatically extract common scan traffic into user-interpretable patterns across any UDP protocol, human-readable or binary. Defenders can, therefore, directly assess how a particular service is targeted by an adversary and in case of a new or unmitigated exploit rollout controls. Even though stealthy scanning distributes scan probes across many hosts and over extended periods of time, any of these scan probes still has to use the same or similar packet payload. As we are matching probes based on commonalities, our approach is effective in identifying such distributed campaigns. With this paper, we make the following three contributions:

- We propose a protocol-agnostic method that can extract commonalities across scans in unsupervised learning, and translate them into understandable templates that help the defender understand the mechanisms and potential aim of the scan. We show that the method is effective for both static scan probes as well as those

that randomize header and payload components on a per-destination basis.

- We demonstrate that such templates effectively track developments for individual scanning groups as well as the scanning landscape. For instance, the template analysis clearly identifies how DNS scanners shifted their attention from the ANY to the TXT query over time.

- As scan campaigns by definition will show some form of commonality, our template method allows us to also identify scan campaigns by matching sources that send an instantiation of an abstract template. We show that this technique can match highly-distributed scans with very low inter-arrival times and link addresses even if they only occasionally emerge.

4.2. RELATED WORK

Lockheed Martin shows that scanning the Internet is often the first step taken to set up a cyber attack [2]. Scanning the Internet for devices can be a lucrative endeavor for cybercriminals. Researchers scanning the Internet show that 13.8% of 3 million devices use standard login credentials [10], making them easy targets for cybercriminals. In order to get a better understanding of attackers scanning the Internet for vulnerable devices, researchers have studied scanning behaviors. Research [4, 11–16] shows that scanning activity can be identified and that attackers adopt different scanning tactics. Attackers target different protocols, use vertical and horizontal scanning strategies, and leverage blacklists to avoid detection. In addition, Lee et al. [17] reveal that scanners use decoy probes to hide their activities. With detecting scanning activities playing an important role in cybersecurity, Coudriau et al. [18] developed a visualization tool, and Iglesias and Zseby developed a method to identify new scanning campaigns [19].

By monitoring and analyzing the tactics used by attackers, researchers have shown the capability to identify the tools used by scanners [7, 13]. The tools are identified using time series analysis or by analyzing the header fields of the received scanning probes. Further studies are conducted showing that scans performed by malware and botnet can be identified [5, 8, 20]. The identified scans performed by malware offer an insight into which services botmasters are targeting. Antonakakis et al. [21] show that the Mirai botnet has increased the number of destination ports scanned during its lifetime in order to increase the number of devices it can infect. Next to monitoring botnet evolution, analyzing the scans performed by botnets can contribute to less evident studies such as measuring the IP churn in autonomous systems [22].

Besides monitoring the scanning activities of botnets, researchers have analyzed distributed scanning campaigns in general. Early research performed by Gates [23] shows that by measuring the overlap in scanned IP addresses, distributed scanning campaigns can be identified. Subsequent research [7, 24, 25] utilizes properties found in the TCP header to identify large distributed scanning campaigns.

The works above show that it is possible to identify distributed scanning campaigns. However, most research only analyzes scans towards services running on top of the TCP protocol. Although 86.3% of the scans are directed towards TCP ports [7], services running on top of the UDP protocol are prone to be abused by cybercriminals setting up

DDoS attacks. Ghiette and Doerr [26] show that after the Memcached vulnerability was exposed, an explosion in scanning activity followed. The UDP protocol is connectionless, forcing scanners to send scan probes containing valid payloads. In the following work, we leverage the necessity of scanners to use probes with valid payloads and introduce a protocol-agnostic method to analyze scan traffic. Using our proposed method, we offer an insight into the UDP scanning landscape, showing a change in scanning behavior over the past seven years. Similar to [7, 24, 25] which uses TCP header values, we show that the UDP payloads can be used to identify distributed scans.

4.3. PROTOCOL-AGNOSTIC TEMPLATE EXTRACTION

In order to make sense of scanning data, it would be useful to identify common patterns in scan traffic to help defenders understand the goals of scanners even in the presence of randomization, and link origins of common scanning patterns to identify campaigns and collaborating entities. In the following, we discuss our approach to identify such patterns in a protocol-agnostic way for UDP. Before we dive into the approach, we will first briefly summarize how adversaries perform UDP scanning to motivate the requirements for template creation.

4.3.1. RANDOMIZED UDP SCANNING

The connectionless nature of the UDP protocol dictates how attackers may scan for devices running a service on top of the UDP protocol. If a UDP packet is sent to a remote host, the protocol specifies that the host returns an ICMP port unreachable notification in case no service is listening to the targeted port. Hence, upon receiving such an answer, the adversaries would know for certain that no service is present, and if no answer is sent, a UDP service should be present. In today's networks, this reasoning does not work anymore as notifications are typically disabled by operating systems and filtered by the network to prevent such scan surveys. To illustrate this, we sent a DNS query to port 53 to 10 million randomly chosen IP addresses on the Internet. 19,553 IPs responded to our query, 389,673 returned an ICMP port unreachable, while 95.9% of the probed IPs remained silent.

To determine whether a port is open or closed, scanners would need to elicit a response from a targeted service. Blindly blasting out random data would however also not provide any insight, as the software listening behind a port would probably parse incoming requests and only respond to those it understands. This thus mandates the scanner

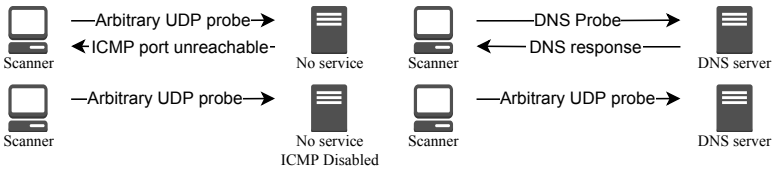


Figure 4.1: UDP scans only return results if a valid request is sent.

to craft valid protocol headers and payloads for each targeted protocol as shown in figure 4.1. A variety of scanning tools feature built-in probes for specific protocols allowing attackers to scan the Internet with little protocol knowledge, as well as allow advanced attackers to further customize the payload.

Such customization includes, for example, the randomization of certain parts of the sent header and payload, for example, the query ID in a DNS packet, which according to the Zmap documentation [27] can be effective in circumventing security measures in multi-homed systems. Creative attackers may also randomize other parts of DNS queries, such as changing the subdomain for each probe, in an attempt to evade IDS signatures or detection and filtering if repeated packet payloads are used to detect scans.

These practices drastically complicate campaign analysis, as we would not know apriori if and how scan probes would look like and how such randomization might take place, in practice these templating options by common scanning tools make each scan probe appear as unique. Previous work identified an entire range of practices, featuring basic randomized header and payload fields to complex relationships such as the encoding of IP/port information in the payload, possibly further masked with bit shifts or encrypted with session secrets [13, 28].

To visualize this heterogeneity in today's port scanning, let us look at scanning traffic collected at a network telescope containing over 65,000 unused IPv4 addresses between 2017 and 2021. Given the large scale of the telescope and intensive port scanning on the Internet today, it is sufficient to only include one month per year in our analysis. We perform our analysis using DNS and NTP scanning traffic, as we show in section 4.3.2 that they possess general traits exhibited by many protocols, yielding in a data set containing 104.5 million DNS and 620,7 million NTP scan probes that we will work with further in this analysis.

Figure 4.2 shows the distribution of unique scan probes sent towards port 53, the standard port for the DNS, in ascending order by the number of probes received. We immediately notice that the distribution of scan probes is heavy-tailed. The 28 most commonly sent DNS packets are received between 50,000 and one million times by our network telescope, while for more than 30,000 payloads, we only receive one probe. This means that randomization is a default practice and clustering identical packets in campaigns is ineffective for the bulk of scanning. Considering that our telescope ranges account for more than 65,000 IP addresses, we also see that the apparent heavy usage of those "typical" 28 payloads is not that distinctive if we receive on average 2.69 such static payloads per telescope IP per month. Effective scan detection and campaign clustering thus requires automatic detection of these packet generation templates.

4.3.2. CHARACTERIZING PROTOCOL TRAITS

In the following, we develop a protocol-agnostic procedure to extract the templates used by scanners when generating UDP probes. Currently, IANA lists over 6,000 services running on top of the UDP protocol. Although each protocol is different, there are some shared commonalities between them. We analyze the seven most commonly scanned UDP protocols [29] and identify general features that can be found across all these protocols, which we will use to design our protocol-agnostics method to handle UDP packets

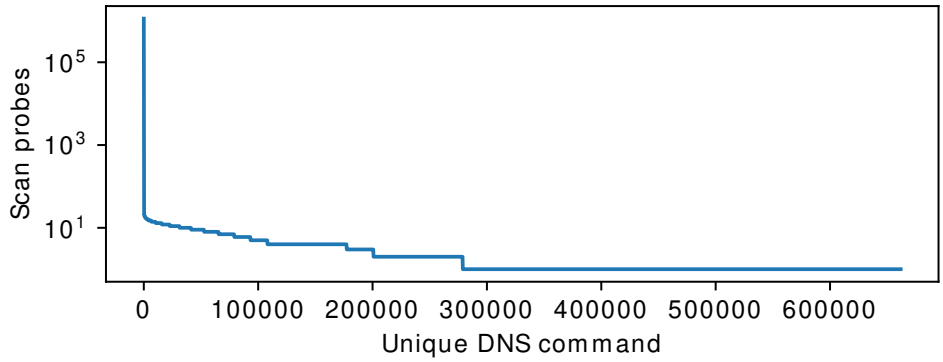


Figure 4.2: Scan probes received per unique DNS command.

in general.

(1) The first characteristic is the usage of binary or text-based data. Protocols such as CHARGEN and SIP exchange textual data while NTP uses binary data. Others such a DNS include a mix of both in their payloads. An effective method for behavior clustering should handle binary, textual, and mixed payloads equally well.

(2) The second aspect relates to the use of randomized data. As UDP is connectionless, it forces protocols to include handles to differentiate connections. Protocols often use fields with per-session randomness to do so. As an example, SIP assigns a random value to sessions, and DNS randomizes its query ID. Therefore, a clustering should be resilient to partially randomized content, if payloads are otherwise "sufficiently" similar.

(3) The third aspect relates to length, or more precisely the *lack* of length requirements. Protocols naturally differ into their packet structure and many protocols allow messages to vary in length. For instance, DNS servers will respond to a short bind command and to longer domain name queries. Similarly, SIP servers allow clients to send a user name, affecting the message length. Therefore, a method should not require any apriori knowledge and be able to relate payloads even if they differ in structure.

(4) The last feature is related to protocols allowing customized commands. Some protocols, such as NTP, are strictly defined in terms of payload content. Others, such as DNS and CHARGEN allow for arbitrary data in the payload. As we do not know "how different" a protocol typically is, this aspect should be learned from the data itself.

4.3.3. PROPOSED METHODOLOGY

We use these four general characteristics to develop a protocol-agnostic process that allows us to extract scan templates used by attackers regardless of the underlying protocol. Figure 4.3 depicts our data processing pipeline for clustering and transforming scan traffic into human-readable templates. In the following, we systematically explain each of the seven steps taken by our approach and justify our design.

(1) We begin our pipeline by randomly sampling packets from incoming scan traffic. Data sampling is necessary due to the intensity of scanning on the Internet. As state-of-

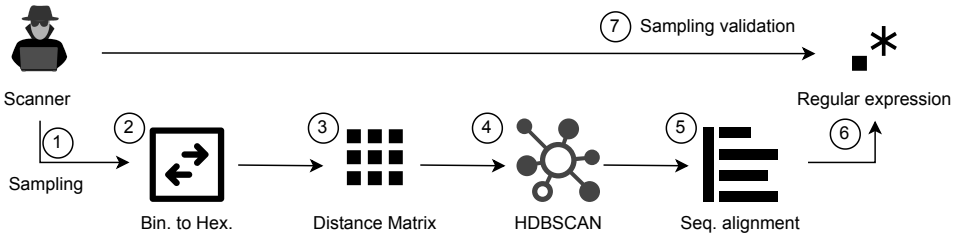


Figure 4.3: Templating process

the-art cluster generation has a complexity of $\mathcal{O}(n \log n)$ and require distance matrices of n^2, it is prohibitive in scenarios like Internet scanning where we receive millions of packets for popular UDP protocols in a given month. In this step, we randomly sample packets to a degree fitting our computational resources. In our work, we randomly selected 70,000 scan probes, as in theory, such sampling could miss out on certain patterns, the pipeline verifies the sampling in the final step and adjusts accordingly.

(2) We convert the payloads to a uniform representation. To accommodate for payloads with text and binary information, we transform all payloads to their equivalent hexadecimal string notation creating a uniform representation. A hexadecimal string notation allows us to use a distance metric that works on both binary and textual commands.

(3) We calculate the distance matrix as input for cluster generation. We utilize the Damerau-Levenshtein distance [30], which counts the minimum number of insertions, swaps, and deletions required to make two strings identical. The Damerau-Levenshtein distance may compute the similarity of strings with different lengths, which makes the algorithm agnostic to different types of protocols and the potential to introduce arbitrary protocol payloads.

(4) We cluster the sampled packets using the HDBSCAN algorithm. HDBSCAN has the nice property to cluster data with the characteristics discussed in the previous section. Furthermore, it does not require knowledge of the number of clusters to generate good results, but uses the data itself to determine a cut-off. Scanners can use templates that introduce varying degrees of randomness, for instance, a template $S1$ generating random DNS queries for subdomains of *google.com* for which the subdomain length is 5 and a template $S2$ of random queries for subdomains of *amazon.com* with a random subdomain length of 20. As the calculated Damerau-Levenshtein distances between the probes generated by $S1$ are smaller than the distances between the probes generated by $S2$, and HDBSCAN can find clusters with varying densities, HDBSCAN will form clusters grouping the commands generated by $S1$ and $S2$ in separate clusters. Additionally, we selected HDBSCAN because it can handle noise in the dataset caused by Internet backscatter.

(5) We translate the identified clusters into a common representation. We anticipate clusters to contain payloads of different lengths. However, different length patterns cannot be harmonized. Therefore, we apply multiple sequence alignment using MAFFT to the clustered probes.

(6) We unify the aligned probes in each cluster into a human-readable template representing all its members. We use the aligned payloads of the clusters to generate regular expressions, specifically as it allows to define random parts in the payloads and can handle the gaps introduced by the sequence alignment process while being easily interpretable by a human. We generate the regular expression by placing a '.' in the regular expression where characters in the probes vary, the gap characters introduced by the alignment are noted by a '.{0,n}' in the regular expression, where n is the number of consecutive gaps. Characters that do not vary throughout the probes remain the same. Figure 4.4 illustrates the template building process using two reverse DNS lookup commands. We illustrate the process using the actual commands, not the hexadecimal translation, as it is easier to understand.


```
Clustered commands
0.0.0.0.in-addr.arpa
111.11.11.111.in-addr.arpa

Aligned commands
--0.-0.-0.--0.in-addr.arpa
111.11.11.111.in-addr.arpa

Regular expression
^.{0,2}.\..{0,1}.\..{0,1}.\..{0,2}.\.in-addr.arpa$
```

Figure 4.4: Example of template extraction.

(7) In the final step, we map the remaining dataset onto the resulting templates to validate the sampling process. Although sampling is randomized, it may affect the detection rate of templates. To measure whether all templates have been identified, we identify any data that cannot be matched by the crafted regular expressions, and rerun for potentially unclassified data. In practice we find that unmatched data is negligible.

4.4. VALIDATING OUR APPROACH

In section 4.3 we introduced a method to extract commands used by scanners to probe for UDP services. Our method bundles HDBSCAN on a sampled data set and explains found clusters using regular expressions. Although HDBSCAN and regular expressions are established techniques, we are not aware of previous work demonstrating its ability to cluster scanning probes. In this section, we validate whether the methodology creates meaningful clusters and templates using domain-specific knowledge and verify that despite sampling, the template generation delivers meaningful results.

Validation data set. In the following, we validate our methodology by clustering DNS probes as this protocol uses all protocol characteristics identified in section 4.3.2, specifically both textual and binary payloads, commands can be of different lengths, and most importantly, the protocol allows for user input of arbitrary lengths, which can be randomized. To analyze the stability of the clustering itself, we cluster three datasets each spanning one month in 2018, believing that over the course of a few months the behavioral evolution of scanners should be minimal, allowing us to evaluate the stability of our cluster generation. On average, each dataset contains 10.1 million DNS packets.

Clustering results. When we apply sampling and clustering as explained before to the test datasets above, we obtain very high cluster assignment scores as shown in table 4.1. HDBSCAN allows for points to be classified as noise and assigns a P value, between 0 and 1, to each point in a cluster representing the probability of that point belonging to the assumed cluster. We see that density-based clustering is very effective for packet data, and only for less than 0.7% of all points the P value is smaller than 1. More than 99.33% of packets are clustered with high confidence across each dataset.

Evaluation of generated clusters. Table 4.1 also shows that the number of clusters significantly varies each month. Clusters are also not equal in size, the top ten most significant clusters contain between 66.73% and 69.91% of the data for each clustered dataset. We manually inspected and analyzed all of the clusters generated by HDBSCAN, to verify that HDBSCAN-based clustering of payload data generated meaningful results. As mentioned earlier, scanners can either use sent static scan probes or dynamically generate their packets. Clusters containing probes generated by a static approach are trivially explained as the cluster members should only contain identical payloads. If a cluster contains probes from dynamic generation, we need to determine whether the payloads in the cluster could have been generated using the same template. We will treat static and dynamic clusters separately below.

a) Static clusters. HDBSCAN identified 72, 53, and 59 clusters for the three monthly data sets. Of the identified clusters, 62, 43, and 45 respectively are composed of identical commands. If the clustering related identical payloads correctly, there must not be any instance in the dataset where a particular static payload from a given cluster is also part of another cluster, as these should have been merged during the clustering process. Indeed, in none of the test datasets we find any instance of such overlap.

b) Dynamic clusters. The validation of clusters containing different commands requires us to ascertain whether all commands in a particular cluster can indeed be generated using the same template. HDBSCAN forms 10, 10, and 14 clusters containing different commands for the three validation data sets. In the following, we showcase the examination of the results for the month of January, and report on the other two.

We analyze the clusters semantically by parsing the binary data and translating it into the DNS packet structure as specified by RFC 1035 [31]. The clusters identified by the algorithm contain commands that differ either by the query ID, domain name, or both. Table 4.2 shows for each dynamic cluster the number of (i) commands, (ii) unique commands, (iii) commands containing a unique domain name, and (iv) those having a unique query ID.

The first type of behavior clustered together are DNS queries with a static domain but every packet containing a random query ID. Clusters 1, 38, and 32 are examples of these, but each of these clusters target a different domain name. All scan packets in each of these clusters can be generated by a template in which all of the DNS fields are fixed except for the query ID field that is randomized for each probe. The use of such a template is recommended by the manual of Zmap [27].

The second type of behavior is where every probe queries a different domain name, but otherwise the packet structure is entirely static. Cluster 2 is an example of this, where every scan packet contains a randomly-generated subdomain all in the form of *xxx.xxxxxxxxx.wc.syssec.rub.de* where x represents an alphanumerical character.

Table 4.1: Clustering results for three months of DNS scans.

Data set	Num. clusters	Noise points	P <1	% Clustered
2018-01	72	380	10	99.44
2018-04	53	275	7	99.59
2018-07	59	457	11	99.33

Table 4.2: Dynamic clusters in the Jan 2018 validation set.

Cluster ID	Number of commands	Unique commands	Unique domains	Unique QIDs
0	119	119	-	-
1	259	257	1	257
2	1003	1003	1003	1
3	1038	1032	1032	1029
4	1115	1107	1107	1103
32	4128	4005	1	4005
38	2331	2226	1	2226
12	153	2	1	2
46	141	2	1	2

The third type of behavior is where different parts, here the domain and query ID, vary per packet. Clusters 3 and 4 are examples of this, but within each cluster the top- and second-level domain is identical. Cluster 3 contains probes for random subdomains of the *openresolvertest.net* domain, and cluster 4 for subdomains of the *openresolverproject.org*. Although both clusters use the same generator where random subdomains are composed of 9 alphanumerical characters, the pipeline treats them as separate behaviors. Although clusters 12 and 46 vary in both aspects, we find that they contain two domain names and two query IDs, and were accidentally formed in an attempt to meet the minimum cluster size of 100. If high precision results are needed, the algorithm could be rerun with lower minimum cluster sizes if some generated templates overlap. Cluster 0 does not contain domains nor query IDs, indicated in table 4.2, as this cluster contains SIP probes that were sent towards port 53. However, all these commands are correctly related as they are generated using sipvicious [32], a scanning tool specially developed for scanning SIP servers.

Using domain-specific knowledge, we analyzed all 184 identified clusters, of which 97.28% represent valid patterns.

Evaluating the regular expressions. After validating the extracted clusters, we also verify the methodologies' fifth and sixth steps for generating regular expressions. Each expression should be unique as it represents a scan template, and no expression should overlap. Furthermore, these expressions should be as tight as possible, a regular expression representing one cluster should not be able to match any command found in other clusters. As we verify the pipeline output for the 184 clusters and regular expression, we see that every expression is unique and does not match packets from any other cluster, with the exception of those two clusters we identified as incorrectly merged due to the minimum size requirement. The remaining 182 generated expressions provide a unique

representation for their clusters.

Evaluating the sampling. The final step in our evaluation is to verify that the sampling performed in the first step of the pipeline did not influence the template detection rate. We use the previously evaluated regular expressions to match probes in their respective data sets. After matching, we find that 96.04% of all the probes are mapped to a regular expression indicating that the sampling had minimal effect.

4.5. LONGITUDINAL STUDY OF COMMAND USAGE

In the previous sections, we proposed and validated a methodology to cluster and extract commands used by scanners. In this section, we apply our approach to perform a longitudinal study of the commands in scan campaigns for DNS and NTP protocols between 2015 and 2021.

4.5.1. EVOLUTION OF COMMAND TEMPLATES

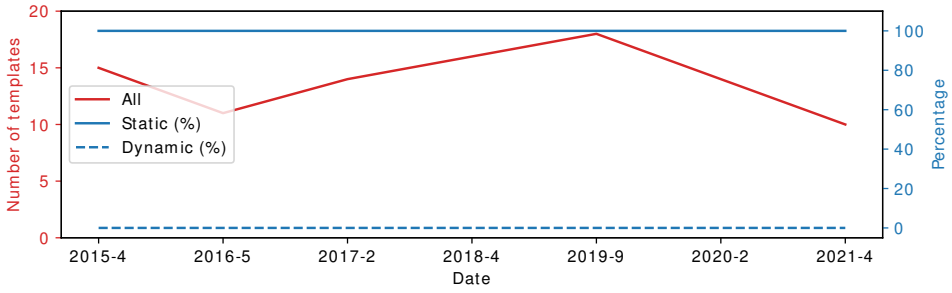
In section 4.4, we showed that attackers use templates to craft dynamic and static probes based on templates, each with its own advantages and drawbacks. In this section, we use our methodology to obtain which types of templates have been used over the years and how the approach to scanning changed over the course of time.

To analyze the development of scanning practices over time, we draw on the dataset of scan traffic collected by 65,000 telescope IPs over the last seven years, and let the algorithm extract the templates used by scanners in each time slot. Figure 4.5a shows the percentage of static and dynamic NTP commands as well as the number of unique commands in use, figure 4.5b depicts the same for DNS. As we see in the graph, everyone who is scanning for NTP services always does so by directing the same scan payload towards different destination addresses[1]. While each scanner behaves entirely static, there is variation between the groups of people who scan for NTP. Over the years, we see between 10 and 18 patterns being employed. The complete lack of dynamic in NTP is not per se surprising, as the protocol specification provides only little opportunity to include arbitrary data. Among the very few possibilities, NTP versions 3 and 4 allow the inclusion of three timestamps to represent the current time configurations. We verified that scanning using randomization provides valid responses in NTPv4, but scanners do not pursue this strategy.

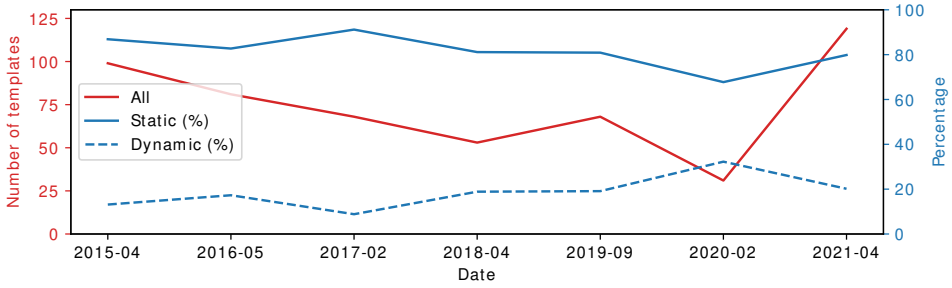
The practice of scanning is however different in case of DNS as shown in figure 4.5b. For DNS, our algorithm identified typically between 53 and 121 distinctive scan patterns per year except for an outlier in 2020, and between 8% and 33% of these identified templates are dynamic and randomized. Aside from a slight increase in DNS, we find that the general tactics and preferences of scan randomization or usage of static payloads has not significantly changed over those years.

While the previous results would indicate a basically static ecosystem with no evolution at all, there is actually significant innovation on what exactly the scanners send.

[1] The single dynamic scanner instance in 2018 is not an exception to this rule, but a scanner who has launched a campaign sending SIP requests to NTP servers.



(a) NTP



(b) DNS

Figure 4.5: Number and share of templates in use in NTP & DNS.

Figure 4.6 shows the percentage of new NTP and DNS commands that are seen for each year, in other words, as we see 99 DNS commands in total in 2015 as shown in figure 4.5b, the value of 85% depicted here for 2016 says that out of the 81 templates we discovered in 2015, 85% or 69 in total were new. As we see in case of DNS, scanners constantly update how they perform the scans, and in 2021 more than 80% of all scan packets used patterns that we did not ever see before in those past seven years. This is due to the widespread adoption of new query types (as we show in the next section), as well as evolution of the protocol.

In contrary to this, the percentage of newly observed NTP commands is minimal and diminishes quickly. Curiously, based on manual inspection, those new templates that *are* used stem mainly from operational mistakes by scanner sending invalid NTP requests, or by scanning for a protocol other than NTP on the IANA NTP default port. Although the NTP protocol was updated three times within the last decade, none of the new features are included by scanners in their activity. As shown by [3] for case of DDoS amplification attacks, adversaries seem to form different communities each focusing particular protocols as their tools of the trade with different degrees of sophistication and techniques. A similar situation also seems to be true in case of scanning.

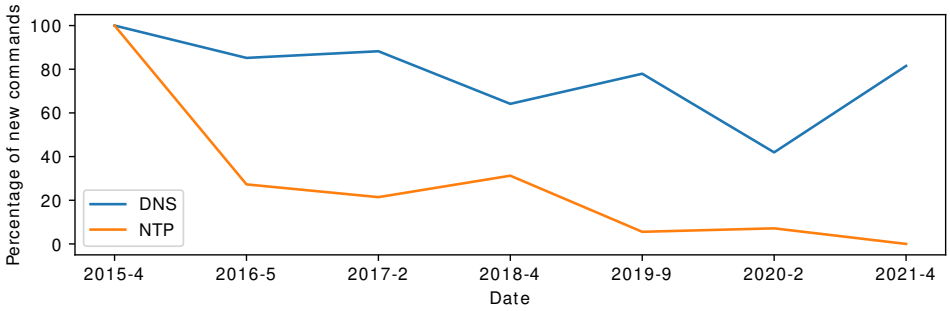


Figure 4.6: Percentage of new templates in NTP & DNS.

4.5.2. EVOLUTION OF COMMAND TYPES

Previously, we showed that scanners probing for DNS display a significant degree of variation in their activities. The advantage of our methodology is that it creates human-readable patterns that we can interpret with respect to the potential target and intention of the activity. In the following, we report on the content and development of these templates over time.

Targeted domain name. One easily changed feature of DNS scan probes is the domain name that is being targeted. If they perform surveys of open DNS resolvers, scanners might change what domain name they query for, as commonly used ones (such as Zmap's default query for google.it) might be flagged or blocked within the destination network. If actors are searching for DNS amplification attacks, they might switch over to zones offering larger amplification or target a different authoritative name server.

We analyze the discovered templates (which might include randomization in headers or payload fields such as random subdomains) with respect to the following three features: first, how often particular domain names are targeted. Second, the percentage of source IP addresses that use a particular domain. Third, the volume of scan packets that are received containing a particular domain name. Figure 4.7 shows the distribution for the three metrics for our snapshot traffic from 2015 until 2021.

In each bar plot, each color represents a particular domain. As the number of domains exceeds the number of meaningful colors, every color is used multiple times in a rotating fashion. However, the same color at the same position symbolizes the same domain name across these seven years. As we are concerned with respect to global developments and not concrete targets, we also omit to break out the 179 targeted domains in a legend. Figure 4.7a depicts how often a particular domain name appears in a particular template, for instance, a campaign enumerating subdomains in the form of `www[0-9]{3,5}.domain.com` would be grouped into one template, a campaign randomly looking up subdomains in the form of `host-[a-z]0[0-9]+.domain.com` would be grouped into another, both however share the second-level domain domain.com in their template. The figure shows no evidence of global trends, domains appear with fluctuating "market shares" across the years, except that over time more domain names appear in the templates of DNS scanners.

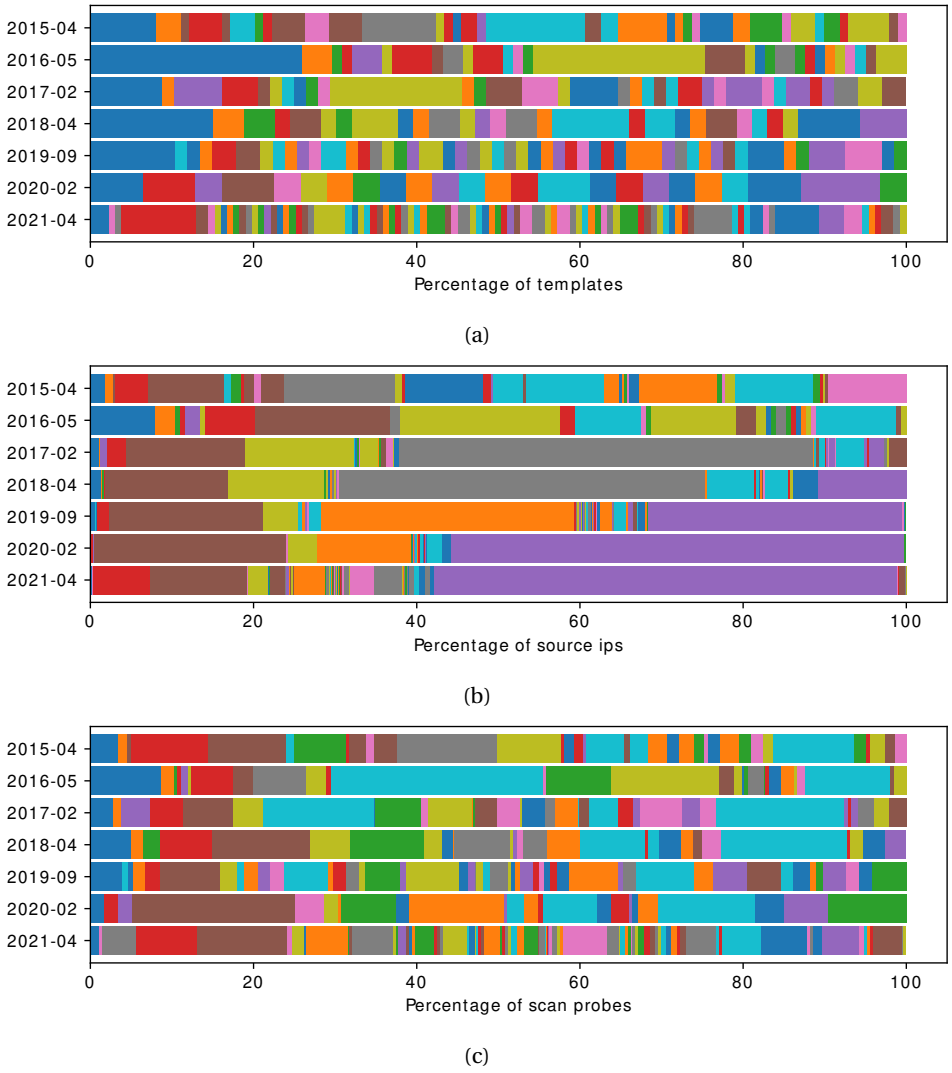
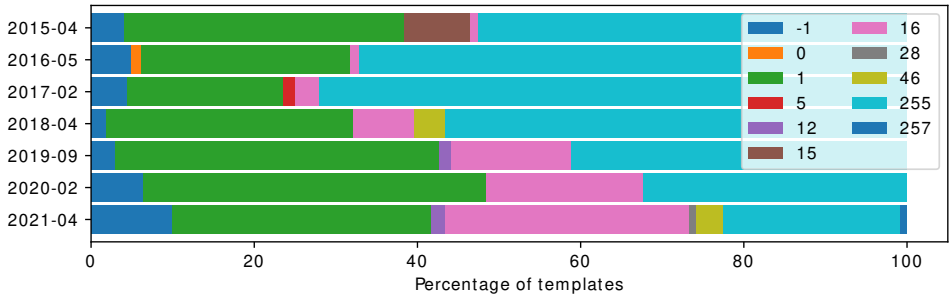
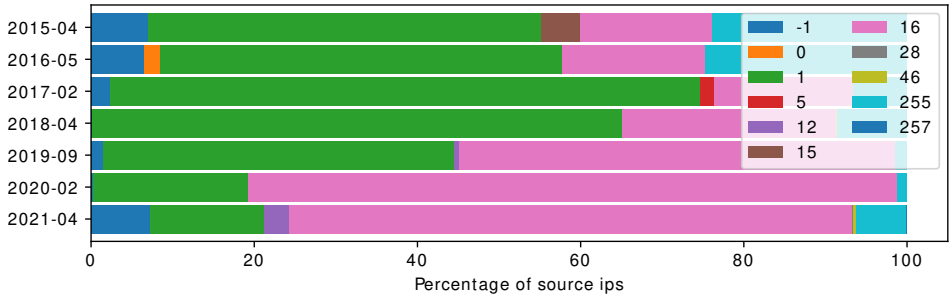


Figure 4.7: Domain name usage of scanners.

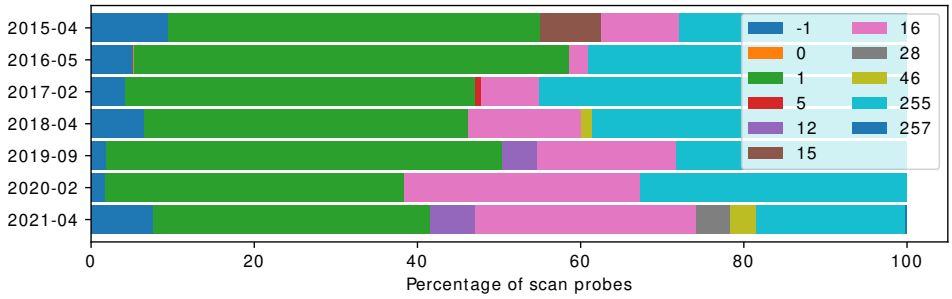
When we look at the percentage of source IPs that have send a probe belonging to a particular template as shown in figure 4.7b, we see that although there is a lot of variety in the ecosystem in terms of what domain names are queried, the absolute bulk of sources are using on a few select templates. Of particular importance here is the purple bar, which depicts the *version.bind.* command, used to discover the software version of the most widely used DNS resolver software bind, for example to execute a particular exploit. While in 2017 only 1.8% of the source IPs sent a probe containing the bind command, in 2021 the percentage increased to 56.7%. Figure 4.7c shows the percentage of



(a)



(b)



(c)

Figure 4.8: Query type usage of scanners.

packets that are sent using a particular template. The figure shows that the number of received probes per template is more or less equally distributed. The different distributions shown by figures 4.7b and 4.7c show that a large number of source IPs using the same template are responsible for a relatively low amount of scanning probes, indicating the presence of large-scale distributed scanning campaigns. In section 4.6 we show that, indeed, scanning campaigns can be identified using templates.

Used query types. Aside from the domain name, also the parameters of the domain query can be adjusted. In the simplest cases, this would entail the type of query, such as

A, NS or ANY, but also instructions whether the target should recurse or validate DNSSec. Figure 4.8 depicts the development in DNS query type over the past seven years based on the same metrics as before, but given the limited number of query types each is labeled by its numerical value as specified in [31].

We see very clearly how the ecosystem of scanners adapts its templates to transition from the ANY (255) query type to using the TXT (16) query type. While in 2017, 72.1% of all the identified templates used ANY, in 2021 this number has dropped to 21.6%. Templates based on ANY – the query for *any* kind of DNS record, a response typically extensive in length and thus useful for DNS amplification attacks – are successively being replaced with those using TXT, which increased from 2.9% to 30.0% during this time. TXT record fields today contain security features such as DNS keys, record signatures, or certificate authority pointers, and are thus also large enough in size to be interesting avenues for abuse.

When we study the number of sources relying on a particular template / query type combination in figure 4.8b, we see that this shift is basically adopted by the entire ecosystem. In 2017, 17.13% relied on TXT queries, this percentage has increased to 69.1% by 2021. Figure 4.8c shows the percentage of probes received for a particular query type and confirms the shift of scanners using TXT queries in favor of the ANY type.

4.6. IDENTIFYING CAMPAIGNS USING PROBE TEMPLATES

In the previous section, we clearly identified signs of distributed scans when we looked at which sources scanned using a particular template. These coordinated activities stem from a variety of origins, for example cyber security firms, universities, as well as malicious actors scaling out port scanning to avoid detection. In this section we show that our methodology is also effective at identifying these campaigns and relating the participating IP addresses to each other.

We refer to a port scan as a sequence of probes sent by a source IP towards one or more destinations host/port combinations in a fixed time period. These individual port scans can be grouped into a scan campaign, which is the systematic scanning activity by multiple entities towards a particular goal. Scan campaigns are usually scaled out to multiple sources, as each device needs to run at less intensity and runs less risk of being detected or blocked. As these devices are under control by the same actor, they are usually managed in a comparable way, which means that the individual devices often show similar behavior, such as comparable if not identical start and end times, activity periods, or in the extreme case even similar port allocations. We showed in section 4.5.2 that scanners often rely on templates to generate their scan probes, where each source would send (randomized) instantiations of a general template, used as a blueprint to specify the scan. We can use these characteristics to identify a campaign and the devices participating in it.

To demonstrate this feasibility, we show the results of campaign extraction given scan traffic received through April 2021. We use our methodology to extract templates from this set and assign every source IP a label based on templates it instantiates. Figure 4.9 depicts the individual scanning activity of hosts over the course of one month, ordered by its first occurrence. The color of each activity indicates its cluster membership, the

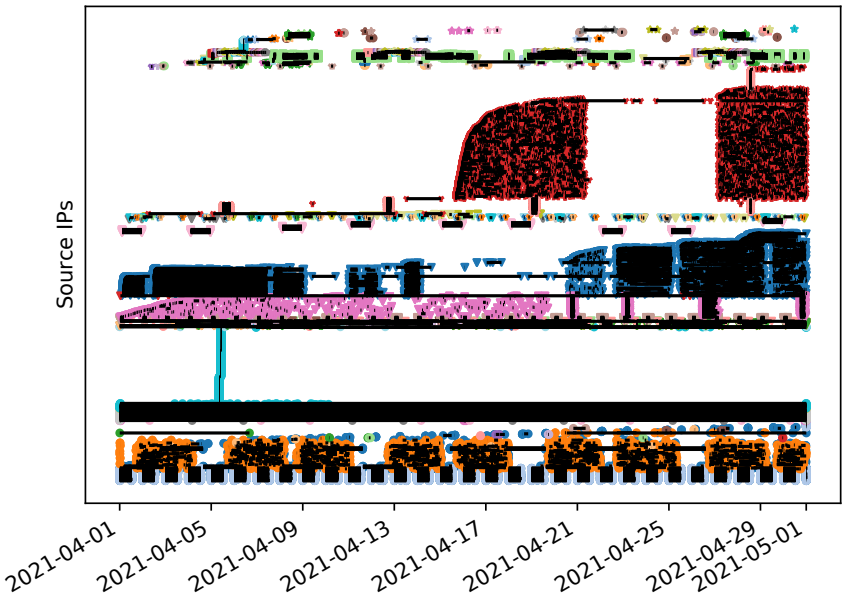


Figure 4.9: Scans performed in 2021 grouped by used template.

beginning and end of each scan is indicated by a black marker. We consider a scan terminated if we receive no packets from a source address for more than six hours.

As we can visually inspect, IP addresses matched on content templates also show other similar behavioral characteristics. For example, the scanning activities performed by the orange IPs all run repeated scans in coordination, and also the behavior of the red IPs is matched which may start and stop simultaneously, but show also a slow startup phase that would be difficult to find based on on/off activity only.

As the methodology can relate common behavior fuzzily, this grouping is also possible if there is little data available. Figure 4.10 shows the speed of each source IP in terms of sent probes per second, the colors and markers match those shown earlier in figure 4.9. We can visually see that the source IPs using the same templates scan at the same speed. Particularly interesting is here the fact that the more sources a campaigns is using, the slower it tends to send probes. The two largest campaigns in the figure emit on average about one packet every $1\ \frac{1}{2}$ hours, compared to some of the faster campaigns at more than 10,000 packets per second. Identifying and linking scanners when operating at such a low rate is typically challenging, as they can only afford such a low rate by using a large number of sources contributing to the same goal, we can still identify them given our template methodology.

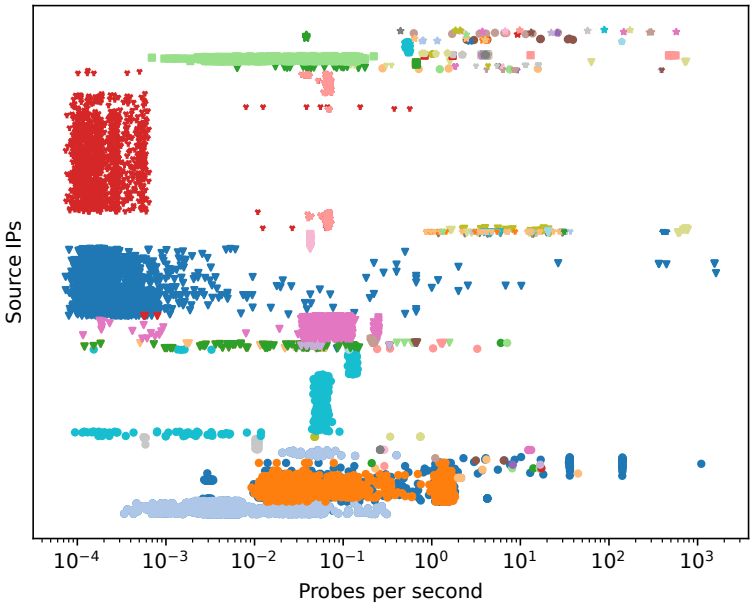


Figure 4.10: Scanning speeds in 2021 grouped by template.

4.7. CONCLUSION

In this work, we presented and validated a novel approach to identify the templates used by scanners. Using our approach, we can parse 96.04% of the scanning traffic and extract the used templates with an accuracy of 97.28%. We use our approach to perform a longitudinal study showing that the DNS scanning landscape has changed. Scanners have shifted from using ANY queries to TXT queries, most likely in response to the introduction of RFC 8482 [33] limiting the information gained by scanners using the ANY query. Finally, we leveraged our templating approach to identify large scanning campaigns.

References

[1] O. Al-Jarrah and A. Arafat. 'Network Intrusion Detection System using attack behavior classification'. In: *IEEE ICICS*. 2014.

[2] Lockheed Martin. *The cyber kill chain*. URL: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.

[3] H. Griffioen, K. Oosthoek, P. van der Knaap and C. Doerr. 'Scan, Test, Execute: Adversarial Tactics in Amplification DDoS Attacks'. In: *ACM CCS*. 2021.

[4] S. Haas, F. Wilkens and M. Fischer. 'Scan Correlation–Revealing distributed scan campaigns'. In: *IEEE/IFIP NOMS*. 2020.

[5] S. Torabi, E. Bou-Harb, C. Assi, E. B. Karbab, A. Boukhtouta and M. Debbabi. 'Inferring and investigating IoT-generated scanning campaigns targeting a large network telescope'. In: *IEEE TDSC* (2020).

[6] M. G. Kang, J. Caballero and D. Song. 'Distributed evasive scan techniques and countermeasures'. In: *SIG SIDAR DIMVA*. Springer. 2007.

[7] E. Bou-Harb, M. Debbabi and C. Assi. 'A systematic approach for detecting and clustering distributed cyber scanning'. In: *Computer Networks* (2013). DOI: https://doi.org/10.1016/j.comnet.2013.09.008.

[8] S. Torabi, E. Bou-Harb, C. Assi, M. Galluscio, A. Boukhtouta and M. Debbabi. 'Inferring, Characterizing, and Investigating Internet-Scale Malicious IoT Device Activities: A Network Telescope Perspective'. In: *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2018. DOI: 10.1109/DSN.2018.00064.

[9] *Junos OS: Remote code execution vulnerability in overlayd service (CVE-2021-0254)*. https://kb.juniper.net/JSA11147. Acccessed 15 February 2021.

[10] G. F. Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2008.

[11] H. Heo and S. Shin. 'Who is knocking on the telnet port: A large-scale empirical study of network scanning'. In: *Asia CCS*. 2018.

[12] M. Alsaleh and P. C. Van Oorschot. 'Network scan detection with LQS: a lightweight, quick and stateful algorithm'. In: *CCS*. 2011.

[13] Z. Durumeric, M. Bailey and J. A. Halderman. 'An Internet-Wide View of Internet-Wide Scanning'. In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. USA, 2014.

[14] P. S. Joshi and H. A. Dinesha. 'Survey on Identification of Malicious Activities by Monitoring Darknet Access'. In: *ICSSIT*. 2020.

[15] A. Affinito, A. Botta, L. Gallo, M. Garofalo and G. Ventre. 'Spark-based port and net scan detection'. In: *ACM/SIGAPP SAC*. 2020.

[16] S. Staniford, J. A. Hoagland and J. M. McAlerney. 'Practical automated detection of stealthy portscans'. In: *Journal of Computer Security* (2002).

[17] C. B. Lee, C. Roedel and E. Silenok. 'Detection and characterization of port scan attacks'. In: *Univeristy of California, Department of Computer Science and Engineering* (2003).

[18] M. Coudriau, A. Lahmadi and J. François. 'Topological analysis and visualisation of network monitoring data: Darknet case study'. In: *IEEE WIFS*. 2016.

[19] F. Iglesias and T. Zseby. 'Modelling IP darkspace traffic by means of clustering techniques'. In: *IEEE CNS*. 2014.

[20] E. Bou-Harb, M. Debbabi and C. Assi. 'On fingerprinting probing activities'. In: *Computers & Security* (2014).

[21] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, Z. Durumeric, J. Cochran, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.* 'Understanding the Mirai botnet'. In: *Proceedings of the 26th USENIX Conference on Security Symposium*. 2017.

[22] 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.

[23] C. Gates. 'Coordinated Scan Detection.' In: *NDSS*. 2009.

[24] S. Pang, D. Komosny, L. Zhu, R. Zhang, A. Sarrafzadeh, T. Ban and D. Inoue. 'Malicious events grouping via behavior based darknet traffic flow analysis'. In: *Wireless Personal Communications* (2017).

[25] C. Fachkha, E. Bou-Harb, A. Keliris, N. Memon and M. Ahamad. 'Internet-scale Probing of CPS: Inference, Characterization and Orchestration Analysis'. In: *NDSS*. 2017.

[26] V. Ghiette and C. Doerr. 'How Media Reports Trigger Copycats: An Analysis of the Brewing of the Largest Packet Storm to Date'. In: *Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity*. 2018. DOI: 10.1145/3229598.3229606.

[27] *Zmap Manual*. https://github.com/zmap/zmap/wiki/UDP-Probe-Module. Acccessed 12 August 2021.

[28] H. Griffioen and C. Doerr. 'Discovering Collaboration: Unveiling Slow, Distributed Scanners based on Common Header Field Patterns'. In: *IEEE/IFIP NOMS*. 2020.

[29] D. R. Thomas, R. Clayton and A. R. Beresford. '1000 days of UDP amplification DDoS attacks'. In: *Symposium on Electronic Crime Research*. 2017.

[30] G. V. Bard. 'Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric'. In: *ACSW frontiers*. Citeseer. 2007.

[31] P. Mockapetris. *Domain names - implementation and specification*. RFC 1035. 1987.

[32] *Sipvicious*. https://github.com/EnableSecurity/sipvicious. Acccessed 12 August 2021.

[33] J. Abley, O. Guomundsson, M. Majkowski and E. Hunt. *Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY*. RFC 8482. 2019.

5

FINGERPRINTING TOOLING USED FOR SSH COMPROMISATION ATTEMPTS

This dissertation aims to showcase the extraction of intelligence at different attack stages as it provides the opportunity to develop in-depth defense strategies. In the preceding chapters, we analyzed scan traffic, collected during the reconnaissance phase of attacks, demonstrating different methods for extracting intelligence during the first stage of cyber attacks. In this chapter, we continue the analysis of attacks based on data gathered at a later stage.

Using a large honeynet we collected millions of SSH brute-forcing attempts from various attackers. Before brute-forcing, attackers need to establish SSH connections and in doing so they need to exchange preferred settings. These settings are often dictated by the libraries or operating systems used to compile the brute-forcing tools. Therefore, analyzing the connection data allows us to identify the underlying libraries attackers use to compile their toolchains. More so, due to the different library versions, we are able to discern different large-scale attacks. We confirm the findings using the time analysis and the brute forcing credential used by the attackers. This chapter showcases the ability to extract cyber threat intelligence, before the actual brute forcing attempts, contributing to detecting and differentiating cybercriminals' toolchains at the early stage of attacks.

5.1. INTRODUCTION

Secure Shell (SSH) is a widely used protocol to operate services on a remote host over a network. One of the commonly used services of SSH is remote terminal access, which allows a user to execute programs on a remote system. The protocol authenticates a user based on a public key or an username/password combination, which prohibits malicious users to connect and exploit the host.

Due to the extensive use of the protocol, SSH is a popular target in brute forcing attacks. While system administrators are able to change the usernames and passwords used by the device, a lot of devices are still configured to use standard username and password combinations. As many devices are left with default configurations, simply trying a list of common username and password combinations proves effective enough for attackers to massively scan for, and attack SSH devices using this method.

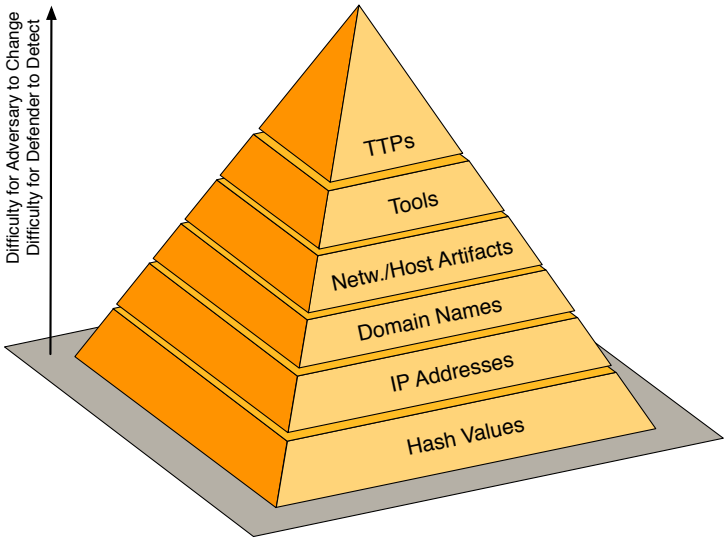


Figure 5.1: While basic Indicators of Compromise (IoC) are easy to gather and distribute, they are trivially changed by an adversary. For effective, more persistent detection it is necessary to assemble threat intelligence that covers behavioral features of the attacker. [1]

While unsophisticated attackers would run through an extensive username/password candidate list in order to gain access, such behavior would be quickly visible in log files, and source addresses with repeated failed attempts are routinely blocked by intrusion detection systems (IDS) or monitoring systems such as fail2ban. Advanced adversaries would thus split the brute forcing out over multiple hosts, but in order to simplify the administration, usage and lower the cost, they would typically run a similar software across systems.

Current detection revolves mostly around simple indicators to detect malicious behavior. Virus scanners or intrusion detection systems for example rely on signatures and hashes to identify malicious activity, and also the IP addresses of malicious hosts scanning and brute forcing logins is enumerated and distributed in IP block lists. As explained by the so-called "pyramid of pain" [1] depicted in figure 5.1, these indicators are however trivially changed for an adversary, for example by simply recompiling a malware or moving the activity to a newly compromised host or proxy. In result, such information is unsuited to stop adversaries for long and at a broader scale. An alternative is the detection based on complex indicators, such as the systems or tools used or the tactics in a compromise, as they are much more difficult and costly for an adversary to change. If we can detect a particular software or modus operandi used for brute forcing SSH, we can reliably identify malicious activity, regardless of the IP address it is coming from and whether this address was participating in such activities before.

In this paper, we introduce methods for fingerprinting software stacks and tools used

in SSH connections. This will help to study and follow the activities of adversaries, as an attacker will most likely distribute the same tool over a number of hosts to leverage the economies of scale. By detecting attacks by their used tools, attackers will have to change their software between campaigns, and even between different hosts. This greatly increases the cost for attackers, and can price them out of the system.

Our approach extracts session negotiation information such as the list and ordering of key exchange algorithms, cipher suites, or compression algorithms which are exchanged in clear text during the SSH session initiation. This means that using this approach, we do not need to interfere with the connection itself, meaning that the method is completely passive, and as the fingerprint is derived from the SSH handshake, we are able to identify brute forcing attempts even before the first password is sent to the system.

This paper makes the following contributions:

- We introduce the concept of fingerprinting to the SSH protocol and demonstrate based on a large corpus of 35 million brute forcing attempts that fingerprinting is suited to identify tools that are used by adversaries. By detecting attacks on this level, the cost for adversaries rises as they need to build new tools for every campaign.
- We deploy the technique to 4,500 honeypots with the aim of gaining cyber threat intelligence about the practices of adversaries. We empirically show the presence of 49 different tools, and show that a cluster of hosts relies on the same toolchains. We furthermore find evidence of large distributed campaigns of collaborating hosts.

The remainder of this paper is structured as follows: Section 5.2 describes the state of the art in fingerprinting and SSH brute forcing. Section 5.3 provides an overview of the SSH protocol and components necessary to introduce the proposed method. Section 5.4 explains the fingerprinting methodology. Section 5.5 provides details about the design and scale of our honeynet. The evaluation of our proposed method is presented in Section 5.6. Using our method, we find a large number of actors, each featuring different strategies, tactics and resources. Finally, Section 5.7 summarizes and concludes our work.

5.2. RELATED WORK

As stated in the previous section, a sustainable cyber defense best focuses not on identifiers of specific malicious instances, but on characteristics that are constant over multiple instances. One way of generating these characteristics is to fingerprint the tools used by attackers. Our main claim in this paper is that we can extract fingerprints from the SSH connection negotiation that can be used to distinguish different tools. Two lines of related work are important to class the proposed work, first previous research on fingerprinting protocols, and second previous research in brute force detection.

First, while fingerprinting has not been done in SSH, differences in cipher suite strings have been used in the SSL/TLS protocol suite to identify server or client software. To fingerprint clients, Husak et al. [2] were able to infer the used client application based on the cipher suites that were used in the connection. The authors found that many applications support different cipher suites for establishing a connection, and some applica-

tions also send the cipher suites in a different order. Therefore, the authors were able to fingerprint client applications using only the cipher suites presented in the handshake. Durumeric et al. [3] applied the analysis of advertised clients (through the HTTP User-Agent) and implemented SSL/TLS handshakes to detect the nature of the client connecting, and thereby identify middleboxes that intercepted the TLS connection between client and server. Fingerprinting specific implementations is also possible by detecting specific patterns in which header fields [4] or packet payloads [5] are set and encapsulated in scan and attack traffic.

Fingerprinting the traffic sent through encrypted channels has been done by Sun et al. [6]. Their algorithm is able to identify which webpages are visited from the amount of traffic sent during the page load. Similar research by Korczynski et al. [7] uses Markov chains to generate fingerprints for different services based on the SSL session. Their research shows that they are able to fingerprint certain applications with a high confidence level. In the case of SSL, research has focused on fingerprinting clients and client behavior. Our SSH fingerprinting method leverages the same intuitions, but is tailored towards fingerprinting adversaries that are attempting to compromise a system.

Second, although there exists no prior work in the literature for fingerprinting SSH endpoints, a selection of previous studies have developed methods for detecting SSH brute forcers. Hellemons et al. [8] have proposed an intrusion detection system method for detecting SSH intrusions using netflows. Similarly, Najafabadi et al. [9] propose a machine learning algorithm to detect brute force attacks in netflow data. The authors have validated their results on the SSH protocol and found that machine learning techniques perform well for detecting these brute force attacks. Nicomette et al. [10] clustered adversaries together based on attempted passwords. The authors find relationships between dictionaries, but at the same time notice that few dictionaries are shared across attackers. All these works focus on the detection of brute forcing attacks at the moment that the system is already under attack, however we show in the following that it is possible to obtain much information about the incoming request during the connection negotiation itself and before the password prompt is shown.

A selection of studies have investigated adversarial behavior after the successful compromise of a honeypot. Ramsbrock et al. [11] followed compromises made into four honeypots, and was able to derive a state machine to describe the actions of adversaries. Barron and Nikiforakis [12] investigated whether adversarial actions differed based on environmental factors, for example depending on the presence of real users on the systems and their usage of files. They were able to distinguish between human and bot login activity, and noticed humans did to a limited extent show interest in stored files while bots generally avoided significant interaction with the file system, and in half of the cases only proceeded to install a proxy gateway.

While proposed methods can identify brute force attacks, they do not allow for tool classification or for pre-emptively stopping these attacks. Given the current threat landscape, in which there is a high number of attackers, identifying attacks in an early stage before actual compromization is increasingly important. By forcing attackers to change their tools every attempt, the cost for attackers increases and many attackers will be priced out of the system. In this paper, we propose a method to fingerprint tools in use by adversaries, which can be used to track their activities over time, relate distributed

attempts to the same toolchain and possibly actor, and thus gain a greater insight into the ecosystem as a whole.

5.3. THE SSH PROTOCOL

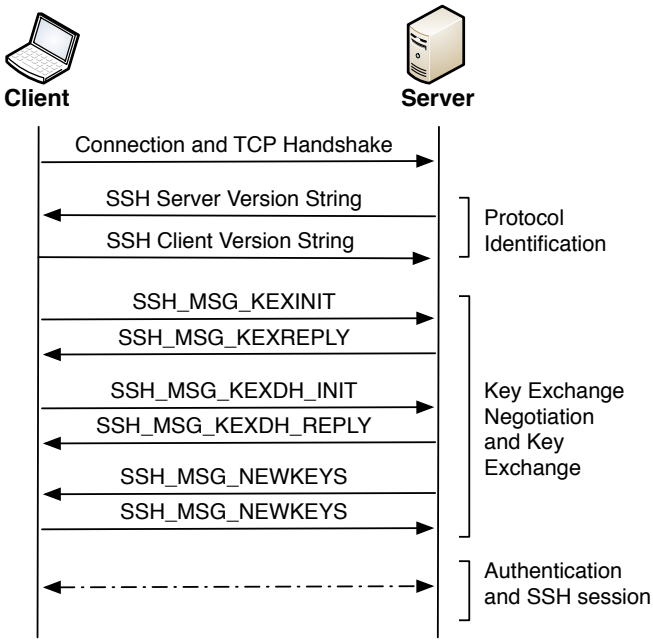


Figure 5.2: Schematic overview of the message exchanges in the establishment of an SSH session.

The secure shell protocol (SSH) is an established protocol for accessing services on a remote host, which is secured by an authentication procedure. In order for an attacker to enter login credentials it is first necessary to set up a secure protocol connection as specified in RFC4253 [13]. The main steps in setting up a secure communication channel between the attacker and its target are shown in figure 5.2 and go through three main phases.

First, after a TCP connection has been established, both parties exchange the version of the SSH protocol they are running in the protocol identification phase. Examples of sent version strings are `SSH-2.0-JSCH-0.1.51` and `SSH-2.0-paramiko_1.7.5`. As will later be explained, the version exchange is one of the components used for profiling attackers.

As SSH provides an authenticated encryption tunnel, both sides need to negotiate key material for the connection. In the second phase of the protocol, client and server negotiate the key exchange mechanism to be used. This negotiation is initiated by the client through a key exchange initialization message (SSH_MSG_KEXINIT), which contains all

the different key exchange algorithms, encryption algorithms, algorithms to compute a message authentication code, and algorithms for compressing the data the client is able to accept. The order of the advertised algorithms is of importance as the algorithms are advertised according to the host's preference, and thus both the presence and order of algorithms shared during this step of the SSH connection can be used to profile a connecting client. After both parties have sent and received the key exchange initialization message, the highest commonly preferred algorithms are selected for setting up a secure connection. Depending on which algorithms have been chosen, the rest of the key negotiation and key exchange procedure slightly varies. After the key negotiation, the actual key exchange is initialized by sending the SSH_MSG_KEXDH_INIT message, after which the key exchange algorithm is run. Once the algorithm is finished, each side signals using a SSH_MSG_NEWKEYS message that the secure connection is set up and ready to be used.

In the third phase, both client and server switch to an encrypted tunnel using the just negotiated key material and perform the authentication, during which the client sends its login credentials. Given the correct credentials, the SSH protocol then makes the requested resource on the remote host available to the client.

5.4. FINGERPRINTING TOOLING

Brute forcing the login credentials to gain access to a shell generally requires a great amount of attempts due to the large amount of possible username/password combinations. Therefore, it is uneconomical for an attacker to perform this task manually, and he or she will likely resort to a tool in order to automate the login attempts. Depending on the knowledge of the attacker, he or she will utilize an existing tool or develop a new one.

If the attacker opts to use known tools, there is no shortage of available material. A quick search on any search engine yields an extensive list of SSH brute forcing tools. Most of these programs are advertised as penetration testing tools, used to assess the security of a network, for example to find servers that use weak login credentials. More so, entire articles and tutorials are dedicated to the usage of those tools, such as Hydra [14], Medusa [15], and Ncrack [16].

When writing an SSH brute forcing tool, one could create an implementation of the SSH protocol, or use a pre-built library that handles the SSH connection. The aforementioned brute forcing tools utilize different libraries implementing the SSH protocol, and only add the logic to perform the attack. Although the libraries are likely to adhere to the standards specified in the RFC describing the SSH protocol, minor differences in connection establishment can be witnessed. One of those difference is the announcement of the SSH version. Libraries such as libssh [17] and libssh2 [18] use a different version string to announce compatibility with the same version of SSH protocol. Both libraries add their name and release version into the SSH version string; libssh version 0.7.1 announces `SSH-2.0-libssh-0.7.1`, whereas libssh2 version 1.8.0 identifies itself as `SSH-2.0-libssh2_1.8.0`. While this provides a first, trivial angle to identify the tools used by attackers, we only use this information as a reference and later combine it with the capabilities implemented by a specific library. Thus, if an adversary is spoofing the

version string, the announced version will not match the fingerprint of the advertised key exchange, encryption, MAC and compression algorithms anymore, which in combination yields an even more distinctive fingerprint for a specific brute forcing tool implementation.

Similarly to the announced version string, the information exchanged during the key exchange initialization varies between libraries. As discussed in the previous section, during the session establishment, different algorithm suites are announced in order of preference. Not all libraries support all key exchange, encryption, MAC, or compression algorithms. More so, not all libraries supporting identical algorithms will order them according to the same preference. This multitude of possible variation of the initialization message increases the likelihood of libraries implementing them differently, which can be leveraged to identify the tool and/or underlying library in an incoming SSH connection request.

In order to compare different key exchange initialization messages, the four exchanged capabilities as discussed in Section 5.3 are used.

The advertised

1. key exchange algorithms (kex),
2. symmetric encryption algorithms (enc),
3. message authentication code algorithms (mac), and
4. compression algorithms (comp)

are concatenated into a single capabilities string. Since we only care about exact matches in capabilities, we hash the concatenated string as a fingerprint for a specific tool. Consider the case of an out-of-the-box SSH server install on Ubuntu 16.04 Desktop, which comes preconfigured with the following configuration:

- *KEX algorithms:* curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256, diffie-hellman-group14-sha1
- *Ciphers:* chacha20-poly1305@openssh.com,aes128-ctr, aes192-ctr, aes256-ctr, aes128-gcm@openssh.com, aes256-gcm@openssh.com
- *MAC algorithms:* umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-sha1-etm@openssh.com, umac-64@openssh.com, umac-128@openssh.com, hmac-sha2-256, hmac-sha2-512, hmac-sha1
- *Compression algorithms:* none, zlib@openssh.com

Thus, we obtain the fingerprint through the MD5 hash of

```
curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh
-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-gro
up-exchange-sha256,diffie-hellman-group14-sha1;chach
```

```
a20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes25
6-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com;
umac-64-etm@openssh.com,umac-128-etm@openssh.com,hma
c-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh
.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,u
mac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac
-sha1;none,zlib@openssh.com
```

resulting in `9f735e5485614bcf6e88b9b848582965`.

Finding or building a tool for SSH brute forcing is only the first step for an attacker towards compromise - next the adversary needs to choose which login credentials to use during authentication. Here the attacker has three main choices:

First, an attacker can opt to incrementally, pseudo randomly, or randomly generate the username and password from a predefined set of characters. Given enough time, and assuming the attacker is not blocked after some attempts, this method is bound to provide access to the targeted machine. Second, an attacker can decide to generate the login credentials using a dictionary attack. In a dictionary attack, words are combined to form the password and or username. Third, an attacker can choose to use predefined login credentials and available password lists. As with SSH brute forcing tools, a quick search on any search engine reveals a plethora of hits advertising password list for download. These lists often contain known default passwords and usernames such as `admin` and `root`.

Here, again, the different options available for the attackers to generate the login credentials offers an opportunity to detect brute forcing campaigns, and we put forth the assumption that the same adversaries will most likely use the same or similar login credentials when trying to exploit systems. Under the premise that an attacker will use the same tool to attack multiple targets, the same fingerprints for a single IP address should be witnessed at different honeypots. While the combination of algorithms provides already a distinctive fingerprint for a specific tool, in the later part of our evaluation we further investigate the relationship between identical tools and password generation algorithms. If an attacker uses multiple machines or IP addresses with the same brute forcing tool and password generation algorithm, the previously described fingerprints can be used to cluster IP addresses belonging to the same attacker.

5.5. DATA COLLECTION

This section describes the setup of our honeypot infrastructure and the data acquisition strategy used in this paper. As the goal of our study is to demonstrate the possibility of fingerprinting the tools and techniques used by adversaries in SSH break-in attempts, we have designed a distributed honeypot system and exposed approximately 4,500 honeypots distributed over three /16 subnets to the open Internet.

5.5.1. HONEYPOT DESIGN

As you recall from section 5.3, the SSH protocol goes through three main phases in connection establishment: first, client and server announce their protocol versions to each

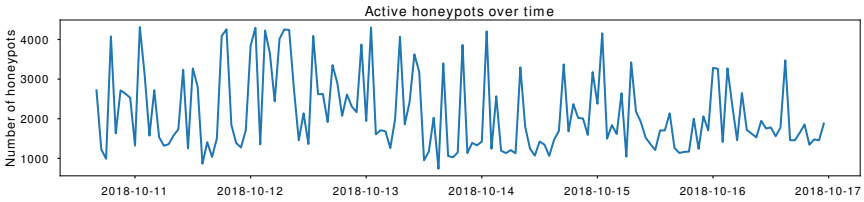


Figure 5.3: Number of active honeypots aggregated by hour over the course of one week in the study.

other, second, the endpoints exchange their ciphering, MAC and compression capabilities and agree on a key, and finally, the tunnel is authenticated by a public key or password before an SSH session is established. While we would clearly expect to see differences in the behavior of adversaries after they have gained access to a particular machine, the SSH protocol is complex enough and contains several configuration options so that the tooling used by attacker may contain implementation differences, that will – as we show in the following – allow recognition of a particular tooling *even before* the SSH protocol advances to the password prompt and an interactive session. The distributed infrastructure was therefore implemented as a honeypot that would negotiate a key exchange and an SSH session with the connecting client, display a login prompt and collect usernames and passwords, but never let any user in. While this simplifies containment and thus reduces the risk of operating such a system, to the connecting user it basically appears like a regular server and an incorrect password guess.

In order to pose as yet another open SSH server, it is essential that the honeypot itself blends in with existing installations found on the Internet, otherwise knowledgeable adversaries could soon identify instances running some honeypot software and avoid individual IPs or even subnets where honeypots were detected. Thus, it would be a major failure if a system meant to identify adversaries based on handshake fingerprints could be fingerprinted itself, which has been found to be an issue for existing common open source honeypots such as Kippo or Cowrie [19]. To avoid this problem, we connect the incoming session to an actual OpenSSH implementation running inside a container, which matched in terms of version strings, list of available algorithms and options and ordering a default Ubuntu 16.04 LTS installation. This way, an adversary probing the system for implementation deviations to unexpected inputs will observe no difference from a typical server, and to an adversary scanning the Internet for banners and key exchanges our honeypots will blend in with what one would expect when connecting to a popular deployed operating system.

5.5.2. ORGANIZATIONAL PLACEMENT

Aside from being identifiable based on specific implementation characteristics, it would also be conceivable that adversaries could spot honeypots based on the way they are deployed and subsequently avoid them. For example, an apparently open SCADA system hosted on an Amazon EC2 cloud IP address should trigger some suspicion in a know-

ledgeable adversary. In our case, an entire block of consecutive IP addresses running SSH servers where otherwise nothing else is open in the network could similarly bias the results, and adversaries be motivated to evade such networks.

In order to create a believable posture and collect representative results, we deployed the honeypot in the enterprise network of an organization. This organization is connected with three /16 networks to the Internet, on these networks approximately 60,000 devices are active and incoming SSH traffic is not filtered by the firewall. These 60,000 official network hosts were of various types and origin, with a mix of servers, workstations, laptops and other mobile devices. While the foremost category would be constantly powered on and accessible, personal workstations, laptops or phones would only be powered on at select times.

In this study, we spread the 4,500 honeypots randomly throughout the network ranges of the organization, so that they would be assigned IP addresses belonging to server, workstation or mobile host subnets. This meant that IP addresses belonging to our honeypot system were located in between sets of servers or regular user machines, thus an adversary exploring the network could not evade the honeypots by skipping select parts of the network ranges, and after scanning several "official" servers our honeypot would appear to the adversary as just another server in the same group. The routing rules of the organization were set up in such a way that IP addresses that were allocated to a host but *also* chosen for the honeypot were forwarded to the official host whenever powered on, and forwarded to the decoy as soon as the official host left the network. This way, adversaries interacting with the organization's hosts would experience an instantaneous seamless handover to our honeypot infrastructure.

Figure 5.3 depicts the number of active honeypots in our deployment over the course of one week. The graph clearly shows a diurnal pattern stemming from a base population of approximately 1,500 active systems in server and workstation ranges, as well as an additional 3,000 decoys which only become activated when the mobile and temporary devices leave the network. Both components of our honeypot deployment strategy make it thus very challenging for an adversary to locate and evade our infrastructure.

5.6. EVALUATION

This section evaluates the results of applying the fingerprinting methods on the dataset. Both the SSH versions string and the fingerprint based on the session negotiation are analyzed. In addition, we use time and password correlation to evaluate the hypothesis

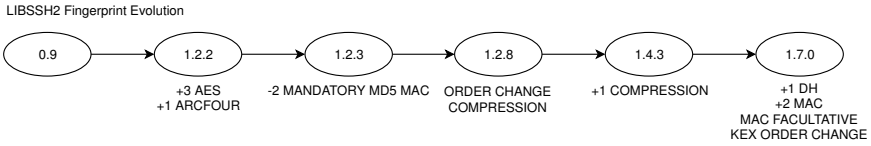


Figure 5.4: Evolution of the algorithms used by the libssh2 library to construct the SSH_MSG_KEXINIT message.

that attackers leverage multiple hosts to brute force SSH servers.

5.6.1. AVAILABLE FINGERPRINTS

During the data collection period, a total of 107,793 hosts tried to brute force the login credentials of one or more honeypots in our network. We only considered source IPs that completed at least one completed SSH key exchange towards our 4,500 honeypots during the entire month, thus excluding mere TCP SYN scans. Within this entire dataset, we observed a total of 123 SSH version strings, and identified 49 distinct MD5 hashes for different libraries and library versions in use.

While the analysis yielded a substantial count of different fingerprints and version strings, we also find that these instances are also surprisingly well spread across source hosts. The distribution of source IPs that use a particular fingerprint follows an exponential decay; while the most commonly used library fingerprint is used by 58% of the sources, already the bottom half of the top 10 fingerprints account only for fractions of a percent. Fingerprints thus have large amounts of variations, and are distinctively correlated to sources: more than 89% are only associated with one fingerprint over the entire observation period. Similar results apply to the version strings; the top 3 version strings are used by more than 75% of all source IPs, also here 90% of all sources only advertise one version string to our honeypot during the entire period.

The large body of fingerprints compared to the number of available tools, as well as the larger number of version strings compared to the amount of fingerprints matches our expectations how brute forcing tools are developed and used. First, as commonly used tools build on system libraries such as libssh or OpenSSH, major updates to the underlying system library implementing the SSH protocol will result in a new fingerprint, even though the adversary uses the same toolchain.

Second, tools (or their users) actively change the version string and configuration advertised by their toolchain, possibly in an attempt to evade detection by signatures. Examples of this are invalid encryption and mac algorithms such as `hmac-sha\x11` and `"lowfIsh` in some of the key exchange messages. Software packages such as OpenSSH allow a user to configure the ciphers used in the key exchange message, even if these algorithms are not implemented in the library itself. Inspection of the source code of for example the libssh2 library reveals that the versions strings announced by the majority of the hosts matches with the one in the source code. When we further analyze the design of available brute forcing tools, we can link common tools back to these libraries. Medusa [15], a tool for SSH brute forcing included in Kali linux, builds on this library in this selected version. The analysis of the 35 million compromisation attempts using the SSH version and key exchange fingerprinting leads to the conclusion that the majority of attackers use readily available tools to perform the brute forcing.

5.6.2. FINGERPRINTS AND LIBRARIES

Different tools generate different combinations in terms of SSH version strings and key exchange fingerprints. If two tools use different libraries implementing the SSH protocol, the announced SSH version string will likely be different. As seen in the data, the name of the SSH library is often included in the announced SSH version string.

Similarly, the fingerprint retrieved from the key exchange can also provide an indication of the tool used. Tools building upon different libraries will announce different algorithms during the key exchange. This is due to different libraries supporting different algorithms and announcing them in different order of preference. Therefore, due to slight variations in used libraries and implementations, tools can be linked to the fingerprints generated by processing the SSH version string and the key exchange initialization message.

In order to investigate the origin of the fingerprints and study user customization, we downloaded 10 commonly used SSH brute forcing tools to inspect which library is linked to realize the SSH protocol, and in addition mined the SSH version strings from all received handshakes for mentions of libraries or implementation stacks. This yielded a total of seven libraries that are utilized across SSH brute forcers we observed, namely (1) Granados, (2) JSCH, (3) libssh, (4) libssh2, (5) OpenSSH, (6) paramiko, and (7) the Erlang standard SSH implementation. For all these libraries, we downloaded every single release, as well as every intermediate version available in the software repositories and manually identified the location in the source code responsible for the SSH connection and parameter selection. A program then identified every intermediate/release version when this code was modified over the entire period of the softwares' past development, and we manually analyzed each changed code segment to extract how this library would advertise itself as in this particular version. This yielded a set of (banner, MD5 hash) tuples, which is on the one hand dependent on the version of a library, on the other hand also on certain options and taken branches in the code, for example if certain other libraries or headers were available on the system where it would be compiled and thus activate `ifdef` blocks. Based on this, we generated a set of possible 57 banner-hash tuples for the seven aforementioned libraries, implemented in a particular software package at any point in time.

Figure 5.4 shows this evolution for the libssh2 library with respect to the construction of the SSH_MSG_KEXINIT message. While libssh2 saw several intermediate releases between version 0.9 and version 1.2.2, and the library advertises itself differently between these two releases, the cryptographic routines remained unchanged during all of these updates leading to an identical fingerprint. In 1.2.2, three options for the data encryption using the AES cipher suite as well as the option to encrypt using RC4 were added, while in 1.2.3 the previously mandatory option to create a message authentication code using MD5 was removed. Later versions such as 1.2.8 only differed in the order they advertised the preference of algorithms. Similar version graphs were created for the other six libraries mentioned above, most of them appeared in our data set announcing different release versions.

When libraries are compiled, the supported algorithms might differ, depending on installed software packages that are required by the algorithms. These dependencies can greatly affect the number of supported algorithms during the key exchange initialization, which is why we have identified all different combinations possible. These also result in a unique fingerprint for a particular installation path and thus allow a peek into the configuration of the attack hosts.

Given the advertised library and version string we can then cross validate whether the

fingerprint obtained from the handshake is consistent with the default behavior of the library, or whether some code changes or configuration changes were introduced. Interestingly, when we look at the 123 SSH version strings and 49 fingerprint hashes that we collected in our honeypots, we find that all 57 theoretically possible tuples from the software libraries were present. When we match the version string for a particular library and the fingerprint hash that *should* have been generated from the honeypot handshake, we find that there is only a match for 26 out of the checked 57 library versions. This indicates that in 31 instances, more than half, the announced version string is spoofed. We manually verified these instances of spoofing, and found that while some of them are attempts to make the version string more generic, others modify the version number of a library or pretend to run a different software stack than the behavior of the library would in practice indicate. For example, a particular brute forcing tool would announce OpenSSH version 4.3, but announces a cipher suite that was not implemented in this particular version. Also, the order of algorithms for some advertised versions of libssh and libssh2 do not match the implementation of the library.

While spoofing of version string is common among attackers, given our tracking of code changes, we were able to trace back 26 out of the 31 spoofing instances to a library that is consistent with the behavior of the brute forcing tool. Overall, we find that we were able to identify more than 91% of the tools used to attack our honeynet using the fingerprint.

5.6.3. COLLABORATING HOSTS

As we have shown in the previous part, the combination of available key exchange algorithms, cipher suites, MAC and compression options together with the advertised version string does contain large amounts of entropy. As an additional verification that this fingerprint can serve as a measure to fingerprint the tooling itself, we look in this part into the behavior of the hosts exhibiting a particular fingerprint. If this relationship holds, we would expect the following two results: First, commonly available tools should see continuous usage, but within this set there could be groups of hosts that use the same tool in a specific way or with a similar behavior that could be clustered together. Second, given that we identified 31 mismatching version strings and fingerprints, we would expect some adversaries to have built custom tools for SSH brute forcing. As these are not publicly available, they should only be in used by a limited group of source hosts, thus the tool fingerprint could be used as a proxy to partially fingerprint the actor.

In the following, we will now investigate the different behaviors of the 49 different key exchange-cipher-MAC algorithm hashes that we initially discovered in our dataset. Figure 5.5 shows the activity of the these fingerprints over the course of the experiment, for compactness of the figure and the discussion, each fingerprint has been assigned a numeric ID from 0 through 48. The number of hosts using a tool with a specific fingerprint at a given time is represented by the size of the marker in the plot, with the area of the markers being proportional to the number of unique hosts using a fingerprint per hour. In the figure we can readily identify five distinct behavioral patterns of fingerprint usage:

- *Popular, commonly available tools* such as Ncrack (fingerprint 30 in cyan), or SSHtrix (fingerprint 16 in red) or Hydra (fingerprint 4 or 14 depending on the

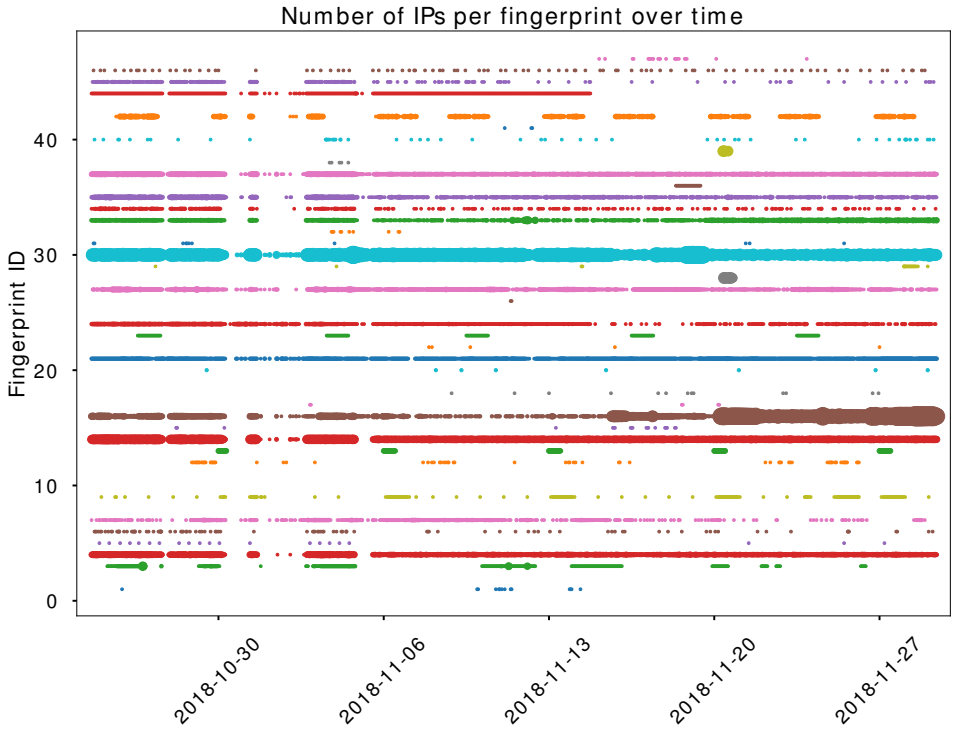


Figure 5.5: Scatter plot showing the number of hosts using a certain key exchange fingerprint over time. The number hosts is indicated by the size of the markers.

operating system it is installed on such as raspbian), see some continuous usage by a diverse and significant number of origins. In the next section, we further show that these groups can be separated by the password list configuration of the brute forcer into distinct subgroups that pursue a common strategy.

- *Custom tools* with a relatively uncommon or unique fingerprint are essentially only in use by distinct groups of adversaries. Often these are rolled out to a large amount of hosts, and from there explore remote hosts simultaneously with synchronized start and stop times. Consider for example fingerprint #23 indicated in green, which performs a weekly recurring scan of the address space, always using a similar amount of resources. Similar behaviors are shown by the groups of hosts using the tool with fingerprint #42 (orange) who scan every other day, or by the groups scanning bi-hourly using the tool with fingerprint #46 (brown).

- Among these, some *custom tools are run at low intensity* only by a select group of origins, which on average make less than 4 login attempts per hour. All installations with fingerprint #36 (purple) operate in this way, and may be classified as a slow brute forcing campaign following the description by Javed and Paxson [20].

- *Distributed hit & run campaigns* only occasionally surface, but then for a short period of time brute force many remote hosts with a large number of resources. Interestingly, we observe these hit & run campaigns to typically exhibit a unique fingerprint and thus employ custom tooling for their activities, which makes these attempts and hosts participating in them easily identifiable by our proposed method. Examples of such fast, concentrated attempts are fingerprints 39 (light green) or fingerprint 29 (gray). The plot shows that both clusters become active during the same time period, around November 20th, and have approximately the same size. A closer inspection reveals that the hosts using the tool identified by fingerprint ID 29 are also using the tool identified by fingerprint ID 39. The IP addresses are located in 103 different /8 subnets indicating that the attacker has the knowledge, resource and intention to spread his or her infrastructure across the Internet, possibly in an attempt to avoid detection.

5.6.4. PASSWORD COMBINATIONS

After the completion of the key exchange and session negotiation, the adversaries were presented a login prompt they could interact with. Past work such as Nicomette et al. [10] have used the entered user credentials to link individual login attempts into related clusters, and for example identified relationships between dictionaries but also noticed that only few of them were reused across adversaries. As discussed before, the economies of scale would imply that an attacker is most likely going to deploy the same setup and tooling on different hosts to launch attacks, thus the same tooling would result in an identical fingerprint, and thus help us gain deeper insights into the activities of the attackers, for example if they are splitting and distributing parts of dictionaries for brute forcing across collaborating hosts. In this section, we will discuss the relationship between groups found based on an identical banner and key exchange, and their associated password lists.

For each of the 49 fingerprints detected earlier, we extracted all SSH sessions from any host that exhibited this signature and assembled the set of credentials (username + password) during login attempts. Table 5.1 shows a selection of 8 fingerprints, which exemplarily shows the spectrum of different key exchange - password list behaviors found throughout the dataset. From the data we can distinguish three types of groups: First, we see clusters where tools and the credential list used are tightly linked together. Second, we clearly see select fingerprints in wide use, which focus on (subsets of) fixed password lists. Third, we observe groups of tooling, where hosts pursue brute forcing with diverse and customized password lists. We will discuss each of these three categories in the following subsections.

HIGH CREDENTIAL / TOOL CORRELATION

For each group advertising the same banner and using the same key exchange algorithm, the table lists the number of IP addresses matching this fingerprint and the number of unique login credentials list used by an attacker belonging to the cluster. To provide a better understanding of the login credential lists used, the number of hosts using the 5 most frequently used credential lists are shown.

Table 5.1: Table illustrating observed relations between key exchange, credentials, and behavior

Type	High credential / tool correlation			Popular tool		Diverse credential lists		
Banner	SSH-2.0-OpenSSH_7.4p1 Raspbian-10+deb9u3	SSH-2.0-OpenSSH_7.4p1 Raspbian-10+deb9u4	SSH-2.0-libssh2_1.8.1_DEV	SSH-2.0-libssh2_1.7.0	SSH-2.0-libssh2_1.8.0	SSH-2.0-Go	SSH-2.0-Go	SSH-2.0-libssh-0.6.3
Kex	0df0d56bb50c6b2426d8d40234bf1826	0df0d56bb50c6b2426d8d40234bf1826	1616c6d18e845e7a01168a44591f7a35	a7a87fbe86774c2e40cc4a7ea2ab1b3c	a7a87fbe86774c2e40cc4a7ea2ab1b3c	c39f4cec145ee3d50fb590595143b9d5	72d744cee7c48197c1b56973e8600140	51cba57125523ce4b9db67714a90bf6e
Cluster	6	7	8	4	5	1	2	3
IP count	684	1138	85	6473	86805	564	208	4479
# cred. lists	9	5	7	2688	635	557	111	4438
Credential list								
Top 1 list	672	1127	79	3602	64140	3	65	18
Top 2 list	2	6	1	16	6024	2	11	3
Top 3 list	2	3	1	16	4488	2	7	3
Top 4 list	2	1	1	13	4347	2	5	3
Top 5 list	2	1	1	10	2832	2	3	2

All clusters in this category exhibit a tight link between the fingerprint and the utilized password list. For example, the fingerprint `0df0d56bb50c6b2426d8d40234bf1826` of cluster 1 is sent by 684 hosts, however within this group only 9 different password lists are used. The vast majority of hosts in this cluster, 672 or 98.2%, always send the exact set of credentials to our honeypots, deviations of the cluster default occur only very infrequently among all remote hosts having connected to our honeypots. In addition to the strong link from a particular fingerprint to a credential list, also the reverse is true: no other attackers have been using this credentials list in the dataset. This would indicate that the proposed fingerprinting method can be used as a predictor for password usage.

POPULAR TOOL

The second category of behaviors we can distinguish in the fingerprint analysis are those tools which are widely deployed but are run similarly configured. In this particular category, we observe the presence of multiple common credential lists, from which hosts pick a subset and brute force all of our honeypots with the same credential set.

An example of this is cluster 5 with 86,805 IP addresses, which employed a surprisingly low number of 625 login credentials lists. The top five groups all settle on a permutation of *(admin, admin), (admin, default), (admin, password)* and move horizontally throughout our ranges. Other groups of hosts choose from lists geared towards specific type of devices, for example credential lists associated with common IoT devices or Raspberry Pi distributions.

DIVERSE CREDENTIAL LISTS

While both previously described clusters could also have be discovered using password-based grouping as adversaries shared significant credentials, we found a third behavior of brute forcing which would have remained undetected to established methods. Clusters 6 through 8 belonged to a new category characterized by a high number of credentials list and a low number of IPs using identical credential lists.

Consider for example the case of group 6, where 557 different password lists appear across 564 different hosts. This is due to the fact that attackers in this cluster use random or pseudo randomly generated passwords in combination with 22 fixed credential tuples. The generated passwords are all 10 characters long consisting of letters and numbers. Next to randomization, each host only performs a limited amount of login credentials, 80% (463 out of 564 IPs) use less than 60 unique login credentials. Another indicator of randomization is the low overlap between passwords, 6,592 of the 10,318 login credentials are used by only one IP address. The password randomization is confirmed by a mean Jaccard index of 0.4 of the credential lists used in the cluster.

While randomization of credentials would make it challenging for a password-based clustering algorithm to detect similarly acting groups, the same will hold true for brute forcers that rely on a *very* long lists of candidate credentials from which username/password combinations are picked. This will dilute possible linkage from the perspective of common credentials, however a clustering based on the fingerprint will find these relationships.

In this section we have shown that clustering based on SSH banners and key exchange algorithms can find different types of clusters. The proposed method is able to find groups using extensive password lists or random password, whereas password-based solutions would struggle. However, password-based solutions can provide better clustering performance for popular tools having multiple credentials lists such as cluster 5, hence a combination of both a fingerprint-based and password-based approach promises to provide more, and complementary findings.

5.7. CONCLUSION

In this work, we have described a method for fingerprinting tools for SSH brute forcing based on version strings and advertised algorithms. As this method distinguishes tools based on the data exchanged during the key initialization, we are able to detect tools prior to even entering the session authentication, and can deploy the mechanism transparently without any changes necessary to an existing enterprise infrastructure.

We have deployed the fingerprinting method over 4,500 honeypots for a period of one month, and from 35 million login attempts been able to detect 49 different fingerprints. The results indicate that different tools are used in different ways, indicating that attackers customize, or develop their own tools. Looking at the behavior of different tools, we were able to identify clear timing patterns originating from different tools, while based on timing patterns we can detect distributed brute forcing campaigns. By fingerprinting the tools used in a campaign, we are able to track and analyze the campaign over time. Additionally password analysis of detected clusters allowed us to identify different brute forcing methods. Both assessments contributed in providing insights into the tactics, techniques and procedures of the attackers.

FUTURE WORK

The work presented in this paper was conducted from the angle of cyber threat intelligence, with the aim of augmenting the portfolio of methods to fingerprint adversarial tooling and gather insights into their activities and behavior. This study led to a variety of different fingerprints, some of which could be traced back to specific brute-forcing tools. In principle, the contribution of this method could however be wider, and potentially be suitable as an additional detection rule within the context of intrusion detection systems. To evaluate its merit for such active threat detection and prevention, further research is however needed to evaluate its efficacy, which has not been done within the scope of this study.

References

[1] D. J. Bianco. *The Pyramid of Pain*. 2013. URL: http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html.

[2] M. Husák, M. ermák, T. Jirsík and P. eleda. 'HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting'. In: *The European Association for Signal Processing Journal on Information Security* (2016).

[3] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman and V. Paxson. 'The Security Impact of HTTPS Interception'. In: *The Network and Distributed System Security Symposium*. 2017.

[4] V. Ghiette, N. Blenn and C. Doerr. 'Remote Identification of Port Scan Toolchains'. In: *IFIP International Conference on New Technologies, Mobility and Security*. 2016.

[5] 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.

[6] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan and L. Qiu. 'Statistical identification of encrypted web browsing traffic'. In: *IEEE Symposium on Security and Privacy*. 2002.

[7] M. Korczyski and A. Duda. 'Markov chain fingerprinting to classify encrypted traffic'. In: *IEEE International Conference on Computer Communications*. 2014.

[8] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre and A. Pras. 'SSHCure: a flow-based SSH intrusion detection system'. In: *Conference on Autonomous Infrastructure, Management and Security*. 2012.

[9] M. M. Najafabadi, T. M. Khoshgoftaar, C. Kemp, N. Seliya and R. Zuech. 'Machine learning for detecting brute force attacks at the network level'. In: *International Conference on Bioinformatics and Bioengineering*. 2014.

[10] V. Nicomette, M. Kaâniche, E. Alata and M. Herrb. 'Set-up and deployment of a high-interaction honeypot: experiment and lessons learned'. In: *Journal in Computer Virology* (2011).

[11] D. Ramsbrock, R. Berthier and M. Cukier. 'Profiling Attacker Behavior Following SSH Compromises'. In: *IEEE/IFIP International Conference on Dependable Systems and Networks*. 2007.

[12] T. Barron and N. Nikiforakis. 'Picky Attackers: Quantifying the Role of System Properties on Intruder Behavior'. In: *Annual Computer Security Applications Conference*. 2017.

[13] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Transport Layer Protocol*. Tech. rep. Internet Engineering Task Force, 2006.

[14] M. Heuse, D. Maciejak and J. Dlabal. *Hydra*. https://github.com/vanhauser-thc/thc-hydra. URL: https://github.com/vanhauser-thc/thc-hydra.

[15] J. Mondloch. *Medusa*. http://foofus.net/goons/jmk/medusa/medusa.html. URL: http://foofus.net/goons/jmk/medusa/medusa.html.

[16] G. Lyon and F. Chantzis. *Ncrack*. https://nmap.org/ncrack/. URL: https://nmap.org/ncrack/.

[17] A. Adamantiadis, A. Schneider, N. Zitzmann, N. Kiesel and J.-P. G. Ballester. *libssh*. https://www.libssh.org/. URL: https://www.libssh.org/.

[18] D. Stenberg, M. Hörsken, V. Szakats and W. Cosgrove. *libssh2*. https://www.libssh2.org/. URL: https://www.libssh2.org/.

[19] A. Vetterl and R. Clayton. 'Bitter Harvest: Systematically Fingerprinting Low- and Medium-interaction Honeypots at Internet Scale'. In: *USENIX Workshop on Offensive Technologies*. 2018.

[20] M. Javed and V. Paxson. 'Detecting stealthy, distributed SSH brute-forcing'. In: *ACM Special Interest Group on Security, Audit and Control Conference on Computer & Communications Security*. 2013.

6

WHY WAS THE IOT BOTNET MIRAI SO SUCCESSFUL? A DEEP DIVE INTO MIRAI'S BRUTE-FORCING STRATEGY

This dissertation contributes to the state of the art by identifying attackers' toolchains during the early attack stages, allowing the study of criminals' behavior. In the previous chapter, we analyzed over 35 million SSH brute-forcing attempts. We concluded that attackers compile their toolchains using various SSH libraries and versions thereof allowing the identification of unique toolchains, which in turn, allows for the behavioral analysis of cybercriminals. This chapter presents a case study analyzing the changes cybercriminals make to the configuration of a successful and widespread botnet.

Each year new botnets are discovered and their tenacity, adoption, and infection rate are notorious, providing vast resources for the criminals operating them. In this chapter, we analyze millions of Mirai brute forcing attempts, extracting information regarding the configurations made by the botmasters. Analyzing, the configurations we can study the behavior of cybercriminals during the brute forcing stages of an attack, and more importantly see how different criminals adopt various strategies.

In our work, we use the brute forcing credentials, their associated weights, and the number of attempts to reconstruct the configurations used by the botmasters. We see evidence of botmasters, trying out new credentials before deploying the new configuration over the entire botnet. We identify criminals using a different number of brute forcing attempts per target, indicating that criminals are adopting either a breadth or depth-first brute forcing strategy. Analyzing the frequency with which bots use certain credentials we reconstruct the weights assigned to the pseudo-random credential selection used by Mirai botmasters. The weighted analysis shows that criminals believe that some credentials are more effective than others. This chapter showcases the ability to extract cyber threat intelligence, during the actual brute forcing attempts, contributing to detecting and differentiating cybercriminals' toolchains at the early stage of attacks.

6.1. INTRODUCTION

Compromised hosts under adversarial control – or bots – are the starting point of many cyber criminal activities. When thousands if not tens of thousands of such bots are assembled into a botnet, they become a platform to launch spam [1, 2], perform distributed denial-of-service (DDoS) attacks [3], or commit click fraud [4], and their owners, the so-called botmasters, have been observed to rent out part of their infrastructure for nefarious use [5].

Botnets are valuable assets for cybercriminals. As the botnet infrastructure is the botmaster's platform to generate revenue, it is of significant value to him and likely to receive attention with respect to maintenance and upkeep. As infected users will at some point notice odd behavior of the host, for example as webpages with advertisements open automatically or the machine is slowed down by the botnet's activities, some portion of the botnet will inevitably disappear on a regular basis, as victims clean their system, reinstall or disconnect it from the Internet. Also Internet Service Providers (ISP) often alert customers about an infection, and have in the past even gone as far as to block Internet access until the infection is removed by the user [6]. Finally, some malware such as Mirai – the largest IoT malware to date – is even volatile and will disappear from an IoT device after a reboot, making the device open for re-infection. Botnets therefore have a tendency to shrink over time.

In order to maintain their installation base or even proliferate, botmasters have to constantly find and add new devices to their setup. Botnets can increase by waiting for new vulnerable devices to appear, taking away devices from other botnets, or innovating and growing into niches targeted previously uninfected devices. Griffioen and Doerr [7] show that the ecosystem of vulnerable IoT devices is finite, and botmasters routinely steal devices from each other. As this is essentially a zero-sum game with no long term advantage, botmasters therefore also constantly search for new username/password combinations that are used as default credentials in Internet-connected devices to unlock a previously unoccupied niche. In their 2017 study of Mirai, Antonakakis et al. [8] found the Mirai malware to contain 371 username/password combinations, of which 46 could be linked to specific device vendors and types. As we show in this paper, this had already grown to more than 2500 unique combinations just one year later, in an attempt to customize and better proliferate.

While previous work notes that IoT malware goes through phases of evolution [7, 8], how exactly botmasters innovate specifically and the advantage such innovation provides has not been shown in detail. In this paper, we focus on credential bruteforcing, which malware uses to break into and gain a foothold on Internet-connected devices. Given the size of IoT-based botnets and their track record in previous large-scale Internet attacks [9], we investigate this evolution for the Mirai IoT malware and report on an installation of 7,800 honeypots designed to trap IoT malware, which recorded 98 million login sessions from 174,742 IP addresses over the course of one month. Even within this comparatively short time period, we can see clear signs of evolution and innovation on how Mirai-based botnets try to break into victims, and we track how 36 Mirai variants adapt their username/password lists, modify the source code, and adjust the weighting system to gain an advantage. While previous papers have remarked that the curation of credential lists is an important element to Mirai's success, this study is the first to in-

vestigate botmasters' techniques, tactics and procedures in depth and characterize the benefit these practices mean in the wild.

With our paper, we make the following four contributions:

- We deploy an environment of 7,800 honeytrap and shows using 98 million login sessions that botmasters update and optimize how they brute-force their victims, for example by tuning weights, introducing new niche credentials or changing the intensity and ordering of attempts.
- The large volume of honeypots and login attempts allows us to empirically reconstruct the credential list used by the different actors and botnet variants, which provides an insight into botmasters' beliefs about the efficiency of brute-forcing credentials.
- We show that botmasters modify Mirai's source code, but follow "good engineering practices" by performing limited test runs on new innovations, before rolling out changes through their botnets.
- We demonstrate that the identified configuration changes criminals make, improves Mirai's effectiveness.

6.2. RELATED WORK

When deploying a botnet, a botmaster has the incentive to target as many devices as possible in order to increase the size of the botnet [10]. However, targeting devices located

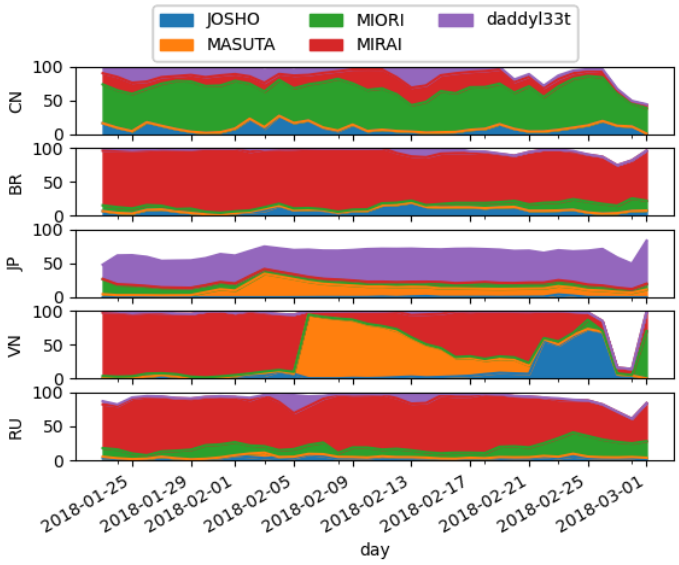


Figure 6.1: Infection percentage per country

across the Internet might be counterproductive. Attacking devices within certain ranges, such as the address space claimed by the United States Department of Defence (DOD), might attract unwanted attention. Therefore, the original authors of Mirai have opted to blacklist the DOD range together with other ranges [8]. Besides blacklisting ranges used by government instances, Guarnizo et al. [11] show that botmasters customize their blacklists to exclude specific geographical locations such as cities. More so, Barron and Nikiforakis [12] determine that attackers tend to avoid targeting devices in IP ranges belonging to certain hosting providers, preferring to target devices at Amazon over devices located in the IP space of Linode. Even botmasters using the same leaked Mirai source code [13] adapt their blacklist to include more subnets. Griffioen and Doerr [7] show that Mirai-based botnets employ different blacklisting strategies. The different blacklisting strategies identified in the works above show that botmasters are willing to modify and adapt their strategies.

Avoiding detection through the use of blacklists is one aspect in which botmasters have diversified. In order to expand a botnet devices need to be captured. Devices running remote login services can be compromised by brute-forcing the login credentials. Several studies [12, 14] show that different botnets share the same brute forcing dictionaries with [14] noting that 81.5% of the brute-forcing attempts use *root* as a username. The commonality of the used brute-forcing credentials is self-evident as devices are often configured using standard login credentials. However, due to the publication of the Mirai source code new variants have emerged [15]. These newcomers target the same vulnerable devices increasing the competition between botnets. As a result, botmasters have adopted new brute-forcing credentials as shown by [7, 8] in order to increase the devices which they can capture.

The adoption of different strategies is not limited to the alteration of blacklists and credential dictionaries. Once a device is brute-forced attackers deploy different strategies to secure the device and add it to their botnets. Lingenfelter et al. [16] show that botmasters use different commands to compromised the captured devices. Next to diversifying the commands used to control devices, attackers have expanded the type of devices which can be added to their botnets. With the wide diversity in IoT devices, attackers can increase their botnet sizes by supporting new architectures. Both [17] and [18] noticed that botnets are supporting a various number of device architectures. Similarly [19] and [17] note that Mirai-like botnets have added new architectures to their arsenal.

The reviewed works show that botmasters continuously adapt their strategies to expand their botnet. In the following, we complement the state of the art by first showing that the simplicity of Mirai's code contributed to its success. Second, we perform an in-depth analysis of Mirai's credential selection procedure, showing that it allows botmasters to control the frequency with which credentials are used during brute-forcing attempts. Third, we show that botmasters change Mirai's brute-forcing procedure. Finally, we simulate botnets using the observed alterations showing that the original implementation has a higher likelihood of successfully brute-forcing devices.

6.3. DATASET

With many different botnets competing for the same resources, botmasters need to innovate to stay ahead of the competition. In addition, due to clean-up operations and general awareness, the number of infectable devices is declining [20]. The decline of the general bot population and the multitude of Mirai variants identified by Griffioen and Doerr [7] indeed suggest that botmasters must adapt their strategies in order to keep their botnets relevant. In this study, we capture different strategies employed by Mirai botmasters to increase the effectiveness of their botnet.

To study the alterations made by botmasters, we require an infrastructure capable of simulating telnet-enabled devices, as these are the devices targeted by Mirai botnets. To fulfill our needs, we use Honeytrap [21], a framework designed to enable the large-scale deployment and monitoring of high interaction honeypots. While Honeytrap allows us to deploy a large-scale honeynet, it cannot deploy honeypots emulating different devices. Research performed by Guarnizo et al. [11] shows that cybercriminals tend to target specific devices due to known exploits. Therefore, we extend Honeytrap with our Honeytrack framework and deploy more than 7,800 honeypots emulating various devices across multiple subnets. Deploying heterogeneous honeypots allows us to capture the different behaviors of attacking bots allowing us to uncover changes made by botmasters. This section provides an overview of our selected honeypot deployment strategy.

6.3.1. DEPLOYING HONEYPOTS IN VARIOUS SUBNETS

Mirai botnets are known to randomly select IP addresses to determine their next victim and use extensive blacklists [7]. Therefore, we decide to deploy a large honeynet across different subnets, providing two benefits over smaller local deployments. First, if attackers are randomly probing the Internet for potential victims, increasing the number of IP addresses used in the honeynet should increase the probability of being attacked. Second, deploying the honeypots across several subnets should increase the number of attacks we can analyze as some attackers might blacklist part of the honeynet ranges.

During our study, we assigned 7,824 IP addresses to the honeynet. The majority of these addresses, 7,749, are part of three /16 subnets belonging to our university. One subnet is used for hosting infrastructure such as servers, employees use the second subnet, and the third is reserved for students and guests. The remaining honeypots are deployed at hosting providers and home users.

We operated our honeynet from the end of January 2018 until the beginning of March 2018. During the collection period, we record 165,387,578 attacks towards our honeypots.

6.3.2. PRESENTING DIFFERENT DEVICES TO THE ATTACKERS

Attackers tend to target different devices [11]. Therefore bots might adapt their attack when confronted with different devices. Attackers having no use for IP cameras might instruct their bot to abort the attack when confronted with such a device. To analyze the different behaviors of bots, we have set up our honeypots to emulate eight different

Table 6.1: Telnet banners used by our honeypots

#	Banner	Rank	Device type
1	This is a honeypot, please don't log in	-	Honeypot
2	Welcome Visiting Huawei Home Gateway Copyright by Huawei Technologies Co. , Ltd. Login:	6	Home modem
3	Welcome to Microsoft Telnet Service Login: **********	34	Microsoft Telnet server
4	Welcome to ZXR10 Carrier-class High-end Routing Switch of ZTE Corporation ********** Username:	144	Switch
5	MikroTik v6.35.4 (stable) Login:	19	Router
6	RICOH Maintenance Shell. User access verification. Login:	162	Printer
7	Remote Management Console Login:	18	Remote Mgmt service
8	User Access Verification Username:	1	Generic switch/router

devices. After connecting to a telnet device, the device sends a banner containing information. The banner might contain information such as manufacturer, device type, and in some cases, the version number of the software. Attackers can use the information gathered from the telnet banner and adapt their attacks accordingly. Table 6.1 shows the eight banners used to emulate different types of devices. The table also includes the type of device and the ranking of the banner. The ranking is retrieved from a telnet scan performed by Censys [22] in late January 2018, grabbing the banners of all telnet devices on the Internet and sorting them by popularity. To ensure all device types are equally present in our honeynet, we randomly select a banner when an attack from a source IP is performed against a honeypot for the first time. Upon subsequent attacks, the same banner is used to ensure we maintain a persistent honeynet.

6.3.3. IDENTIFYING MIRAI ATTACKS

Our goal is to study the adaptations made by Mirai botmasters. Therefore, we have set up a honeynet emulating telnet device. However, Mirai is not the only botnet targeting telnet devices [23]. To exclude attacks performed by non-Mirai botnets, we use the same method used by Griffieon and Doerr [7]. After sending the login credentials, Mirai bots immediately send a fixed set of commands ending with */bin/busybox Mirai* without waiting for a response from the targeted device. The last command is used to identify whether *Busybox* is installed. Both [7] and [15] show that Mirai variants alter the last command changing *Mirai* to another name. As we are only interested in Mirai and its variations, we filter out any attacks which do not match Mirai's behavior allowing the word *Mirai* in the last command sent to be changed into another name. In the remainder of this paper, we use the word of the last command sent to name Mirai variants. After filtering, our dataset contains 98,817,811 attacks performed by 174,742 bots using one of 36 Mirai variants.

6.4. SIMPLICITY THAT CONTRIBUTED TO MIRAI'S SUCCESS

The Mirai botnet is one of the most successful botnets in recent times. Despite competing with various other botnets [23] and the emergence of awareness campaigns and cleanup operations, the botnet managed to infect many devices, with Antonakakis et al. [8] estimating Mirai to contain 300,000 devices and news reports estimating that the botnet contains over two million devices. The leaked source code [13] also contributed to the success of Mirai bots in general, allowing botmasters to create custom variants [7]. In this section, we provide an insight into the success of Mirai. We first show that Mirai bots can target more than 95% of the telnet devices on the Internet despite their limited codebase. Second, we show that attackers with limited skills can set up a Mirai botnet.

6.4.1. SIMPLE YET EFFECTIVE SOURCE CODE

The Mirai botnet spreads by infecting new telnet devices. There are many types of telnet-enabled devices ranging from industrial-grade routers to IoT devices. These devices often use different banners to identify themselves and to present a login prompt. From an Internet scan performed by Censys in the last week of January 2018, over 3 million

telnet-enabled devices were active on the Internet using over 300,000 different banners. Botnets, such as Mirai, wanting to brute force telnet devices must be able to parse the sent banners in order to identify a login prompt allowing them to enter brute-forcing credentials. The variety in advertised banners, together with no requirements imposed by the RFC [24] for these banners, poses a challenge for attackers targeting telnet devices. In this section, we analyze the source code used by Mirai to identify login prompts and show that the authors implemented a simple yet effective login prompt detection mechanism, allowing them to identify and thus brute-force 95% of the telnet devices.

MIRAI'S TELNET IMPLEMENTATION

The leaked source code of Mirai [13] allows us to have an insight into the brute-forcing mechanism used by Mirai. The entire code base responsible for finding and brute-forcing vulnerable devices is less than 1000 lines, which is relatively small compared to the 2500 lines of code used by the GNU Inetutils telnet implementation [25]. Analyzing the entire source code responsible for infecting devices is out of the scope of this research. Therefore, we provide a summary of the steps taken by a Mirai bot to attack a device. The first step taken by a Mirai bot is to identify new vulnerable devices. A bot scans the Internet for machines running the telnet service. Once the bot identifies a telnet device, it sets up a TCP connection using the *connect* function provided by the Operating System. After establishing the TCP connection, the bot initiates a telnet connection. The bot exchanges its preferred telnet options as required by the telnet specifications [24]. After completing the negotiations, the bot attempts to identify the login prompt sent by the device under attack. After successfully identifying a login prompt, the bot sends login credentials followed by a string of commands. Suppose the bot fails to identify a login prompt. In that case, the bot continues targeting other devices, and the connection eventually times out, causing the bot to increment the number of failed brute-forcing attempts for that device. When a brute-forcing attempt is successful, the bot informs the command and control server about the brute-forced device. Each bot attempts to brute force the same devices ten times, after which a new device is targeted.

We previously mentioned that in 2018 there were over 300,000 different banners advertised by telnet devices. Therefore bots must be able to identify the login prompts in different banners, as failing to do so results in failed brute forcing attempts. The function responsible for identifying login prompts is shown in listing 6.1. From the code, we can derive that bots will identify login prompts using a character set (*':'*, *'>'*, *'$'*, *'#'*, *'%'*) and substrings (*'ogin'*, *'enter'*). When the function finds such a character or substring, it returns the position of the found object, and the bot will send a login credential. If no login prompt is found, the function returns -1, and the bot will let the connection timeout and continue its routine. The login prompt detection implemented by Mirai's authors may seem crude as it relies on five characters or two substrings being present in the banners. However, in the following subsection, we show that this approach allows Mirai bots to target 95% of the telnet devices.

USED TELNET BANNERS

We show in the previous section that the function used by Mirai bots to identify login prompts in telnet banners is relatively small. Only five characters and two substrings

```
static int consume_user_prompt(struct scanner_connection *conn) {
  char *pch;
  int i, prompt_ending = -1;

  for (i = conn->rdbuf_pos - 1; i > 0; i--) {
    if (conn->rdbuf[i] == ':' || conn->rdbuf[i] == '>' || conn->rdbuf[i] == '$' || conn->rdbuf[i] == '#' || conn->rdbuf[i] == '%') {
      prompt_ending = i + 1;
      break;
    }
  }

  if (prompt_ending == -1) {
    int tmp;

    if ((tmp = util_memsearch(conn->rdbuf, conn->rdbuf_pos, "ogin", 4)) != -1)
      prompt_ending = tmp;
    else if ((tmp = util_memsearch(conn->rdbuf, conn->rdbuf_pos, "enter", 5)) != -1)
      prompt_ending = tmp;
  }

  if (prompt_ending == -1)
    return 0;
  else
    return prompt_ending;
}
```

Listing 6.1: Mirai telnet username handler

are used to identify potential login prompts. A scan performed by Censys in January 2018 shows that telnet devices use more than 300,000 different banners. The small codebase combined with the abundance of different banners used by telnet devices raises the question of what percentage of the telnet device population Mirai bots can brute force. In this subsection, we show that Mirai bots were able to attack 95% of the telnet devices found on the Internet despite its simple login detection function.

For our analysis we first require to collect the banners used by telnet devices. We use scans grabbing the banners of all telnet devices found on the Internet performed by Censys. Six different scans separated by a week each are used to account for fluctuations in the telnet device population. To identify the devices that can be attacked by Mirai bots we implement the same login prompt identification function as used by Mirai. Table 6.2 shows the amount of banners which can or cannot be parsed by Mirai. In addition the table shows the number of devices using banners which can be parsed by Mirai, providing an indication of the number of devices which can be brute forced. On average there were 3,354,566 telnet devices connected to the Internet of which 95.31% could be brute force by Mirai.

6.4.2. A SIMPLE BOTNET TO SETUP

In the forum post leaking the source code of Mirai, the original author states that the botnet is easy to set up, requiring one hour of work and two servers to set up a functional botnet [13].

The ease of setting up a Mirai botnet is reflected by the 36 variants identified in our dataset spanning one month. This ease must have contributed to the wide adoption of the botnet software, and we see also lesser-skilled attackers deploying the software.

A curious visualization of unskilled usage of Mirai is the incorrect inclusion of new credentials. The original author of Mirai decided to include the brute-forcing credentials in the source code. These credentials were obfuscated by XORing each character with an 8 bit key, probably to evaluate anti-virus signatures. During brute-forcing, the list of pass-

Table 6.2: Telnet devices which Mirai can or cannot brute force

Date	Can brute force	Devices	Banners
2018-01-30	False	157798	17300
2018-01-30	True	3218585	319680
2018-02-06	False	157089	17129
2018-02-06	True	3158442	319363
2018-02-13	False	157101	17129
2018-02-13	True	3159015	319371
2018-02-20	False	158345	17224
2018-02-20	True	3195341	320908
2018-02-27	False	156791	17073
2018-02-27	True	3219014	343499
2018-03-06	False	157014	17139
2018-03-06	True	3232866	345596

Table 6.3: Botnet marketshare per device banner

Banner	Variant	%
=== IMPORTANT ==========	NGRLS	78.44
Use 'passwd' to set your login password	MIRAI	18.69
this will disable telnet and enable SSH	SORA	0.82
————————	RBGLZ	0.62
K2 login:	MIORI	0.41

Banner	Varian	%
	MIORI	69.95
	JOSHO	10.84
zzgx login:	daddyl33t	7.39
	MIRAI	3.94
	Cult	2.96

Banner	Variant	%
	MIRAI	52.96
	JOSHO	19.02
(none) login:	daddyl33t	5.71
	MIORI	4.85
	Zeus	4.46

words is decoded with the fixed key. In our dataset, we identify two Mirai variants named Ngrls and Word that were sending both meaningful credentials as well as gibberish. As the "key" is only 8 bits, it is trivial to test which "key" is necessary to turn these non-sense credentials into meaningful username/password combinations. For instance, k = 125 would turn `"\x1c\x19\x10\x14\x13 \x1c\x19\x10\x14\x13\x1f\x13DD"` into *admin adminbn99*, a standard credential used in Barracuda firewalls. This at first obscure behavior can be explained by uninformed copy&pasting of botnet owners, who are unaware of the exact working of Mirai. These credentials (and used key) can be traced back to different code repositories on sites such as GitHub [26] as well as the dark web, where variants of Mirai have been posted. Botmasters seem to copy credentials of other repos and include it in their own branch, not correcting for the obfuscation. This allows us to trace where and when someone obtained new credentials from. The owner of the Ngrls variant of Mirai copy & pasted new credentials in multiple round, but after 10 days recognizes the issue and rolls out a patch. The perpetrator behind the Word variant did not notice the issue and continued until the end of our study as-is.

6.5. BOTNETS ADOPTING NEW CREDENTIALS

We have mentioned before that botmasters need to innovate in order to keep their botnets from shrinking. An obvious solution is to include more credentials into their brute-forcing dictionaries. By increasing the number of brute-forcing credentials bots will have a higher chance of succeeding in their attack. In this section, we show that botmasters do indeed add new credentials to their brute-forcing dictionaries. In addition, we show

that botmasters adopt different strategies when selecting the credentials to add to their brute-forcing dictionaries.

6.5.1. DIVERSE BRUTE FORCING CREDENTIALS

Adding new passwords to a brute-forcing dictionary is a tactic that botmasters can employ to increase the bot population. This subsection shows that botmasters add new brute-forcing credentials and that the added credentials differ from one botnet to another. In order to analyze the used brute-forcing credentials for the different botnets, we first need to identify different botnets. Second, we need to extract the brute-forcing credentials used per botnet. Finally, we need to compare the dictionaries used by different botnets to each other to assess their similarity.

In section 6.3.3 we demonstrate how different Mirai variants can be identified by using the last command sent during a brute-forcing attack. After identification, we extract a brute-forcing dictionary using the brute-forcing attempts performed by the bots of each variant. Our data set contains 36 different Mirai variants using 2521 unique credentials. Later, in section 6.6, we show that the credential selection process is semi-random. To ensure we have gathered the entire brute-forcing dictionary, we analyze the variants, having performed a minimum of 200,000 brute-forcing attempts. The resulting dataset contains 14 variants.

To compare the similarity of the used dictionaries, we require a method to calculate the similarity between lists. Therefore, we use the Jaccard index, a metric that divides the length of the intersection by the length of the union of two lists providing a number between zero and one. A score of one indicates that the lists are identical, while lists with no items in common have a score of 1zero. Figure 6.2 is a heatmap showing the calculated Jaccard indices of the brute-forcing dictionaries used by different Mirai variants. The mostly blue-toned heat map shows that the used dictionaries have few passwords in common. Indeed, 73.46% of the calculated indices have a Jaccard index lower than 0.2 suggesting that most botmasters include various credentials in their lists. Besides using different credentials in their brute-forcing lists, botmasters also have different credential lists sizes. The recorded dictionary sizes fluctuate between 14 and 653 entries, with a median and mean of 195 and 204.21.

6.5.2. INCORPORATING NEW BRUTE FORCING CREDENTIALS

The previous subsection shows that botnets use different brute-forcing dictionaries varying in size and credentials. A botmaster can adopt new brute-forcing credentials in several manners. The botmaster can simply instruct the entire botnet to use new credentials, as revealed in section 6.4.2 showing a lesser skilled attacker updating the credential list for the entire botnet. Alternatively, a botmaster can first test the brute-forcing efficiency of new credentials before adding them to the brute-forcing dictionary. A botmaster has an incentive to maintain an effective brute-forcing dictionary. The source code of Mirai limits bots to perform a maximum of ten brute-forcing attempts per targeted device [13]. Furthermore, we show in section 6.6 that the credentials chosen during the brute-forcing procedure are semi-randomly chosen. Therefore, during a brute-forcing attack, less effective credentials can take the place of the more effective ones causing

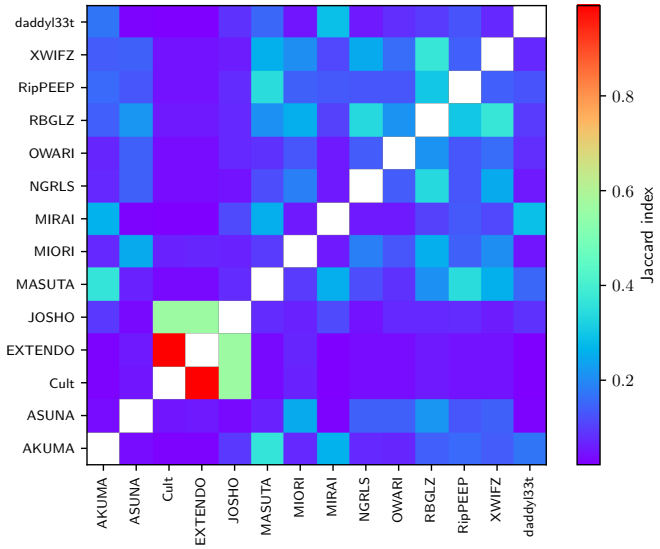


Figure 6.2: The jaccard similarity between the credential dictionaries used by different botnets.

the brute-forcing to fail. This section shows a botmaster testing the efficiency of credentials prior to updating the entire botnet. In order to analyze the testing and adoption of new credentials, we must analyze the credential adoption over time, revealing whether the botmaster tests new credentials prior to updating the brute-forcing dictionary of the botnet.

We identify different Mirai botnets using the same method explained in the previous sections. Again, we use the 14 variants for which we have collected at least 200,000 brute-forcing attempts ensuring we have captured the entire brute-forcing dictionary. To analyze the password adoption strategy, we measure the percentage of bots using a particular credential per day. Measuring the number of bots using certain credentials can reveal whether the botmaster uses a specific set of bots to test new credentials. Out of the 14 identified variants, we find one botnet named Rbglz, for which the botmaster tests credentials using a single bot. The other thirteen botnets do not exhibit signs of botmasters testing out credentials.

Figure 6.3 shows the process of the Rbglz botmaster trying out new credentials and updating the dictionary of the botnet. The figure shows the percentage of bots using one of 53 credentials per day in the form of a scatterplot. In addition, we overlay the figure with the credential used by a single bot, indicated by black ticks inside the dots of the scatter plot. The bot indicated by the black ticks is the only bot using 21 of the 53 credentials on the 23^{rd} and 24^{th} of January. On January the 25_{th} the bot ceases its activities, and none of the 21 credentials are recorded. The day after, the 21 credentials are used by 21.81% of the botnet, which increases to 38.27% the day after. We acknowledge that not all bots are seen using the newly introduced credentials. This behavior is expected because the

number of brute-forcing attempts per targeted device is limited, and the brute-forcing dictionary has grown in size. Furthermore, the difference in the percentage of bots using specific credentials is also expected. We show in section 6.6 that the credential selection process is not random but semi-random, causing some credentials to be used more frequently than others. The fact that a single bot uses specific credentials, after which a large portion of the botnet starts using the same credentials, indicates that the botmaster is testing out new credentials prior to deploying them.

In addition to seeing the botnet adopting new credentials tested by a single bot, a secondary behavior can be noticed involving the same bot. The bot testing out the new credentials does not use three credentials previously used by the botnet. The bot does not use the *bin:12345*, *root:jvbzd*, and *root:xc3511* credentials while 91.16% of the other bots do. The same day the botnet adopts the new credentials, the three credentials not used by the single bot are used less often by the general bot population. The percentage of bots using the three credentials drops from 97.85% on January the 25th to 45.16% and 17.16% the following two days. That not all the bots stop using the three credentials can be explained by the botmaster adjusting the semi-random credential selection process to favor other credentials over the previously mentioned ones.

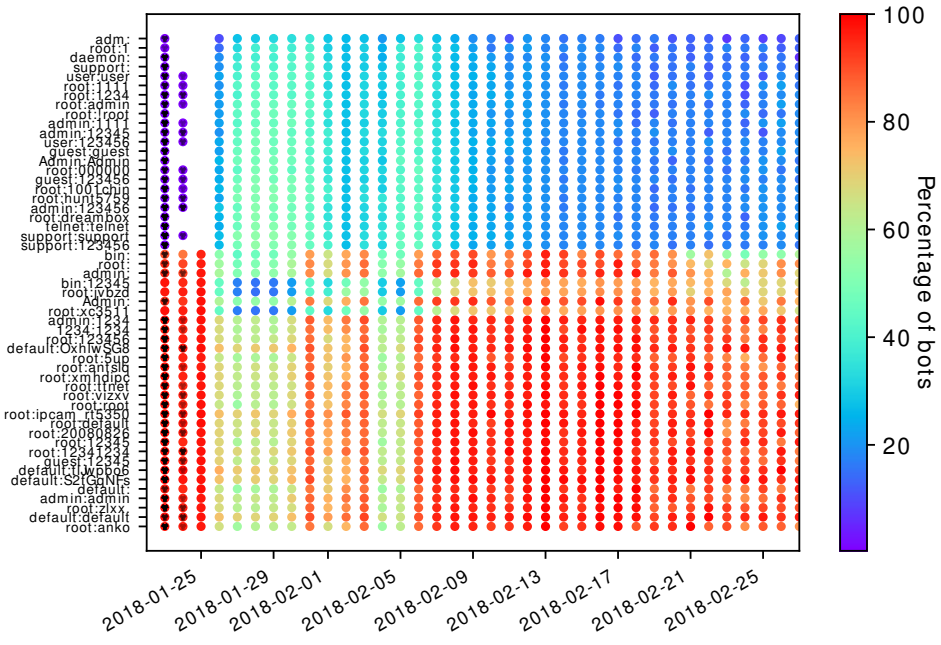


Figure 6.3: Adoption of new brute-forcing credentials for the Rbglz botnet

6.6. MANAGING GROWING CREDENTIAL LISTS

In section 6.5 we show that botmasters add new credentials to their brute-forcing dictionaries. With a growing number of brute-forcing credentials, botmasters must manage which credentials to use while brute-forcing. Some standard credentials such as *admin:admin* might be more effective in brute-forcing devices than more complex or customized credentials. This section shows that botmasters use three approaches to solve the problem of growing credential dictionaries. First, we show that Mirai botnets use a weighted random selection process to pick brute-forcing credentials. Second, we show that botmasters have increased the number of brute-forcing attempts per targeted device. Third, we show that others have opted to fix the credential order, ensuring that each credential is used during a brute-forcing attack.

6.6.1. USING A WEIGHT SYSTEM

With a growing amount of credentials in a brute-forcing dictionary and a fixed number of attempts per targeted device, botmasters must implement a credential selection process. In this subsection, we first analyze the source code of Mirai, showing that the authors have implemented a weighted system determining the frequency with which credentials are used. Second, we show the effects of the implemented strategy. Third, we show that botmasters adopt different weighting strategies.

CODE ANALYSIS

The Mirai authors have implemented the credential selection process in the form of a random selection from a weighted table [13]. When starting, a Mirai bot sets up a table named *auth_table* designed to hold the brute-forcing dictionary. The table has a property named *auth_table_max_weight* keeping track of the total weight of the credentials in the table. Each credential entry possesses three attributes, the credential, minimum weight (*weight_min*), and maximum weight (*weight_max*). When adding a credential to the table, the minimum weight property of the credential entry is set to the current maximum weight of the table. Then, the maximum weight property of the credential is set to the sum of the maximum table weight and the weight of the credential. After adding a credential, the weight of the credential is added to the maximum weight of the table. The process described above is repeated whenever a credential is added to the table. After completing the setup, the bot starts brute-forcing devices using a credential from the table. Listing 6.2 shows the source code in charge of retrieving a credential from the table. The source code shows that the selection process is not truly random and that the chance of selecting a credential depends on its associated weight. A random number *r* is generated lower than the total weight of the table *auth_table_max_weight*. The table is traversed using a for loop, and the credential for which the random value *r* lies between the credential's minimum and maximum weight is returned. By increasing or decreasing a credential's weight, the botmasters can influence the frequency with which the credential will be used during brute-forcing attempts.

```
static struct scanner_auth *random_auth_entry(void) {
  int i;
  uint16_t r = (uint16_t)(rand_next() % auth_table_max_weight);

  for (i = 0; i < auth_table_len; i++) {
    if (r < auth_table[i].weight_min)
      continue;
    else if (r < auth_table[i].weight_max)
      return &auth_table[i];
  }

  return NULL;
}
```

Listing 6.2: Mirai weighted random ccredential selection

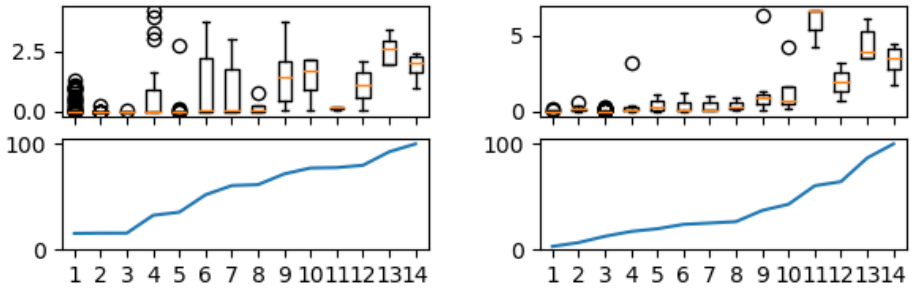


Figure 6.4: Weight Distribution of daddyl33t and JOSHO botnets

EFFECTS OF THE WEIGHT SYSTEM

Mirai's weighted credential selection process dictates the frequency with which credentials are used during brute forcing attempts. By analyzing the frequency of the used credentials during brute-forcing attacks, we can approximate the weight assigned to credentials by the botmasters. Figure 6.5 illustrates the effect of using a weighted system for the Josho variant. The figure shows the percentage of brute-forcing attempts per day per credential (translated to numerical value on the y axis). The data shows that when the brute-forcing dictionary was small, between the 22nd and 31st of January, the credentials were used between 2.26% and 9.85% of the time. However, when the dictionary size increases on the 7th of February, the usage percentage varies between 0.001% and 7.12%. Even though over 200 credentials are added, the weighted selection process allows the botmaster to control the frequency with which credentials are used. While the newly added credentials are seldomly used due to their low weight, the initial credentials are more frequently used during brute-forcing attempts. The example above illustrates that the weighted selecting process allows a botmaster to prioritize the usage of particular brute-forcing credentials enabling the addition of new brute-forcing credentials without losing control over the frequency with which credentials are used.

DIFFERENT WEIGHTING STRATEGIES

The weight system allows botmasters to manipulate the frequency with which credentials are used during brute-forcing attacks. Botmasters likely assign a higher weight to

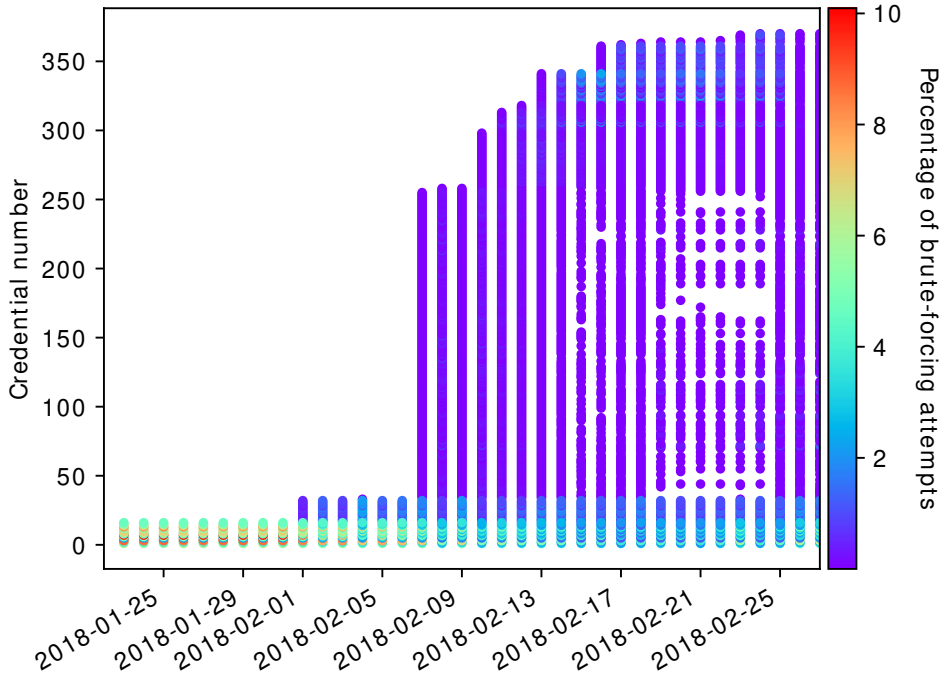


Figure 6.5: Percentage of credentials used during burte-forcing attacks by the Josho botnet

effective credentials than to lesser ones. Therefore, the weights assigned to the different credentials can provide an insight into which credentials a botmaster believes to be effective. Table 6.4 shows the top 5 used credentials and associated weights for the Mirai, Miori, and Rbglz botnets. The weights are expressed in the percentage of usage during brute forcing attempts. Table 6.4 shows that botmasters favor different credentials. All top 5 used credentials, except for credential *admin:admin*, are different for the represented botnets suggesting that botmasters have different beliefs about the effectiveness of credentials.

Table 6.4: Top 5 credentials and associated weights of the Mirai, Miori and Rbglz botnets

Mirai		**Miori**		**Rbglz**	
credential	**weight**	**credential**	**weight**	**credential**	**weight**
root:xc3511	3.97	root:ipcam_rt5350	4.52	default:OxhlwSG8	10.35
root:vizxv	3.63	default:OxhlwSG8	4.23	default:S2fGqNFs	10.23
admin:admin	3.58	default:S2fGqNFs	4.22	root:7ujMko0admin	7.36
root:root	2.98	default:tlJwpbo6	4.22	admin:admin	7.05
root:admin	2.84	default:default	3.97	root:hunt5759	6.12

Table 6.5: Percentage of brute-forcing sessions with unique credentials

Variant	% unique	Variant	% unique	Variant	% unique
AKUMA	29.94	MASUTA	34.85	RBGLZ	20.60
ASUNA	13.55	MIORI	22.08	RipPEEP	33.81
Cult	38.65	MIRAI	60.94	XWIFZ	24.18
EXTENDO	44.03	NGRLS	23.26	daddyl33t	24.17
JOSHO	41.43	OWARI	53.41		

Table 6.6: Top 5 credential weights for countries

China

Credential	%
default:OxhlwSG8	6.07
default:S2fGqNFs	5.93
admin:admin	5.40
support:support	4.51
root:7ujMko0admin	4.50

Brazil

Credential	%
admin:admin	3.82
root:vizxv	3.65
root:xc3511	3.59
root:root	2.93
support:support	2.77

Japan

Credential	%
root:default	3.18
root:hunt5759	3.17
support:support	2.99
default:	2.95
admin:admin	2.90

Russia

Credential	%
admin:admin	3.19
root:xc3511	2.50
support:support	2.50
root:vizxv	2.35
user:user	2.34

In addition to preferring different credentials, botmasters assign different weights to credentials shared between them. Figure 6.6 shows the weights of 10 shared credentials between the Mirai, Rbglz, and Miori botnets over time. The figure shows that botmasters have different confidence levels in the brute-forcing success of the shared credentials as their relative order changes from one botnet to the other. The Mirai botnet prefer the *root:vizxv* and *admin:admin* over other credentials while Miori botnet prefers the *admin:admin* and *support:support*.

ADJUSTING WEIGHTS

In section 6.5.2 we show that prior to introducing new credentials, a botmaster assesses the effectiveness of the new credentials. Botmasters can also assess the effectiveness of credentials while the botnet is running. After a successful brute-forcing attack, a bot sends the used credentials to the command and control server. A botmaster can use this information to evaluate the efficiency of the currently used brute-forcing credentials and adjust their weights. Figure 6.6 shows that on the third of February the weight of both the *admin:admin* and *support:support* credentials change for the Miori botnet. Prior to February the third, the weight of the *admin:admin* credential was on average 5.93, while afterward, the weight increased to 7.89. The inverse can be noted for the *support:support*

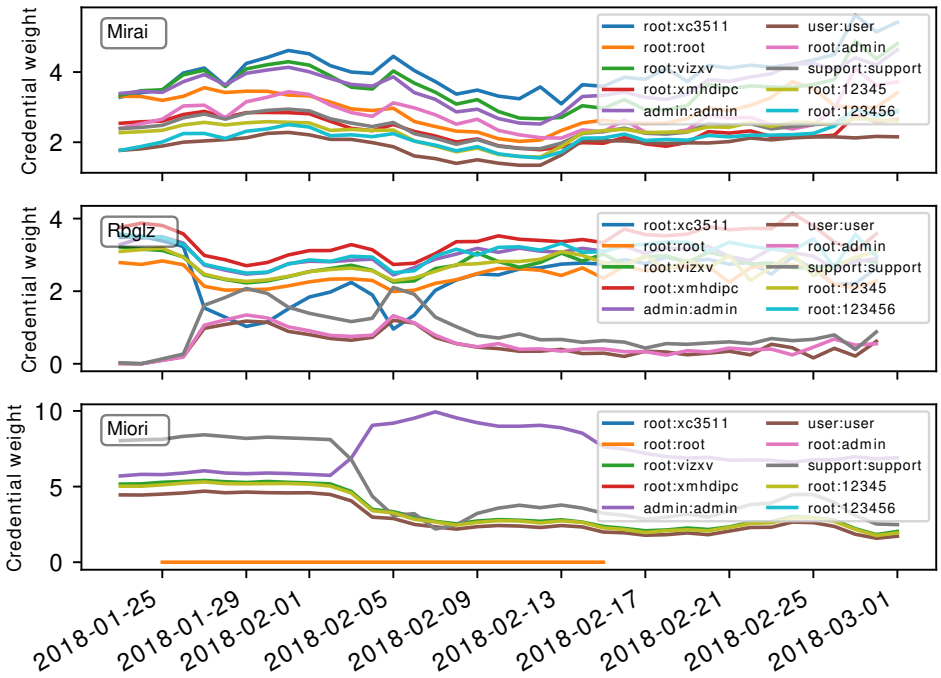


Figure 6.6: Credentials with their associated weight over time for the Mirai, Rbglz and Miori botnets

credential when prior to February the third, the weight was 8.08 and dropped to 3.35 afterward. The sudden shift of weight of credentials suggests that the botmaster analyzes the success rates of the credentials and adjusts their weights accordingly.

6.6.2. INCREASING THE NUMBER OF BRUTE FORCING ATTEMPTS

The original source code of Mirai limits a bot to perform at most ten brute-forcing attempts per targeted device [13]. Increasing the maximum number of brute-forcing attempts could improve the success rate of an attack as more credentials are tried per attacked device. In this subsection, we analyze the number of credentials used during brute-forcing sessions showing that botmasters have altered the source code of Mirai, increasing the number of brute-forcing attempts per targeted device.

To analyze the number of brute-forcing attempts bots perform during a brute-forcing session, we count the number of credentials a bot sends towards a particular honeypot. To ensure we only count the number of brute-forcing attempts during a single session, we limit a session only to include received credentials within one hour of each other. Due to the random target selection of Mirai, it is unlikely that a bot will select the same target within one hour. Figure 6.7 shows the PDF of the number of brute-forcing attempts performed by bots targeting a honeypot. The figure shows four extensive peeks 10, 20, 45,

and 90 brute forcing attempts. The extensive peeks show that botmasters have altered the source code of Mirai to increase the number of brute-forcing attempts per targeted device, probably in an attempt to improve the botnet's success rate.

6.6.3. FIXING THE CREDENTIAL ORDER

Using a weighted brute-forcing dictionary allows botmasters to introduce new credentials while maintaining control over the frequency with which the credentials are used. An alternative method to control the credential usage frequency is to fix the order of the credentials. Using a fixed credential order has two advantages over a weighted random one. First, it assures that each credential is only used once while brute-forcing a target. An analysis of the brute-forcing credentials used by Mirai bots reveals that in 41.87% of the brute-forcing attacks, bots use the same credentials multiple times when targeting a single honeypot. The second benefit of fixing the credential order stems from the guarantee that all credentials are used when brute-forcing a target. The drawback of fixing the credential order is that a bot must increase the number of brute-forcing attempts per target when the credential list grows. This subsection shows that Mirai botmasters are using a fixed credential list.

Modifying the source code to accommodate an ordered credential selection is trivial and lies within reach of every attacker capable of writing simple code. Listing 6.3 shows a simple modification of the source code shown in the listing 6.1 enabling an ordered credential selection. We have modified the code to allow for an additional function parameter *i* defining the credential to select. When a bot has to retrieve the next brute-forcing credential, it can pass the number of failed brute-forcing attempts. The bot already keeps track of the number of brute-forcing attempts and uses it to limit the maximum number of brute-forcing attempts per target.

In order to uncover whether bots are using a fixed credential list, we first must assess whether such lists are used. The source code of Mirai, shown in listing 6.2, randomly selects credentials based on their assigned weight. However, the code does not influence the order in which the credentials are selected. Therefore, we can analyze the frequency

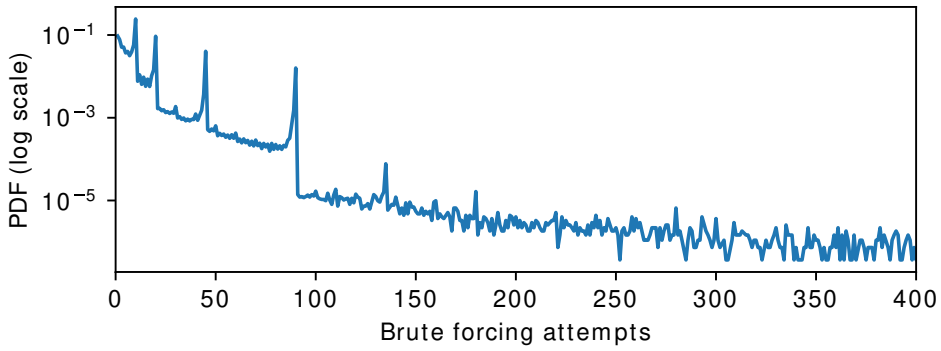


Figure 6.7: PDF of number of brute forcing attempts

of the order in which credentials are used during brute forcing attempts. Suppose bots are using a fixed credential list. In that case, the number of occurrences of that sequence should be higher than sequences generated by bots using a weighted random credential selection process.

We extract the credential sequences used by the bots against our honeypots. We define a credential sequence as the consecutive credentials used by a single bot towards a single honeypot for which the time in between received credentials does not exceed one hour. We utilize a one-hour time window to exclude data pollution created by bots randomly targeting the same honeypot twice. The chance of a bot attacking the same honeypot within an hour is small as Mirai randomly selects its targets and the chance of selecting the same honeypot twice is approximately $1/2^{64}$.

In total we identify 1,880,014 different credential sequences. A large number of credential sequences is expected. Mirai variants use brute-forcing dictionaries containing between 14 and 498 entries, and bots will perform between 10 and 90 brute-forcing attempts per target. As a simple reference: a bot performing ten brute-forcing attempts per target using a dictionary size of 14 can generate over 1.1 million ($C^r(14,10)$) possible password sequences. Of the identified sequences, two are used more frequently than others, with 4.00% and 1.58% of the conducted attacks using the same sequence originating from attacks performed by 1.37% and 1.42% of the bots. To put these numbers in perspective, on average a credential sequence is used in less than 0.00005% of attacks. Figure 6.8 shows for the 40 most commonly used credential sequences the percentage of uses in terms of attacks. We have colored the two most commonly used credential sequences green and red. The following credential sequences having the same color indicate subseries of the most commonly used ones, starting with the same credential. Coloring matching sequences reveals that 19 of the 40 most commonly used sequences can be related to the two most commonly used sequences. Shorter similar sequences can result from bots aborting their attack, thereby not generating the entire credential sequence. The data represented in figure 6.8 indicates that botmasters have altered the Mirai source code to implement a sequential credential selection in favor of the weighted

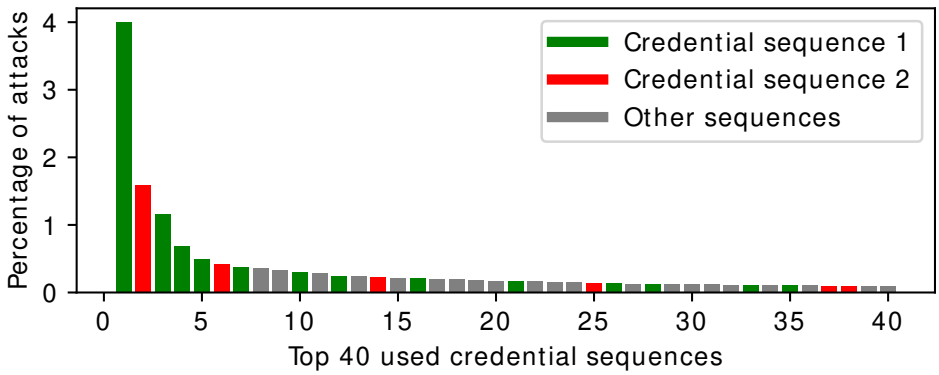


Figure 6.8: Top 40 most commonly used credential order

one.

6.7. DO MODIFICATIONS PROVIDE AN ADVANTAGE?

Through the empirical analysis in the previous section, we have seen essentially all bot masters significantly augment and modify the Mirai source code in terms of credentials, but only some also modifies the source code algorithmically. In this section, we investigate whether algorithmic changes to the weights, credential order, or attempts do make a difference when running a botnet at Internet scale.

6.7.1. A MULTI-AGENT SIMULATION OF MIRAI

While the effect of new passwords on a variant's market share is clear – the botmaster can compromise a new set of devices –, it is from the outside not evident whether modifications to the way Mirai brute-forces would provide a competitive advantage in the Internet. The cause for the different market shares of Mirai variants observed by [7] could be the result of many factors, potentially even interactions between factors. To cleanly evaluate the impact of these strategies, we turn to a multi-agent simulation of Mirai.

For this setup, we build on the Mirai source code which we run in a multi-agent simulation framework. In this setup, we simulate an Internet of 2^{32} endpoints with a set of IoT devices, a proportion of them exposing telnet and being susceptible to Mirai's credential list, which turn into bots themselves once discovered and successfully compromised by brute-forcing. Modeling Mirai via a discrete event simulation is a partial abstraction of Mirai's routine. Mirai's simple architecture continuously spins in a single while loop in order not to overwhelm the resource-constrained device. A basic scheduler launches a new scan every second while the remainder of the routine focuses on handling the brute forcing attempts. In our simulations, we scan every tick, representing a second, and put the discovered devices in a list. Then the devices in the list are brute-forced with the appropriate amount of credentials before the next tick is launched. We acknowledge that brute forcing all discovered devices in between scans can be an overestimation of the processing power and therefore the brute forcing rate of bots. However, Mirai sends 160 scan probes per second randomly targeting devices on the Internet, and there are roughly 3.5 million telnet devices connected to the Internet [22]. Therefore, a Mirai bot will only discover a new target every 7.6 seconds. This would leave enough time for a bot to brute force multiple targets simultaneously. Hence, our design decision of performing the brute forcing attempts within a tick should not affect the real-world representation of our simulations.


```
1 static struct scanner_auth *random_auth_entry(int i) {
2   if (i < auth_table_len)
3     return &auth_table[i];
4
5   return NULL;
6 }
```

Listing 6.3: Altered Mirai source code

6.7.2. PARAMETERS

The parameter of our simulations reflects a real-world scenario and takes into consideration Mirai's standard settings and the observed modification made by botmasters. Table 6.7 summarizes the default setting used to configure the simulated botnets. Some of the settings are self-explanatory such as using the entire Internet address space for our simulations. Others, such as the number of telnet-enabled devices are based on scan data from Censys [22]. Botnet configurations, such as scan speed, brute forcing list length, and number of brute forcing attempts are based on Mirai's original source code [13]. We also take into consideration and simulate infected devices resetting after an amount of time as shown by Griffioen et al. [27].

Parameters for which we do not have accurate data are derived from a combination of studies, Mirai's source code, and simulations. One of those is the number of infectable devices. Reserach suggests Miria infected 600,000 devices [8]. However, a botnet's size depends on its brute-forcing credentials. To eliminate the uncertainty caused by the different credentials list we opted to use the 62 used by the original Mirai and set 600,000 devices to use one of those credentials according to a linear distribution based on the assigned weights in Mirai's source code. The final parameter, the number of starting bots is set to 1000. We ran multiple simulations involving ten competing botnets each starting with 10, 50, 100, or 1000 bots. The simulation showed large variations in the botnets' final sizes when starting with fewer than 1000. The recorded variations are likely caused by the combination of random credential selection and random target selection. When small starting botnets are competing infecting one device before the other botnets do provide a significant advantage. This advantage diminishes when the size of the starting botnets is increased.

Table 6.7: Simulation settings

Parameter	Value
With empirical data	
Number of IP addresses	2^{32}
Number of active telnet devices	3.5 million [22]
Scanning speed of Mirai devices	160 IPs / second [13]
Brute-force target list	128 IPs [13]
Attempts per target	10 [13]
Average time between infections	33 h, std.dev. 103 h [27]
Infectable telnet devices	600,000 [8]
Without empirical data	
Number of alternative telnet port 2323	10%
Initial infections	1000

6.7.3. CHANGING THE NUMBER OF CREDENTIALS PER TARGET

The empirical analysis in the previous section shows that Mirai botmasters increase the number of brute-forcing attempts from the original 10 to 20 per target. Due to Mirai's implementation increasing the number of attempts per target affects the overall attack strategy of the botnet. Mirai's original author used a connection table to keep track of the attacked targets and has a hard-coded length of 128. Therefore, increasing the number of brute-forcing attempts per target increases the likelihood of the table filling up, not allowing newly found targets to be added. Thus, increasing the number of brute-forcing attempts per target changes the attack strategy from one targeting as many devices as possible to another in which fewer devices are targeted but more credentials are tried. To analyze the effect of changing the number of brute-forcing attempts per target we simulated 20 starting botnets competing against each other, with 10 launching 10 attempts per target and 10 performing 20 attempts per target. Figure 6.9 shows the growth

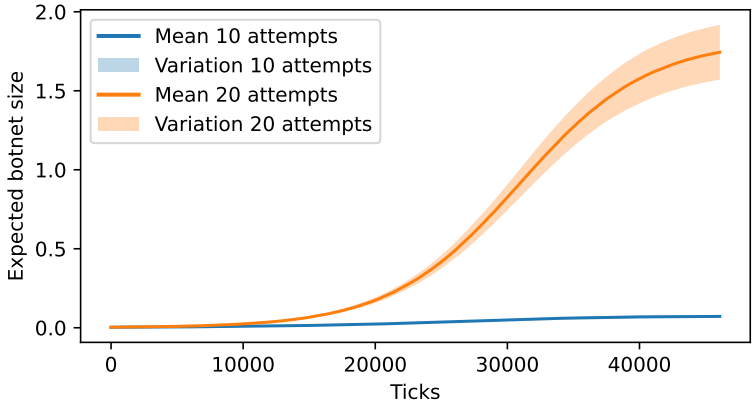


Figure 6.9: 10 botnets 10vs20 attempts

of the simulated bots, the y-axis representing the expected botnet size, and the x-axis representing the number of ticks. The expected size is calculated by dividing the botnet size by the number of infectable devices divided by the number of competing botnets. If all botnets were equally successful, they would all achieve an expected botnet size of 1 indicating all infectable devices are conquered by the botnets equally. Figure 6.9 clearly shows that increasing the number of attempts per target device significantly increases a botnet's success rate. On average the bots performing 10 attempts achieve an expected size of 0.07, whereas the botnets performing 20 attempts achieve an expected botnet size of 1.74. This simulation shows that increasing the number of attempts per target, as performed by some of the observed botmasters, increases a botnet's success.

The previous simulation illustrates botnets simultaneously targeting previously uninfected devices. However, with multiple active botnets, the population of infectable devices will decline. In such a situation botnet's growth will rely on infecting previously infected and becoming uninfected due to resets as shown by Griffioen [27]. In such situations targeting as many devices as possible might prove more successful than making

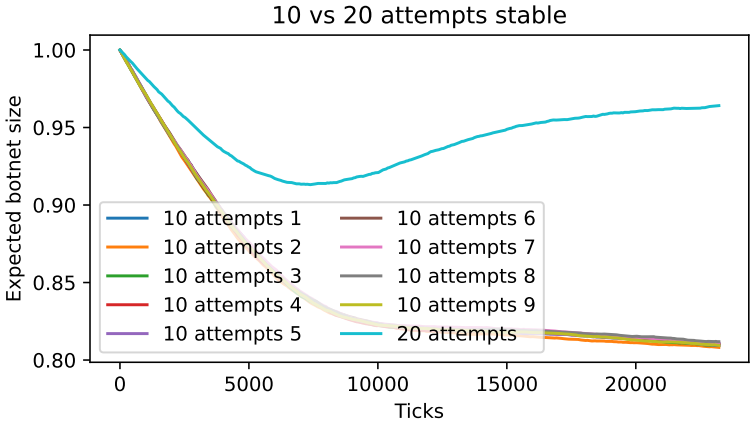


Figure 6.10: steady state 20vs10 attempts

more brute forcing attempts per target. Therefore, we simulated 10 competing botnets operating in an environment in which all infectable devices are part of one of the active botnets. Figure 6.10 shows the expected botnet sizes of 10 botnets, 9 performing 10 brute forcing attempts per target and one performing 20. The figure clearly shows that both types of botnets suffer losses from resetting devices. However, the botnet performing 20 brute-forcing attempts per target recuperates most of its lost bots.

Both simulations reveal that increasing the number of attempts per target is beneficial for a botnet's growth. We ran additional simulations involving botnet making 40 and 60 attempts per target and found similar results. Therefore, we can conclude that increasing the number of brute-forcing attempts per target increases a botnet's growth in both a starting situation and in a situation in which all devices are infected. Hence, the simulations support the identified changes cybercriminals made to their botnets, as increasing the number of brute forcing attempts provides an advantage for botmasters in both a starting and a saturated market situation. The question remains, why haven't all botmasters adopted this strategy? The most likely answer is linked to Mirai's source code being publicly available. The ease with which a Mirai botnet can be set up results in less experienced criminals deploying botnets. Indeed, in section ref{sec:simpletosetup we identified botmasters who are unfamiliar with the inner workings of Mirai and sent obfuscated brute-forcing credentials. These inexperienced criminals would not have the know-how to change the configurations, resulting in only the experienced cybercriminals adopting a strategy of increasing the number of brute forcing attempts, confirming the findings of the previous sections.

6.7.4. CHANGING THE CREDENTIAL SELECTION PROCESS

The empirical analysis in the previous section shows that Mirai botmasters have fixed the order in which credentials are used when brute-forcing a target. Fixing the credential order has a major advantage over using a weighted system, namely, the same credentials

will not be reused multiple times while attacking a device. We simulate fixing the credential order by implementing a botnet that does a weighted selection process but without replacement, effectively eliminating the possibility of a bot reusing the same credential twice during an attack while maintaining the importance of the weights assigned to each credential. We opted for this solution rather than letting bots attempt all the available credentials as we have seen that increasing the number of credential attempts has a positive effect on a botnet's growth and we want to isolate the effect of different settings. We ran two simulations, one in which the botnets are started in a world without infected devices and one in which all devices are infected. We measure the success of a botnet using the expected size, which is a botnet's size divided by the number of infectable devices divided by the number of competing bots. Using the expected size allows us to express the botnet's success in terms of relatively conquered devices. When starting in a world with uninfected devices the difference in expected size is small ranging from 0.01 to 0.16. When the bots start in a world in which all devices are infected the difference between the expected size is smaller and ranges between 0.04 and 0.06. The simulations reveal that using a credential selection strategy without replacement does not provide a significant advantage for a botmaster. The small impact of deduplicating credential usage is surprising. We show in section 6.6.3 that 41.87% of the brute-forcing attacks from a bot towards a honeypot have duplicate credentials. Therefore, we can conclude, that the duplicated credentials during an attack have little effect on whether or not a botnet will be able to successfully brute force its target. Furthermore, modifying Mirai's code to remove duplicate credentials during brute-forcing attacks requires knowledge about the workings of the botnet, which likely not all botmasters have as argued in the previous section. In addition, the alterations required for duplication credential usage would require the allocation of additional resources towards the bot process, which could render the, often weak devices Mirai runs on unstable causing them to reboot more frequently than they already do [27]. The low impact of credential deduplication on a botnet's success rate, combined with the necessity to understand Mirai's workings and the additional resources required to run such an altered version is likely the cause for seeing few botmasters applying such a strategy.

6.7.5. RETAKING CONQUERED BOTS

Griffioen et al [27] show that telnet devices are prone to reset. In another paper, the authors show that devices are recaptured after being restarted [7]. Using an infrastructure to locate and reinfect resetted devices can provide an advantage as botmaster can reinfect devices before competing botnets do. We simulated 10 competing botnets to quantify the advantage such a strategy might provide. Nine of the competing bots are regular Mirai instance while one reinfects devices after they have been reset. We perform two simulations one in which the bots start in a world without infected devices, and a second one in which all devices are infected by the competing botnets.

Figure 6.11 shows the growth of the simulated bots starting in a world with only uninfected devices. The y-axis represents the expected botnet size, and the x-axis represents the number of ticks. The expected size is calculated by dividing the botnet size by the number of infectable devices divided by the number of competing botnets. The figure clearly shows that the botnet using a retake strategy has a higher growth rate than the

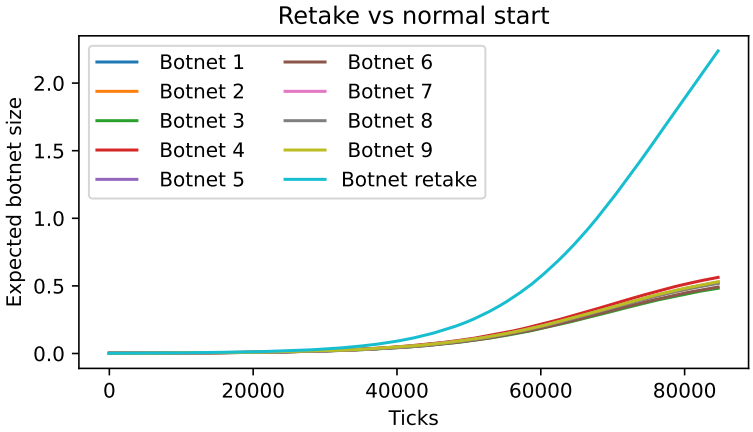


Figure 6.11: Retake c&c starting from 10 bootstrap, percentage growth

other. The botnet retaking lost devices has an expected growth of 2.32 while the other botnets have an average expected growth of 0.51. The growth rate confirms that using a retake strategy is beneficial even when the botnets are targeting uncaptured devices. The results suggest that recapturing devices that were lost due to takeovers or resets at the early deployment stages provides a considerable advantage for the botmaster. This advantage is likely due to the momentum a botnet builds, or rather the prevention of lost momentum. The growth of the simulated botnets follows a logistics curve which is often used to predict growth [28]. Logistics growth is characterized by a time point in which the growth rate dramatically increases. Due to the botnet's ability to reconquer lost devices, it is able to reach this time point earlier than competing botnets, allowing it to conquer a large part of the devices and build momentum increasing its growth, as shown in figure 6.11.

Figure 6.12 shows the expected botnet size of the competing botnets when the botnets start in a world in which all infectable devices are equally allocated to each botnet. The figures show that in a state where all devices are infected implementing a retake strategy is beneficial. The simulated botnet with such a strategy has an expected botnet size of 1.8, while the other botnets suffer losses and have an expected size of 0.77. The success of a retake strategy in a situation where all devices are conquered can be explained by the low likelihood a botnet will target a reset device. At the start of the simulation, each botnet counts 60,000 bots. Each bot scans the Internet using 160 probes per second. Therefore, a botnet would need, in the ideal situation of each bot targetting a different IP address, 447.39 seconds to cover the entire Internet. Hence, implementing a retake strategy in which lost bots are re-infected within a limited time frame is an effective manner to outperform the competition and ensure the botnet's growth.

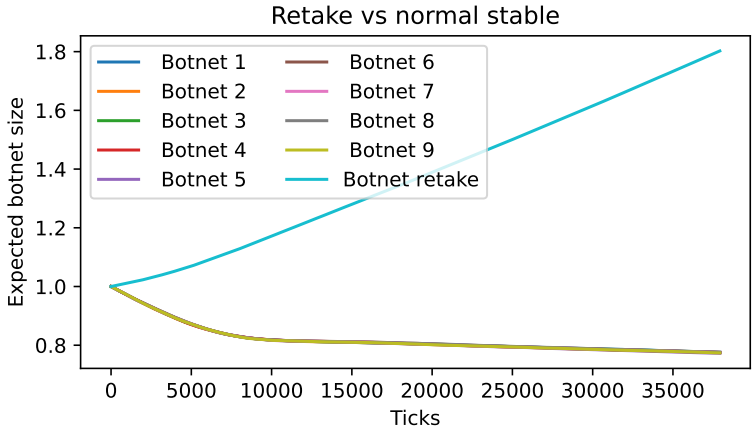


Figure 6.12: Retake c&c stable situation, percentage growth

6.8. CONCLUSION

Undoubtedly, Mirai can be called a successful botnet. In this paper, we have analyzed different aspects of the botnet contributing to its success. First, we show that Mirai bots can attack 95% of all telnet devices on the Internet despite its simple codebase. Second, we show that due to Mirai's simplicity, even lesser-skilled attackers can set up botnets, probably contributing to the emergence of the many Mirai variants and the widespread use of the malware. With increasing competition from other botnets and the many Mirai variants, botmasters must innovate to stay ahead of the competition. We show that botmasters add new credentials to their brute-forcing dictionaries. In addition, by analyzing the credential selection process used by Mirai we show that botmasters have different beliefs about which credentials are more successful. Furthermore, we record some botmasters altering the source code of Mirai, probably in an attempt to gain an advantage over other botnets. Finally, we analyze the impact of the alterations made to Mirai by simulating multiple botnets in different settings showing that the observed alterations, such as increasing the number of brute forcing attempts, and actively reinfecting lost devices improves a botnet's success rate.

References

[1] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten and I. Osipkov. 'Spamming botnets: signatures and characteristics'. In: *ACM SIGCOMM Computer Communication Review* 38.4 (2008), pp. 171–182.

[2] W. Z. Khan, M. K. Khan, F. T. B. Muhaya, M. Y. Aalsalem and H.-C. Chao. 'A comprehensive study of email spam botnet detection'. In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2271–2295.

[3] N. Hoque, D. K. Bhattacharyya and J. K. Kalita. 'Botnet in DDoS attacks: trends and challenges'. In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2242–2270.

[4] P. Pearce, V. Dave, C. Grier, K. Levchenko, S. Guha, D. McCoy, V. Paxson, S. Savage and G. M. Voelker. 'Characterizing large-scale click fraud in zeroaccess'. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014, pp. 141–152.

[5] D. Manky. 'Cybercrime as a service: a very modern business'. In: *Computer Fraud & Security* 2013.6 (2013), pp. 9–13.

[6] Cyber Clean Center. *The Fight Against the Threat from Botnets: Report on the Activities of the Cyber Clean Center*. https://www.telecom-isac.jp/ccc/en_index.html. 2011.

[7] H. Griffioen and C. Doerr. 'Examining Mirai's battle over the Internet of things'. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020.

[8] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, Z. Durumeric, J. Cochran, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.* 'Understanding the Mirai botnet'. In: *Proceedings of the 26th USENIX Conference on Security Symposium*. 2017.

[9] P. Nicholson. 'Five most famous DDoS attacks and then some'. In: *A10 Networks. Source: https://www. a10networks. com/blog/5-most-famous-ddos-attacks/[accessed April 2022]* (2020).

[10] G. Bottazzi and G. Me. 'The botnet revenue model'. In: *Proceedings of the 7th International Conference on Security of Information and Networks*. 2014, pp. 459–465.

[11] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai and Y. Elovici. 'Siphon: Towards scalable high-interaction physical honeypots'. In: *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*. ACM. 2017, pp. 57–68.

[12] T. Barron and N. Nikiforakis. 'Picky attackers: Quantifying the role of system properties on intruder behavior'. In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM. 2017, pp. 387–398.

[13] J. Gamblin. *Mirai Source Code*. 2016. URL: https://github.com/jgamblin/Mirai-Source-Code.

[14] S. Udhani, A. Withers and M. Bashir. 'Human vs Bots: Detecting Human Attacks in a Honeypot Environment'. In: *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*. 2019, pp. 1–6.

[15] Y. Liu and H. Wang. 'Tracking Mirai variants'. In: *Virus Bulletin* (2018), pp. 1–18.

[16] B. Lingenfelter, I. Vakilinia and S. Sengupta. 'Analyzing Variation Among IoT Botnets Using Medium Interaction Honeypots'. In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2020, pp. 0761–0767.

[17] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama and C. Rossow. 'Iotpot: A novel honeypot for revealing current iot threats'. In: *Journal of Information Processing* 24.3 (2016), pp. 522–533.

[18] 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.

[19] J. Carrillo-Mondejar, J. M. Castelo Gomez, C. Núñez-Gómez, J. Roldán Gómez and J. L. Martínez. 'Automatic analysis architecture of IoT malware samples'. In: *Security and Communication Networks* 2020 (2020).

[20] 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.

[21] R. Verhoef. *Honeytrap*. 2018. URL: https://github.com/honeytrap/honeytrap.

[22] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey and J. A. Halderman. 'A Search Engine Backed by Internet-Wide Scanning'. In: *22nd ACM Conference on Computer and Communications Security*. Oct. 2015.

[23] T. Bajto, P. Sokol and T. Mézeová. 'Virtual honeypots and detection of telnet botnets'. In: *Proceedings of the Central European Cybersecurity Conference 2018*. 2018, pp. 1–6.

[24] J. Postel and J. Reynolds. *Telnet Protocol Specification*. STD 8. RFC Editor, May 1983. URL: http://www.rfc-editor.org/rfc/rfc854.txt.

[25] G. O. System. *Inetutils - GNU network utilities*. https://www.gnu.org/software/inetutils/. 2022.

[26] RootSec. *Mirai Variants Source Code*. 2019. URL: https://github.com/R00tS3c/DDOS-RootSec.

[27] 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.

[28] J. R. Miner. 'Pierre-François Verhulst, the discoverer of the logistic curve'. In: *Human Biology* 5.4 (1933), p. 673.

7

CONCLUSIONS

Over the past years, cybercrime has become an increasingly burden on society causing billions of dollars in damages [1, 2]. Unsurprisingly, security firms, universities, and nations responded to increased cybercrime by developing and contributing to the field of cyber threat intelligence (CTI) and translating the gained knowledge into defense systems to handle cybercrime. For further motivations on the importance and impact of CTI in countering cybercrime we refer to chapter 1. Despite the many efforts, there are still opportunities to improve the current state of the art as most of the intelligence is based on low-value indicators and criminals continuously update their operations requiring defending parties to continuously revise their defense strategies. Therefore, in this dissertation, we presented works advancing to the state of the art of CTI and answering the overarching research question:
Can we, in the early stages of attacks, extract information characterizing the criminals toolchain and identify behavioral traits?

This concluding chapter is structured as follows: in the first subsection, we summarize the key findings of each chapter. The following subsection discusses the contributions and how they contribute to the state of the art in relation to the research question. Finally, we identify potential improvements to our contribution which can be addressed in future works.

7.1. SUMMARY OF THE FINDINGS

In the previous chapters, we presented five works that contributed to answering our research question. In this subsection, we summarize the key findings.

- In the second chapter, we analyzed the effectiveness of simple scan detection methods. We have shown that simple defense mechanisms based on blocklisting can detect more than 95% of incoming scan traffic. However, the large volume of daily scan traffic received results in hundreds of undetected network scans. Therefore, solely relying on blocklists as a defense mechanism can lead to a false sense of security for defending parties. Furthermore, we showed that attackers change the locations from which they launch their attacks making location-based defense strategies less valuable. Finally, we have shown that in recent years scanners re-

duced the number of scan probes sent from a single source IP making it more difficult to identify scans, especially for small network operators.

- In the third chapter, we presented a case study analyzing the scan behavior of criminals targetting Memcached servers. We analyze the scan landscape after a proof-of-concept was disclosed showcasing the protocol's potential in DDoS attacks. We find that immediately after the disclosure, scan activity towards Memcached servers increased significantly indicating that criminals are quick to respond to new vulnerabilities. Furthermore, we studied scan probes' payload identifying different tools attackers use, showing that initially, attackers develop their own tools. However, as time passes, attackers converge to using the same tools, indicating that attackers are likely to use off-the-shelf solutions when available.

- In the fourth chapter, we proposed a new methodology to cluster scan traffic based on payloads and reconstructed randomized templates used by cybercriminals. Our methodology has an accuracy of 97.28% and is able to classify 96.04% of the received scan traffic. Using our methodology we identified large distributed scan campaigns. Furthermore, we monitored changes in the scan landscape such as criminals collectively changing their scan payloads in recent years.

- In the fifth chapter, we analyzed SSH brute-forcing attempts and identified the libraries and their respective versions used by attackers to compile their toolchains. Using the different SSH key exchanges used in various libraries we were able to pinpoint which versions criminals used to compile their toolchains. By fingerprinting the criminal's toolchains, we identified different brute-forcing campaigns and identified the behavioral characteristics of attackers.

- In the sixth chapter, we analyzed Telnet brute forcing attempts, showing that we can differentiate between attacks despite criminals using the same software. Analyzing Mirai brute forcing attempts, we reconstructed the configurations used by cybercriminals to set up their botnet, allowing for the identification of various botnets and the strategies used by botmasters prior to them compromising the targeted machines. Furthermore, we simulated botnets configured with the identified changes and showed that the altered botnets are more successful in compromising their targets.

7.2. DISCUSSION

In this subsection, we focus on relating the findings of the presented works to the research question and more importantly explain how the results contribute to the state of the art. In the following paragraphs, we discuss our contributions and reflect on their impact in the field of CTI and their limitations.

The second chapter of this dissertation analyzed the scan landscape in the past eight years assessing the effectiveness of defense systems using indicators of compromises located at the bottom of the pyramid of pain [3]. Our findings show that over 95% of the attacks are detected using simple tools such as blocklists. However, hundreds of scans remain unnoticed due to the sheer volume of attack traffic. The proposed research demon-

strated that utilizing defense systems relying on low-value intelligence does not provide adequate security against cybercriminals, and demonstrated the necessity of defending parties to utilize more valuable intelligence located at higher levels of the pyramid of pain. Although our research examined the effectiveness of commonly used low-value intelligence there are some limitations to the study. One of those limitations is the omission of evaluating the effectiveness of defense systems combining multiple sources of intelligence. Combining sources can be done using open-source CTI feeds, such as combining multiple public blocklists, which theoretically should improve defense systems' effectiveness. A second omission is the analysis of the effectiveness of stacking multiple defense strategies. An example of stacking would be combining IP blocklists with payload blocklists in order to increase the defense systems' effectiveness. Studying both aforementioned limitations would provide insight into the effectiveness of a defense-in-depth strategy which is considered a best practice in cyber security.

The second, third, and fourth chapters of this dissertation analyzed the change in the scan landscape. In the second chapter, we showed that scanners have changed their scan tactics over the past years. They tend to change the location from which they send scan probes and we noticed a trend of criminals adopting evasion techniques such as reducing the amount of probes sent per source address and reducing the amount of destination scanned from each source address. In the fourth chapter, we showed that scanners have changed the type of payloads used in scan probes in the past years, notably changing the DNS query type from ANY to TXT most likely in response to RFC8482 [4] which reduces the sizes of ANY responses making them less desirable for performing DDoS attacks. The third chapter revealed that cybercriminals quickly adopt new scanning strategies when new vulnerabilities are disclosed. All three studies showed that the scan landscape is continuously evolving due to cybercriminals' changing tactics. Our presented works therefore show that CTI related to scanning should be continuously evaluated with regard to the current scan landscape as changing criminal behavior can affect the effectiveness of defense systems, especially if these defense systems are based on outdated information. Both the second and fourth chapters analyzed the scan landscape using multiple years of data the findings of the work in the third chapter are based on a single event. We acknowledge that a single case study analyzing the changes in the scan landscape after a new vulnerability was disclosed cannot be considered representative of the behavior exhibited by criminals after similar events occur. To overcome this limitation similar events should be analyzed showing whether the behavior witnessed during the case study is common, emphasizing and confirming the need to quickly react and gather CTI when new vulnerabilities are disclosed.

Having established that the cybercrime landscape is continuously evolving and that CTI belonging to the bottom of the pyramid of pain does not translate into effective defense systems, we presented, in the fourth chapter a methodology to identify the templates cybercriminals used to craft their scan probes. Furthermore, we showed that these templates can then be used to identify large distributed scan campaigns. Providing the possibility to identify scan campaigns allows future research to identify and extract the TTPs cybercriminals use. Indeed as we have shown in the fourth chapter and mentioned earlier, using our methodology we were able to detect changes in the global scan landscape. Even though our methodology is 97% accurate we must address its potential limit-

ations. First, the methodology has been tested using scan probes targeting two services, hence assessing its effectiveness against scanners crafting probes targeting two protocols. Some protocols allow for more degrees of freedom than others. One of those is DNS allowing for arbitrary contents in the payloads. Others such as NTP are more strict on payload contents. Therefore, our proposed methodology's effectiveness should be assessed against scan probes targeting a broader range of protocols confirming its robustness. Another limitation concerning our methodology relates to the changing landscape. We have shown in both chapters three and four that scanners converge to using the same payloads in scan probes. Our methodology contributes to identifying the TTPs of criminals by reconstructing the unique templates criminals use to craft scan probes. If criminals use the same probe payloads, our methodology will still be able to recreate the common template, which in turn can be used to identify scan traffic, however, it will no longer contribute to discerning between the scan campaigns launched by criminals increasing the difficulty to analyze the behavioral aspects of scanners and thus identify their TTPs.

Although relying on the output of cybercriminals' tools allows for their behavioral analysis, the methodology has limitations. When the entropy of the outputs is reduced, analyzing them will not allow the distinction between campaigns. Therefore, we demonstrated, in the fifth chapter, an additional approach enabling the characterization of the toolchains used by attackers by means of detecting the libraries used to compile them. We fingerprinted the SSH key exchanges prior to brute forcing attempts allowing the identification of the libraries and their versions cybercriminals use to compile their toolchains. Using our fingerprinting approach we can identify different distributed bute forcing campaigns. The identified fingerprints allow defending parties to detect ongoing attacks and allow the analysis of the TTPs attackers use during brute forcing. Indeed, we were able, among others, to link the credential usage to the versions criminals use to compile their toolchains demonstrating the possibility of predicting the future behavior of attackers based on the early stages of the attacks. Although our fingerprinting method allows for the identification of attacks, and the analysis of criminal's TTPs, there are some limitations. First, the fingerprinting method can only be used to identify and discern between attackers' targeting services requiring a key exchange, and to a greater extent the exchange of parameters, making it unsuitable for other protocols that do not require such exchanges to set up their connections. A second limitation concerns the ability to use our fingerprinting technique to discern between different brute-forcing campaigns. Similar to scanners using the same templates, cybercriminals brute forcing devices may op to use off-the-shelf toolchains, which are precompiled, and therefore, have the same fingerprints, making it impossible to distinguish between their campaigns, increasing the difficulty of extracting the TTPs cybercriminals use.

The final contribution of this dissertation concerns identifying the different configurations cybercriminals use when deploying similar botnets. In the sixth chapter, we analyzed the brute forcing attempts of Mirai bots revealing that criminals configure their botnets in different manners. Our analysis showed that criminals adopt different brute-forcing strategies and assign different weights to brute-forcing credentials favoring some over others. Furthermore, we showed that the custom configurations used by certain botnets increase their success rate, revealing that botmasters customize their botnet in

order to outperform the competition. Our analysis allowed us to distinguish between bots running the same software but which are configured differently by their botmasters allowing us to identify different campaigns at the early attack stages. Furthermore, we showed that criminals are continuously tweaking the settings of their tools to gain an advantage over competing criminals, indicative of an evolving and competitive cybercrime landscape. Although our approach, allows for the identification and study of different botnets, the study has some limitations. First, the study is performed on one botnet family, therefore, other widespread botnets should be studied to confirm whether criminals adjusting their botnet's settings is common. Second, similar to the previous studies, our approach is not suitable for distinguishing between botmasters using the same configurations, increasing the difficulty of analyzing the TTPs of criminals using similar setups.

In conclusion, we have answered our research question by showing *that it is indeed possible, at the early stage of attacks, to extract information characterizing the criminals toolchain and identify behavioral traits.* We started by showing that the scan landscape is continuously evolving and that CTI belonging to the top of the pyramid of pain is invaluable in contributing to the development of defense systems. Then, we propose a methodology to identify attackers behavioral traits based on the output of attackers' toolchains. We follow, by showing that we can differentiate between the toolchains attackers use, allowing for the analysis of cybercriminals' behavior. Finally, we show that we can identify the different configurations criminals use for the same toolchains, allowing us to identify different campaigns and thus analyze the TTPs used by criminals contributing to the generation of actionable intelligence. In summary, the presented studies contribute to advancing the state of the art by demonstrating the generation of CTI based on the characteristics at different levels of criminal toolchains. The generated CTI spans multiple layers of the toolchains attackers use, providing actionable intelligence at the early attack stages. The generated intelligence can, therefore, contribute to the design of defense. More so, it allows for the design of defense-in-depth systems which aim at halting ongoing attacks during multiple attack stages increasing the likelihood of stopping an ongoing attack.

7.3. FUTURE WORK

Answering the research question led us to develop new methods for analyzing cybercriminals' toolchains allowing for the generation of CTI. In the following paragraph, we discuss how our methods could be leveraged to solve other challenges in the field of cybersecurity and suggest further research topics complementing this work.

We developed a clustering methodology allowing for the clustering of scan probes generating the templates cybercriminals used to craft scan payloads. We use this methodology to analyze scan campaigns and reveal the behavioral aspects of scanners. Internet service providers (ISPs) could use our algorithm to detect scan traffic and other types of traffic, such as DDoS-related packets, allowing them to take action before the probes reach their targets or analyze criminals's behavior on a larger scale. ISPs are incentivized to use our method as they would benefit twofold from detecting attack probes. First, it allows them to shield their clients from incoming attacks. Second, it will enable them to intercept outbound attack traffic, lowering the risk of being classified as untrusted or

high-risk networks. The results from applying our methods on a dataset from an ISP would reveal new behavioral characteristics as other attacks, which do not appear in Internet telescope datasets, can be identified and analyzed, providing additional insights into cybercriminal behavior. Additionally, it would reveal whether our methodology can be applied on a large scale and whether adjustments should be made to the proposed algorithm.

Our methods rely on identifying attack traits resulting from cybercriminals customizing and assembling their toolchains. The presented works show that criminals have a tendency to use similar toolchains when performing attacks. We notice that scanners have all switched towards using the same query when scanning for DNS servers. We see that attackers converged towards using the same scan probe payload when targetting Memcached servers. One of the recurring limitations of our works is that when attackers use identical toolchains entropy of the toolchain's output is reduced, rendering our methods unsuitable for distinguishing between different campaigns. Therefore, we suggest researching the application of time series analysis on received scan probes to assist in distinguishing and identifying scan campaigns. Throughout the different works, we identify campaigns and plot their characteristics against time revealing behavioral traits suggesting that time series analysis might be used to identify scan campaigns. Successfully using time series analysis on scan traffic can be challenging as campaigns can be launched from single or multiple addresses, and the speeds with which the probes are sent vary as shown in our research. Solving the problem of identifying campsites based on time series analysis could improve the identification and detection of campaings, contributing to the generation of CTI.

7

References

[1] Beyond Identity. *How Has a Decade of Cybercrime Impacted the United States?* 30th Aug. 2021. URL: https://www.beyondidentity.com/blog/rise-cybercrime-study.

[2] S. Morgan. 'Global Cybercrime Damages Predicted To Reach $6 Trillion Annually By 2021'. In: *Cybersecurity Ventures Official Annual Cybercrime Report* (26th Oct. 2020). URL: https://cybersecurityventures.com/annual-cybercrime-report-2020.

[3] D. J. Bianco. 'The Pyramid of Pain'. In: *SANS* (17th Jan. 2014). URL: http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html.

[4] J. Abley, O. Guomundsson, M. Majkowski and E. Hunt. *Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY*. RFC 8482. 2019.

ACKNOWLEDGEMENTS

As I bring this dissertation to a close, I would like to take a moment to express my heartfelt appreciation to those people who supported me through this long and sometimes challenging journey. First and foremost, I am deeply grateful to my promoters for giving me the opportunity to pursue this PhD. I would especially like to thank Jan for his extraordinary support and guidance, which were invaluable in helping me finalize this dissertation. My sincere thanks also go to Christian whose daily supervision, constructive feedback, and countless opportunities provided along the way kept me grounded. I extend my appreciation to the members of the committee for their time, insights, and thoughtful feedback. I am also thankful to all my colleagues with whom I had the pleasure of working. In particular, I would like to thank Harm for the stimulating discussions, the collaborations, and memorable moments during this experience. My thanks also go to Sandra, for smoothing out bureaucracy and organizing the much-needed coffee breaks and very welcome fun lunches. A special thanks to my family, to my mother and sister, thank you for always being there with encouragement and understanding. And to my wife, I cannot thank you enough for your patience, your belief in me has been a constant source of motivation. Lastly, I thank my friends for their support and companionship during this chapter of my life.

Curriculum Vitæ

Vincent Ghiette

26-04-1989	Born in Etterbeek, Belgium.

Education

1992–2007	School Lycée français Vincent van Gogh
2007–2013	Bachelor of Computer Science Delft University of Technology
2013–2016	Master of Computer Science Delft University of Technology
2025	PhD. Cybersecurity Delft University of Technology *Thesis:* Threat Intelligence in the Early Stages of Cyberattacks *Promotors:* Em. prof. dr. ir. J. van den Berg and Prof. dr. C. Doerr.

List of Publications

- Ghiette, V. and Doerr, C., *How media reports trigger copycats: An analysis of the brewing of the largest packet storm to date*, Workshop on Traffic Measurements for Cybersecurity 2018, https://doi.org/10.1145/3229598.3229606
- Ghiette, V. and Griffioen, H. and Doerr, C., *Fingerprinting tooling used for SSH compromisation attempts*, 22nd International Symposium on Research in Attacks, Intrusions and Defenses 2019, https://www.usenix.org/conference/raid2019/presentation/ghiette
- Ghiette, V. and Doerr, C., *Scaling website fingerprinting*, IFIP Networking Conference 2020, https://ieeexplore.ieee.org/document/9142795
- Ghiette, V. and Doerr, C., *Clustering Payloads: Grouping Randomized Scan Probes Into Campaign Templates*, IFIP Networking Conference 2022, https://doi.org/10.23919/IFIPNetworking55013.2022.9829757
- **Draft**, Ghiette, V. and Doerr, C., *UDP scanning and the limitations of behavioral and blocklist based countermeasures*.
- **Draft**, Ghiette, V. and Doerr, C., *Why was the IoT Botnet Mirai so successful? A Deep Dive into Mirai's Brute-Forcing Strategy*.