Uncovering Network Tarpits with Degreaser

Lance Alt 
Naval Postgraduate School
laalt@cmand.org

Robert Beverly 
Naval Postgraduate School
rbeverly@nps.edu

Alberto Dainotti 
CAIDA, UC San Diego
alberto@caida.org

ABSTRACT

Network tarpits, whereby a single host or appliance can masquerade as many fake hosts on a network and slow network scanners, are a form of deceptive cyber-deception. In this work, we develop degreaser, an efficient fingerprinting tool to remotely detect tarpits. In addition to validating our tool in a controlled environment, we use degreaser to perform an Internet-wide scan. We discover tarpits of non-trivial size in the wild (prefixes as large as /16), and characterize their distribution and behavior. We then show how tarpits pollute existing network measurement surveys that are tarpit-naïve, e.g. Internet census data, and how degreaser can improve the accuracy of such surveys. Lastly, our findings suggest several ways in which to advance the realism of current network tarpits, thereby raising the bar on tarpits as an operational security mechanism.

Categories and Subject Descriptors
C.2.0 [Computer-Communication Networks]: General—Security and protection; C.4 [Performance of Systems]: Measurement techniques

Keywords
Tarpits; Internet Census; Sticky Honeypot; Deception

1. INTRODUCTION

Networks face a continual barrage of abusive and malicious attacks. Among available network security defenses is the class of deceptive network strategies. The high-level idea is simple: provide to adversaries the illusion of vulnerable targets, or promote an appearance of greater complexity than actually exists. For example, network honeypots [31] have long been used to attract abusive traffic and attacks. Traffic that arrives at the honeypot can then be analyzed in order to build databases of e.g. IP reputation [19, 34], malware signatures [6, 37], and to provide early detection of emerging attacks [25]. Such databases of abusive traffic characteristics can then be leveraged to help mitigate attacks elsewhere in the network.

A more advanced view of deception includes not only providing a believable target, but actively influencing the adversary through deceit [10, 28, 29]. For example, Trassare’s work on deceptive network topology [32] seeks to alter the adversary’s notion of the attack surface in an effort to change their decision making process, whereas moving target defense systems [18] frequently change the IP addresses of resources to increase the complexity of an attack against them.

By extension, network “tarpits” (or “sticky honeypots”) seek to slow automated scanning traffic or frustrate and confuse a human adversary [21, 35, 13, 16, 30, 15]. These tarpits can be configured to use inactive IP addresses or address blocks within a network, thereby providing the illusion of a large pool of active hosts. For each fake host, a single tarpit machine can answer all incoming connection requests. Some tarpits answer both to ICMP requests and on all TCP ports – providing the illusion of both fake hosts and services. In addition, by employing TCP flow control mechanisms to hold the connection open without allowing data transfer, the tarpit can both slow and penalize a scanning attacker.

In this work, we investigate the ability to detect network tarpits, and provide empirical evidence of their presence on today’s Internet. Our work is motivated by defensive security objectives, where the efficacy of a tarpit is limited by the degree to which the deception is believable; an adversary, whether automated or human, will avoid known tarpits. For instance, an automated scan that can detect tarpits in real-time could skip those tarpits and tarpit networks, improving the scan performance (both in time and accuracy).

We develop an active, on-line probing methodology to efficiently (2 to 6 packets per target) detect a variety of different tarpits in real-time. Our methodology is based on TCP options fingerprints and TCP flow control behavior. By sending a series of specially crafted probe packets, we discern real TCP stacks from tarpit hosts. We synthesize our methodology into a publicly available open source tool [5], degreaser, that infers the presence of tarpits among a set of input target networks.

In addition to validating our methodology against known-ground truth tarpits, we utilize degreaser to perform an Internet-wide scan. To facilitate large-scale scanning and avoid triggering anomaly detectors, degreaser uses permutation scanning [7, 12] to pseudo-randomly iterate through the IP address space when probing. Our real-world Internet scan, which probes at least one address in each /24 network in the Internet, discovers 107 different tarpit subnetworks...
ranging in size from /16 (with up to 2^{16} fake hosts) to /24 (with up to 2^{8} fake hosts). As some of these subnetworks have both real hosts and tarpits interleaved, we characterize the occupancy of fake addresses, both as a function of tarpit type and network size. In all, we find over 215,000 active IP addresses that are fake. These provider, customer, and university networks spread across 29 different countries and 77 autonomous systems. While these numbers represent non-trivial portions of address space, they are small relative to the size of the Internet. However, we note that even small blocks of tarpit addresses can greatly slow automated scans as part of their intended capturing behavior.

To better understand how fake hosts and subnetworks are observed in Internet measurement campaigns, we examine publicly available measurement data from an Internet-wide ICMP census [8] and HTTP scans [4]. We find that the networks inferred by degreaser as filled with tarpits, appear instead as fully occupied in the census data. Thus, not only are tarpits affecting abusive network scans, they also successfully deceive legitimate measurement surveys. As such surveys are used in policy decisions (e.g. understanding IPv4 address space exhaustion and allocating new addresses), we offer degreaser as a means to improve their accuracy.

In general, we demonstrate that tarpit deception is operationally deployed in the Internet. We therefore make four primary contributions:

1. We devise fingerprinting techniques to efficiently detect network tarpits in real-time. We synthesize our approach in degreaser, an open source tool that discerns deceptive tarpit hosts and networks (Section 3).

2. We discover tarpits in the wild by running Internet-wide scans using degreaser, showing that such security tools are actually employed in real world scenarios and characterizing their deployment (Section 4.1).

3. We empirically examine how tarpits pollute network measurement studies, including the Internet census [2] (Section 4.2). With degreaser, we suggest that such measurement surveys can return more accurate results.

4. We suggest improvements to make tarpit deception more realistic (Section 5), thereby raising the bar for this operational security mechanism.

2. BACKGROUND

Cyber-deception has been used to great effect in understanding and mitigating network attacks. For instance, honeypots [31] may attempt to lure an attacker away from a true network resource to some deceptive resource, thereby gaining additional intelligence on the tools, techniques, and procedures (TTP) employed in an attack. Similarly, darknets and network telescopes [24, 35] capture traffic destined to unused blocks of address space, passively providing insight into attack behaviors.

Not only does deceit permit intelligence gathering, it can induce an attacker to expend time evaluating potential attack vectors on fake resources, slow their progress and keeping them in an intelligence gather phase – increasing the likelihood of being discovered [29].

In this work, we restrict our analysis to a particular type of deception: fake hosts of network tarpits. In this section, we review the salient features and operation of network tarpits, as well as prior work in identifying network deception.

2.1 Network Tarpits

Network tarpits were originally conceived in response to aggressive scanning worms, e.g. code-red [23]. Analogous to physical tarpits (which trap animals in sticky tar), network tarpits attempt to hold open remote incoming TCP connections that are likely to be abusive. Once held (or “stuck”), the tarpit both actively prevents data transfer over the connection and discourages the remote end from disconnecting. By preventing such connections from performing any useful work, the tarpit both slows the scanner and consumes the scanner’s resources. Penalizing abusive connections, whether from spammers, worms, or other malicious activities, not only makes the target appear less attractive, it helps to slow the global rate of scanning and permits introspection over the behavior of the scanner.

Network tarpitting has been applied at both the transport and application layers. For instance, SMTP tarpits attempt to slow and penalize email spammers once the application-layer detects that the incoming email is likely spam [13, 16]. The tarpit mechanism employed by an email server could be at the SMTP-layer, for instance by taking an arbitrarily long time to respond to incoming SMTP commands.

Other tarpits are able to extend their deceptive operation by co-opting unused addresses on their local subnetwork. Network tarpits such as those in LaBrea [21] and the Netfilter TARPIT plugin [15] can answer and hold connections to multiple IP addresses on a network, including addresses that are not in use. In this fashion, not only do these tarpits slow connections to active machines on the network, they fake the existence of servers for every IP address in network prefix.

2.2 Tarpit Operation

Network tarpits employ three primary mechanisms: 1) a means to determine which IP addresses on a subnetwork are unused and thus may be faked; and 2) a strategy to impersonate hosts by responding to TCP, UDP, or ICMP probes destined to fake IP addresses; and 3) a method to hold the TCP connection open.

The IP addresses for which the tarpit responds may be statically configured, or dynamically inferred. When inferred, the tarpit typically acts conservatively and relies on the lack of a layer-2 address resolution protocol (ARP) response as a indication that the IP address in question is not in use by a real host. For example, Figure 1 shows the sequence of packets observed on a subnetwork running an instance of LaBrea [21]. The local router, 10.1.10.1, has a packet with destination 10.1.10.210 to deliver. Because the network is an Ethernet, it must first determine the layer-2 address of the destination and sends an ARP request. Note that the first three ARP requests in Figure 1 go unanswered and are spaced roughly one second apart. LaBrea promiscuously listens on its network interface and observes these unanswered ARP requests. On the fourth request, LaBrea conservatively infers that there is no true host with the requested IP address and instead responds with its own layer-2 address on behalf of that IP address. In this way, LaBrea takes over responsibility for the IP address, and will do so similarly for other unused addresses.

Next, TCP connections must be held open. A TCP-level tarpit typically listens on all TCP ports and responds to SYN connection initiation with a SYN/ACK. However, the general strategy of the tarpit is to use the 16-bit window field in the TCP header [26] to lock the client in a fully
established TCP connection, while simultaneously preventing it from transmitting data. In the TCP protocol, the window field implements flow control – providing an indication to the sender of the number of bytes the host is able to accept [26], thus relieving the network of the burden of transmitting packets that can only be dropped by the receiver. The tarpit exploits such mechanism by advertising a small initial window, and then replying to incoming packets with a window size of zero. Per the TCP specification, the remote host will send “zero-window” probes periodically to the tarpit to determine when it can resume sending data. The tarpit will never increase the window, keeping the remote host locked in a persistent connection. In practice, most operating systems at the client will eventually terminate the TCP connection after a certain amount of time has elapsed without an increase in the window size. When the remote host decides to terminate the connection using the normal FIN process, the tarpit ignores these packets forcing the operating system to maintain connection resources until the FIN-WAIT period expires. During the whole connection, the client’s socket resources are consumed to keep connection state (while the tarpit instead maintains no state), but the client is unable to perform any useful work.

While the mode of operation described above is known as persistent mode, tarpits support also a non-persistent mode in which all the subsequent packets on the connection sent by the client after the initial handshake are simply ignored. The client is unable to perform any useful work. During the whole connection, the client’s socket resources are consumed to keep connection state (while the tarpit instead maintains no state), but the client is unable to perform any useful work.

While the mode of operation described above is known as persistent mode, tarpits support also a non-persistent mode in which all the subsequent packets on the connection sent by the client after the initial handshake are simply ignored. The client is unable to perform any useful work. During the whole connection, the client’s socket resources are consumed to keep connection state (while the tarpit instead maintains no state), but the client is unable to perform any useful work.

Two of the most widely known tarpit applications are LaBrea [21] and the Linux Netfilter TARPIT plugin included in the xtables-addons package [15]. In total, we consider the following types of tarpits in this work:

- **LaBrea**: LaBrea [21] runs as a user-space application using raw sockets to intercept all packets arriving from the network. TCP responses are crafted without using the operating system network stack, thus requiring no peer connection resources. LaBrea supports hard capturing a specific list of IP addresses, or it can utilize unused IP addresses within a network block using its ARP-timeout mode. In ARP-timeout mode [21], LaBrea will intelligently release IP addresses it responds for if another device joins the network. This allows a subnet to remain “full” even as hosts join and leave the network. LaBrea can be run in either persistent or non-persistent modes. Herein we refer to these as “Labrea-P” and “Labrea-NP” respectively.

  - **Netfilter**: The xtables-addons package [15] includes two deceptive plugins: TARPIT and DELUDE. Netfilter interfaces directly with the operating system, providing additional target firewall rules to which packets can be directed. Netfilter rules are implemented by the user via the iptables application. Because the plugins are integrated in to Netfilter, it permits use of the full compliment of firewall rules, for instance permitting more complex decisions than LaBrea when choosing which connections to tarpit. However, Netfilter is more limited when establishing a tarpit over a wide range of addresses, since the network interface must be bound to all IP addresses covered by the tarpit. Additionally, Netfilter does not provide an ARP-timeout mode, requiring all tarpit IP addresses to be specified.

The xtables-addons TARPIT plugin holds TCP connections in the same fashion as LaBrea, but does so in a unique way what that allows us to distinguish it from LaBrea as we discuss in the next section. In addition, xtables-addons includes a DELUDE plugin that replies to incoming SYN connections with a zero-window SYN/ACK, but sends a RST to terminate the connection for all other packets. Delude is designed to fool network scanners looking for open ports. While DELUDE does not explicitly attempt to hold the connection open, it provides deception by responding for non-existent IPs and services. Herein, we refer to the Netfilter TARPIT plugin as “iptables-T” and the Netfilter DELUDE plugin as “iptables-D.”

- **Other**: Last, we find a class of tarpitting addresses that act as tarpits, but do not behave in a way that is consistent with either LaBrea or the Netfilter plugins as configured by default. The primary distinguishing characteristic of these addresses is that they deterministically respond with a zero-window in the SYN/ACK. We observe that some of these addresses respond once, but then implement a timed blacklist whereby subsequent connections go unanswered for a period of time. While we cannot definitively attribute the variety of behaviors in the group to a particular implementation, they act as tarpits. We therefore herein refer to this group as “Other.”

### 2.3 Detection

Fundamentally, the tarpit detection problem is based on fingerprinting. Both active and passive network stack fingerprinting has been employed with large success to identify operating systems [36, 22]. These methods rely on
Figure 3: Ground-truth testing on LaBrea network in a /29 subnet.

implementation-specific differences as exhibited by various TCP/IP fields, for example IPID, TTL, options, etc. However, these existing fingerprinting tools fail to identify tarpits. More complex physical device fingerprinting [20] techniques similarly fail due to the lack of TCP timestamp option support in tarpits.

Instead, this work seeks to use additional characteristics to identify tarpits by eliciting tarpit-specific behavior via active probing. To the best of our knowledge, degreaser is the first tool to reliably infer the presence of tarpits.

3. METHODOLOGY

In this section, we seek to identify traffic characteristics of network tarpits. We empirically analyze various traffic properties of real production networks as compared to tarpits, and describe features that we find to not provide sufficient power to discriminate tarpits. Then, by running known tarpits in a controlled environment, we develop a set of discriminating traffic characteristics as the basis for degreaser. From these characteristics, we detail degreaser’s inference algorithm. Last, we describe the permutation scanning used by degreaser to perform Internet-wide probing.

To verify our hypotheses, we establish a ground-truth LaBrea network on the Internet. As shown in Figure 3, we install LaBrea on a /29 subnetwork and use PlanetLab [9] to probe from multiple vantage points the entire /24 aggregate to which the /29 belongs. We scan the /24 network by attempting to establish TCP connections to each IP address in the subnet and capture the packets for further analysis.

3.1 Non-Features

We first describe features that, intuitively, might provide an indication of a tarpit host. These features, however, proved unreliable or unfeasible.

- **Layer-2 Address:** An observer with access to the same network segment as LaBrea can trivially discern traffic from fake hosts by examining the link-layer addresses. LaBrea uses a specific, hardcoded, Ethernet MAC address (00:00:0f:ff:ff:ff) for all of its responses regardless of its physical network adapter address.

In contrast to a fixed address, the Netfilter plugin uses the network adapter’s MAC address for all packets it generates. However, in normal operation, a single interface may have multiple assigned IP addresses. Thus, the same MAC address across multiple IP addresses is only a weak indicator of a Netfilter tarpit. Since a tarpit would typically be used to combat threats outside the local network, our efforts instead focus on remotely identifiable characteristics, i.e., those that are discernible from outside the network segment containing the tarpit host.

- **Active IPs in a Subnet:** Intuitively, we might expect high-occupancy subnets to be good indicators of possible tarpits. To this end, we initially investigated using a hitlist of probable tarpits as inferred from the /24 subnets with more than 240 responding web hosts in the scans.io [4] survey. However, we found this to be a poor selection criterion as many of these subnets belong to well known Content Distribution Networks and web hosting services. We did however, make use of other aspects of the data in [4] when building degreaser as detailed in §3.3.

- **Open TCP Ports:** An IP address that answers for all TCP ports is indicative of a tarpit. However, ascertaining the set of open ports requires $2^{32}$ probes per host. Even a search for hosts with more than a particular threshold of open ports imparts an exponential impact on the total number of probes required to scan the Internet. Further, some tarpits answer only to one or a small number of ports. We therefore use open TCP ports only as a last test to disambiguate instances of possible tarpits.

- **Response Time:** As described in §2.2, LaBrea can use ARP-timeout mode where it waits for multiple ARP requests to go unanswered before using an IP address. The remote host will therefore observe a delayed response time between sending the SYN packet and receiving the SYN/ACK. Intuitively, the default timeout period of three seconds provides a discriminating characteristic to identify a possible LaBrea host. However, this ARP-induced behavior is unreliable as it only occurs when the router’s ARP cache is not already populated. Given the large amount of typical network scanning traffic and noise, we could not reliably use such criterion to detect our ARP-based tarpit.

More importantly, in §4 we show that LaBrea typically operates in hard-capture mode, where it does not utilize ARP and does not introduce extra delay. The Netfilter tarpit instead, does not provide an ARP-timeout mode and is not susceptible to exploitation of this characteristic.

3.2 Discriminating Characteristics

We identify two characteristics which, taken together, enable reliable remote detection, via active probing, of fake hosts created by tarpits: TCP window size and options.

3.2.1 TCP Window Size

Fundamental to the tarpit-like behavior of both LaBrea and the Linux Netfilter TARPIT plugin is TCP flow control (§2.2). We observe that LaBrea and the Netfilter plugin return by default a TCP window size of 10 and 5 respectively. While LaBrea’s window size is configurable, Netfilter’s value is hard-coded into the source.

Given these known window values, we first examine the more general distribution of window values (scaled as required by any TCP window scaling option) as observed in two different traffic captures and summarized in Table 1:

1. **Equinix:** One minute of traffic from a 10Gbps link within the San Jose Equinix exchange point on December 19, 2013 as anonymized and made available by CAIDA [1]. This trace contains approximately 5.4M flows, 31M packets, and 24Gbytes of traffic (average of 456Mbps). 94.8% of the traffic is TCP.

2. **Campus:** One hour of traffic from our academic department’s border router. This trace contains approximately 1.2M flows, 48M packets, and 34Gbytes of traffic (average of 9Mbps). 94.7% of the traffic is TCP.
Based on our detection criteria, we build a publicly available open source tool, degreaser, to automatically detect tarpitting hosts and networks. Degreaser runs on a standard Linux host using raw sockets and supports multi-threaded scanning. Degreaser is designed to scan a list of subnets and classify each responding IP. When tarpit-like behavior is detected, degreaser can determine which of the two most popular tarpit applications, LaBrea or the Netfilter plugin, is being used.
Algorithm 1 Degrease(Dst)

1: \text{SYNACK} ← SendSYN(Dst)
2: \text{W} ← \text{Window(SYNACK)}
3: if (\text{Options(SYNACK)} \setminus \text{MSS} = \emptyset) and (W < 20) then
4: \text{ACKResponse} ← SendACK(Dst)
5: \text{if ACKResponse} = \text{RST} then
6: return(delude)
7: \text{else if Window(ACKResponse)} = 0 then
8: return(iptables-T)
9: \text{if W} = 0 then
10: \text{FINResponse} ← SendFIN(Dst)
11: \text{if FinResponse} = \emptyset then
12: return(other)
13: \text{else}
14: \text{DataResponse} ← SendData(Dst, size = W – 1)
15: \text{if DataResponse} = \emptyset then
16: \text{ZeroWinResp} ← SendZeroWinProbe(Dst)
17: \text{if ZeroWinResp} = \emptyset then
18: return(labrea-NP)
19: \text{else}
20: return(labrea-P)
21: return(real)

3.4.1 Scanning Algorithm

Because degreaser circumvents the host TCP stack by sending raw IP packets, we take care to prevent the operating system from receiving unexpected replies. Degreaser uses ports outside the ephemeral port range and integrates with the Linux firewall to block incoming packets to that port range. This ensures that the host’s operating system does not receive probe responses and send RSTs.

The degreaser pseudo-code is given in Algorithm 1. The scan is initiated by sending a TCP SYN packet to the remote host. The packet is a standard SYN that contains common TCP options (MSS, WSOPT, SACK, and TSOPT) [17]. A response timeout of five seconds is used for all outgoing packets and if no response is received within the timeout, the host is marked as not responding. If a SYN/ACK is received, degreaser classifies a host as non-tarpitting if the receive window size is greater than the 20 byte threshold determined in 3.2.1. Similarly, if the received SYN/ACK contains any TCP options it classifies the host as non-tarpitting since neither LaBrea nor Netfilter include TCP options in their replies. During testing however, we observe that some paths proactively add the MSS option for all TCP connections when none is present. Previous work has shown that middleboxes are known to add the MSS option [11, 14], therefore degreaser can selectively ignore the presence of MSS.

This simple algorithm is sufficient to classify the nodes in our test network as either tarpits or real hosts with perfect accuracy and no false positives or negatives. However, it is possible that a legitimate host would advertise a small window and also not include any TCP options. In addition, we wish to distinguish between the variety of different tarpits defined in §2.2. The remainder of the algorithm provides this functionality.

Next, degreaser sends an ACK to complete the three-way handshake. If the response to this ACK is a RST, we infer an iptables using delude. However, if the response contains a window of zero, we infer an iptables target.

Typically, however, we do not expect nor receive a response to the ACK packet. In this case, and when the SYN/ACK window was zero, we send a FIN packet to elicit a response from a valid host. If we receive no response to our FIN, we infer that the target is an unknown tarpit.

Otherwise, if the SYN/ACK window was non-zero, we wish to distinguish between a real-host with a small advertised window and the two types of LaBrea tarpits. We transmit a data packet with a payload size one byte smaller than the advertised receiver window from the remote host. A legitimate host will respond in one of two ways. If the legitimate host is still busy and has not drained its receive buffer, it will send an ACK with the receive window decreased by the amount of data we send. Alternatively, a legitimate host will increase its window if the buffer pressure is relieved. In contrast, LaBrea will not respond. However, in persistent mode, LaBrea will respond to zero window probes. We therefore distinguish between persistent and non-persistent LaBrea by sending a zero window probe and observing whether we receive a response.

Thus, degreaser requires only one TCP connection and a maximum of four packets per probed IP address, and avoids non-deterministic network behavior such as response time measurements.

3.4.2 Random Permutation Scanning

To facilitate large-scale network scanning and avoid triggering anomaly detectors, degreaser includes the ability to pseudo-randomly scan large network subnets using a cryptographic cipher. We use the RC5 block cipher with a 32-bit block size to create a pseudo-random permutation over a much smaller domain. Degreaser automatically switches between either a prefix-cipher or cycle-walking cipher, as described in [7], depending on the number of addresses to scan. Multiple disjoint subnets are combined in to a single continuous domain to provide pseudo-random scanning across the entire scan range.

4. FINDING TARPITS IN THE WILD

In our quest to find live tarpit hosts on the Internet, we first searched on the web for organizations that publicly admit to using tarpits. Our search revealed only one company that indicates using a tarpit. In fact, their website provides a publicly viewable statistics page that shows a list of all the IP addresses captured by their three tarpitting hosts. However, we sought to better understand the wider operational deployment of unadvertised (and previously unknown) tarpitting on the Internet.

4.1 Probing the Internet

Realizing that as a network defense mechanism, very few organizations would provide detailed information revealing tarpit hosts they are running, we scanned the Internet for tarpits. We used degreaser with the pseudo-random permutation scanning described in §3.4.2 to scan approximately 20 million IP addresses in May, 2014. Using permutation scanning allowed us to scan at least one host in all of the \( \approx 14.5 \times 10^6 \) routed /24 subnets over the course of 30 days. Out of the \( \approx 20 \) million addresses probed, degreaser discovered 1,451 IP addresses exhibiting tarpit-like behavior (either LaBrea or the Netfilter TARPIT plugin). We manually verified a random sample of these hosts, and found that they did indeed exhibiting tarpit-like behavior, confirming that our detection methodology works correctly.

From these seed tarpit IP addresses, we used degreaser to perform an exhaustive scan of each /24 subnet containing
one of the 1,451 tarpit IPs. Often, the majority of these subnets are completely full of fake tarpit IP addresses. Several subnets however were mixed, having tarpits intermingled with real hosts and non-responding IPs. We more completely characterize the tarpits in the next section.

We then expanded degreaser’s search to adjacent subnets to determine if the tarpitting /24 belonged to a larger aggregate tarpitting subnetwork. Overall, we found several larger subnets (up to /16 blocks) filled with tarpits and totaling over 215,000 fake hosts.

4.2 Characterization

From our Internet-wide scan, we assimilate a list of 107 different tarpit subnetworks. These networks are spread across 29 different countries and 77 autonomous systems, indicating that multiple independent organizations are using network tarpits. Additionally, the subnet ownership was diverse, with 51 university subnets, 36 provider subnets, 19 customer subnets, and 1 government subnet exhibiting tarpit-like behavior. This non-trivial presence of operational tarpits in the Internet speaks to one aspect of cyber deception currently used in real networks today. Our survey allows us to understand more about their properties and to validate some of our reasoning.

Figure 6 shows a breakdown of the various tarpit subnet sizes, as a function of tarpit type. Of note are the existence of six large /16 tarpit networks, where a /16 has a total of $2^{16}$ possible IP addresses. The Netfilter plugins, dehuide and obfuscated identifier. While the inter-quartile range is small, the outliers are significant. The lack of ground truth and unresponsiveness of organizations suspected of running tarpits makes determining degreaser’s false-positive rate difficult to calculate. Lab testing resulted in 100% accuracy, however, due to the numerous configurable options in existing tarpit software and the position, while other subnets are a mix of real and tarpitting IPs. Figure 8(a) shows the cumulative fraction of tarpit networks versus their occupancy for each of the fix tarpit types. We observe a variety of occupancy’s, with persistent LaBrea being the most highly occupied.

Figure 8(b) again shows the cumulative fraction of tarpit networks versus their occupancy, but broken down by the subnet size. Approximately half of all /24’s have an occupancy of 95% or greater, while more than 60% of the /22’s and /23’s have an occupancy of 95% or more. The occupancy’s of the six /16’s vary more widely; two of the /16’s are fully occupied (more than 99%), while the other four are between 15-30% occupied.

Next, we note that many tarpits answer all TCP ports. To better understand the port-specific behavior of the tarpits we discover, we probe all addresses within each tarpit subnet for TCP ports 80, 443, and 34343. TCP port 34343 is not assigned to any service, and therefore would not typically be expected to respond. Figure 8(c) shows the cumulative fraction of tarpit networks versus their occupancy as a function of port number. Unsurprisingly, we observe that by probing port 80 we find a higher occupancy than port 34343. Of note however, is that the difference between port 80 and 34343 is relatively small, suggesting that most of the tarpits we find answer for all ports.

Finally, with a substantial list of subnets running network tarpits, we sought out ground truth to further confirm that our detection methodology is accurate. Since we were unable to directly find confirmation on any of the subnet owner’s websites, we utilized Whois[3] records to make email contact. After waiting over two weeks, we had only received responses from two of the organizations we queried. One organization’s response was the creation of a “trouble ticket,” for which we never received further information. The second organization that responded was helpful and confirmed that they indeed were running LaBrea on the subnets in question. The lack of ground truth and unresponsiveness of organizations suspected of running tarpits makes determining degreaser’s false-positive rate difficult to calculate. Lab testing resulted in 100% accuracy, however, due to the numerous configurable options in existing tarpit software and the position, while other subnets are a mix of real and tarpitting IPs. Figure 8(a) shows the cumulative fraction of tarpit networks versus their occupancy for each of the fix tarpit types. We observe a variety of occupancy’s, with persistent LaBrea being the most highly occupied.

Figure 8(b) again shows the cumulative fraction of tarpit networks versus their occupancy, but broken down by the subnet size. Approximately half of all /24’s have an occupancy of 95% or greater, while more than 60% of the /22’s and /23’s have an occupancy of 95% or more. The occupancy’s of the six /16’s vary more widely; two of the /16’s are fully occupied (more than 99%), while the other four are between 15-30% occupied.

Next, we note that many tarpits answer all TCP ports. To better understand the port-specific behavior of the tarpits we discover, we probe all addresses within each tarpit subnet for TCP ports 80, 443, and 34343. TCP port 34343 is not assigned to any service, and therefore would not typically be expected to respond. Figure 8(c) shows the cumulative fraction of tarpit networks versus their occupancy as a function of port number. Unsurprisingly, we observe that by probing port 80 we find a higher occupancy than port 34343. Of note however, is that the difference between port 80 and 34343 is relatively small, suggesting that most of the tarpits we find answer for all ports.

Finally, with a substantial list of subnets running network tarpits, we sought out ground truth to further confirm that our detection methodology is accurate. Since we were unable to directly find confirmation on any of the subnet owner’s websites, we utilized Whois[3] records to make email contact. After waiting over two weeks, we had only received responses from two of the organizations we queried. One organization’s response was the creation of a “trouble ticket,” for which we never received further information. The second organization that responded was helpful and confirmed that they indeed were running LaBrea on the subnets in question. The lack of ground truth and unresponsiveness of organizations suspected of running tarpits makes determining degreaser’s false-positive rate difficult to calculate. Lab testing resulted in 100% accuracy, however, due to the numerous configurable options in existing tarpit software and the position, while other subnets are a mix of real and tarpitting IPs.
sibility of tarpits not based on the stock LaBrea software or the Netfilter TARPIT plugin, we can not claim perfect accuracy in the wild. We have not publicly disclosed our list of suspected tarpits due to security concerns, but we do encourage organizations or researchers interested in our work to contact the authors for access to our results.

4.3 Effect on Internet Scans

With several identified tarpitting subnets we explored how these tarpits were reflected across several different Internet scans [4, 2]. For scans that utilized ICMP-based approaches such as ping, we find these subnets appearing as fully occupied with responding hosts.

For example, Figure 9 shows screenshots of three sections of the IP address space as viewed from the ISI ANT Internet census browser [2]. The census browser visualizes subnet utilization by laying out subnets on a Hilbert Curve and then using a heatmap where increasingly bright green corresponds to proportionally higher occupancy. Dark regions indicate regions inferred to be empty, while red indicates negative replies. Figure 9(a) shows one of the /22 tarpit subnets we discover, within a region of otherwise high-occupancy. Figure 9(b) highlights a /16 aggregate where all of the green corresponds to the 58 of 256 tarpit /24 subnetworks within the larger aggregate. Last, Figure 9(c) shows a region of green surrounded by black – this green is a fake /20 tarpit-subnet within a larger aggregate that does not respond otherwise.

Next, we examine the relation between the data we extracted from the logs of Project Sonar’s HTTP scan from April 2014 [4] and our inferred tarpit subnets. To facilitate comparison, we broke each tarpit network into its respective /24’s. For each /24, Figure 10 evaluates the number of half-responding IP addresses from the HTTP scan (introduced in §3.3) versus the number of tarpit IP addresses we infer.

We observe two dominant characteristics of the density plot. First, we see close agreement between the inferred tarpit occupancy vs. Project Sonar half-responding count.
work tarpits to make them less easily distinguishable. Below we present a few recommendations to improve network state, tarpit applications are relatively easy to detect and require conforming implementations to handle this behavior. While violating the recommendations of the standard, this robustness requirements assures that our improved tarpit would continue to effectively trap hosts.

- **TCP Retransmissions**: One final improvement is in exploiting the nature of TCP retransmission to improve the stickiness of the tarpit. Since most TCP implementations will attempt retransmission at least three times before closing the connection, an improved tarpit would discard the first two packets, and wait for the third before responding. This incurs the cost of having to remember unacknowledged packets for each TCP connection, but can effectively slow the connection by several RTTs. The space overhead of remembering packets could be reduced by only storing a hash of the TCP header under the assumption that the host will not change the header during a retransmit.

6. CONCLUSIONS

In this work, we build degreas er, a tool to infer the presence of fake tarpitting hosts and networks. By probing at least one address in each /24 network in the Internet, degreas er uncovers more than 100 different tarpit subnetworks. Notably six of the tarpits are /16’s with two of the /16’s using all of their addresses to tarpit. Overall, we find over 215,000 active IP addresses that are fake in 29 countries and 77 autonomous systems.

While the size and extent of tarpits we discover on today’s Internet is small relative to the entire Internet, we are pleased that degreas er is able to discover these needles in a haystack. However, even small blocks of tarpit addresses can greatly slow automated scans as part of their intended capturing behavior.

Further, our results emphasize that cyber-deception is real and requires additional research attention. At present, it is unknown whether the deception we observe is security or policy related – for instance an attempt to influence address allocation policies. In general, it is an open question as to whether the use of deception is becoming more popular. We thus plan to run degreas er periodically in order to perform longitudinal study.

While we are confident that our classification of LaBrea and Netfilter-based tarpits are accurate, our scans reveal several other behaviors that are inconsistent with either type tarpit or real hosts. We encounter non-tarpitting hosts that accept TCP connections on well known ports, ACK data packets, but provide no application layer response (such as a HTTP Bad Request). Some of these hosts eventually terminate the connection using FINs, others do not. More exotic behaviors include networks that accept our connection attempts but after scanning several hosts, suddenly stop responding. A second attempt scanning the same network results in no successful connections, while scans from a different origin network are successful. The combined use of deception and temporal blacklisting warrants future study.

The effectiveness of tarpits is difficult to measure against the ever increasing range of attacks plaguing the Internet. This work has shown the simplicity in detecting tarpits and
we can only assume that as malicious scanning tools evolve, they will become more resilient to the effects of tarpits. Furthermore, we believe that our detection methodology could easily be incorporated into operating system TCP stacks in order to automatically skip tarpits, providing immunity to all network applications (abusive or otherwise).

Due to the negative effect these tarpitting subnets have on a variety of legitimate network scans, we suggest more explicit distribution of known-tarpits among whitehat communities in conjunction with using techniques developed in degreaser. Further, we presented avenues for future research into making tarpits more resilient to detecting and more believable to adversaries. We thus hope this work serves to raise awareness of a particular form of network deception that is popular in the wild, and its present-day implications.

Acknowledgments

We thank Mark Gondree, kc kayffy, John Heidemann, and our reviewers for feedback. Nathan Desso helped with early experiments and coined the degreaser moniker. This work supported in part by the Department of Homeland Security (DHS) under contract N66001-2250-58231 and by U.S. NSF grant CNS-1228994. Views and conclusions are those of the (DHS) under contract N66001-2250-58231 and by U.S. NSF supported in part by the Department of Homeland Security experiments and coined the degreaser moniker. This work our reviewers for feedback. Nathan Desso helped with early investigations.

7. REFERENCES


[33] USC/LANDER Project. Internet ipv4 address space census. PREdict ID USC-LANDER/internet_address_survey_it58w-20140122


