Could you clean up the Internet with a Pit of Tar? Investigating tarpit feasibility on Internet worms

Harm Griffioen
Hasso Plattner Institute
University of Potsdam, Germany
harm.griffioen@hpi.de

Christian Doerr
Hasso Plattner Institute
University of Potsdam, Germany
christian.doerr@hpi.de

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

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

I. Introduction

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

The bulk of the scanning and initial compromise activity comes from automated tooling and scripts [20] that either rely on common passwords [11], attack services based on pre-defined signatures, or blindly launch exploits directed at common and frequently unpatched vulnerabilities. Given the plethora of Internet devices and generally low security awareness, it is only a matter of time before one of these attackers "gets lucky". As Internet-connected devices are often set up using weak, common, and insecure passwords [7] or are not updated to address vulnerabilities [39], there are enough victims to make automated scanning and exploitation a profitable activity. This explains figures such as [23] which report that 63% of Internet scans target port 23 and [16] that the portion of malicious Internet traffic is heavily increasing.

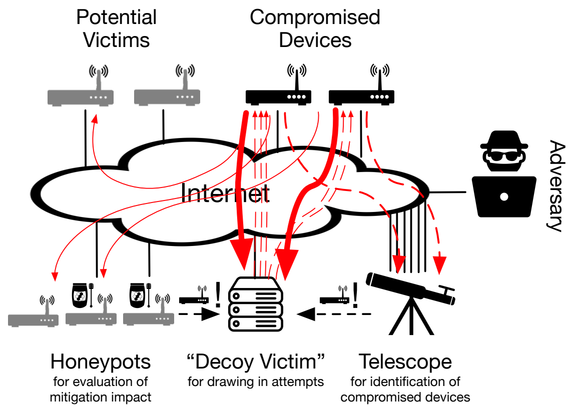


Fig. 1: A network telescope detects infected devices and informs a tarpit to offer itself as a decoy victim. Honeypots measure the effect from reduced compromization activity.

As many insecure devices expose themselves on the Internet and automatic scanning is both low effort and lucrative, neither problem will go away on its own. Today, botnets comprised of hundreds of thousands of low-powered devices [28] are together capable of launching major distributed denial-of-service attacks [25], originate SPAM [36], or participate in cryptocurrency fraud [13]. This poses the question of whether it is possible to prevent large-scale exploitations by addressing the link between perpetrator and victim: can we somehow interfere that these scripts do not efficiently find victim devices to exploit? As we cannot and do not want to interfere with the network itself, we advertise decoy devices so that adversaries concentrate the bulk of their malicious activities on them and have, in turn, fewer resources to go after real victims.

One of the key engineering principles to make high-speed scanning and exploitation possible is that devices do not keep track of past and ongoing connection requests they have sent [17]. As the book-keeping from making contact to an IP address and retrying on a non-response would be way too slow to cover significant parts of the Internet, modern scanning tools blast out connection requests via raw sockets without actually going through the operating system to establish a connection. If an answer comes back, the tool checks based on header field values whether this response could be related to the previous scan and hones in on those IP addresses that have responded.

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

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

II. BACKGROUND

The work put forward in this paper is applicable to the containment of any self-propagating malware. In the following, we focus on the Mirai IoT malware as one of the most significant Internet threats [10], which is responsible for 87% of all telnet compromization attempts worldwide [20]. Cleaning up Mirai and its siblings will therefore drastically reduce brute-forcing activity as a whole, but the presented method does also work on other stateless malware (which we also trapped but excluded in the discussion). To understand why trapping would be effective in today's high-performance scanning routines, we first need to introduce some background about the Mirai ecosystem and the way it selects its targets.

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

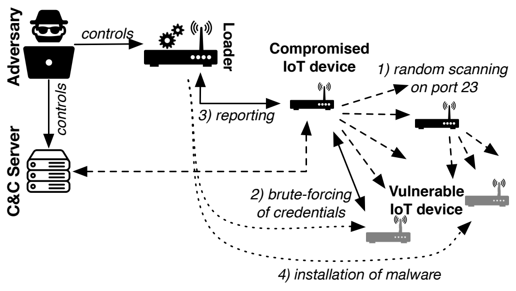


Fig. 2: Devices infected with Mirai scan the Internet for vulnerable hosts, and report their findings to a loader.

Scanning and Brute-Forcing Processes. Although the malware loader technically makes the infection, Mirai behaves like a worm as the compromised devices are responsible for discovering and breaking into additional victims. In order to spread effectively, Mirai uses some architectural designs commonly found in high-performance port scanning software such as ZMap or Masscan. As opening TCP sockets in the operating system comes with significant overhead and the OS limits open connections, consecutive connections would be too inefficient. Instead, Mirai performs scanning using RAW packet injection: the malware crafts TCP SYN packets to randomly generated destination IPs and directly injects them into the network. If the destination IP is a device listening on TCP port 23/2323, it will answer with a TCP SYN+ACK. Once a SYN+ACK is received, the malware opens a "real" network socket to the destination and begins brute-forcing. By avoiding opening connections for scanning, Mirai does not have to save any internal state and therefore progresses fast; in our experiments, we saw IoT devices routinely scanning at speeds exceeding 8500 packets/second.

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

NATs and IP Churn. Mirai infections are not limited to end users' Internet routers, any device opening a telnet port with weak login credentials is in principle susceptible. This means that for dealing with Mirai infections we have to consider three scenarios as depicted in figure 3:

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

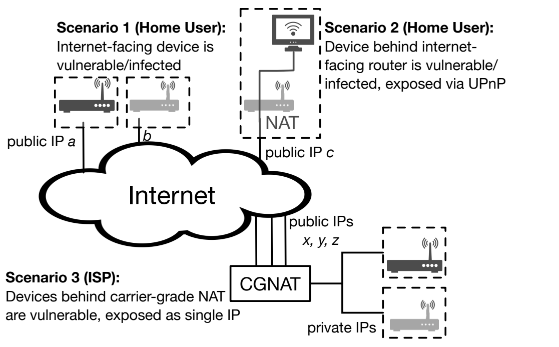

Scenario 1 (Home User):
Internet-facing device is vulnerable/infected
Scenario 2 (Home User):
Device behind internet-facing router is vulnerable/infected, exposed via UPnP
NAT
public IP a
b
public IP c
Internet
public IPs x, y, z
Scenario 3 (ISP):
Devices behind carrier-grade NAT are vulnerable, exposed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

Fig. 3: Vulnerable devices could be exposed directly to the Internet or via UPnP at public IPs, or hidden behind NATs.

b. Counting the number of scanning/brute-forcing public IPs will reveal the number of infected devices on the Internet.

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

Scenario 3: Carrier-grade NAT. As publicly routable IP addresses are in short supply, many ISPs do not hand out individual public IPs to each customer. Instead, the routers of home users are given a local IP address and connect to the Internet via a carrier-grade NAT (CGNAT). Thus, an arbitrarily large number of home installations would be exposed via the same IP address to the Internet. The CGNAT tracks the state and matches incoming responses from the Internet to an earlier request to deliver it to the correct local client. For tracking Mirai infections, this creates the complication that more than one device might be infected behind a public IP. Occasionally, CGNATs are assigned a pool of public IPs (x, y, and z in figure 3), and may re-assign a particular home user to a different public IP in regular time intervals, creating so-called IP churn. This creates another complication for tracking, as the same infection would be recorded multiple times over its lifetime if only counting scanning IP addresses. IP churn is known as a major factor in over- and under-estimating infections [24].

The particular setup of Mirai's port scanning routines conveniently addresses both issues. As upon startup, the malware picks a random source port and window size, this 32-bit value is fixed during the entire infection period. Suppose scanning packets originate from the public IP address x with two different sourcePort/windowSize combinations. In that case, we are confident that two devices behind that IP address have to be infected, as Mirai binds to port 23 and thus prevents two concurrent infections from being active on port 23 [21]. Similarly, if scanning traffic stops on IP x and shortly after commences on IP y with the same sourcePort/windowSize combination, there is only a one in four billion chance that this is not the same infection. We can thus also reliably track and count infections behind carrier-grade NATs.

III. Related Work

The default action against botnets in the recent past has not been based on tarpits but rather through *sinkholing*, where the command and control (C&C) infrastructure is taken down by researchers or law enforcement [33], [15]. The coordinated takedown of a C&C is very labor-intensive and often escapes long-term effects as malware authors will simply restart their activities [18] and sightings frequently get reported with weeks of delay [19]. Major botnets such as ZeuS or CodeRed could be reactivated in a matter of days after such takedown [18], [22]. Furthermore, over the past decade, malware has evolved to use peer-to-peer-based overlays [24], fast fluxing [34], proxy layers [9], or bitcoin-based signaling [37], making it increasingly hard to find and take down a C&C server.

Some studies have shown that so-called *tarpits* can be used to reduce impact of DDoS attacks [30], SPAM [12] and Web Crawlers [38] by occupying infected devices. While researchers have focused on creating and evaluating tarpits, no work has evaluated the effect of a tarpit on large malware installations. We are, however, interested in how this approach would work in practice with a sufficiently large monitoring infrastructure, and whether a large tarpit infrastructure would be capable of slowing down the spread of a botnet.

Tarpits aiming to trap malicious devices have been around for a long time. The first tarpit was called "LaBrea" and was introduced by Tom Liston [1] in response to the CodeRed worm [31] that exploited a vulnerability in Microsoft IIS web servers to spread through the Internet. LaBrea monitors TCP traffic towards a host and replies with a SYN+ACK that completes the TCP handshake and makes the malicious program wait for new replies that are never sent. Since its inception, other research has developed tools to reduce network attacks based on tarpits. Borders et al. [14] created OpenFire, which fills unused address space with honeypots and makes the entire port space appear to be open to an attacker. Modern scanning software and malware have implemented both multi-threaded infrastructures and network timeouts, limiting the impact of OpenFire. As tarpits and honeypots in general introduce a bias for researchers scanning the Internet, Alt et al. created "Degreaser" [8] which identifies and filters tarpits. Adversaries could use this to evade these infrastructures, but this would take extra effort to identify and stop the scanning of these devices. Especially in distributed environments such as a botnet this will be hard to implement.

When a worm connects to a tarpit that attempts to keep the malware occupied on the network layer, the worm does not receive the expected application layer data, and the malware author can use a timeout to detect the trap. Walla and Rossow [40] propose an automated method to identify implementation mistakes in the malware that can be used to halt the program execution remotely. Other researchers have devised methods to scale tarpits to IPv6 [26], but only evaluated the method on a real-world /24 network as the IPv6 network did not receive packets. Furthermore, the authors only report on general trapping durations and not the effects on a worm.

Given the increased difficulty and diminishing returns of takedowns, tarpits have been recently rediscovered and suggested as an alternative to contain malware [40], but no works have identified whether this would work at-scale. As replacing a single C&C is easy, taking away the foundation of a botnet is more challenging to replace and might offer longer-lasting results. From epidemiological evaluations of the Mirai malware, this might very well work, as researchers show that the Mirai-malware, which is a large botnet, has a low reproduction number $R_0 = 1.0033$ and is barely self-sustaining [20]. If it would be possible to slow down the infection process with a tarpit significantly, it might lead to a collapse of the botnet. For this study, we will evaluate the effect on a botnet in-the-wild, for which the epidemiological results will be transferable to other botnets. We evaluate whether we can contain botnets based on Mirai's scanning [11] and significantly disrupt the spread and impact of these botnets.

IV. Methodology

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

A. Data collection

To identify devices infected by a variant of Mirai, we use two complementary datasets that are also shown in Figure 1:

1. Network telescope - We use a large network telescope of approximately 65,000 IP addresses in an enterprise environment to record scanning attempts. As these IP addresses never respond, they will only receive Internet scans and backscatter. Scattered throughout actively used enterprise IP address space, we find them not to be on the blocklists of Mirai malware [2].

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

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

B. System Architecture

The setup of the entire system is shown in figure 1. Our network telescope and honeypots forward infected IP addresses and the corresponding source and destination ports towards our tarpit, which starts to forge artificial reply packets and maintains connections to compromised devices. The tarpit is designed to send out 100 replies per second (at 64 bytes each, we thus send 6.4 kbps) to infected devices (the motivation for this will be shown in section VI). It will timeout after the device has not responded for one minute so that we will send at most 375 kilobyte towards devices cleaned up after a reboot.

As the tarpit will send 100 packets per second (pps) to infected hosts and respond to all incoming connections from infected devices, the system needs to send large amounts of packets if we aim to address Mirai infections worldwide from a single device. Raw Ethernet sockets would therefore be too expensive, as it uses the `sendto` system call, which performs a context switch and transfers packets through kernel space. Similar to [6] we use the PF_RING ZC interface, which works as a kernel bypass and allows user-space code to access the NIC directly, eliminating the need for context switches [3]. Bypassing the OS and direct packet injection allows us to maintain very large numbers of concurrently open connections from a single device, as we are offloading all connection book-keeping from the OS. We implemented a simplified TCP stack in Golang to construct valid responses. The machine running the tarpit was comparatively moderate, with a quad-core *Intel Xeon E3-1225* from 2011 at a clock speed of *3.20 GHz*, a *1 Gbps* uplink and *32 GB* of memory.

C. Tarpitting Strategies

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

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

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

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

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

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

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

Dataset	**Date (2021)**	**Description**	**Open sourced**	**Size**
Network telescope	Feb 19 - Apr 16	Set of 65.000 unused IP addresses used to identify infections and required packet header fields	List of IP addresses infected with Mirai-based malware	2 GB
Cloud honeypots	Feb 19 - Apr 16	100 honeypots used for: (1) collecting application-level traces on port 23, 1023 and 2323. (2) Identifying new infections and required packet header fields	Logs and raw PCAP dumps	2 TB
Main tarpit	Mar 3 - Apr 8	Machine used to trap infections identified by honeypots and telescope	Logs and raw PCAP dumps	24 TB
Extra tarpit	Apr 2 - Apr 8	Machine used to trap infections identified by honeypots and telescope	Logs and raw PCAP dumps	2 TB
Tarpit & simulator	-	Source code of a tarpit and simulator responding to scanning probes	Full source code	–

D. *Could the tarpit be weaponized?*

IP addresses can be spoofed, so an adversary might attempt to make a connection request on behalf of an unsuspecting victim. Any of the strategies above require first the completion of a TCP handshake in which we reply with a random ISN, before we sent out SYN+ACKs from the tarpit. As a spoofing attacker could not guess our ISN, the handshake does not complete and the spoofed IP will never enter our queue and be considered by the tarpit. Hence, the system will only trap actual infections and cannot be weaponized by an adversary against others.

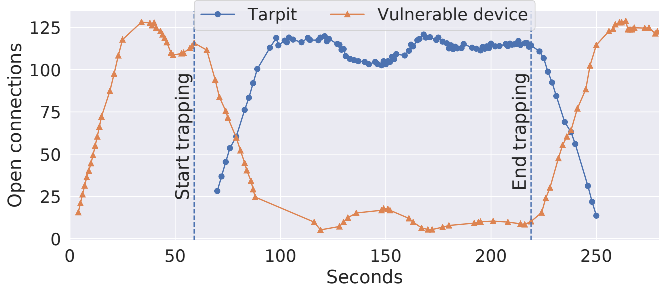


Fig. 4: Connection trapping against a base version of Mirai.

V. Verification and Ethics

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

A. *Verification of the trapping of infected hosts*

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

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

To verify this concept, we infected an IoT device in a controlled environment with the original Mirai malware [2], and measured its scanning and brute-forcing activity before and after we initiate our trapping. Figure 4 shows the number of brute-forcing connections towards a vulnerable device located in a network of 20 devices. This means the malware has a 5% chance of discovering the vulnerable device every time it sends out a scanning probe, which is magnitudes larger than the probability of finding a vulnerable device during an Internet-wide scan. The orange line shows the number of connections that are active towards the vulnerable device identified by the malware. The blue line shows the number of connections towards the tarpit, which advertises itself by forging 100 SYN+ACKs per second. In this best case scenario for the malware of 5% discovery chance and without any connection slow-down, the trapping can already reduce the amount of brute-forcing by a factor of 20.

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

While the trapping works as intended, it is essential to verify that this procedure does not impair the stability or performance of the device to the end user, and instead only prevents the spread of the botnet. To test this, we created the verification setup as shown in figure 5, in which we benchmark the behavior of four Internet routers, ranging from a $20 budget device to mid-range home routers.

In our environment, these routers are setup to connect an end-user to the Internet either via a 100 Mbps or 1 Gbps Ethernet cable and operate the router at its maximum transmission rate. We connect a high-end router on the Internet uplink of the infected device that drops the outgoing scan probes to prevent pollution and links the setup to our tarpit. We then measure with iperf the performance of the link between the end user machine and the router's uplink in terms of packet loss, latency and throughput, and at the same time monitor the "health" of the router via its CPU load. This allows us to

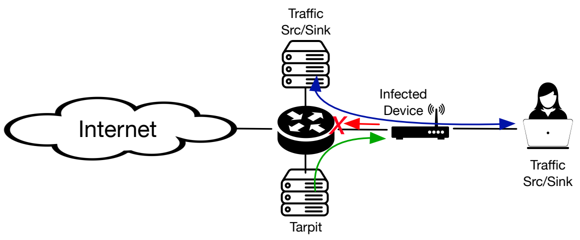
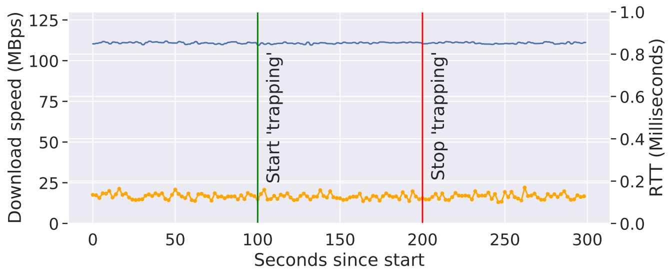
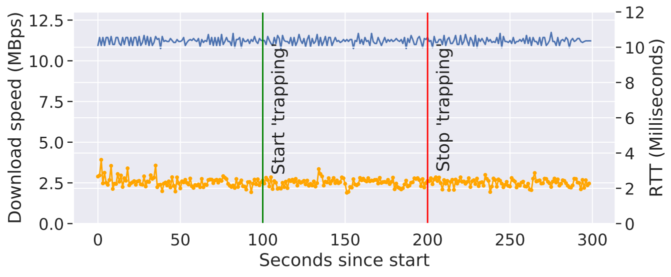


Fig. 5: Experimental setup to test for potential impairment of the router in forwarding capacity, delay and CPU utilization.



(a) Asus GT-AC5300, Gigabit uplink



(b) TP-Link TP-WR841N, 100 Mbps uplink

Fig. 6: Tarpit impact on forwarding speed and latency. The blue line shows download speed, the orange line packet RTT.

quantify whether either the end user experience is impaired or the device itself has to process an abnormal workload.

Each router is tested for 300 seconds in the following scenario. We first establish the iperf connection measuring the performance of the end user's connection. We infect the router, which triggers the device to begin scanning the Internet for other devices. We record this baseline activity for 100 seconds. We then activate our tarpit to trap the device at the intensity of 100 SYN+ACKs per second, identical to the speed we will use throughout our experiments. This trapping continues for 100 seconds. At second 200 into the experiment, the tarpit shuts down so that we can monitor for the remaining 100 seconds the rebound behavior of the router. We perform this experiment for a variety of routers, ranging from a $19 TP-Link TP-WR841N budget router with only 32 Megabyte RAM, two mid-range routers Zyxel VMG8825-T50, a TP-Link AD7200 to a high-end Asus GT-AC5300. The first two devices have 100 Mbps Ethernet ports (technically the Zyxel has Gbps, but is capped at 100 Mbps), the others feature a Gigabit Ethernet interface.

Figure 6 shows exemplarily the download speed and the round-trip time (RTT) latency of the Asus GT-AC5300 and the TP-Link TP-WR841N during the course of the experiments. The graphs are depicted separately, as the devices have different link speeds. As we can already see visually, the tarpitting at 100 SYN+ACKs per second has no impact on the forwarding performance of the router, also the CPU load does not measurably increase after the tarpit starts and while it is running. A Kolmogorov-Smirnov test confirms that the router behavior is not impaired by our trap. While only two examples are shown, all routers behave identically.

Finally, we have to evaluate a potential impact on the network in which the devices are located, as they have to forward our tarpit responses to the infected devices. To estimate the impact, we take a worst-case scenario approach to obtain an upper bound on the amount of traffic that is forwarded through a network. Let us consider the most infected AS in the world, AS4134, in which we have seen *a total* of 125,000 IPs being infected historically. If we suppose that we would trap all 125,000 devices ever seen concurrently at the tarpitting speed we utilize in this work, we would send a total of 12 Mbps towards this AS. This is approximately half of the bandwidth requirement that would be generated by a single customer watching a 4K video from a streaming platform. We hence can consider the additional traffic on ISPs and the Internet to be negligible, especially considering that by trapping the infected devices, the compromised device would not generate further exploitation traffic towards other victims.

We presented the study setup and verification results to the Institutional Review Board. As the study does not directly relate to human subjects, they referred us to the data protection officer. They evaluated the study and found no risk to cause harm to people and their devices, and raised no objections.

VI. Trapping Mirai IoT infections

During the 35 days in which we operated our tarpit, we observed a total of 423,810 distinct IP addresses scanning and brute-forcing the Internet using the Mirai fingerprint. These devices originated from 178 countries and 3,150 Autonomous Systems. Since its inception, Mirai's principles have been adopted by other malware, which also diversified in terms of targets: we found that in addition to telnet, the diversified Mirai ecosystem now also targeted 17 other ports. As all of these malwares rely on Mirai's scanning routines, our tarpitting is effective against all of them. While all of these malwares were captured by our technique, we focus in the following on Mirai as the largest IoT malware. During our study we successfully trapped 202,003 (48%) of these IP addresses in our tarpit.

Although we observed hundreds of thousands of infected IP addresses, the *momentary* number of infections is significantly lower: Mirai-infected devices frequently crash and become available for reinfection [20], furthermore ISPs often dynamically reassign IP addresses – so-called IP churn –, which means that a unique infection might reappear from a different IP. We account for IP churn using the methodology in [21] and find on average 13,298 unique infections on a given day. We designed and executed a series of experiments to understand the dynamics of the environment and the resulting requirements on a tarpit. In this section, we report on the success of trapping these infections with respect to the following questions:

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

A. The impact of a tarpit in steady state

During the entire experiment, we encountered Mirai infections at 423,810 IPs, of which we were able to trap a total of 202,003. As discussed above, the number of momentary infections is much lower, and of the 13,298 addresses infected on average per day, we were able to trap on average 6,326 (47.5%). In this subsection, we report on the effectiveness of the infrastructure once the tarpit was fully active, namely when the telescope and honeypots (see figure 1) fed their sightings to the tarpit, enticing hacking attempts. In VI-B and VI-C, we report how long it took to get there and the performance degradation if we would not keep track of new infections.

Even a single tarpit can tie up the brute-forcing of thousands of devices. As discussed in section II, Mirai uses separate routines for finding victims and exploiting them. By allegedly responding to its scanning probes, our theory was to tie up a device's brute-forcing resources so that other concurrently identified victims could not be compromised. In practice, this turned out to be remarkably effective: The original Mirai source code supports 128 concurrent brute-forcing sockets, none of the variants we observed have ever increased this limit. To measure the effectiveness of our infrastructure, we count the number of brute-forcing connections we can concurrently trap, figure 7 shows a CDF over all connection attempts made to the tarpit, showing the average number of new connections per minute. On average, we see 356 new sockets per minute, 2.8 times the maximum number of connections Mirai can maintain at a single point in time. As the sockets timeout on either the OS or in the application itself, the malware immediately creates a new socket towards the tarpit. This means that when the duration-before-connections timeout is longer, the number of sockets created is lower. We find that the duration it takes for a socket to timeout is indeed inversely proportional to the number of new sockets created over time. When we combine our measurements, we see that we trap on average 126.2 threads per infected IP and thus absorb nearly the entire brute-forcing capability of devices through our tarpit.

All tarpitting strategies are equally effective. As described in Section IV, tarpitting could take place at layer 4 or layer 7 using different strategies. Our tarpit supports three application protocol-agnostic connection handling strategies and two additional ones geared towards telnet. Different reply strategies generally do not increase the time a socket is kept open. This means that tarpits could be designed much simpler: after completing handshakes with a SYN+ACK, we can forget about the connection, limiting the strain on the tarpit itself. The duration for which tarpit connections last however differs by protocol, as noted in appendix XI-B.

Devices can be trapped for a long duration until the network interferes. The success of the tarpit largely depends

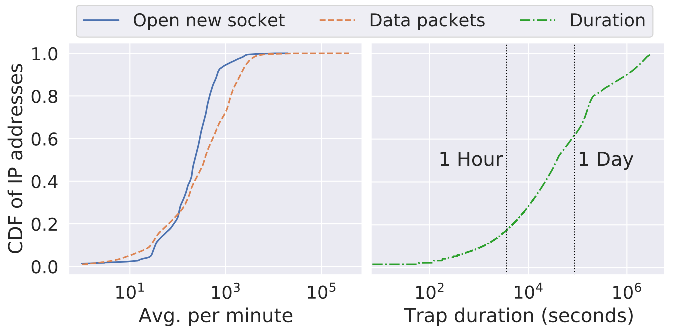


Fig. 7: CDF of average number of connections per minute for trapped IPs and the total duration an IP address is trapped.

on the duration for which it can trap devices, as during this time the device will connect to fewer vulnerable devices. Figure 7 shows a CDF of the *total duration* successfully trapped devices spend contacting the tarpit, where 60% of the IP addresses are only trapped for less than a day and 19% for even less than one hour. IP-based measurements are often biased due to ISP's IP churn [32], where networks forcibly disconnect customers' Internet connections and let them rejoin with a new IP address [35]. The infection would however persist and reappear at the new IP address. The authors in [35], [21] show that the bulk of ISPs reset in intervals between 4 and 24 hours, which would heavily skew the distribution as many churning IP addresses have a short lifetime and can therefore not be trapped for an extended period. Using the methodology of [21], we can link Mirai infections before and after a churn event together and thus estimate the total trapping duration, which allows us to estimate the effect of IP churn for non-Mirai infected devices as well. We find that infected devices are trapped for on average 6 days when accounting for IP churn. This 6-day timeframe is the average lifetime of Mirai infections we see active on a particular day. In other words, if an infected device contacts us and responds to the tarpitting event, we find that we can keep it trapped in 100% of the cases until the end of the infection. Once devices fall off our trap, none of the honeypots ever record any scanning traffic afterwards from such IP addresses, which means they can hold onto it entirely and thus neutralize the infection impact.

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

Due to the differences in how NATs are used and configured in different countries and Autonomous Systems [32], the effect of our tarpit largely varies between geographical regions. For example, in the two countries with the largest IP space [4], the US and China, the share of trapped devices are respectively 30 and 95%. This is beneficial as Mirai infections are far more prevalent in China than in the US [20]. The reverse

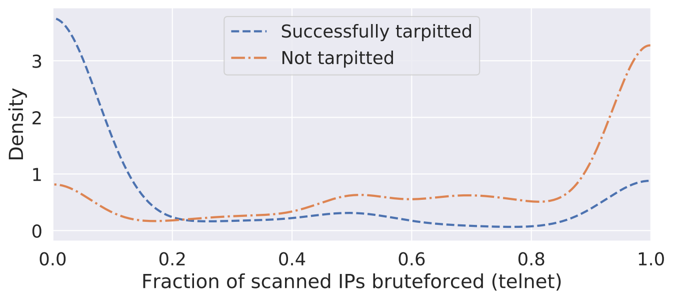


Fig. 8: PDF of the brute-force attempts on port 23, 2323 and 1023 after a device is found after a scan. Devices that responded to tarpits show significantly reduced brute-forcing.

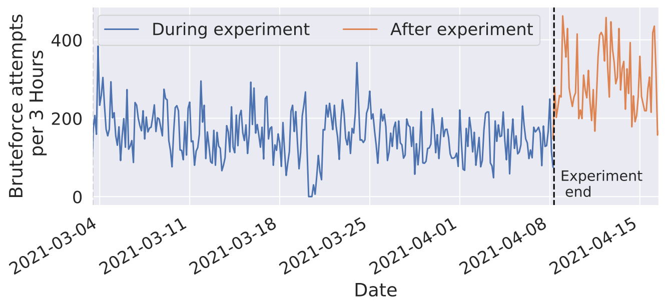


Fig. 9: Bruteforce attempts on port 23, 2323 and 1023 by 2,150 long-term infections, during and after tarpitting.

effect of network policies is also observed, as some networks have much higher churn rates than other networks. When a device is allocated a new IP address, the tarpit must again learn of its existence. This means that in networks with higher churn, infected devices are temporarily disappearing from the tarpit and are thus able to perform their regular operation. In the US, we find that the successfully trapped devices are much less likely to churn than devices located in China. The tarpit traps the US-based devices for more extended periods without any downtimes when the device is unknown. The effect of this IP churn downtime is however partially mitigated because of the brute-forcing queue of Mirai, where it remembers the tarpit IP address as a vulnerable device even after the infected device is allocated a new IP. The infected device will then connect to the tarpit from the new IP address, revealing its presence.

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

When the first scan packet from an infected device arrives, and the tarpit entices it to brute-force, the infected IoT devices will keep scanning the Internet and eventually reach our cloud honeypots. We analyze what percentage of the infected devices attempt to brute-force the cloud honeypot after it was discovered. Figure 8 shows the Probability Density Function (PDF) of the brute-forcing attempts made after a scanning packet is observed in a honeypot, in blue those that responded to the tarpit, in orange those that ignored our tarpit requests. To account for the case that both tarpit and honeypot were concurrently discovered and the brute-forcing was already about to happen before the tarpit reached the IoT device, the first 60 sec at the on-set of the tarpit activity are excluded. We see that the devices trapped in the tarpit are much less likely to brute-force other devices on the Internet after they discover potential victims. While for non-tarpitted devices, on average, 84% of the discovered hosts are subsequently brute-forced, for those trapped by our tarpit, the average is only 17%.

To demonstrate the difference tarpitting makes, let us zoom into those devices continuously infected during the entire experiment, which means that we can exclude confounding factors such as IP churn. Figure 9 shows the number of recorded brute-forcing attempts in 3-hour intervals for these 2,150 long-running infections, both outside and inside of a NAT. The blue line shows the brute-forcing intensity while we tarpitted infected devices and the red line depicts the intensity after we released them and continued to monitor for another week. After the devices were released, the average number of connections to the cloud honeypots almost immediately doubles. With about half of the devices shielded behind a NAT and half affected by the tarpit, we see a total reduction by half in brute-forcing.

While during the tarpitting the brute-forcing capability of IoT devices significantly reduces, it does not entirely drop to zero. The reason is the concurrency between Mirai's scanning and brute-forcing threads: while we fill up the brute-forcing queue with our tarpit as decoy victim, we compete for slots in this queue with Mirai's scanning routine. Whenever the saturated list drops one entry, it is a matter of chance whether our tarpit's response or a newly discovered victim enters its place. When we refer back to figure 4, we already see this phenomenon in the controlled experiment of our validation. The solution to this problem is simple though: always have a packet waiting in line as the queue clears a spot, which is essentially a question of sending tarpit requests fast enough. By using the same setup as in figure 4, we find that with 10 times the intensity, we can reduce the number of requests that slip through by 92% to 127.8 out of 128. While we could thus eliminate the problem almost entirely, this creates a strain on the devices already victimized by the malware. In this work, we limit our tarpit requests to the mentioned 6.4 kbps, or 0.09% of the average worldwide Internet connection speed [5], which, as we saw above, already occupies 126.2 threads.

Tarpitting the brute-forcing threads also reduces scanning speed. The main goal of our tarpit is to prevent the infection of others by filling up the queues of the Mirai malware, with a volume we specifically designed to be negligible for an end user's Internet uplink. A bit unexpected, though, was that devices in practice also effectively scanned slower by tying up the brute-forcing functionality. This reduced scanning across the Internet also implies a reduced probability of identifying vulnerable devices, which further slowed the speed of infection. Figure 10 shows the average scanning speed per IP address in pps before, during, and after the experiment for port 23 and port 5555. For port 5555, the average scanning speed nearly halves from an average of 2,872 pps to 1,516 pps,

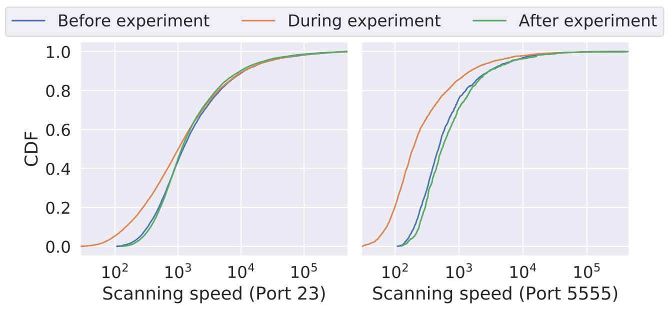


Fig. 10: CDF of (estimated) scanning speeds in packets per second before, during and after our experiment for two ports.

TABLE II: Effect of the tarpitting experiment on scan speed.

Port	# IP	PPS (normal)	PPS (tarpit)
Telnet: 23	150,661	8,498	7,617
RSFTP: 26	3,171	9,545	4,280
HTTP: 80	1,299	8,007	2,755
Telnet - alt: 1023	3,413	1,573	543
Telnet - alt: 2323	26,463	4,706	3,884
ADB Worm: 5555	10,347	2,872	1,516
Overall	174,167	7,822	7,105

for port 23 it declines by 10% from 8,498 to 7,617 pps. Table II lists the measured effects on the scanning speed across the ecosystem during the tarpitting experiment. This decrease in scanning speed results in vulnerable devices being less likely to be identified, slowing down malware spread. As we see the scans from infected bots decrease more quickly than the decrease in the number of bots, we can attribute the reduction to our tarpit eating up computing cycles in the Mirai malware that would have otherwise gone to scanning. Appendix XI-A investigates the reduction in scanning speed per AS in more detail, where we can see that we have a positive impact on approximately 60% of networks in the Internet.

One tarpit slows world-wide Mirai infections by 21%. As discussed above, the tarpit proved highly effective in trapping Mirai instances when they were accessible from the Internet. While effectivity could be further increased by sending more requests from the tarpit, we could significantly decrease both scanning and brute-forcing speeds of infected IoT devices, which means that it will take more time for the malware to find potential victims and infect them. Although only 48% of all infected devices responded to our requests, the tarpitting still significantly affected the worldwide Mirai ecosystem. When tracking the number of infected devices and the average time between infections (as IoT devices fairly often crash and are reinfected later [20]), we see that during our tarpitting, the average re-infection interval increased from 12.1h to 14.6h.

We can assess the slow down we accomplished by measuring how long it would take a random IP in the telescope to become infected by a compromised device before our tarpitting and during, and therefore measure the reduced propagation ability of Mirai as a whole. Even when reaching only half of all Mirai instances worldwide, a single tarpit would thus slow down the malware spread by 21% after locking up almost half of the brute-forcing capabilities in the ecosystem. The reinfection rate drops less than the brute-forcing capacity due to the saturation of the ecosystem where most victims are already infected and the botnet's brute-forcing power, thus being higher than the total power needed to infect every other device.

B. *System startup: How long before we achieve these effects?*

To measure the time it takes the tarpit to reach a steady-state and the time it takes for the measurement infrastructure to capture devices, we completely disable and start up the tarpit three times starting from March 23. On average it takes the tarpit about a day to start up and reach the steady-state, where we have seen all Mirai infections at least once. To identify the effect that the size of the measurement infrastructure has on the performance of the tarpit, we take random samples of our network telescope and average over 20 samples. We find that the decrease in devices trapped in the tarpit is not proportional to the size of the measurement infrastructure, as 2,500 honeypots identify almost 60% of the devices observed by 65,000 honeypots in the same time span. The difference in trapped devices also fades over time as a large network observes all infected devices, and smaller networks are still identifying new devices. While a large network would thus be faster in identifying the infections, a smaller network would still be suitable for identifying and trapping a botnet.

C. *System retention and scaling constraints: how much agility is needed in practice?*

The Mirai ecosystem is highly volatile and although there are a plethora of vulnerable devices, only a small percentage is infected at any given point. This poses the obvious question of whether this high volatility impacts the performance of the tarpitting. If we would not continuously feed updates into the tarpit, how much would the trapping degrade?

Our experiment uses the input from a network telescope with approximately 65,000 IP addresses to discover infected devices. When the tarpit reaches a steady-state, where on average 6,326 hosts are trapped, such large collection system might not be needed and might instead rely on a small set of monitoring points to remain in a steady state. To identify whether this retention exists, we conduct two separate experiments where we eliminate a large portion of monitoring IPs. In the first test on March 7, only 1,000 IP addresses instead of 65,000 were used to monitor for newly infected devices. As shown in Figure 11 the number of trapped devices gradually decreases, mainly due to IP churn where devices are still infected but moved to another IP. In the steady-state of this degradation, the tarpit trapped 4,613 devices, 27% less than the tarpit monitoring 65,000 IPs. Thus it would, in principle, be realistic to jumpstart a tarpit with a lot of IP addresses and maintain a reasonable performance with a much smaller installation. In our second test on March 17, only 100 cloud instances were used for monitoring. While still some new devices were identified, a large portion of devices disappeared from the system. The steady-state of the tarpit with only 100 monitors lies at 2,318 trapped devices. The system's retention is mainly influenced by IP churn, where devices leave the trap.

Finally, as a single tarpit reduces Mirai infections by 21% as 103 billion connections hit it, is the containment of Mirai just a question of scaling out? While our four-core server on

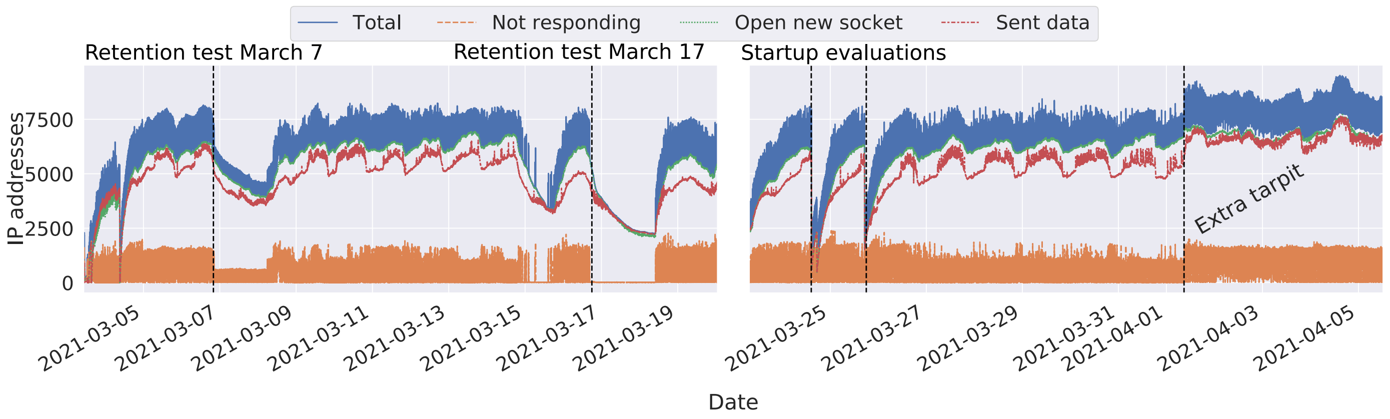


Fig. 11: Timeline of the experiments running from March 3 - March 20 and March 23 - April 8 showing the trapped IP addresses per minute. A fraction of the IPs that were sent SYN+ACK never respond. The vertical lines indicate when we tested the retention, startup time, and tested for network congestion through a second tarpit.

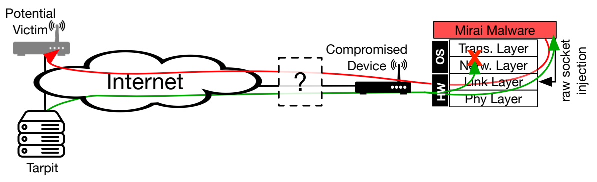


Fig. 12: Mirai sends and receives with a raw socket, which means that return packets are processed by OS and malware.

a Gigabit uplink is never saturated on CPU load or uplink capacity, it might be that the large volume of packets leads to packet drops further upstream. To learn about potential upstream limitations, we added a second tarpit to load-balance trapping activity. As shown in the last part of figure 11, a doubling of the infrastructure increased the number of trapped devices by 5%. The tarpits received more response packets from successfully trapped devices, where more devices send application layer data. There is a slight but statistically significant increase in the number of devices trapped per minute (R=1.02, p<.05). The effect on the containment as a whole is however marginal, thus the single tarpit was not held back.

VII. Reaching through NATs and firewalls

The problem we faced with trapping devices behind NATs results from the Mirai's connection handling. As the author intended to scan the Internet as fast as possible, Mirai does not use regular network sockets of the OS to establish connections but directly crafts and injects TCP/IP frames utilizing raw sockets. This allows the malware to scan the Internet at speeds otherwise impossible. This bypassing of the regular OS networking stack for outgoing packets means that return packets are received by both the malware and the networking stack of the OS. To see how this creates an issue for a NAT, let us first consider a malware connected directly to the Internet.

As shown in figure 12, Mirai would scan the Internet with TCP SYN packets via a raw socket to find victim devices. When a reply arrives, the TCP SYN+ACK response (in green) would be received by Mirai as well as the operating system. Mirai adds the destination to its brute-forcing queue, but the OS would not know how to treat this SYN+ACK, having never sent a SYN request to the destination. The TCP protocol mandates the OS to respond with a TCP RST [27], asking the remote party to tear down this connection. When infected IoT devices are directly connected to the Internet, this does not pose any problem, as Mirai would initiate several new brute-forcing connections shortly afterwards. When our tarpit floods Mirai with alleged SYN+ACKs, these connections would be unknown to the OS, but Mirai still takes note of them.

An issue however occurs when Mirai is located behind a network appliance such as a NAT or firewall. These devices usually keep state, and when a NAT or firewall receives the TCP RST from the host OS, it will take note of this unwanted traffic and in the future block packets from this IP address to the IoT device. For regular Mirai operation, this is not a problem because soon Mirai's brute-forcing threads open new connections and the NAT will again allow this traffic through. This is however an issue for tarpitting because we can only get tarpitting requests into the brute-forcing queue until the IoT's OS has returned a TCP RST and the network device shuts off the connection. While tarpitting in principle also works for NATted hosts, the window for trapping is limited.

To investigate this window, we experimentally test how many packets will be allowed through three different NATs before they shut off incoming traffic: two standard consumer home routers with NAT functionality, (1) a Zyxel VMG8825-T50 router and (2) a Fritzbox 7590, as well as (3) a Carrier-grade NAT of a large telecom provider for mobile Internet. A SYN packet is sent out from a device located behind the NAT to a tarpit in the experiment. The tarpit then responds with 128 SYN+ACKs, varying the speed between 25 and 1000 packets per second. This allows us to measure the typical processing time window of these devices.

Surprisingly, both the NAT on the Zyxel router and Carrier-grade NAT ignored the device's RSTs. In such non-compliant TCP implementations, the NAT would remain open and allow a tarpit to trap devices indefinitely. The NAT on a Fritzbox 7590 correctly closed on a TCP RST, leaving only a tiny window to send packets. We empirically find that the NAT closes after around 850 milliseconds of receiving the first SYN+ACK. While we can still trap all brute-forcing threads in a device behind this NAT, this tarpitting is only active for a limited time depending on our speed of SYN+ACKs and how large the brute-forcing queue of the infected device is. This behavior and the non-compliance of devices to the TCP specification explains why we still obtain some effect even in networks that utilize NATs.

VIII. Can distributed tarpits eradicate botnets from the Internet?

Middleboxes such as NATs and firewalls not only shut off a connection if the internal device responds with a TCP RST, they also keep a state of the outgoing and incoming connections. This is necessary because an Internet Service Provider might connect hundreds of customers using a single public IP to the Internet. When the customer connects to the outside, the Carrier-Grade NAT (CGNAT) records this request and later matches the response to forward it to the correct host. This is however a significant problem for tarpit operation: as we would have learned about an infected device from a honeypot, the brute-forcing invites from the tarpit towards the Mirai infection could not be matched in the CGNAT to a previous flow and thus would be discarded. For this reason, about half of the Mirai infections we observed in our experiments never ended up brute-forcing. In a sense, Mirai infections behind a NAT are therefore immune. However, the solution is trivial: do not only brute-force from a central location, but also respond to brute-forcing attempts from other honeypot locations when they are contacted. As each of these return flows can now be matched, every honeypot increases our leverage on Mirai infections and slightly reduces the spread of the malware.

To evaluate the feasibility of this approach and evaluate the number of instances required for potential eradication, we implemented a discrete-event simulator, which we calibrated based on the earlier experimental results. The framework simulates and predicts the scenarios presented before and will be shared with the community after publication.

A. Simulator design

Mirai is a collection of independently acting endpoints. The most logical approach for a simulation is an agent-based, discrete event simulation where all infected devices are autonomous, independent entities. Previous work has used similar modeling to test defense methods against, for example, botnet-based DDoS attacks [29], and a similar method can be used to simulate the spread of a botnet. We assume a Susceptible-Infected-Susceptible (SIS) model, where devices return to the pool of vulnerable devices after they have been rebooted or cleaned up. For Mirai, this is a valid assumption, as we observe devices to become reinfected frequently.

In each time step, every agent generates a number of scanning probes and infects newly found vulnerable devices. We randomize the scanned destinations using the randomization method of the Mirai source code. Behind an IP address, a device can locate other agents, honeypots, or vulnerable devices. If an agent encounters a honeypot, it will enter a "trapped" state that reduces the likelihood of infecting other devices. If it finds a vulnerable device, the agent will try to infect it, remain blocked for several time-steps to simulate the brute-forcing, and succeed with some likelihood.

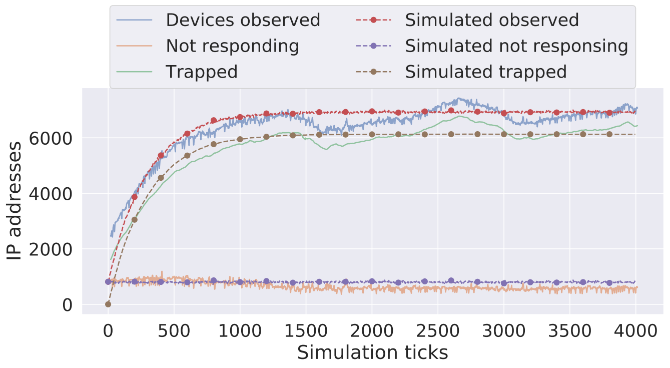


Fig. 13: Simulation of the system on the experiment.

In a tarpit scenario, there are two ways for an infected device to leave the "trapped" state: the device can be (1) cleaned up, or (2) IP churn changes the IP address of the device, which makes it unreachable for the tarpit so that the device needs to be detected again by a honeypot. A percentage of devices can also be placed behind a NAT, where adversaries can infect these devices, but it is not possible for the tarpit to address these devices and subsequently trap them. The simulator supports all three aspects, and we will evaluate these variables in the following. Table III lists all parameters used for the simulation, as well the experimentally measured value from section VI.

B. Simulation validation

To validate the simulation framework, we perform an agent-based simulation of the Mirai ecosystem with one tarpit, based on parameter values that we experimentally measured in section VI and previous work. We are particularly interested in two aspects during validation: first, does the agent-based simulation reproduce the same characteristics we experimentally measured in the steady-state in section VI-A, and second, does it also match the significantly more sensitive transient state when the system is initialized and scales up in section VI-B?

Figure 13 shows an excerpt of our simulation result together with the measured experimental results from initialization until steady state. The experimental measurements are smoothed using a rolling average of 20 data points to allow for easier readability. The figure shows similar proportions of devices we can trap and those that evade tarpitting, and we see especially in the dynamic scaling a close match. A two-sample Kolmogorov-Smirnov test confirms the similarity as we cannot reject the null hypothesis with $p > 0.1$.

C. What is needed to eradicate a botnet?

Given this simulator, we can now evaluate the sensitivity of the Mirai botnet to those parameters that we cannot change in the real world through our experiments, namely the prevalence of NATs and IP churn, in other words the percentage of

TABLE III: Parameters of the agent-based discrete-event simulation

Parameter Name	Description	Experimental value (Sec. VI)
1. Vulnerable devices	Total number of devices on the Internet vulnerable to infections	13,298
2. Honeypots	Number of honeypots on the Internet monitoring the botnet and trapping infected devices	65,000
3. Infected devices	Initial amount of infected devices at the start of the simulation (randomly chosen from the vulnerable devices)	100%
4. NATted devices	Number of the vulnerable devices behind a NAT and thus "immune" against the tarpit	40%
5. Average scan rate	Number of packets sent per average per tick, proportional to the spreading rate	15,000
6. Churn rate	Probability that IP churn occurs on an IP per simulation step	0.05%
7. Cleanup rate	Probability that an infected device is cleaned up and becomes "vulnerable" again	0.02%
8. Blocked time	Number of ticks an agent is blocked having identified a vulnerable device and is thus brute-forcing it	1
9. Infection probability	Likelihood that a brute-forcing is successful and leads to an infection	90%
10. Time steps	Number of ticks for which the simulation is run.	4,000
11. Trapping start	Time step at which the trapping starts, giving the infection time to spread before the tarpit is initialized.	0

devices hidden behind address translation and the turnover of IP addresses which influences how long devices are turned loose before a tarpit captures them again.

"Immunity" through NATs. Devices can be immune to a centralized tarpit behind a NAT if it drops unsolicited SYN/ACKs. These infected devices will however remain in operation and keep infecting others. In the experiment, we capture all detected devices that are not behind a NAT, but the ecosystem remains steady as there are too many immune systems. Running a tarpit directly on a honeypot would however solve this issue: packets will go through the NAT and reach the device because the connection was instantiated from the device behind the NAT. When we simulate a scenario where there are thus no devices immune to the tarpit, all attacking threads are captured after a device is identified. When honeypots are traversing the NAT themselves after observing a scanning packet from a device, we find that tarpits consisting of only 1,000 devices can effectively mitigate botnets consisting of 100,000 devices.

When devices are placed behind a NAT and honeypots cannot reach them, we find that the system will not trap every device that is located outside a NAT. Depending on the number of devices that "break free" due to IP churn or device cleanups, there is an equilibrium where as many devices are *trapped* every tick as there are devices that are leaving this state. An example of this is shown in Figure 13, where 60% of the devices are not behind a NAT and could therefore be trapped, but only 49% are trapped in the tarpit. Immunity to the tarpit from some parts of the ecosystem thus also affects the other devices in the ecosystem, which will not all be trapped.

Permanent tarpit vulnerabilities. Our system needs to continuously contact an infected device to work. Trapped devices can thus break free when a network operator churns the IP addresses in the network. As other research has shown, there are also tarpit vulnerabilities in malware itself that can permanently block a thread of the malware [40]. When a permanent vulnerability is found and exploited to block all malware threads using the method shared in this work, a device contacting a single honeypot can be permanently trapped. Our simulations show that in such a case large botnets could be completely eradicated already using a small number of honeypots. We perform different simulation experiments using various honeynet sizes based on a scenario where tarpit vulnerabilities are permanent and also lock up a device behind a NAT. While these simulations are performed using 50,000 vulnerable devices, this number does not influence the simulations much, as an increase in infections will also increase overall Internet scanning, making it more likely for the honeypots to find an infected device. Overall it takes a honeynet consisting of only 1,000 devices 3 days to trap 80% of all infections.

Reducing the speed of infection spread. Walla and Rossow note that a tarpit can be very effective in slowing down infection spread from a device [40]. While one device can be significantly impaired and therefore be slowed down, the effects of such slowdowns on the entire ecosystem are not yet known. We simulate a botnet scenario where 1% of the vulnerable population is infected at the start and a tarpit is active, aiming to disrupt the botnet from spreading through the network. We find that in practice, the effect of this slowdown is marginal as a botnet grows exponentially, and a device has to be observed by the tarpit network first before the slowdown occurs. Suppose the number of vulnerable devices is larger than the size of the tarpit. In that case, infected devices have a higher chance of finding and infecting a new device than ending up in the tarpit, keeping the infection spread alive. In our simulations, we find that when using a tarpit network of 100 devices, the infections reach their highest point after 1,424 ticks, whereas a tarpit of 1,000 devices would slow the infection spread such that the highest point is reached after 1,577 ticks. Only a tarpit consisting of more honeypots than the number of vulnerable devices would slow the tarpit down enough to remove the infection from the Internet immediately.

Cleanup rate. If a device is cleaned up by an operator or loses the infection due to a device crash or reboot, it will become vulnerable again if no additional action is taken to secure the device. While it is a good thing that infections are removed from the ecosystem, the cleanup rate of systems counter-intuitively slows down the tarpitting efforts significantly as devices can be cleaned up when they are in the "trapped" state and become vulnerable again to malware infection. In our simulations, we have used an average cleanup rate of 7 days. However, when decreasing this rate to once every day, the system cannot trap all devices anymore depending on the scanning speed of the malware, the size of the honeynet, and the amount of vulnerable devices. Thus, in this case, the tarpit would only be effective in slowing down the spread of the infections. When operators however secure their systems after cleanup, a tarpit network contains the spread of the malware and will eventually eradicate the malware.

IX. Discussion

Can we clean up the Internet with a pit of tar? Although it sounds a bit counter-intuitive to take on botnets bottom-up by spinning up decoy victims, in practice, this turns out to be an effective strategy against self-propagating malware, especially as C&C takedowns are getting increasingly tricky and provide diminishing downtimes. Tarpits have been known for a while but mainly were overlooked to address the threat of botnets, and to this date, there has surprisingly not been a study that evaluated their merits at scale.

A. *Community Tarpits*

Even though NATs are somewhat of a deterrent in theory, their impact is much less pronounced in practice. First, we can easily scale out so that the individual honeypots not only monitor but respond. As SYN+ACK flows of 6.4 kbps already trapped 98.5% of Mirai's brute-forcing threads, such a scale-out would require trivial resources and not impair the Internet connectivity of organizations or volunteers operating such a device. Second, even if we ignore immune devices, those trapped are missing in the further propagation. At a reproductive rate $R_0 = 1.0033$, this is already enough to contain Mirai, even though we cannot fully eradicate it, as infected pockets behind NATs continue to jump-start the infection. Lastly, we have seen that implementations can be faulty and allow us to punch through with many packets, not just single ones. Together with the stateless scanning now prevalent in port and vulnerability scanning, this gives an exploitable edge even in single tarpit scenarios. As we can frequently get at least one packet through a NAT, it would be possible to turn the tables entirely. Walla and Rossow [40] developed an automatic toolkit to discover vulnerabilities in malware that will cause the software to lock up. Combined with our tarpit at scale, such vulnerabilities would further amplify our impact. Given a permanent vulnerability, even a tarpit infrastructure of trivial size could scale a botnet down.

The easiest route forward – and the one that malware authors could not adapt to – would be to democratize botnet cleanup. Given that enough devices participate – where the required number is low enough that this could be done as a community project –, it will be possible to contain and eliminate this threat platform that has significantly heightened the Internet threat landscape in the past years. We release a flexible, turn-key honeypot/tarpit solution as open-source to leverage this insight. Organizations and individual volunteers can run their part of a botnet sinkhole on spare devices, requiring minimal resources from the hardware and the network. We provide this platform as part of this work, and we invite others to join this effort. Given a low entry barrier and a clear path towards mitigation, it will be possible to accomplish this jointly. The software and related documentation is provided at *https://www.malwaretarpit.com.*

B. *Honeypot Evasion*

As the tarpits were highly effective in containing Mirai, would it not be logical for adversaries to simply change the source to evade being trapped and make this effort futile? As it turns out, this would not be trivial to do without fundamentally changing how Mirai finds its victims. Let us look at three strategies for handling connection parameters.

Case 1. Current Mirai. Recall how Mirai picks a session-static source port and sets the ISN as the destination IP, and verifies this relationship for the returned SYN+ACKs. This means that the source port is randomized but used for all destination IP addresses. A honeypot that operates at IP a can inform the tarpit at IP b about the used setting, so that it can reply with ACK# = b as the expected fingerprint. This is the current operating principle of our system.

Case 2. Randomizing connection parameters. Obviously, this randomization could be extended to all other connection parameters. If an ISN is randomly generated and verified like the random source port, the same reasoning as in case 1) applies. Honeypot a would tell tarpit b the currently used ISN, and trapping of bots would also succeed.

Case 3. Keying parameters with a secret. The most challenging case would be if parameters would be keyed with a secret, for example ISN = hash(secret + destIP + time). Here if a connection attempt is made at a and honeypot a forwards this information to tarpit b, b could not trap the infected device, as SYN+ACKs from b would be identified as invalid responses and we could not infer the key.
This situation is however identical to the case of NATs. If an infected device behind a NAT connects to honeypot a, we cannot respond to the NAT's public IP address from b, as the NAT does not have any record of a prior connection attempt from one of its clients to b and cannot therefore route back the answer. A single, centralized tarpit cannot reach infections behind a NAT. As a solution, we introduced the community tarpits for which show that already a low number of volunteers would be able to eradicate Mirai. These community honeypots will also address case 3). If a Mirai instance contacts a community honeypot with a keyed ISN and random source port, the community endpoint responds and traps this particular device locally as it has a valid ISN and keeps getting new ones due to its tarpitting activity. As we show in the paper, if a device can be trapped, it stays trapped until the end of the infection. The community component therefore addresses also this advanced situation.

What could the adversary otherwise do to evade our system? Basically, it would be necessary to randomize and key connection parameters and keep track of probes and responses, which requires the software to save state *per outgoing SYN*, for 2^{32} possible IP addresses. The scanner routine could send randomized probes, count how many it has sent to a particular destination, match responses, and discard any SYN+ACKs not related to previous requests. This book-keeping will result in significant overhead and slow downs. Connection-based scanning is what scanners traditionally used, but was superseded over the past decade by state-less scanning to realize speed and complexity advantages. This would need to be undone.

X. Conclusion

Taking down a botnet is generally done by removing or taking over the botnet's C&C server or by arresting the perpetrators behind the botnet. These measures become increasingly complex as botnets use sophisticated means to hide or make their infrastructure more resilient. In this paper, we evaluate a possible alternative whether a collection of *tarpits* can be used to contain malware, and we find that even with one device of minimal requirements on CPU and bandwidth, the propagation can be significantly slowed down, and complete containment of these threats is within reach.

References

[1] [Online]. Available: https://labrea.sourceforge.io/Intro-History.html

[2] [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code

[3] [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/

[4] [Online]. Available: https://worldpopulationreview.com/country-rankings/ip-address-by-country

[5] [Online]. Available: https://www.fastmetrics.com/internet-connection-speed-by-country.php#average-speed-internet

[6] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, "Zippier zmap: Internet-wide scanning at 10 gbps," in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. San Diego, CA: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/woot14/workshop-program/presentation/adrian

[7] T. Alladi, V. Chamola, B. Sikdar, and K.-K. R. Choo, "Consumer iot: Security vulnerability case studies and solutions," *IEEE Consumer Electronics Magazine*, vol. 9, no. 2, pp. 17–25, 2020.

[8] L. Alt, R. Beverly, and A. Dainotti, "Uncovering network tarpits with degreaser," in *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014, pp. 156–165.

[9] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, "Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus," in *2013 8th International Conference on Malicious and Unwanted Software:" The Americas"(MALWARE)*. IEEE, 2013, pp. 116–123.

[10] M. Anisetti, C. Ardagna, M. Cremonini, E. Damiani, J. Sessa, and L. Costa, "Security threat landscape."

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

[12] A. A. Antonopoulos, K. G. Stefanidis, and A. G. Voyiatzis, "Fighting spammers with spam," in *2009 International Symposium on Autonomous Decentralized Systems*. IEEE, 2009, pp. 1–5.

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

[14] K. Borders, L. Falk, and A. Prakash, "Openfire: Using deception to reduce network attacks," in *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007*. IEEE, 2007, pp. 224–233.

[15] D. Dittrich, "So you want to take over a botnet..." in *5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 12)*, 2012.

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

[17] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 605–620.

[18] C. Gañán, O. Cetin, and M. van Eeten, "An empirical analysis of zeus c&c lifetime," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 97–108.

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

[20] H. Griffioen and C. Doerr, "Examining mirai's battle over the internet of things," in *ACM Conference on Computer and Communications Security (CCS)*, 2020.

[21] ——, "Quantifying autonomous system ip churn using attack traffic of botnets," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–10.

[22] J. Healey, N. Jenkins, and J. Work, "Defenders disrupting adversaries: framework, dataset, and case studies of disruptive counter-cyber operations," in *2020 12th International Conference on Cyber Conflict (CyCon)*, vol. 1300. IEEE, 2020, pp. 251–274.

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

[24] 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.

[25] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in ddos attacks: trends and challenges," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2242–2270, 2015.

[26] K. G. Hunter, "Scaling network tarpits for ipv6," Naval Postgraduate School Monterey, CA United States, Tech. Rep., 2018.

[27] P. John, "Transmission control protocol," *RFC 793*, 1981.

[28] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[29] I. Kotenko, A. Konovalov, and A. Shorov, "Simulation of botnets: Agent-based approach," in *Intelligent Distributed Computing IV*. Springer, 2010, pp. 247–252.

[30] J. Lathrop and J. B. O'Kane, "A case study in opportunity reduction: Mitigating the dirt jumper drive-smart attack," in *2014 IEEE Joint Intelligence and Security Informatics Conference*. IEEE, 2014, pp. 224–227.

[31] D. Moore, C. Shannon, and K. Claffy, "Code-red: a case study on the spread and victims of an internet worm," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, 2002, pp. 273–284.

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

[33] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee, "Beheading hydras: performing effective botnet takedowns," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 121–132.

[34] J. Nazario and T. Holz, "As the net churns: Fast-flux botnet observations," in *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2008, pp. 24–31.

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

[36] A. Pathak, F. Qian, Y. C. Hu, Z. M. Mao, and S. Ranjan, "Botnet spam campaigns can be long lasting: evidence, implications, and analysis," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, pp. 13–24, 2009.

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

[38] A. H. UW, Y.-K. Liu, and A. R. UW, "Counter-attacks for cybersecurity threats," 2005.

[39] K. Vaniea and Y. Rashidi, "Tales of software updates: The process of updating software," in *Proceedings of the 2016 chi conference on human factors in computing systems*, 2016, pp. 3215–3226.

[40] S. Walla and C. Rossow, "Malpity: Automatic identification and exploitation of tarpit vulnerabilities in malware," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 590–605.

XI. Appendix

A. *Impact of tarpitting on Mirai's scanning speed*

The overall reduction in Mirai's activity is not a result of merely containing a small share of highly active bots, but a reduction of its activity across a significant share of the Internet. Figure 14 displays a cumulative density function of the fraction of scanning speeds encountered by AS during tarpitting. We see that while 40% of ASse are unaffected due to their wide usage of CGNATs, for the remaining 60% drastic reductions apply. For 20% of all autonomous systems in the Internet, the scanning speed per AS is reduced to 0.3 (or 30%) of its pre-tarpitting activity, and for about half of the trappable ASes their activity is reduced by 50% or more.

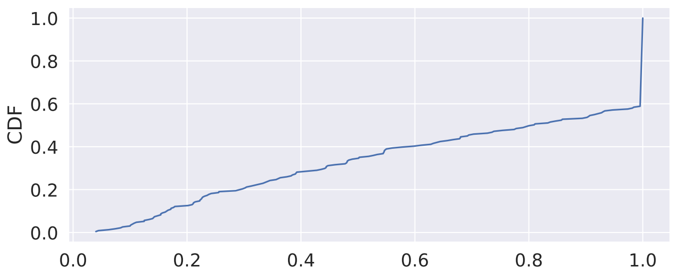


Fig. 14: CDF of the fraction of the scanning speed per AS while the tarpit is active opposed to its normal scanning rate.

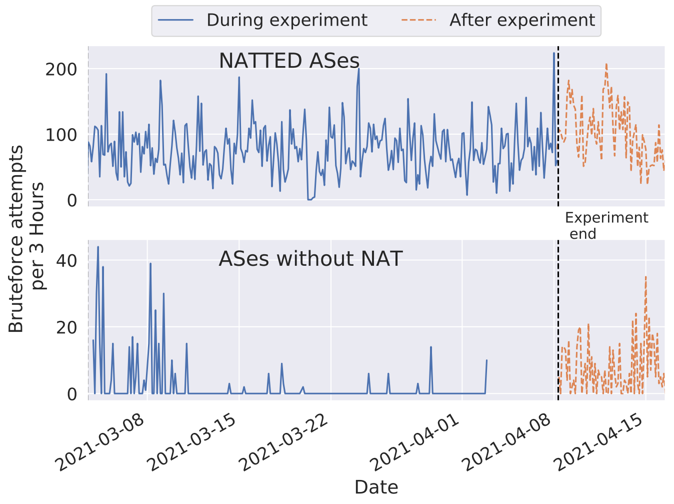


Fig. 15: Brute-forcing attempts during and after tarpitting, for autonomous systems without NATs and those utilizing NATs.

This dependence on the usage of NATs becomes even more visible when we revisit the results from figure 9 within the context of those ASes where we find evidence of the usage of CGNAT and those without. Figure 15 splits the results from figure 9 in these two groups. While it is clear that in autonomous systems with a NAT our tarpit has almost no effect, in those without a NAT the outcome is drastic. After the startup of the tarpit, new brute-forcing attempts almost vanish, with the exception of newly infected hosts until they are discovered by our infrastructure.

B. *Effect of different tarpitting strategies*

While the type of tarpitting (such as sending TCP keepalives or replying with random data) does not have any measurable impact on the connection duration, the duration of a connection to the tarpit greatly differs between protocols. For example, connections from malware targeting port 23 usually last longer than half a minute, while connections to port 445 are closed within 10 seconds regardless of the type of reply sent. For port 80, only completing the handshake even traps the median connection longer than other strategies as the client waits for data and hangs up on unexpected data. A notable exception is port 5555 (Android Debug Bridge), where sending random data increases significantly the duration of connections to the tarpit. Figure 16 shows CDFs of IP addresses for the average connection duration split on different ports.

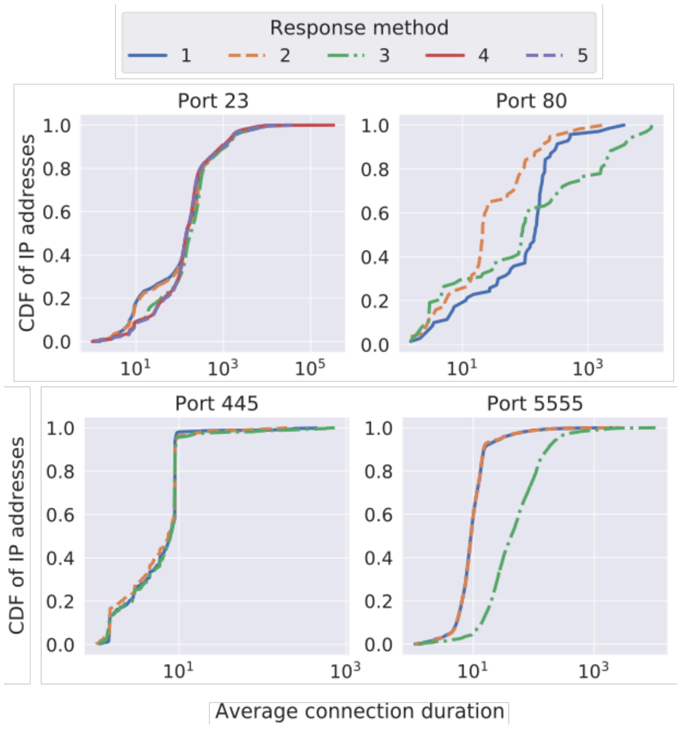


Fig. 16: Effect of different tarpit strategies on different ports.