On Using Application-Layer Middlebox Protocols for Peeking Behind NAT Gateways

Teemu Rytilahti
Ruhr University Bochum
teemu.rytilahti@rub.de

Thorsten Holz
Ruhr University Bochum
thorsten.holz@rub.de

***Abstract*—Typical port scanning approaches do not achieve a full coverage of all devices connected to the Internet as not all devices are *directly* reachable via a public (IPv4) address: due to IP address space exhaustion, firewalls, and many other reasons, an end-to-end connectivity is not achieved in today's Internet anymore. Especially *Network Address Translation* (NAT) is widely deployed in practice and it has the side effect of "hiding" devices from being scanned. Some protocols, however, require end-to-end connectivity to function properly and hence several methods were developed in the past to enable crossing network borders.**

In this paper, we explore how an attacker can take advantage of such application-layer middlebox protocols to access devices located behind these gateways. More specifically, we investigate different methods for identifying such devices by using only *legitimate* protocol features. We categorize the available protocols into two classes: First, there are *persistent protocols* that are typically port-forwarding based. Such protocols are used to allow local network devices to open and forward external ports to them. Second, there are *non-persistent protocols* that are typically proxy-based to route packets between network edges, such as HTTP and SOCKS proxies. We perform a comprehensive, Internet-wide analysis to obtain an accurate overview of how prevalent and widespread such protocols are in practice. Our results indicate that hundreds of thousands of hosts are vulnerable for different types of attacks, e. g., we detect over 400,000 hosts that are likely vulnerable for attacks involving the UPnP IGD protocol. More worrisome, we find empirical evidence that attackers are already actively exploiting such protocols in the wild to access devices located behind NAT gateways. Amongst other findings, we discover that at least 24 % of all open Internet proxies are misconfigured to allow accessing hosts on non-routable addresses.

I. Introduction

Port scanning is a common phenomenon on the Internet. In 2007, Allman et al. [2] presented a 12-year-long view of Internet scanning based on data collected starting from 1994 until 2006. According to their analysis, the year 2001 marked the end of an era where the number of regular connection attempts dropped below those of scanners. *Nmap* [46] is probably the most popular port scanning tool, but it was not designed to perform high-speed scans over large portions of the Internet. As a result, tools such as *ZMap* [20] and *Masscan* [27] were developed. They take advantage of the steadily increasing available bandwidth and enable efficient and scalable Internet-wide scans. In 2014, Durumeric et al. [19] studied the network scanning activity with the help of a large darknet, concluding that both researchers (e. g. [39], [60]) and malicious actors (e. g., for locating vulnerable hosts for reflection attacks) are leveraging these tools for large horizontal scans of the Internet. Nowadays, searchable databases exist for exploring results from such scans such as *Censys* [36] and *Shodan* [63], allowing anyone to explore the scan results efficiently.

Unfortunately, typical port scanning approaches do not achieve full coverage of all devices connected to the Internet because they require a *direct* network connection to the to-be-scanned device. In practice, however, not all devices are directly reachable via a public (IPv4) address: due to IP address space exhaustion, firewalls, and many other reasons, an end-to-end connectivity is not achieved in today's Internet anymore. Mainly due to the rising number of connected devices in our times, the number of available IP addresses to give out has depleted, and this has become a problem in practice. To address this obstacle, different solutions have been developed, e. g., IP version 6 (IPv6) which uses 128 bits for addressing, but its adoption has been slow even though growing [34]. The Internet as a whole has not hurried to use IPv6—instead, Network Address Translation (NAT) is widely deployed in order to allow the expansion of the Internet to continue without moving to IPv6. Lately, Carrier-grade NATs (CGN), where this translation is done on the Internet Service Provider's (ISP) premises to allow bundling several clients behind a single IP address, are commonplace and increasing in numbers [45]. By assigning clients a non-publicly routable address space (see RFC 1918 [59]), a single Internet-facing device can be used as a gateway device for a multitude of clients. For example, in their study on the prevalence of middleboxes, Huang et al. [33] reported in 2017 that 7 % (or 695) of the autonomous systems they investigated using Luminati were evidently behind middleboxes.

The basic idea of NAT is to use other information besides IP addresses for routing by *translating* the address when the packet is being sent to the Internet. This can be done by saving the information obtained from outgoing communications, i. e., storing a tuple of source and destination addresses, respective ports, and the protocol (UDP, TCP, etc.) to allow identifying the connection later on. The device performing this translation keeps a collection of these tuples in a so-called *connection tracking table*. When receiving a packet destined to an external network from LAN, NAT will adjust the source address and port transparently to the sender and save it in the tracking table.

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24389
www.ndss-symposium.org

Upon receiving a response, it will check its connection tracking table to see if it is part of a known flow. If a match is found, it will rewrite the destination address and port accordingly to those values that were stored in the table earlier. As a side-effect, NAT also builds a feeling of security, as the assigned IP addresses of the internal clients are not routable and thus not directly accessible from the Internet. As a result, these devices are "hidden" from network scanners and cannot be probed with existing scanning tools.

In this paper, we explore different approaches to access networks that should be inaccessible by the fact that they are behind (NAT) gateways. More specifically, we investigate different methods for enumerating such devices by using legitimate features of protocols designed to cross network borders. Our approach bases on the insight that there are also downsides for mangling the connections via NAT/CGN. Most importantly, NAT breaks the end-to-end connectivity between devices. For example, some application layer protocols storing IP address information in their payloads, such as Session Initiation Protocol (SIP) commonly used for Voice over IP (VoIP), are affected by NAT as the address does not match anymore. Second, some protocols such as File Transfer Protocol (FTP) may use multiple ports for communication, requiring special handling to allow communication to those other ports. Third, and most importantly, NAT prevents incoming connections to the devices in the internal network if there is no indication where the packets are to be relayed, making hosting services behind them impossible without NAT traversal techniques.

To overcome these challenges, several application-layer middlebox protocols were developed and deployed on the Internet to enable crossing network borders. We show how these protocols can be utilized by an adversary to peek behind gateways. More specifically, we systematically explore how different types of protocols can be used to access devices behind a NAT gateway or relay traffic to external systems. For example, we show how the UPnP Internet Gateway Device (UPnP IGD) protocol can enable access to internal networks that would otherwise be out of reach for attackers. Generally speaking, we categorize the available protocols into two classes based on their temporal behavior. On the one hand, there are *persistent* protocols that are port-forwarding based: such NAT traversal protocols are used to allow local network devices to open and forward external ports to them, hence bypassing the restrictions of NAT. In practice, the three commonly used protocols are UPnP Internet Gateway Device (UPnP IGD), NAT Port Mapping Protocol (NAT-PMP), and its successor Port Control Protocol (PCP). On the other hand, the *non-persistent* proxy protocols allow merely relaying packets for the life-time of the proxy connection. Famous examples are HTTP and SOCKS proxies, where especially SOCKSv5 is much more powerful compared to HTTP proxies given that the protocol also allows UDP connections.

Throughout the paper, we assume the following threat model: we investigate how an *external* attacker can leverage these protocols to pivot to networks that would otherwise be inaccessible for them. This ability could be used to exploit unpatched devices deemed to be secure due to their lack of a public IP address. For NAT traversal protocols, the premise is that the normally only LAN-accessible control interfaces are mistakenly also open to the Internet. For proxies, where the expected functionality is to be open to the Internet, the premise is a misconfiguration of access control mechanisms. We emphasize that the described methods could also be exploited for further access by an attacker who already has a foothold into the network.

In a first step, we perform Internet-wide scans to study the prevalence of persistent protocols in the wild. Among other findings, our measurements discover over 400,000 (~15 %) of UPnP responsive hosts that are likely vulnerable to this class of attacks. Most interestingly, we find that over 60,000 hosts contain traces of attempts to misuse such protocols, which serves as empirical evidence that attackers are already actively exploiting endpoints behind the corresponding gateways. In addition, we provide a comprehensive analysis of the usage of NAT port-mapping with the help of UPnP's enumeration feature. Our analysis shows that these vulnerable gateways are still more commonly used for their designed purpose, e. g., to allow BitTorrent and chat software, such as WeChat or WhatsApp, to function. We also perform a brief measurement study on the existence of NAT-PMP/PCP, which is a competing standard for controlling port mappings. Our results indicate that although there exist Internet-exposed NAT-PMP/PCP endpoints, they do not seem to be vulnerable for protocol-conforming forward attacks like the UPnP endpoints we identified during our scans.

Second, we also study the non-persistent, temporary relay protocols HTTP and SOCKS, and investigate if open network proxies can also be used for similar malicious purposes. To this end, we scan the Internet on commonly used proxy ports and make requests on reachable open network proxies to understand their functionalities and potential misconfigurations. We find that merely 3 % of all Internet-exposed proxies are open proxies, while the majority of the closed ones are running outdated Squid instances located in just a few networks, hinting at ISP-wide installations. Among other findings, we show that 47 % of open SOCKSv4 proxies support DNS extension, that merely 9 % of SOCKSv5 proxies support IPv6, 10 % support UDP relaying, and that 76 % perform DNS resolving. Worse, we discover that 23 % of all open proxies (potentially up to 40 %) are misconfigured and can be abused by an attacker to access internal networks. In addition, by analyzing the responses from these misconfigured proxies, we show that they can be leveraged to access *internal* services such as router configuration pages and SSH, which are not otherwise externally accessible. We also discovered a large population of about 200,000 open HTTP proxies located on a single autonomous system of a large European ISP. We argue that even when open proxies are small in numbers on a global scale, many users are typically located behind such proxies (e. g., in corporate environments) and they could be exploited via such misconfigurations.

In summary, we make the following key contributions:

1) We explore different methods that can be used for scanning internal networks via protocols that enable us to connect devices across network borders. We divide these protocols into two types: *persistent* NAT traversal protocols such as UPnP IGD and NAT-PMP/PCP, where the target changes the routing behavior, and *non-persistent* protocols such as

HTTP/SOCKS proxies, where the target acts as an intermediary.

2) We perform extensive, Internet-wide measurements to provide a comprehensive overview of hosts on the Internet implementing these protocols. Based on this empirical data, we analyze the potential attack surface while taking both ethical and legal considerations into account. Amongst other results, we find empirical evidence that attackers are already actively exploiting the techniques studied by us in the wild.
3) In contrast to previous studies, we provide a more holistic and comprehensive view of the proxy ecosystem by also analyzing the non-open proxies to the extent possible.

This paper is structured into several main sections that each contains a preface describing its intent, and where suitable, we separate the measurement approach, evaluation, and our key findings inside sections for clarity. After these three main sections, we discuss related works (Section V), reflect on ethical considerations and some limitations of our study (Section VI), and conclude in Section VII with a summary.

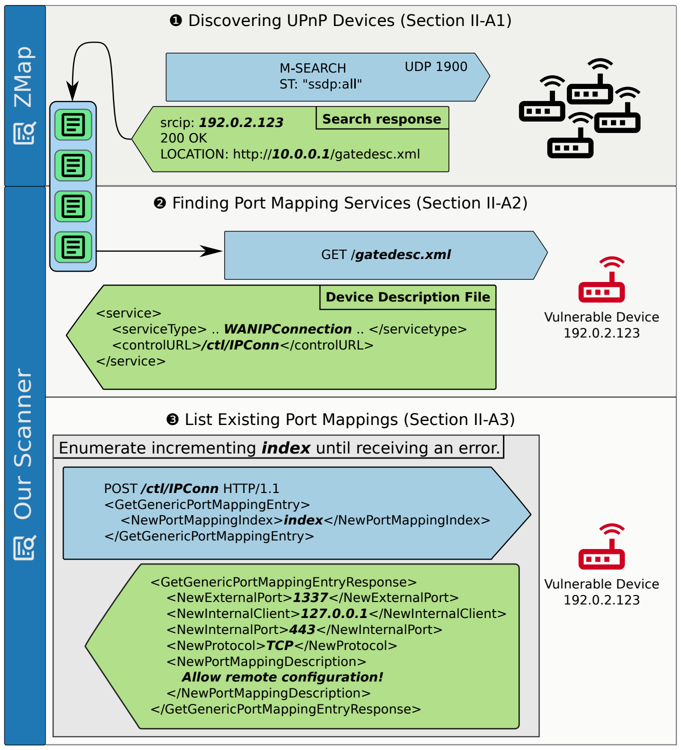
II. UPnP Internet Gateway Device

Universal Plug and Play (UPnP) is a marketing term used for a set of protocols which aim at enabling consumer devices to discover and control other UPnP-enabled devices effortlessly [54]. In practice, UPnP is typically used in the context of home-entertainment systems for media streaming and playback controlling. In the context of this paper, we are interested in the UPnP Internet Gateway Device (IGD) profile, which is a suite of UPnP services for configuring gateway devices. The offered capabilities vary by implementations, but commonly exposed services include capabilities for querying the state of external connectivity, controlling potential integrated services (such as DHCP), and controlling port mappings.

In this section, we investigate the port mapping control functionalities of the UPnP IGD profile, especially concentrating on how external actors could misuse this profile to insert new port forwards to access otherwise inaccessible networks. Although we concentrate on how this functionality can be misused by an external attacker due to the endpoints being exposed to the Internet (while they should only be available on LAN interfaces), we emphasize that the same functionality can also be misused by an adversary who has already compromised a device on the network. However, as measuring this is a hard task, we perform Internet-wide scans to find hosts which expose this interface to the WAN interface and enumerate over the existing port forwards to obtain insights on how this feature is used for both benign and malicious uses.

A. Measurement Approach

In the scope of our work, we are only interested in existing port forwards on those devices. Obtaining that information is a three-step process, as illustrated in Figure 1: (i) discovering UPnP devices by sending discovery requests with a portscanning tool such as ZMap (Section II-A1), (ii) downloading the *device description file* from responsive hosts to see if they are exposing the services of our interest (Section II-A2), and finally (iii) enumerating over existing port forwards (Section II-A3). In the following, we describe each step in detail.


ZMap
Our Scanner
❶ Discovering UPnP Devices (Section II-A1)
UDP 1900
Search response
❷ Finding Port Mapping Services (Section II-A2)
Device Description File
Vulnerable Device
192.0.2.123
❸ List Existing Port Mappings (Section II-A3)
Enumerate incrementing **index** until receiving an error.
Vulnerable Device
192.0.2.123

```
M-SEARCH
ST: "ssdp:all"
```

```
srcip: 192.0.2.123
200 OK
LOCATION: http://10.0.0.1/gatedesc.xml
```

```
GET /gatedesc.xml
```

```
<service>
  <serviceType> .. WANIPConnection .. </servicetype>
  <controlURL>/ctl/IPConn</controlURL>
</service>
```

```
POST /ctl/IPConn HTTP/1.1
<GetGenericPortMappingEntry>
  <NewPortMappingIndex>index</NewPortMappingIndex>
</GetGenericPortMappingEntry>
```

```
<GetGenericPortMappingEntryResponse>
  <NewExternalPort>1337</NewExternalPort>
  <NewInternalClient>127.0.0.1</NewInternalClient>
  <NewInternalPort>443</NewInternalPort>
  <NewProtocol>TCP</NewProtocol>
  <NewPortMappingDescription>
    Allow remote configuration!
  </NewPortMappingDescription>
</GetGenericPortMappingEntryResponse>
```

Fig. 1. UPnP scanning method: ❶ We start by scanning with ZMap for SSDP responsive hosts and feeding the results to our scanning system (Sec. II-A1). ❷ We extract the location of the *device description file* from responses and try to fetch it (Sec. II-A2). ❸ We enumerate the existing port forwardings from responsive hosts and save results to a database for analyses. (Sec. II-A3)

1) Discovering UPnP Devices: Although the Simple Service Discovery Protocol (SSDP) uses HTTP-like requests on the UDP multicast group 239.255.255.250 with port 1900 for UPnP discovery, the architecture specification mandates that all devices shall also listen for unicast search messages [54]. The discovery begins when the requesting party sends a discovery request containing a *Search Target* (ST) header indicating which types of services it is looking for. All matching devices supporting the searched service shall send a unicast reply to the requester, one for each matching service in case there exist multiple matching services. In our example (see the first step in Figure 1, or Listing 1 in Appendix A for more details), we use the wildcard target `"ssdp:all"` to elicit responses about all available UPnP services on all devices receiving the request. As a side note: this feature is also the very same that is widely used for DDoS attacks [60].

We used ZMap's UPnP probe payload and saved the results into a list in a Redis database. The results were read simultaneously from the list by our scanner to avoid potential IP churn. We had to modify ZMap because the stock version captures only responses with the source port 1900 (same as destination), and we found that this significantly underestimates the actual number of UPnP hosts on the Internet.

2) Finding Port Mapping Services: The first step our crawler takes is extracting the location of the *device description file* (contained in the `LOCATION` header of the discovery response, see the search response in Figure 1), replacing the (potentially internal, 10.0.0.1 in the example) IP address with the source of the SSDP reply (192.0.2.123 in the example) and finally trying to fetch the file (step 2 in Figure 1). This XML file contains general information about the device (including its name, manufacturer, serial number, etc.), a list of exposed devices, and the services that are currently being offered. An abbreviated example is available in Listing 2 in Appendix A. In this paper, we are interested in interfaces implementing any version of the `WANIPConnection` or the `WANPPPConnection` service (later `WAN*Connection`). Each service entry element contains the location of a *service description file* containing, e. g., what *actions* are provided, and what their parameters and return values are. As we work on UPnP-standardized services, we do not need to parse the service description files, but simply mark down the SOAP service endpoints (`controlURL`) necessary for our next step.

3) Listing Existing Port Mappings: Invoking a UPnP action happens by sending a specifically crafted, SOAP-formed HTTP POST request to the service endpoint. This request contains a `SoapAction` header describing the action to execute and its body is an XML-encoded SOAP document containing the parameters specific to the action.

There are currently two versions of the `WAN*Connection` service which support different sets of actions for obtaining existing port mappings. The more widely supported version 1 exposes an index-based `GetGenericPortMappingEntry`, which allows enumerating the existing mappings by executing the call with incremented index until an error is received. Version 2 introduces `GetListOfPortMappings` to query all available port mappings without enumeration. Based on our preliminary investigation on SSDP responses from vulnerable devices, this is not widely available in practice. As the versions are backward-compatible, we use the former in the remainder of this paper.

To accommodate for potential sparseness in forward lists (e. g., due to removal of forwards in-between), we continue iterating up to five times after receiving the first SOAP error. The response contains the forwarded port as well as the target host, port, used protocol (TCP/UDP), and a description, among other information. A condensed example of the response is shown in the bottom-most response in Figure 1. Note that the values found in the responses mirror the ones for `AddGenericPortMapping` calls used to create port mappings and which resides under the same service endpoint (i.e., we assume is, that they share the same access controls).

B. Evaluation

We performed an Internet-wide scan in January 2019 with a patched version of the ZMap scanner [20]. The total runtime of the whole, Internet-wide crawling process was approximately 12 hours, including the ZMap scan. A summary of the results can be seen in Table I. For readibility, we round the numbers in text and refer our readers to the table for exact numbers. In the following, we use the terms "port mapping" and "(port) forward" interchangeably.

1) Responsive UPnP Hosts: Our ZMap scan received responses from ~2,800,000, from which the majority (66 %) would have been ignored by the standard ZMap due to its port filter (we saw replies from 44,075 distinct source ports). We contacted the ZMap developers and submitted a patch to address this problem, the patch was accepted and merged. We

TABLE I. RESPONSIVE UPNP HOSTS

	Total	**WAN*Connection**	**Vulnerable**†
SSDP responses	2,789,823	480,563	408,080
From port 1900	957,031	221,176	192,759
From other ports ††	1,832,792	259,387	215,321
Countries	223	200	195
Unique AS#	11,984	5,256	4,381
Exposed HTTP endpoints ‡	1,067,035	480,563	408,080
Manufacturers	1,545	677	237
Models	15,038	2,125	716
Implementations	28,963	749	445
WANIPConnection	393,775	393,775	326,312
WANPPPConnection	100,063	100,063	88,244
With forwards	130,899	130,899	118,556
Internal	127,133	127,133	114,790
External	17,704	17,704	17,704

† As defined in Section II-B5.
‡ Endpoints that allowed downloading the service description file (Section II-A2).
†† Hosts undiscoverable with vanilla ZMap.

found that more than half of the hosts were from Asia: 19 % from South Korea, followed by China (15 %), Taiwan (8 %), Vietnam (7 %), and Japan (5 %). The remaining 46 % were spread over 209 other countries.

2) Hosts With Port Mapping Service: From these hosts, 38 % (almost 1,100,000) exposed their HTTP endpoint, i. e., we were able to fetch the corresponding device description file. 45 % (~480,000) of hosts had either `WANIPConnection` (~394,000), `WANPPPConnection` (~100,000), or both of them (~13,000) exposed. The majority of these devices (almost 350,000) had no active forwards. However, 86 % of them returned a SOAP error indicating that they support the interface, but that there are no active forwards. The remaining devices represent various errors caused by network outages or protocol handling differences.

3) Existing Port Mappings: We now concentrate only on the endpoints having forwards, totaling 27 % of all `WAN*Connection` exposing endpoints with a total of almost 5,000,000 forwards, covering almost all possible ports. After filtering out non-active forwards (either by being explicitly disabled, or not having information about the ports and hosts) and removing duplicates (where the forwards share the same description as well as the target host and port), we are left with almost 3,300,000 forwards.

For simplicity, we categorize the forwards into three groups: (a) "Galleta Silenciosa" forwards targeting mainly TCP ports 139 and 445, (b) forwards targeting Internet-routable IP addresses, and (c) innocuous port mappings. We note that the two former groups are not mutually exclusive, whereas the innocuous group contains all forwards *not contained* in those two. This does not necessarily mean that all forwards in this group are definitely benign. Table II provides an overview of numbers of unique hosts, forwards, and locations for each of these categories. Table III on page 6 provides a closer look into the most frequently seen descriptions, internal ports, and targeted subnets (we summarize IP addresses to the corresponding `/24` subnets for readability). For clarity, we also clean the descriptions by removing unique identifiers (e.g., WhatsApp (⟨ID⟩) to WhatsApp, and group descriptions consisting only of `host:port` under "IP+port match" if there is a match.

TABLE II. UPnP GATEWAYS WITH PORT FORWARDS

	Total	External	Galleta Silenciosa	Innocuous
Hosts	130,899	17,704	42,401	113,388
With internal forwards	127,133	0	39,354	113,388
With external forwards	17,704	17,704	10,425	0
Countries	183	90	116	180
AS#	3,340	593	1,067	3,187
Forwards	3,265,311	586,471	1,625,220	1,080,607
Internal forwards	2,678,840	0	1,598,233	1,080,607
UDP forwards	525,369	0	0	525,369
TCP forwards	2,147,450	0	1,598,233	549,217
External forwards	586,471	586,471	26,987	0
UDP forwards	7,410	7,410	0	0
TCP forwards	579,000	579,000	26,987	0
Target hosts	62,535	31,984	39,603	15,678
Target subnets	16,963	15,052	7,609	1,816
Target ports	64,600	11,942	8	64,492
Descriptions	407,028	13,464	1	393,888
Cleaned descriptions	35,479	3,251	1	32,486
SSDP				
SSDP source ports	15,027	225	676	14,839
HTTP endpoint ports	12,431	1,323	3,883	11,379
SSDP implementations	399	149	160	363
Manufacturers	471	145	197	460
Models	1,468	423	609	1,411

a) Galleta Silenciosa – Silent Cookie: Our first category consists of hosts with forwards with the exact description "galleta silenciosa" (Spanish: "silent cookie"), accounting for up to 37 % of all existing forwards. In total, 32 % (~42,000 from all hosts had these forwards, predominantly targeting TCP ports 139 and 445 (87 %) across all private, RFC1918 address ranges. In total, we found ~1,600,000 forwards to ~22,000 unique IP addresses on ~7,600 different /24 subnets. A closer analysis revealed that this observation indicates that actual attacks are performed by adversaries in the wild that exploit UPnP to reach hosts behind a NAT gateway. The most common 20 subnets had forwards to *each* of the hosts in these subnets, while the top 100 had a median of 201 hosts being targeted, which indicates that an attacker has aimed to locate endpoints behind these NAT gateways. These forwards target primarily the network indicated in the Location header of the discovery response. Although these mappings were seen on hosts from almost 700 different SSDP response ports, we emphasize that 97 % of them were from devices responding to discovery from port 1900. The device description files were predominantly hosted on ports 2048 and 5431 (up to 62 %), the rest were seen in arbitrary ports and non-standard file paths. This indicates a more sophisticated approach from the attacker than simply trying to use the commonly used endpoints.

Independently from us, Seaman also detected these port mappings (or "NAT injections") [62], which confirms that we are actually observing attackers exploiting this attack vector in practice. Seaman speculated that the goal of the attacker has been to exploit SMB implementations vulnerable to the famous SMB vulnerabilities *EternalBlue* and *EternalRed*. Concurrently to our work, a security company identified a malware family using the hardcoded string "galleta silenciosa" for its port forwards [15]. Whether this is the origin of these forwards or just a copycat remains unknown to us.

b) External Port Mappings: The second category of malicious port forwards involves forwarding to *external* hosts. In total, we detected over 18,000 gateway devices (14 % from all forward-having devices) containing such forwards targeting on 32,000 different IP addresses. More specifically, the most commonly seen descriptions are "galleta silenciosa" (~27,000 forwards on ~10,000 hosts), "MONITOR" (~530,000 on 5,100 hosts), and "node:nat:upnp"[1] (over 1,800 on ~1,400 hosts), which seem to be of malicious nature and used by different actors based on their usage. Whereas the external "galleta" injections mostly link to other routers in this same group of gateways, the "node:nat:upnp" ones are pointing to DNS ports, while the "MONITOR" forwards' target addresses that are not contained in our data set. These forwards are mostly on HTTP(S) ports on 11,239 unique IP addresses, of which ~7,000 had reverse DNS records indicating various advertisement networks, VPS providers, and CDNs as targets, indicating their potential use for domain fronting as indicated by Akamai's research. In total, we observed 205 different 2nd-level domains under 29 top-level domains.

Given the variety of malicious activities we observed related to these forwards, we speculate that this activity belongs to different kinds of actors. This observation also corresponds to the interpretation by researchers from Akamai [1], who speculate that these injections are part of a botnet linked to the "Inception Framework" identified by Symantec [67]. Again, these observations demonstrate that attackers are actively misusing such protocols to peek behind NAT gateways.

c) Innocuous Port Mappings: Our last group consists of non-malicious port mappings, which were not included in the two previous groups described above. This group contains the majority of the devices with forwards, totaling ~114,000 hosts with over a million of non-duplicate forwards. As shown in Table II, these one million forwards divide equally to UDP and TCP forwards.

In total, ~25,000 hosts in this group contained forwards to privileged ports including HTTP(S) ports (port 80 on ~11,000 hosts, port 443 on over 6,000), with appearances of telnet (660 hosts), SSH (1,216), SMTP (611), and FTP (1,238). On average, gateways in this group had three internal clients with a total of 8 forwards. The most common internal networks were unsurprisingly 192.168.0.0/24 and 192.168.1.0/24 (seen in Table III). The most common, legitimate applications were BitTorrent and chat programs (e. g., WhatsApp or WeChat) on 20 % of devices, and gaming (e.g., UDP 9308 is apparently used for PlayStation multiplayer games).

4) Forwards on Port 0 and Broadcasts Addresses: The widespread occurrence of port 0 on both innocuous and external mappings is a mystery—the specification states that it is not allowed as forwarding target and shall only be used as a wildcard to indicate that *all* unmapped ports are to be forwarded to the specified client (seen on almost 500 gateways for external and on ~3,100 for internal target addresses). This feature could potentially be misused by an attacker to capture all potential traffic, assuming the functionality is implemented as it is specified in the standard. The innocuous forwards on 255.255.255.255 are likewise odd—in total, almost 1,300 hosts had one for TCP (most with internal port 44382) and over 2,100 for UDP. We could not find a reason for the TCP forwards (all but three had description "miniupnpd"), nor should they be valid forwards. Note that the UPnP specification explicitly states that UDP mappings must be supported for

[1]The default value for the Node.js UPnP library, https://github.com/indutny/node-nat-upnp

TABLE III. MOST COMMON UPNP FORWARD DESCRIPTIONS, INTERNAL TARGET PORTS, AND SUBNETS PER CATEGORY. THE PERCENTAGE INDICATES THE AMOUNT OF RESPONSIVE HOSTS IN THE CATEGORY HAVING THE DESCRIBED FORWARD.

Galleta Silenciosa (TCP)		**External Forwards TCP**		**External Forwards UDP**		**Innocuous TCP**		**Innocuous UDP**		**Everything (any)**	
Forward Descriptions											
galleta silenciosa	100.0 %	galleta silenciosa	63.01 %	node:nat:upnp	50.79 %	uTorrent	15.97 %	WhatsApp-prefixed	16.49 %	galleta silenciosa	32.33 %
–	–	MONITOR	30.76 %	miniupnpd	25.75 %	libtorrent	11.85 %	libtorrent	13.21 %	libtorrent	10.21 %
–	–	proxy	5.37 %	HCDN	4.37 %	Network	11.68 %	wechat voip	11.95 %	WhatsApp-prefixed	10.09 %
–	–	Skype-prefixed	3.32 %	WhatsApp-prefixed	4.01 %	PiXel	10.44 %	IP+port match	11.45 %	uTorrent	9.63 %
–	–	HTTP_FORWARD	2.7 %	Teredo	3.96 %	miniupnpd	7.21 %	HCDN	10.47 %	IP+port match	7.86 %
Unique values	1	Unique values	3,019	Unique values	277	Unique values	27,195	Unique values	4,652	Unique values	35,490
Internal Ports											
139	76.23 %	80	68.83 %	53	51.45 %	80	14.31 %	9308	10.82 %	139	25.01 %
445	72.55 %	0	22.14 %	8290	2.59 %	4433	11.32 %	6881	10.49 %	445	23.86 %
80	18.25 %	443	21.14 %	6881	1.78 %	8195	8.54 %	0	6.94 %	80	16.16 %
0	13.95 %	8100	12.59 %	4194	1.68 %	65003	8.39 %	19132	5.88 %	6881	8.15 %
443	0.07 %	4450	5.36 %	0	1.17 %	65004	8.23 %	3027	3.52 %	0	7.55 %
Unique values	8	Unique values	7,357	Unique values	5,246	Unique values	60,159	Unique values	60,822	Unique values	64,600
Target Subnets											
192.168.0.0/24	54.71 %	182.161.73.0/24	30.38 %	199.217.119.0/24	44.95 %	192.168.1.0/24	43.91 %	192.168.1.0/24	49.43 %	192.168.1.0/24	46.5 %
192.168.1.0/24	25.66 %	43.227.116.0/24	28.35 %	200.0.0.0/24	23.67 %	192.168.0.0/24	42.41 %	192.168.0.0/24	36.69 %	192.168.0.0/24	36.0 %
192.168.10.0/24	2.84 %	172.217.25.0/24	28.28 %	82.163.142.0/24	5.99 %	192.168.2.0/24	2.27 %	192.168.2.0/24	3.02 %	182.161.73.0/24	3.83 %
10.0.0.0/24	1.08 %	216.58.197.0/24	28.15 %	144.1.0.0/24	3.3 %	192.168.10.0/24	1.89 %	255.255.255.0/24	2.69 %	43.227.116.0/24	3.58 %
192.168.9.0/24	0.87 %	183.111.131.0/24	26.95 %	88.2.0.0/24	3.1 %	255.255.255.0/24	1.72 %	10.0.0.0/24	2.55 %	172.217.25.0/24	3.57 %
Unique values	7,609	Unique values	14,651	Unique values	614	Unique values	1,167	Unique values	1,235	Unique values	16,963

broadcast purposes. The most commonly seen broadcast user was an obscure peer-to-peer video streaming solution with the description "HCDN" (on almost 1,800 hosts).

5) Estimating the Number of Vulnerable Hosts: While trying to understand why some of these hosts are left unabused without trying to add our own mappings on them, we developed a way to estimate the number of vulnerable devices based on what we know about the already abused ones. As it turned out that 98 % of abused gateways were responding to discovery using the source port 1900, we speculated that the attackers were misled by the same ZMap issue as we did in the early stages of our research. However, it does not seem to be so clear as there exists a population of almost 200,000 routers using that port with no forwards whatsoever, so unfortunately we have no conclusive answer to this observation.

Therefore, instead of estimating the number of vulnerable hosts based on the availability of `WAN*Connection` interfaces (the middle column in Table I on page 4), we decided to use the information from known-to-be-vulnerable hosts (i. e., hosts from the first two groups). To this end, we create a tuple of $\langle manufacturer, modelname, modelnumber \rangle$ and we consider all matching devices *definitely* vulnerable (shown in the rightmost column in Table I). Based on this, ~480,000 SOAP endpoint exposing devices are potentially vulnerable, while 86 % of them are definitely vulnerable.

C. UPnP Internet Gateway Device Honeypot

Based on the empirical results presented in the previous section, we were also interested in tracking abuse not only through active analysis via Internet-wide scans, but also developed a honeypot to study potential attacks. More specifically, we developed a honeypot that implements both `GetGenericPortMappingEntry` (allowing enumeration up to five items) and `AddPortMapping` (responding successes) and use our responses from a real, vulnerable device. Knowing that SSDP is used as a source for DDoS attacks, we limited the number of responses to two per requester in an hour to prevent abuse. Furthermore, we modified the location of the device description file in order to ascertain whether the potential attackers parse the replies instead of leveraging commonly used locations. We set our SOAP endpoints to listen on tens of the most commonly used ports and answering only to requests (and device description requests) on tens of known paths or path prefixes based on data we collected for previous analyses. We make the source code of our honeypot available at https://github.com/RUB-SysSec/MiddleboxProtocolStudy/ to foster further research on this topic.

We analyzed the saved interactions between December 9, 2018 and February 3, 2019. In total, we observed 791 HTTP requests from 52 distinct IP addresses, while 821 IP addresses sent a discovery request. Of those, 33 addresses requested the actions we were interested in 299 times. Two scanner instances (one using a single IP address, another distributing the scan to happen from tens of addresses) enumerated through the port mappings after fetching the description file from our non-standard location, which we categorize as research scanners based on their behavior although neither of them announced themselves as such directly. The first one used a single IP address from Hongkong, which crawled first based on the SSDP response, but extended later to check for other common device description locations. The second instance was an orchestrated, simultaneous scanning using 22 IP addresses from China which tried to fetch the first 45 forwards while ignoring our error responses (with user-agent "Firefox 5.0").

We also observed 15 calls to adding a port mapping (from ten different networks and countries), but a closer inspection revealed that these were all targeting the same port in hopes to exploit a command injection vulnerability[2] in one of the input fields. With this, we conclude that either we were unlucky or that malicious activities were not active during our observation period. We have kept our honeypot running also after the observation period described here, but we have not noticed any malicious activities involving insertion of forwards.

[2]Looks like a variant of CVE-2014-8361 as analyzed by Montonen [50]

D. UPnP IGD: Key Findings

In conclusion, our research shows that 30 % of all SSDP responsive hosts expose also their control endpoints, potentially allowing different kinds of misuses in practice. We were able to enumerate port forwards from about 130,000 hosts, while finding that ~480,000 hosts expose their port forwarding controls to the Internet. Our analysis indicates that malicious actors have used this feature for either scanning hosts behind these NAT gateways, or as jump hosts to forward their traffic over the Internet to masquerade their activities. Most commonly seen innocuous uses were use of Bittorrent or VoIP softwares, with the majority of hosts using either 192.168.0.0/24 or 192.168.1.0/24 as their internal network. During our investigation, we also found out that ZMap's response filtering misses a large fraction of SSDP responsive hosts, making them invisible for perpetrators using the tool for collecting potential SSDP amplifiers for DDoS attacks.

III. NAT-PMP and PCP

NAT Port Mapping Protocol (NAT-PMP) is an IGD competitor, first introduced and defined by Cheshire et al. [12] in 2005 while working for Apple. Support for it was first shipped with OS X 10.4 released earlier that year. The draft got updated several times, until an informational RFC 6886 [11] was released (posthumously) in 2013 alongside its IETF-standardized successor, Port Control Protocol (PCP, RFC 6887 [71]).

NAT-PMP uses a simple UDP-based binary protocol on port 5351, supporting only three operations: (i) `ANNOUNCE` for determining the external address of the NAT gateway and for server to client signaling, (ii) `Map UDP`, and (iii) `Map TCP` for requesting forwarding. The client requests a port mapping from the NAT gateway by sending one of the mapping requests containing the port where to forward the traffic to, the *suggested* external port (the gateway will choose the port it maps for the client), and the wanted lifetime of the mapping. It must be emphasized that in contrast to IGD, it is not possible to explicitly state the forwarding destination, but the source address of the request serves as that.

Port Control Protocol (PCP) is the successor of NAT-PMP using the same port, a compatible packet format, and similar operational semantics. The protocol was extended to support IPv6, the management of outbound mappings (PEER opcode), and more. PCP was designed from the start to be extendable by having a simple base header followed by an opcode-specific payload and potential options. An illustration of a complete PCP request with MAP opcode and THIRD_PARTY option to request a mapping 10.0.23.221:12345 $\Rightarrow$ 127.0.0.1:1024 is shown in Figure 2. For the `ANNOUNCE` command, only the base header is necessary.

A. Measurement Approach

In comparison to UPnP IGD, neither NAT-PMP nor PCP offer functionality to enumerate over existing port mappings. Instead, we can only try to verify the existence of misconfigured implementations with the carefully planned Internet-wide scans. Our NAT-PMP/PCP scans involve three different payloads to gain an understanding of whether similar attacks as for UPnP IGD are possible. As with UPnP IGD, we carefully crafted the payloads and tested them against the software

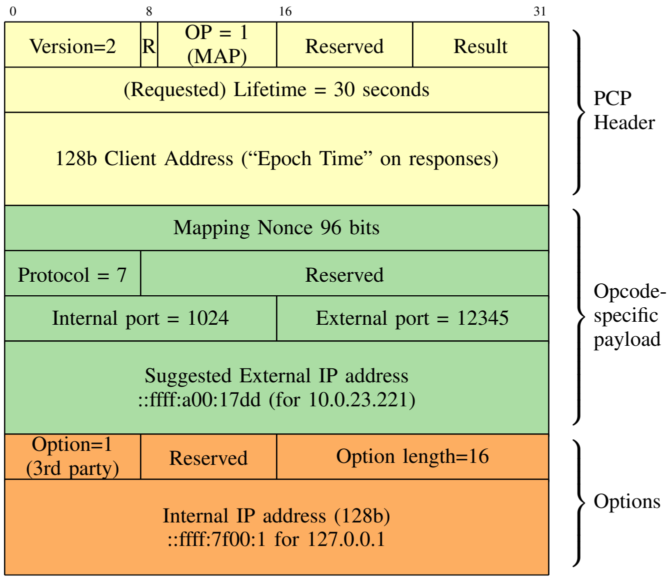


Fig. 2. PCP request with the common header (top, yellow), MAP payload (middle, green) and the 3rd party option (bottom, orange).

implementation *miniupnpd* in our lab environment in different configurations to make sure they were working as expected and did not cause any unintended side effects.

1) Discovering PCP Servers: For the initial host discovery, we leverage the backward compatibility of PCP to NAT-PMP's packet structure, meaning that we can do a single query to discover servers supporting either of the protocols by sending a single PCP `ANNOUNCE` request. For this, we prepare a request containing only the PCP base header (Figure 2, the topmost part) including our IP address as the requester. We set the version field to "2" (PCP) and the lifetime to "0", and run a scan on port 5351 on the whole Internet with ZMap [20].

Correctly configured PCP servers should silently drop these packets as they are not arriving from the internal network. If that does not happen and the server processes the packet correctly, we expect to receive `SUCCESS` in the result field with the epoch field filled with device's current uptime. If the host supports only NAT-PMP or a vendor-specific implementation (version 1 in the payload), an `UNSUPP_VERSION` [71, "Version Negotiation"] response is expected.

2) Checking for 3rd party option: Our second check is used to understand how many of those PCP supporting servers would support the THIRD_PARTY option for creating arbitrary forwards. To this end, we sent a specifically crafted MAP request to create a forward on the very same IP address sending out these requests, i. e., our scanner server. If the option is not supported at all by the PCP server, it should respond with a `UNSUPP_OPTION` result code indicating that fact. Recalling back to the implicit forward destination, the RFC mandates that the third-party forward target has to be different from the source address, and violations must be reported back with an `MALFORMED_REQ` error.

a) Testing for Potential Vulnerabilities: Our last check is also done on the same PCP-supporting population, which we use to verify if the server allows the creation of forwards. While knowing that this may affect the routing of

TABLE IV. RESULTS FROM PCP SCANS

	ANNOUNCE		MAP THIRD_PARTY	
	NAT-PMP	PCP	Invalid IP	127.0.0.1
ADDRESS_MISMATCH	–	4	–	–
MALFORMED_OPTION	–	–	2	2
MALFORMED_REQ	–	–	31,046	–
NETWORK_FAILURE	–	9	8	9
NOT_AUTHORIZED	–	70,437	423	45,892
SUCCESS	114	5	–	–
UNSUPP_OPCODE	–	1	–	–
UNSUPP_VERSION	554,487	–	–	–
Total	554,601	70,456	31,479	45,903

misconfigured devices, we want to emphasize that we treated the topic carefully by following the RFC-recommended way for obtaining the external IP address [71, Section 11.6]. This involves requesting a short-lived mapping for obtaining the bound external IP address, which in complex setups cannot be known before the mapping has already been made. As such, we sent a legitimate request to these devices that follows the standard. For the actual measurement, we chose the unlikely used TCP port 9 (discard) with a short-lived lifetime (in our case one second) as instructed in the RFC. Again, we extensively tested this request in our lab environment in different configurations to make sure that it does not create any unintended side effects.

B. Evaluation

1) Responsive PCP Endpoints: Our Internet-wide scans revealed a total of 625,057 exposed endpoints. The results (shown in Table IV) show that a majority of 89 % of all responding servers supported only NAT-PMP by responding with `UNSUPP_VERSION`. Almost all PCP-supporting ("version 2") servers furthermore responded with `NOT_AUTHORIZED`, indicating that this feature is either disabled or that we are not allowed to access it.

2) Checking for THIRD_PARTY Support: The results (see Table IV, rightmost column) indicate that some 31,000 servers reported back as expected, and to our surprise, there were *no* servers not supporting this option or they responded with a different error code, or simply silently ignored our request. The lower number of responsive hosts is at least partially due to IP address churn, as the follow-up scans did take place one day after the initial `ANNOUNCE` scan.

3) Creating a Port Mapping: This scan resulted in similar observations as during our initial `ANNOUNCE` scan: Most of the services reported that we are not authorized to perform this action, concluding that the PCP server population seems to be securely configured against this type of attack. Nevertheless, we want to note that these over 600,000 hosts should not be responsive to our probes in the first place.

C. Hijacking Internal Traffic

We found that also a small population of misconfigured NAT-PMP enabled routers exist which report an internal IP address as their external IP address. First reported by Hart [31] in 2014, this confusion may allow creating mappings which cause traffic destined *to* the router's given port to whoever creates the mapping. Although not as bad as having the ability to allow arbitrary port mappings, this may still allow hijacking traffic (e. g., DNS queries) destined to the router. To analyze this aspect, we downloaded recent scan results from Rapid7's NAT-PMP scan and found that 1,3 % of NAT-PMP supporting devices (out of about 480,000) were still reporting an RFC1918 IP address as their external address.

D. NAT-PMP/PCP Key Findings

In conclusion, our findings can be summarized as follows: while we received responses from several hundred thousands of NAT-PMP hosts that should not be exposed to the Internet, we could not confirm that these could be misused for accessing internal hosts, as was the case with UPnP. Only a fraction of these hosts supported the newer PCP, indicating to a better security posture of newer installations. Therefore, we hope that this part of our paper will help raise knowledge of this rather obscure protocol and safer configurations that do not expose these devices at all will be deployed in the future.

IV. NETWORK PROXIES

After having extensively studied NAT traversal protocols, we now focus on network proxies as a complimentary example of application-layer middlebox protocols given that such proxies are typically used to route packets between network edges. In contrast to the previously discussed protocols, which form *permanent* port mappings by changing the routing tables to allow external connections to hosts behind NAT, proxies act as *temporal* conduits between the client and its targets, passing messages in between. Hence proxies are more often used to control access to *external* networks, e. g., for blocking access to unwanted websites or filtering malicious content. Although the importance of network proxies has decreased due to Tor [17] and cheap VPN solutions, there are still many open network proxies which can for example be used to bypass geo-blocking by a simple configuration change.

In this section, we cover two types of network proxies in detail: HTTP proxies (Section IV-A) and SOCKS proxies (Section IV-B). After introducing these protocols, we describe our measurement approach in Section IV-C. We first focus on finding proxy candidates and how we check if they are proxies, followed by checking if they are vulnerable for allowing access to internal networks. We evaluate our findings on the proxy ecosystem in Section IV-D, which is followed by our findings on services hosted on internal networks of vulnerable proxy systems and finally complement our Internet-wide scans by crawling for Internet proxies.

A. HTTP Proxies

HTTP proxy servers act as an intermediary between the client and the target server, and there are two ways defined in RFC 2616 [21] to do that:

(i) Using an absolute URI, where the client requests the full URI instead of the path (i.e., requesting `GET http://localhost/ HTTP/1.1`). In this case, the proxy acts as an intermediary by conversing HTTP with both participants. The HTTP 1.1 standard mandates that even non-proxying implementations must accept this absolute addressing form for future compatibility.

(ii) Using the more powerful CONNECT method [37], [4], in which the proxy acts merely as a conduit between the endpoints, allowing non-HTTP protocols (e. g., TLS for HTTPS, or SSH [56]) to be tunneled through it. In response to connect requests, the proxy either responds with a "200" status indicating that the tunnel has been established, or an error message (such as "407 Proxy Authentication Required").

The standard HTTP error codes [23] are used with both methods. For brevity and the wider applicability (i. e., not being limited to HTTP requests), we concentrate on CONNECT-supporting proxies, if not noted otherwise.

B. SOCKS Proxies

SOCKS is a protocol for relaying arbitrary, TCP-based communication on the Internet and was presented first by Koblas [38] in 1992. Currently, there are two major, wire-incompatible (for comparison, packet headers are shown in Figure 4 and the status codes in Table XIII in Appendix B) versions of SOCKS: version 4 (as defined by Lee [41]) and the first IETF-standardized version 5 (RFC 1928 [44], 1996), which both use port 1080 for communication. We now briefly introduce both deployed versions of the protocol and explain the main differences between them.

1) SOCKS4(A): SOCKS4 [38] defines only two commands: CONNECT for establishing a tunnel and BIND for creating a binding to allow connections *behind* the proxy to connect back to the original client (e. g., for FTP active mode).

To establish a tunnel, the client sends a CONNECT request to the proxy (potentially containing the username), which either grants the connection or responds with an error (Table XIII in Appendix B lists all standardized error codes). If the connection is granted, the server replaces the destination address and the port with those it has bound for the outgoing connection and the communication between the endpoints can begin. SOCKS4A [42] extends SOCKS4 with DNS resolving capabilities by using a non-routable IP address as the destination, and appending a domain name ending with a null-byte after the username.

2) SOCKS5: SOCKS5 [44] is the first RFC-defined version which was created to fix several limitations of SOCKS4. Most notably, it adds support for authentication negotiation, IPv6 and UDP proxying (UDP ASSOCIATE command), and the ability to delegate DNS resolving to the proxy.

In the initial handshake, the client offers its list of supported authentication methods for the server to choose from. Depending on the chosen method (e. g., no authentication or username & password [43]), the authentication protocol has to be completed before SOCKS commands can be sent. As we concentrate on open proxies, we omit further details of different authentication methods. To proxy UDP packets, the server assigns an external port, on which the client shall send UDP datagrams to be forwarded to the target destination. The created UDP conduit is kept open as long as the control connection stays alive.

C. Measurement Approach

Our measurement approach contains three separate steps, as illustrated in Figure 3. In the first step (Section IV-C1) we locate potential proxy candidates by Internet scans or by crawling, and add them into our working queue. In the second step, we verify if the host is a proxy (Section IV-C2), followed by checking for internal access (Section IV-C3) if the given host is an open proxy.

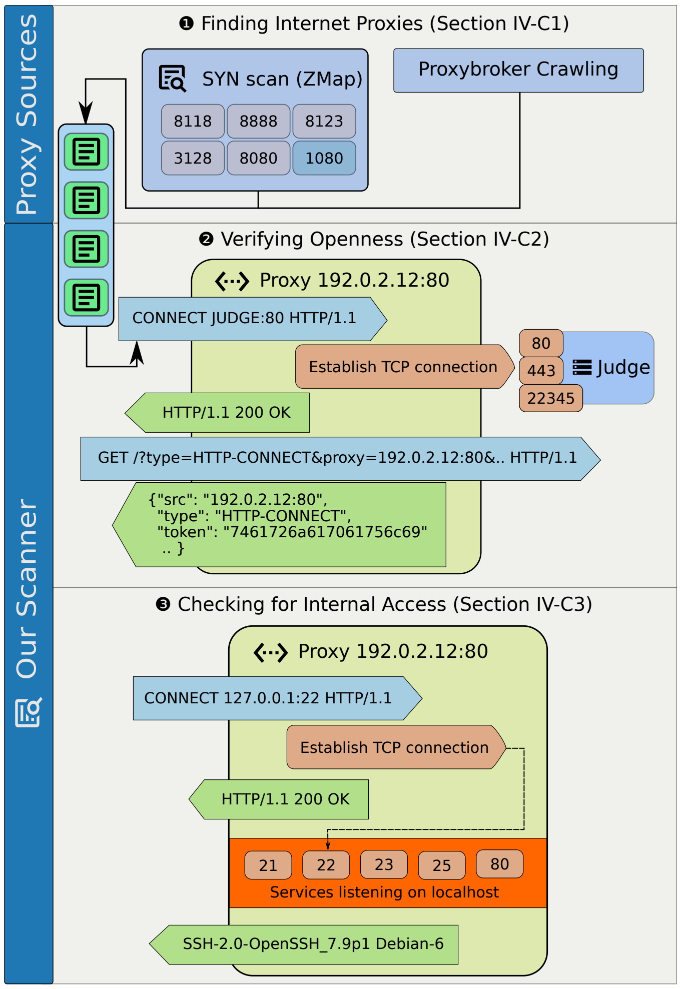

Proxy Sources
Our Scanner
❶ Finding Internet Proxies (Section IV-C1)
SYN scan (ZMap)
8118
8888
8123
3128
8080
1080
Proxybroker Crawling
❷ Verifying Openness (Section IV-C2)
Proxy 192.0.2.12:80
CONNECT JUDGE:80 HTTP/1.1
Establish TCP connection
80
443
22345
Judge
HTTP/1.1 200 OK
GET /?type=HTTP-CONNECT&proxy=192.0.2.12:80&.. HTTP/1.1
{"src": "192.0.2.12:80", "type": "HTTP-CONNECT", "token": "7461726a617061756c69" .. }
❸ Checking for Internal Access (Section IV-C3)
Proxy 192.0.2.12:80
CONNECT 127.0.0.1:22 HTTP/1.1
Establish TCP connection
HTTP/1.1 200 OK
21
22
23
25
80
Services listening on localhost
SSH-2.0-OpenSSH_7.9p1 Debian-6

Fig. 3. Proxy measurement approach: ❶ Scan the Internet with ZMap and complement the results with crawling. The responsive hosts are added to a work queue for the next step. (Sec. IV-C1). ❷ The work queue is read simultaneously and our scanner checks if the host is an open proxy (Sec. IV-C2). ❸ If the given host is an open proxy, we check if it allows requesting internal resources (Sec. IV-C3).

1) Finding Internet Proxies: To find Internet proxies, we leverage a two-step analysis pipeline. We run ZMap SYN scan on corresponding ports and seed the work queue for our crawler similarly to our previous scans. However, instead of having a single worker, we launched 20 concurrent crawlers reading from the same work queue. Each scan was finished in approximately eight hours in sync with the corresponding ZMap scan. All tests were run sequentially within one week in February 2019.

Based on the related works ([47], [69], [57], [61], see summary in Table XII) and our investigation on popular proxy software, we decided to scan the following ports: 1080 (SOCKS), 3128 (Squid), 8080 (common HTTP proxy port), 8118 (Privoxy), 8123 (Polipo), and 8888 (Tinyproxy). We emphasize that before we conducted any scans, we extensively

TABLE V. Overview of Tested Network Proxies

	Name	Security in default configuration	Updated
SOCKS	3proxy	Example config requires authentication, disallows connections to 127.0.0.1	18.4.2018
	Srelay	Example config given, but not directly usable. Allows everything if started directly.	25.12.2017
	antinat	Default: allow in only from RFC1918 ranges, disallows connections to 127.0.0.1	20.2.2017
	Dante	Default config has variety of examples, blocks everything per default	6.2.2017
HTTP	Squid (3128)	Binds to all interfaces. Inbound from RFC1918. Allowed ports: 443, 80, 21, 70, 210, 280, 488, 591, 777, 1025-65535, 443 (connect)	2.7.2018
	Tinyproxy (8888)	Binds to all interfaces. Inbound from 127.0.0.1. CONNECT only on 443 and 563	1.1.2016
	Polipo (8123)	Binds only to 127.0.0.1. Allows GET (80-100, 1024-65535), CONNECT (22, 80, 109-110, 143, 443, 873, 993, 995, 2401, 5222, 9418)	15.5.2014
	Privoxy (8118)	Binds only to 127.0.0.1	26.8.2016

tested all specifically mentioned proxies in our laboratory environment to understand their behavior and restrictions. Table V lists these implementations.

2) Verifying Openness: For verifying the functionality of proxies, we try to use them to access a website ("proxy judge") created and hosted by ourselves. To this end, we encode some information (including query type, proxy address, and port) to the requested URIs. This information is mirrored back to us with some additional information (e. g., the requesting IP address and request headers). Based on the responses, we tag each proxy (consisting of a tuple $\langle ipaddress, port \rangle$) with the information we use to summarize our findings. We call a proxy *open* if it has delivered us our expected payload. If we receive an unexpected response, we differentiate between the protocol-conforming responses (e. g., requiring us to authenticate by sending an HTTP `407` status or SOCKS error) and non-conforming (e.g., a regular web-page being delivered). We mark the former as *proxy* and the latter *responsive*, accordingly.

a) Verifying HTTP Proxies: To verify the functionality of proxies, we issued GET requests on the following three ports on our judge server: (i) 80 to verify regular HTTP functionality, (ii) 443 to confirm if HTTPS requests are possible, and (iii) 22345 to verify if the target port can be arbitrarily chosen. On port 443, we serve clients with TLS encryption using our self-signed certificate, and we do not check for the validity of the certificate in our scanner. For each potential proxy, we launch these three requests simultaneously. When receiving an HTTP response (no matter the response content or status code), we launch two more requests with CONNECT, one for port 80 and one for port 22345. The first check is used to ascertain if such requests are generally allowed (also on non-TLS ports) and the second one if we are limited to specific port ranges.

b) Verifying SOCKS Proxies: We use the same GET request payloads and probe the same ports for both SOCKS versions. In case of a successful SOCKS connection, we create additional requests based on the protocol version: (i) for SOCKS4, we also try to check for DNS resolving (SOCKS4A) support by requesting our proxy judge with its domain name, and (ii) for SOCKS5, we try to verify DNS support as well as support for UDP and IPv6 connections. To verify UDP connectivity, we send a datagram containing information about the proxy similarly to our HTTP queries to our judge server. Furthermore, to understand if *identd* authentication is actively used for SOCKS4, we also host a program capturing the incoming requests on TCP port 113.

3) Checking for Internal Access: This phase is done only on open proxies, with the additional requirement for HTTP proxies, where we expect that the proxy allows proxying using the CONNECT method.

In this phase, we send several additional requests to understand if the proxy is misconfigured and allows access to internal hosts. For this purpose, we use the targets "127.0.0.1" and "192.168.0.1", and create connections without sending any payload on banner-yielding ports 21, 22, 23, and 25. Additionally, we send an HTTP GET request to the regular HTTP port. We chose the target hosts based on the intuition that when regular sockets are used by proxy implementations, the packets passed through them are routed as any other network traffic, allowing us to detect if the proxy is vulnerable for misuses targeting non-Internet-routable addresses. For HTTP, we consider a proxy to be *potentially vulnerable* if we receive a `200` status code for any of our CONNECT requests, indicating that a successful connection to the target host has been made. On the other hand, for SOCKS we settle for receiving a status code indicating success.

In order to report on definitely *vulnerable* proxies, we deploy the following port-specific heuristics to decide whether the received payload is protocol-conforming: (1) SSH (22) begins with `SSH-`, 2) FTP and SMTP (21, 25) begins with `220`, and (3) HTTP (80) has an HTTP status line with status code "200", and (4) Telnet (23) payload has to contain word "telnet" (after a preliminary empirical analysis) after the proxy connection establishment (i.e., status code `200` for CONNECT, or a successful SOCKS reply).

4) Complementary Proxy Crawling: To complement our network scans, we also crawl proxy lists by utilizing freely available ProxyBroker [58] as also done by Mani et al. [47]. However, instead of leveraging its built-in functionality checks, we use it only to download a list of available proxies which we process using the method described above. The results from this analysis are handled separately from the Internet-wide results in our evaluation.

D. Evaluation

In the following, we first provide an overview of all proxies on the Internet, based on proxy-conforming responses (Section IV-D1). This is followed by an analysis of open proxies (Section IV-D2) and an analysis of the results from proxies allowing access to internal networks (Section IV-D4). The summarized results can be seen in Table VI. For readibility, we round the numbers in text and refer our readers to the table for exact numbers. After that, we discuss our crawling results (Section IV-D5), and discuss a case study of misconfigured proxies hosted by a large European ISP (Section IV-D6).

1) Global View on Internet Proxies: In order to quantify the total number of the HTTP proxies on the Internet, we leverage the authorization requests sent back to our requests. If authentication is required, the proxy sends an HTTP status `407` and must add a `Proxy-Authenticate` header informing how and on which realm the user needs to authenticate [24]. In total, almost 615,000 proxies sent this header with only a few

TABLE VI. SUMMARY OF FUNCTIONALITIES OF PROXIES IN STANDARD PROXY PORTS

	Total	**Squid (3128)**	**Generic (8080)**	**Privoxy (8118)**	**polipo (8123)**	**tinyproxy (8888)**	**SOCKS4**	**SOCKS5**
Total	33,968,960	3,962,196	12,316,479	3,560,946	4,392,588	4,838,874	4,897,877	4,897,877
Responsive	10,470,875	1,187,146	7,430,128	90,138	45,363	1,467,605	250,495	250,495
Proxy	688,112	387,732	178,677	1,652	2,162	66,251	31,706	19,932
Open Proxy	**19,723 (2.9 %)**	**5,098 (1.3 %)**	**8,604 (4.8 %)**	**933 (56.5 %)**	**263 (12.1 %)**	**1,878 (2.8 %)**	**1,518 (4.8 %)**	**1,429 (7.2 %)**
Open Proxy (GET)	19,545	4,991	8,561	909	263	1,874	1,518	1,429
80	16,954	3,641	7,777	847	247	1,811	1,388	1,243
443	14,549	4,116	4,966	851	232	1,704	1,405	1,275
22345	16,926	3,437	7,945	846	250	1,788	1,397	1,263
Open Proxy (CONNECT)	11,856	2,770	4,522	605†	124	888	1,518 *	1,429 *
80	10,831	2,655	3,976	584	117	868	1,388 *	1,243 *
22345	11,011	2,490	4,386	566	39	870	1,397 *	1,263 *
Potentially vulnerable	**7,981 (40.4 %)**	**2,117 (41.5 %)**	**3,290 (38.2 %)**	**424 † (45.4 %)**	**53 (20.0 %)**	**679 (36.2 %)**	**636 (41.9 %)**	**782 (54.7 %)**
Vulnerable ‡	**4,545 (23.0 %)**	**1,642 (32.2 %)**	**1,209 (14.1 %)**	**252 (27.0 %)**	**15 (5.7 %)**	**412 (21.9 %)**	**489 (32.2 %)**	**526 (36.8 %)**
FTP (21)	332 (1.7 %)	63 (1.2 %)	28 (0.3 %)	22 (2.4 %)	1 (0.4 %)	40 (2.1 %)	92 (6.1 %)	86 (6.0 %)
SSH (22)	2,717 (13.8 %)	1,409 (27.6 %)	145 (1.7 %)	218 (23.4 %)	12 (4.6 %)	316 (16.8 %)	327 (21.5 %)	290 (20.3 %)
Telnet (23)	310 (1.6 %)	1 (0.0 %)	2 (0.0 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)	128 (8.4 %)	179 (12.5 %)
SMTP (25)	673 (3.4 %)	184 (3.6 %)	42 (0.5 %)	61 (6.5 %)	2 (0.8 %)	27 (1.4 %)	133 (8.8 %)	224 (15.7 %)
HTTP (80)	1,636 (8.3 %)	343 (6.7 %)	1,077 (12.5 %)	51 (5.5 %)	8 (3.0 %)	157 (8.4 %)	n/a ††	n/a ††
Localnet	354 (1.8 %)	64 (1.3 %)	150 (1.7 %)	6 (0.6 %)	0 (0.0 %)	27 (1.4 %)	26 (1.7 %)	81 (5.7 %)

† Ignoring 202,313 open, 201,950 access to localhost – case "Large European ISP" (Section IV-D6 on page 13).
‡ Percentage in parentheses signifies the amount of vulnerable proxies from all open proxies.
†† Missing due to a failure in measurement system.
* CONNECT and GET are the same for SOCKS, duplicated just to make comparison easier.

(8,000 proxies) using other than basic authentication (such as digest or NTLM). These proxies reported with a total of 2,705 unique realms, the ten most common ones being shared by 85 % of proxies, as shown in Table VIII. The most common realm "Private port. Please go away and have a nice day"[3] related to a Network Functions Virtualization platform was used by 28 % of the proxies, followed by Squid's default realm ("Squid proxy-caching web server", 16 %) and "Proxy_Auth" (of unknown origin, 13 %). 62 % (~420,000) of all proxies were located in the USA, followed by Zambia with 72,000 proxies, while the rest were spread out all around the world. The top four AS consisted of 40 % of all proxies and had each over 50,000 proxy instances running seems to indicate that these are probably hosted by service providers. Table VII shows the distribution of proxies among continents per port.

a) Proxy Implementations: We also analyzed `Server` headers from these responses (provided by 93 % of all proxy responses). There were a total of 460 unique server strings, but by taking only the first part (i. e. "squid" from "squid/4.0.20") into account, we were left with 113 unique implementations. The most commonly deployed proxy software on any port was Squid, totaling up to 96 % of all proxies revealing their identity. The eleven most popular, uncleaned versions (excluding one without a version number) were Squid adding up 83 % of proxies, all running old, unmaintained versions between 3.1.23 and 4.0.20. The most common with 27 % was a two-year-old Squid 4.0.20, followed by 3.5.12 (released in Nov 2015) with 15 % and 3.5.27 (Aug 2017) with 12 % of all responses. Luckily, most of these are too old to be vulnerable to a recently found unauthenticated code execution flaw [52]. The first non-Squid responses were CCProxy (12th most common, 5,569 proxies) and Zscaler (14th, 4,588). Squid was also the most common implementation on all our scanned ports, including about 140,000 installations on non-default ports. Table IX lists the most common implementations from all tested ports.

[3]https://github.com/T-NOVA/Squid-dashboard/blob/master/squid/squid.conf

b) SOCKS Proxies: For SOCKS, we saw replies from almost 32,000 SOCKS4 and ~20,000 SOCKS5 proxies (from almost five million SYN-responsive hosts), totaling to ~34,000 (50 % supporting both versions) unique proxies.

2) Open Proxies: From all the scanned ports and out of almost 690,000 proxies, just under 3 % (~20,000) were open proxies. The most popular port for open proxies in absolute numbers was surprisingly not Squid's 3128 (which came second with 5,100 proxies), but the generic port 8080 with 8,600 open proxies (4.8 % from all on that port). While 71 % (11,869) of all open HTTP proxies supported the `CONNECT` command for HTTPS connections, more importantly, 53 % supported it also on non-HTTPS ports. In contrast to all proxies, the most open proxies (totaling to 23 %) were located in China, followed by the USA (17 %), Brazil (5 %), and Russia (4 %). Proxies from these four countries add up to over 50 % of all open proxies. We refer to Table VII for details on geographical location and openness of proxies among continents. A notable detail is that although the number of proxies is much lower in Asia and South America, the proxies in these continents were more likely to be open than the ones in Europe or North America. The likelyhood of being an open proxy was much higher for the privoxy port 8118 than for the rest of the ports.

a) Open SOCKS Proxies: Out of 34,216 SOCKS proxies, 6 % were open proxies, 38 % supporting both SOCKS versions. On SOCKS-specific features, 47 % of open SOCKS4 and 76 % of open SOCKS5 proxies supported the DNS extension. Furthermore, 9 % of SOCKS5 proxies were IPv6 enabled, and 10 % allowed successful UDP relaying. During our investigation, our identd server observed merely 543 ident requests, surprisingly only from 28 addresses for the SOCKS port 1080. Most of the request came for ports 8080 (260), 8888 (163), and 3128 (88), which indicates that these were likely caused by our crawling activities.

3) Vulnerable Proxies: When ignoring the large Privoxy population we are going to discuss later in Section IV-D6 and the proxies allowing CONNECT only on port 443, we

TABLE VII. PROXIES PER PORT PER CONTINENT (PERCENTAGE SHOWS THE AMOUNT OF OPEN PROXIES)

	Total	Squid (3128)	Generic (8080)	Privoxy (8118)	polipo (8123)	tinyproxy (8888)	SOCKS (1080)
North America	445,990 (0.83 %)	241,912 (0.78 %)	134,828 (0.47 %)	785 (43.57 %)	1,375 (1.67 %)	54,019 (0.43 %)	13,071 (4.28 %)
Europe	84,731 (4.94 %)	45,313 (2.98 %)	19,419 (9.34 %)	333 (41.14 %)	437 (6.86 %)	3,020 (17.78 %)	16,209 (1.96 %)
Africa	73,770 (0.47 %)	73,103 (0.05 %)	571 (46.76 %)	0 (0 %)	3 (33.33 %)	29 (68.97 %)	64 (31.25 %)
Asia	54,961 (16.25 %)	21,586 (6.06 %)	20,420 (24.09 %)	525 (85.52 %)	311 (66.88 %)	8,336 (12.46 %)	3,783 (26.67 %)
South America	6,352 (26.31 %)	3,107 (15.58 %)	2,486 (36.97 %)	2 (100.00 %)	9 (0.00 %)	66 (54.55 %)	682 (33.72 %)
Oceania	3,624 (2.65 %)	1,654 (1.93 %)	768 (6.12 %)	7 (42.86 %)	27 (3.70 %)	781 (1.66 %)	387 (0.00 %)
Unknown	1,262 (0 %)	1,057 (0 %)	185 (0 %)	0 (0 %)	0 (0 %)	0 (0 %)	20 (0 %)
Total	670,690 (2.82 %) †	387,732 (1.31 %)	178,677 (4.82 %)	1,652 (56.48 %)	2,162 (12.16 %)	66,251 (2.83 %)	34,216 (6.25 %)

† SOCKS results are combined for brevity, causing the deviation from the totals shown in Table VI.

TABLE VIII. MOST COMMON PROXY-AUTHENTICATE REALMS

Port	Realm	Proxies	%
3128	Private port. Please . . .	174,373	27.40 %
8080	Proxy_Auth	84,275	13.24 %
3128	Squid's default realm	80,966	12.72 %
3128	proxy	58,919	9.26 %
8888	Access denied	47,846	7.52 %
8080	proxy	36,880	5.79 %
3128	Anonymous proxy	21,595	3.39 %
8080	⟨Header missing⟩ †	12,885	2.02 %
8080	Squid's default realm	12,191	1.92 %
3128	Squid Basic Auth . . .	11,158	1.75 %
Other		95,386	14.99 %
Total	2,705 unique	636,474	100.00 %

† Header missing although host identified to be a proxy.

TABLE IX. MOST COMMON PROXY SERVER IMPLEMENTATIONS

Port	Server	Proxies	%
3128	squid	370,630	58.23 %
8080	squid	138,831	21.81 %
8888	squid	61,495	9.66 %
8080	⟨Header missing⟩ †	23,215	3.65 %
3128	⟨Header missing⟩ †	15,859	2.49 %
8080	Zscaler proxy	4,588	0.72 %
8080	CCProxy	3,454	0.54 %
8080	Mikrotik HttpProxy	3,429	0.54 %
8080	Proxy	2,042	0.32 %
8888	CCProxy	1,976	0.31 %
Other		10,955	1.72 %
Total	113 unique	636,474	100.00 %

† Header missing although host identified to be a proxy.

are left with a total of 8,909 CONNECT-supporting HTTP proxies, from which 74 % signaled with status "200" that they accepted connections to the localhost making them potentially vulnerable. Additionally, 40 % of these CONNECT-supporting proxies delivered an expected payload, marking them as *definitely vulnerable*, adding up to 21 % of all open HTTP proxies being vulnerable.

a) SOCKS Proxies: From ~1,500 (5 %) open SOCKS4 proxies, 42 % claimed to allow connections to the localhost, and 32 % were definitely vulnerable. In comparison, out of ~1,400 SOCKS5 proxies, over half (55 %) allowed such connections, with 37 % being definitely vulnerable. While there was no significant difference between the amount of potentially vulnerable SOCKS and HTTP proxies, open SOCKS proxies were more likely definitely vulnerable than HTTP proxies.

To conclude, 40 % of all open proxies in any protocol claimed to result in a successful connection creation on localhost on any of the tested ports, while 23 % were also returning protocol-conforming responses for our probes. We now analyze services hosted on these definitely vulnerable proxies.

4) Services Hosted on Vulnerable Proxy Systems: To understand more about the services behind definitely vulnerable hosts, we parse and categorize the responses we received from vulnerable hosts. The summary of our categorization is seen in Table X. Note that the absolute numbers differ from Table VI as the table does not differentiate between SOCKS versions.

Based on the heuristics defined in Section IV-C3, the most widely exposed service was SSH with over half of the vulnerable proxies responding with a valid SSH banner. Most commonly seen implementations were different versions of OpenSSH on different operating systems, but there were also over a hundred hosts with Mikrotik's SSH implementation (ROSSSH). The second most common was HTTP on 36 % of vulnerable proxies, the most commonly exposed service being the router configuration interface of Mikrotik routers. When comparing these results to those from SSH, it is interesting to note that over 1,200 SSH banners were from Ubuntu-based devices, followed by over 700 with a generic and ~300 with the Debian banner, so it is not just Mikrotik routers that are exposing these services. Among other HTTP exposing services were some default sites of common web servers or frameworks. On the other hand, SMTP and FTP were not so common—merely 432 SMTP services and 214 FTP servers were exposed. Telnet was the least seen service (only "CCProxy Telnet" on 80 hosts), so we omit it in the table.

5) Complimentary Crawling for Proxies: During our two weeks of crawling (end of January until the beginning of February, 2019) we collected in total 96,863 $\langle host, port\rangle$ combinations from 56,861 different IP addresses using 20,438 different ports. In total, merely 16 % (~16,000) of proxies were open, hosted on over 5,500 different ports. While the majority of the proxies were HTTP proxies (88 %), there were also 14 % of SOCKS proxies (10 % SOCKS4, 3 % SOCKS5, 677 proxies supporting both versions). In total, 67 % (~11,000) of all functioning proxies supported HTTP CONNECT or SOCKS, making them candidates for our attacks. As can be seen in Table XI, only 9 % of open SOCKS proxies were hosted on the standard port 1080. Out of all open SOCKS proxies

TABLE X. SERVICES HOSTED ON VULNERABLE PROXY SYSTEMS

Internet-wide (23 %, 4,545 out of 19,723 vulnerable)							
FTP		HTTP		SSH		SMTP	
MikroTik	100	Mikrotik Config	1,144	OpenSSH (Ubuntu)	1,252	Postfix	243
vsFTPd	36	⟨Other⟩	192	OpenSSH	727	CCProxy	76
⟨Other⟩	32	Apache2 Default	165	OpenSSH (Debian)	293	Sendmail	58
ProFTPD	29	Bootstrap Theme	66	ROSSH	115	Exim	55
Pure-FTPd	24	IIS default	42	OpenSSH (Raspbian)	19	⟨Other⟩	17
Microsoft FTP	14	nginx default	24	OpenSSH (*BSD)	13		
Filezilla	10	Squid error page	3	⟨Other⟩	6		
wuftpd	1			OpenSSH (Mikrotik)	2		
Total	246		1,636		2,427		449
Crawled Proxies (19 %, 2,961 out of 15,832 vulnerable)							
MikroTik	64	Mikrotik Config	1,912	OpenSSH	631	Postfix	470
ProFTPD	15	⟨Other⟩	69	OpenSSH (Ubuntu)	161	Exim	31
vsFTPd	13	Apache2 Default	48	OpenSSH (Debian)	98	Sendmail	14
⟨Other⟩	13	nginx Default	5	ROSSH	82	CCProxy	6
Pure-FTPd	13	IIS Default	4	OpenSSH (Raspbian)	6	⟨Other⟩	4
Filezilla	5	Bootstrap Theme	1	OpenSSH (*BSD)	5		
Microsoft FTP	2	Squid error page	1	⟨Other⟩	3		
				OpenSSH (Mikrotik)	1		
Total	125		2,040		987		525

TABLE XI. MOST COMMON PORTS FOR CRAWLED OPEN PROXIES

HTTP			**SOCKS**		
Port	Count	%	Port	Count	%
8080	3,290	22.97 %	4145	584	37.75 %
3128	1,089	7.60 %	1080	146	9.44 %
9999	825	5.76 %	9999	82	5.30 %
80	745	5.20 %	6667	28	1.81 %
53281	669	4.67 %	9050	21	1.36 %
Totals	14,324	–	–	1,547	–
Uniques	5,127	–	–	535	–

hosted behind 535 different ports, almost 40 % were using port 4145, which seems to be used as a backdoor for malicious activities [68]. In total, merely a third of all open proxies found by crawling were behind any of the standard proxy ports we used for our Internet-wide scans discussed earlier. 47 % of all open proxies indicated that they were able to form a connection to localhost (i. e., were potentially vulnerable). Out of these proxies, 42 % delivered us an expected payload (i. e., were definitely vulnerable) with 19 % (~3,000) of all open proxies in comparison to 23 % from our Internet scans. 65 % of these vulnerable hosts expose Mikrotik's configuration interface, while SSH was being exposed less often (seen in Table X). These vulnerable proxies were found in 122 countries and ~1,100 ASes – the most common locations for vulnerable proxies was China with 14 % of the proxies, followed by ~7 % of each by Russia, Indonesia, India, and Brazil.

6) Case Study: Large European ISP: During our scans, we found over 200,000 inadvertently open proxies supporting `CONNECT` proxying while returning a 400 error ("Invalid header received from client") for our absolute-URI requests. All of these systems were located in a single autonomous system of a large European ISP, spanning over 152 different subnets in a single country. Further investigations revealed the error on absolute-URI proxy requests occur when the mandatory "Host" header [22] is sent to the proxy. To confirm our suspicions, we requested the configuration page located under `config.privoxy.org` manually on one of the systems, which succeeded and we were greeted with a reasonably recent (3.0.26, released at the end of 2016) Privoxy configuration page. However, the same version of Privoxy we tested in our laboratory setting does not exhibit this erroneous behavior. Privoxy is a non-caching, filtering proxy that uses so-called actions to modify content proxied via it. In order to understand more why these proxies are deployed, we also did request the list of actions from a single proxy, which contained a single "action" adding `Link-Account` header containing a presumably unique identifier of this device. Although the configuration was perfectly fine (we tested it in our lab setup), this header was not delivered to our server. We can only guess that either the rule is not working correctly for some unknown reason, or that the ISP is using this only internally in their network. We disclosed this vulnerability to the ISP at the beginning of 2019 in various ways (e.g., e-mail and security contact form on their website). As of the end of the year 2019, this issue appears to be fixed.

E. Network Proxies: Key Findings

Our Internet-wide scans revealed that only a small percentage of services running on default proxy ports are actually proxies. Also, merely 3 % (~20,000) of all proxies are open proxies. 23 % of these are definitely misconfigured and allow unauthorized access to internal networks (i.e., an adversary can misuse them to obtain access to systems behind these proxies). Further, we found that up to 40 % are likely misconfigured, but our probes were not targeting the correct ports. For identifiable proxies, Squid was the dominant implementation with ~96 % of all hosts announcing the implementation, also on other ports besides its default 3128. We identified a population of over 200,000 open, modified Squid instances located in an ASN of a large European ISP. These proxies require a slightly off-standard requests to function, and which—according to their configuration—append an extra tracking header to their outcoming requests. Our two-week-long crawling with ProxyBroker totaled to almost as many open proxies as our Internet-wide scans on several ports. These proxies were mostly found on non-standard ports and were over 20 percentage points more likely to be definitely vulnerable, which hints that they are unlikely to be open on purpose, but rather vulnerable systems.

V. RELATED WORK

In the following, we discuss how our work relates to previous work in this area.

A. NAT Traversal Protocols

Already in 2006, Hemel [32] reported on the lack of destination address filtering in several UPnP IGD implementations, and described how this could be used to expose other internal hosts to the Internet as well as to proxy traffic to external hosts. In 2008, Squire [65] reported finding a small number of devices exposing their SOAP endpoints on the WAN interface, and three years later, Garcia [25] released a tool to scan for exposed SOAP endpoints and reported finding over 150,000 endpoints on the Internet. The first in-depth security analysis of UPnP was done by Moore [51] in 2013. He reported finding over 81 million SSDP responsive devices with 17 million exposed SOAP endpoints. In 2017, McAfee reported on sightings of malware leveraging UPnP to proxy C&C connections. [35] Concurrently and independently to our study, Akamai researchers [1], [62] analyzed malicious port mappings with similar conclusions compared to the ones we obtained via our study. Our work differs from theirs by

TABLE XII. COMPARISON OF OUR WORK TO RELATED PROXY ECOSYSTEM STUDIES

Internet-wide Scans					
	This paper	**ACSAC'18** [47]	**NDSS'18** [69]	**WWW'18** [57]	**CCC'15** [61]
Ports scanned	3128, 8080, 8118 8888, 8123, 1080	–	–	3128, 8080, 8118 8081	3128, 8080, 8123
Total	33,968,960	–	–	29,100,000	2,133,646 †
Responsive	10,470,875	–	–	6,450,000	–
Proxies	688,112 (890,425 ‡)	–	–	–	28,608 †
SOCKS any	34,216	–	–	–	–
SOCKS4	31,706	–	–	–	–
SOCKS5	19,932	–	–	–	–
Working	16,358 (218,671 ‡)	–	–	2,518	1,880 †
GET	14,389	–	–	–	–
CONNECT	11,869	–	–	–	–
SOCKS4	1,518	–	–	–	–
SOCKS5	1,429	–	–	–	–

Crawling					
	This paper	**ACSAC'18** [47]	**NDSS'18** [69]	**WWW'18** [57]	**CCC'15** [61]
Duration	2 weeks Jan–Feb'19	50 days Apr–May'18	2 months Apr–Jun'17	10 months Jan–Oct'17	–
Total	96,863	107,034	65,871	180,000	–
Responsive	19,259	54,996	49,444	–	–
Proxies	16,259	31,000	–	–	–
SOCKS	1,927	–	–	–	–
Working	15,832	20,893	19,473	39,143	–
GET	14,324	–	–	–	–
CONNECT	9,012	9,625	–	17,350	–
SOCKS	1,547	74 ††	–	–	–
SOCKS4	1,255	–	–	–	–
SOCKS5	528	–	–	–	–

† The paper reports details only for port 3128.
‡ When not ignoring the accidentally open proxies as described in Section IV-D6.
†† Only daily median reported.

not only targeting a single `WANIPConnection` interface using brute-force search. Instead, we carefully implemented the UPnP specification to obtain all the relevant interfaces and adapted our crawling based on the responses from the endpoints. As shown in Table I, this allows us to cover ~21 % of the hosts exposing `WANPPPConnection` that could have been missed by the brute-force approach. Besides reporting on the malicious forwards, we extend our work to also show that this feature is used for benign purposes.

UPnP was also in news [18], [70] when someone accessed Chromecast devices located behind NAT gateways. While this was not an attack against the protocol itself, it raised awareness of the protocol and its ability to expose devices to the Internet without users' noticing it. Other uses of UPnP have also been explored: in 2012, DiCioccio et al. [16] leveraged a software installed on end-users' computers to complement end-host based bandwidth measurements using router-reported data, e.g. connection speeds. Their results from 120,000 hosts indicated that merely 35 % had an UPnP enabled router. Related to our honeypot implementation, Hakim et al. [30] introduced a concept of generating UPnP honeypots based on UPnP description files.

There is little research available on either NAT-PMP or PCP. Some indications (such as UPnP-PCP bridge defined in RFC 6970 [9], and the support for cascading NATs) suggest that especially PCP is more aimed to be deployed by ISPs rather than home users. The only relevant study on NAT-PMP was done by Hart [31] in 2014, where he analyzed some potential attack scenarios and reported on finding 1.2 million (i.e., twice as much as our scans) exposed NAT-PMP endpoints. To the best of our knowledge, there have been no reports on insecurities in PCP deployments.

B. Internet Proxies

Various studies on the *open* proxy ecosystem exist [61], [57], [69], [47]. Common to these studies is that they are limiting their analyses to either crawling *or* Internet-wide scans, while not reporting out enough concrete numbers to allow understanding of the whole proxy ecosystem. A detailed comparison of these works to ours is summarized in Table XII and we discuss the main differences next.

In their work from 2015, Scott et al. [61] analyzed how open HTTP proxies are used by analyzing the statistics provided by management interfaces of some proxy implementations. Their work also included Internet scans on several ports (3128, 8080, 8123) to locate proxy servers, but unfortunately they left out many details. A complementary study involving both crawling and Internet-wide scans was performed by Perino et al. [57]: they leveraged existing proxy lists and did ZMap scans to quantify the free proxy ecosystem and to analyze its trustworthiness. Results from both of these studies indicate that scanning for default proxy ports are not very fruitful—from millions of SYN-responses, only a handful are real proxies in the end. We confirm and particularize these results in our study.

In 2018, Tsirantonakis et al. [69] showed that 38% of their observed open proxies did modify the sent data and that 5 % of open proxy servers could be classified as malicious. They leveraged crawling for their data collection. Mani et al. [47] studied the open proxy system also from the perspective of maliciousness of the offered services. They were the first to explicitly discuss HTTP CONNECT and SOCKS proxies shortly in this context.

In contrast to related work, we decided to report not just on *responsive* proxies (i. e., hosts which responded to a TCP SYN), but we also introduce a new category ("proxy") to report on hosts responding using a proxy protocol (i. e., SOCKS error, or `407` for HTTP proxies requesting to authenticate for access). This enables us to provide a more holistic view of both closed and open proxies on the Internet.

C. Abuse of Proxy Protocols

In 2004, Pai et al. [55] reported on different sorts of malicious activities done over their open proxies, including spam. In 2005, Andreolini et al. [3] described a honeypot system to track spam activities via proxies, but they did not report any findings. In 2008, Steding-Jessen et al. [66] analyzed the spam ecosystem using a low-interaction honeypot implementing HTTP CONNECT and SOCKS proxies. During their over a year-long study, they collected over 500 million spams while reporting that the vast majority of the connection attempts were targeting the SMTP services hosted elsewhere. SOCKS has been reportedly used by malware to offer connect-back features, e.g., by SpyEye [64].

D. Peeking to Internal Networks

WebSockets have been shown to allow attackers to probe arbitrary ports of internal devices in specific situations. Tools such as *JSrecon* [40] and *sonar.js* [10] were developed to demonstrate the practical feasibility of such scans of devices hidden behind NAT gateways. In 2013, Grover et al. [29] used custom firmware installed in over a hundred routers around the world. Their measurement indicated that an average household has seven devices connected. Huang et al. [33] reported in 2017 that 7 % (695) of the autonomous systems they investigated with the help of Luminati were evidently behind middleboxes. Unfortunately, we could not find a publicly available data set for this study, nor did the authors respond to our contact attempts. As a result, we could not verify what percentage of these are proxies. Earlier in 2019, Mi et al. [49] analyzed residential proxies provided by actors like Luminati, where they also used the same mechanism we introduced in this paper to fingerprint proxy hosts by accessing the localhost.

VI. Discussion and Limitations

A. Ethical Considerations

Considering that we are studying live systems on the Internet, we aim to prevent (or at least minimize) any potential harm on the target systems by avoiding changes to these systems. With all the following precautions in place, we try to balance between the benefits and the potential harm caused by our scanning, as discussed in the Menlo report [7]. We argue that the understanding of these phenomena outweigh the potential harms and now discuss in detail the steps we undertook to guide our measurements. Scanning approaches involving multiple application-layer requests have previously for example been used to detect TLS vulnerabilities [48].

First of all, we utilize standard-conforming querying functionality whenever possible (e. g., UPnP IGD enumeration). Where it is unavoidable (e. g., understanding potential PCP vulnerabilities), we follow the corresponding RFC guidance. For example, RFC 6887 [71, Section 11.6] recommends using a short-lived forward for obtaining the external address for PCP and we craft our payloads accordingly. All requests we sent conform to the respective standards, and we did not try to misuse or exploit any vulnerability.

We undertook several steps in order to make clear that our purposes are benign based on the recommended practices introduced by Durumeric et al. [20]. For HTTP requests, we use a user-agent string indicating that the requests are made for research purposes, and include our contact information. The server we used for scanning also hosts a website explaining our scanning activities as well as our contact information for exclusion from future scans. Considering the amount of network scanning traffic on the regular HTTP port 80, we are not able to quantify how many visitors arrived to our page due to our scanning activities.

The reverse DNS record of the scanning host was set to indicate its use for research purposes. Furthermore, the whois information for its IP address contained our abuse e-mail address, which received mostly automated mails noting that our scans have been detected. We promptly responded to these e-mails requesting for information on the networks to be blocked. In the end, only a single individual responded to our inquiries asking for the networks to be excluded from our scan, which was promptly done.

Before conducting Internet-wide scans, we tested our scanning system, including the probes we used, extensively in our laboratory environment while empirically verifying that they were not causing any unexpected side effects. The list of proxy software used in our laboratory setup can be found in Table V on page 10. For UPnP tests, we deployed a widely used miniupnpd [8] and Linux-IGD [26], the former being used as a sole PCP implementation we tested. We configured all the software (where necessary) to be as permissive as possible to verify our approach.

For proxy protocols, we limit our actions to connecting and, in case of HTTP, performing single requests. In order to obtain evidence of misconfigurations, we had to target some non-routable addresses. We deliberately chose two target addresses (localhost and "192.168.0.1") that were likely to provide us the confirmation without really trying to access any networked devices behind the target host. Our scanning approach in this case is similar to the approach taken by Mi et al. [49].

All collected data is stored on secured servers, and only authorized persons have access to this data. We did not collect any kind of personal data; our university does not require an IRB approval for this type of network scans.

B. Potential Remediations

As UPnP is not designed to be accessible over the Internet, the mitigation would be patching these vulnerable devices. However, considering that UPnP/SSDP has been misused for amplification attacks for years, it is doubtful that the manufacturers are going to provide fixes for these CPE devices. Therefore, the current recommendation and industry best practice is to filter the discovery port 1900 (e.g., [28], [13], [6], [14]). Due to the large number of ports for SOAP endpoints, blocking them is not feasible without negative impact on regular use cases. Therefore, patching or replacing vulnerable devices is the only remedy in the long term. For Internet proxies, it is necessary to implement access controls to disable accesses on unwanted networks.

C. Limitations

Besides the protocols discussed in this paper, there are also other protocols for relaying traffic which could be susceptible for misuse. One example are Traversal Using Relay NAT (TURN) relays, which are mainly used for VoIP and WebRTC when no direct peer-to-peer connectivity is available. Case in point, it was recently reported that Cisco's Meeting Server acting as a TURN gateway is vulnerable for arbitrary TCP relaying [5]. However, based on our brief investigation, their population seems to be restricted to service providers and access to these servers requires authentication, i. e., there is no openly accessible TURN server population in the same sense as for proxies, so we omitted their analysis in this paper. For Internet proxies, there is also a recent IETF draft for a non-backwards compatible SOCKS6 [53] aimed for today's protocol designs (e. g., extendability and reduction of initial round trips) with no public implementations yet.

VII. Conclusion

While Internet-wide scans have been used to understand the Internet in general, a number of hosts remain invisible to these scans due to their location behind NAT gateways. In this work, we investigated a number of application-layer middlebox protocols which an attacker could use to scan such hosts and networks. For example, we showed how Internet Gateway Device (IGD) of the UPnP protocol stack could be misused to access internal networks which would otherwise be out-of-reach for attackers. We also studied both HTTP and SOCKS proxies, given that these protocols also allow contacting devices across network borders.

To assess the attack surface, we performed several Internet-wide scans and a comprehensive analysis of the collected data. We found a large number of hosts on the Internet that are potentially vulnerable to such attacks. Most importantly, we found empirical evidence that attackers are actively abusing such protocols to reach hosts that would otherwise not be reachable. We also quantified the results from several previous studies related to open proxies in order to provide a more accurate and complete view of the proxy ecosystem than reported before. Furthermore, we showed that a large number of open proxies are misconfigured to allow likely unwanted connections on non-Internet accessible hosts. In summary, we think that our holistic approach on understanding the whole proxy ecosystem (instead of limiting ourselves to just open ones) is a useful contribution to guide future work in this area. We make the source code of our honeypot available at https://github.com/RUB-SysSec/MiddleboxProtocolStudy/ to enable further research on this topic. We have contributed a patch to fix ZMap's behavior to allow it to detect the UPnP devices responding using a non-standard source port.

Acknowledgment

We would like to thank the anonymous reviewers and our shepherd Adam Bates for the constructive feedback and guidance. Furthermore, we thank the Shadowserver Foundation for their help figuring out the ZMap issue, Chad Seaman for his insights and discussion on the UPnP issue, and Nils Wagner for the initial work on this topic. This work was supported by the German Research Foundation (DFG) within the framework of the Excellence Strategy of the Federal Government and the States – EXC 2092 CaSa – 39078197. In addition, this project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 786669 (ReAct). This paper reflects only the authors' view. The Research Executive Agency is not responsible for any use that may be made of the information it contains.

References

[1] Akamai, "UPnProxy: Blackhat Proxies via NAT Injections," Akamai, Tech. Rep., 2018. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/white-paper/upnproxy-blackhat-proxies-via-nat-injections-white-paper.pdf

[2] M. Allman, V. Paxson, and J. Terrell, "A brief history of scanning," in *ACM SIGCOMM Conference on Internet Measurement*, 2007.

[3] M. Andreolini, A. Bulgarelli, M. Colajanni, and F. Mazzoni, "HoneySpam: Honeypots Fighting Spam at the Source." *SRUTI*, vol. 5, 2005.

[4] Ari Luotonen, "Tunneling TCP based protocols through Web proxy servers," IETF Secretariat, Internet-Draft draft-luotonen-web-proxy-tunneling-01, 1999. [Online]. Available: https://tools.ietf.org/html/draft-luotonen-web-proxy-tunneling-01

[5] R. Arrouas, "Vulnerability disclosure – Cisco Meeting Server (CMS) arbitrary TCP relaying," ImmunIT, Tech. Rep., 2018. [Online]. Available: https://www.immunit.ch/en/blog/2018/06/12/vulnerability-disclosure-cisco-meeting-server-arbitrary-tcp-relaying-2/

[6] AT&T, "AT&T Network Practices," AT&T, Tech. Rep., n.d. [Online]. Available: https://about.att.com/sites/broadband/network

[7] M. Bailey, D. Dittrich, E. Kenneally, and D. Maughan, "The Menlo Report," *IEEE Security & Privacy*, vol. 10, no. 2, 2012.

[8] T. Bernard, "MiniUPnP Project HomePage," 2017. [Online]. Available: http://miniupnp.free.fr

[9] M. Boucadair, R. Penno, and D. Wing, "RFC 6970: Universal Plug and Play (UPnP) Internet Gateway Device - Port Control Protocol Interworking Function (IGD-PCP IWF)," RFC Editor, Tech. Rep., 2013.

[10] M. Bryant, "sonar.js – A Framework for Scanning and Exploiting Internal Hosts With a Webpage," 2015. [Online]. Available: https://thehackerblog.com/sonar-a-framework-for-scanning-and-exploiting-internal-hosts-with-a-webpage/index.html

[11] S. Cheshire and M. Krochmal, "RFC 6886: NAT Port Mapping Protocol (NAT-PMP)," RFC Editor, Tech. Rep., 2013.

[12] S. Cheshire, M. Krochmal, and K. Sekar, "NAT Port Mapping Protocol (NAT-PMP)," IETF Secretariat, Internet-Draft, 2005. [Online]. Available: http://www.ietf.org/internet-drafts/draft-cheshire-nat-pmp-00

[13] Comcast, "Blocked Internet Ports List," Comcast Xfinity, Tech. Rep., n.d. [Online]. Available: https://www.xfinity.com/support/articles/list-of-blocked-ports

[14] Cox, "Internet Ports Blocked or Restricted by Cox," Cox, Tech. Rep., n.d. [Online]. Available: https://www.cox.com/residential/support/internet-ports-blocked-or-restricted-by-cox.html

[15] Dataproof Communications, "Plurox: Modular backdoor," Dataproof Communications, Tech. Rep., 2019. [Online]. Available: http://www.dataproof.co.za/index.php/2019/06/18/plurox-modular-backdoor/

[16] L. DiCioccio, R. Teixeira, M. May, and C. Kreibich, "Probe and Pray: Using UPnP for Home Network Measurements," in *International Conference on Passive and Active Network Measurement (PAM)*, 2012.

[17] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-generation Onion Router," in *USENIX Security Symposium*, 2004.

[18] P. Ducklin, "Don't fall victim to the Chromecast hackers – here's what to do," Sophos, Tech. Rep., 2019. [Online]. Available: https://nakedsecurity.sophos.com/2019/01/04/dont-fall-victim-to-the-chromecast-hackers-heres-what-to-do/

[19] Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-Wide View of Internet-Wide Scanning," in *USENIX Security Symposium*, 2014.

[20] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *USENIX Security Symposium*, 2013.

[21] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee, "RFC 2616: Hypertext Transfer Protocol - HTTP/1.1," RFC Editor, Tech. Rep., 1999.

[22] R. T. Fielding and J. F. Reschke, "RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," RFC Editor, Tech. Rep., 2014.

[23] ——, "RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC Editor, Tech. Rep., 2014.

[24] ——, "RFC 7235: Hypertext Transfer Protocol (HTTP/1.1): Authentication," RFC Editor, Tech. Rep., 2014.

[25] D. Garcia, "Universal plug and play (UPnP) mapping attacks," in *DEF CON 19*, 2011. [Online]. Available: http://toor.do/DEFCON-19-Garcia-UPnP-Mapping-WP.pdf

[26] G. George, E. Wirt, and D. J. Blueman, "Linux UPnP Internet Gateway Device," 2007. [Online]. Available: http://linux-igd.sourceforge.net

[27] R. D. Graham, "MASSCAN: Mass IP port scanner," 2014. [Online]. Available: https://github.com/robertdavidgraham/masscan/

[28] B. Greene, "Filtering Exploitable Ports and Minimizing Risk to and from Your Customers," Tech. Rep., 2017. [Online]. Available: https://www.senki.org/exploitable-port-filtering/

[29] S. Grover, M. S. Park, S. Sundaresan, S. Burnett, H. Kim, B. Ravi, and N. Feamster, "Peeking Behind the NAT: An Empirical Study of Home Networks," in *ACM SIGCOMM Conference on Internet Measurement*, 2013.

[30] M. A. Hakim, H. Aksu, A. S. Uluagac, and K. Akkaya, "U-PoT: A Honeypot Framework for UPnP-Based IoT Devices," in *IEEE International Performance Computing and Communications Conference (IPCCC)*, 2018.

[31] J. Hart, "R7-2014-17: NAT-PMP Implementation and Configuration Vulnerabilities," Rapid7, Tech. Rep., 2014. [Online]. Available: https:

//blog.rapid7.com/2014/10/21/r7-2014-17-nat-pmp-implementation-and-configuration-vulnerabilities/

[32] A. Hemel, “Universal Plug and Play: Dead simple or simply deadly?” in *5th System Administration and Network Engineering Conference*, 2006.

[33] S. Huang, F. Cuadrado, and S. Uhlig, “Middleboxes in the Internet: a HTTP perspective,” in *Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2017.

[34] Internet Society, “State of IPv6 Deployment 2018,” Internet Society, Tech. Rep., 2018. [Online]. Available: https://www.internetsociety.org/resources/2018/state-of-ipv6-deployment-2018/

[35] S. Karve, “McAfee Discovers Pinkslipbot Exploiting Infected Machines as Control Servers; Releases Free Tool to Detect, Disable Trojan,” *McAfee Blogs*, 2017. [Online]. Available: https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/mcafee-discovers-pinkslipbot-exploiting-infected-machines-as-control-servers-releases-free-tool-to-detect-disable-trojan/

[36] B. Kelly, Z. Durumeric, D. Adrian, D. Corcoran, and J. A. Halderman, “Censys,” 2018. [Online]. Available: https://censys.io

[37] R. Khare and S. D. Lawrence, “RFC 2817: Upgrading to TLS Within HTTP/1.1,” RFC Editor, Tech. Rep., 2000.

[38] D. Koblas and M. R. Koblas, “SOCKS,” in *USENIX Summer 1992 Technical Conference*, 1992. [Online]. Available: https://www.usenix.org/conference/usenix-summer-1992-technical-conference/socks

[39] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, “Exit from Hell? Reducing the Impact of Amplification DDoS Attacks,” in *USENIX Security Symposium*, 2014.

[40] L. Kuppan, “JS-Recon,” 2010. [Online]. Available: http://www.andlabs.org/tools/jsrecon/jsrecon.html

[41] Y. Lee, “SOCKS: A protocol for TCP proxy across firewalls,” NEC, Tech. Rep., 2005. [Online]. Available: https://ftp.icm.edu.pl/packages/socks/socks4/SOCKS4.protocol

[42] ——, “SOCKS 4A: A Simple Extension to SOCKS 4 Protocol,” NEC, Tech. Rep., n.d. [Online]. Available: https://www.openssh.com/txt/socks4a.protocol

[43] M. Leech, “RFC 1929: Username/Password Authentication for SOCKS V5,” RFC Editor, Tech. Rep., 1996.

[44] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones, “RFC 1928: SOCKS Protocol Version 5,” RFC Editor, Tech. Rep., 1996.

[45] I. Livadariu, K. Benson, A. Elmokashfi, A. Dhamdhere, and A. Dainotti, “Inferring Carrier-Grade NAT Deployment in the Wild,” in *IEEE Conference on Computer Communications (INFOCOM)*, 2018.

[46] G. Lyon, “Firewall/IDS Evasion and Spoofing,” 2019. [Online]. Available: https://nmap.org/book/man-bypass-firewalls-ids.html

[47] A. Mani, T. Vaidya, D. Dworken, and M. Sherr, “An Extensive Evaluation of the Internet’s Open Proxies,” in *Annual Computer Security Applications Conference (ACSAC)*, 2018.

[48] R. Merget, J. Somorovsky, N. Aviram, C. Young, J. Fliegenschmidt, J. Schwenk, and Y. Shavitt, “Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities,” in *USENIX Security Symposium*, 2019.

[49] X. Mi, Y. Liu, X. Feng, X. Liao, B. Liu, X. Wang, F. Qian, Z. Li, S. Alrwais, and L. Sun, “Resident Evil: Understanding Residential IP Proxy as a Dark Service,” in *IEEE Symposium on Security and Privacy*. IEEE, 2019.

[50] C. Montonen, “Solar and Solstice - Two Mirai Variants,” Tech. Rep., 2019. [Online]. Available: https://blog.race-conditions.net/posts/solar-and-solstice-two-mirai-variants/

[51] H. Moore, “Security Flaws in Universal Plug and Play: Unplug. Don’t play,” Rapid7, Tech. Rep., 2013. [Online]. Available: https://hdm.io/writing/SecurityFlawsUPnP.pdf

[52] S. Neti and S. Sivakumaran, “CVE-2019-12527: Code execution on squid proxy through a buffer overflow,” Trend Micro, Tech. Rep., 2019. [Online]. Available: https://www.zerodayinitiative.com/blog/2019/8/22/cve-2019-12527-code-execution-on-squid-proxy-through-a-heap-buffer-overflow

[53] V. Olteanu and D. Niculescu, “SOCKS protocol version 6 - draft,” RFC Secretariat, Tech. Rep., 2018. [Online]. Available: https://tools.ietf.org/html/draft-olteanu-intarea-socks-6-02

[54] Open Connectivity Foundation, “UPnP Standards & Architecture,” 2019. [Online]. Available: https://openconnectivity.org/developer/specifications/upnp-resources/upnp

[55] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson, “The Dark Side of the Web: An Open Proxy’s View,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, 2004.

[56] Pat Padgett, “Corkscrew – A tool for tunneling SSH through HTTP proxies,” 2001. [Online]. Available: https://github.com/bryanpkc/corkscrew

[57] D. Perino, M. Varvello, and C. Soriente, “ProxyTorrent: Untangling the Free HTTP(S) Proxy Ecosystem,” in *World Wide Web Conference (WWW)*, 2018.

[58] ProxyBroker Developers, “Proxybroker,” 2019. [Online]. Available: https://proxybroker.readthedocs.io

[59] Y. Rekhter, B. G. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, “RFC 1918: Address Allocation for Private Internets,” RFC Editor, Tech. Rep., 1996.

[60] C. Rossow, “Amplification Hell: Revisiting Network Protocols for DDoS Abuse,” in *Symposium on Network and Distributed System Security (NDSS)*, 2014.

[61] W. Scott, R. Bhoraskar, and A. Krishnamurthy, “Understanding open proxies in the wild,” *Chaos Communication Camp*, 2015.

[62] C. Seaman, “UPnProxy: EternalSilence,” Akamai, Tech. Rep., 2018. [Online]. Available: https://blogs.akamai.com/sitr/2018/11/upnproxy-eternalsilence.html

[63] Shodan Developers, “Shodan,” 2019. [Online]. Available: https://www.shodan.io

[64] A. K. Sood, R. J. Enbody, and R. Bansal, “Dissecting SpyEye – Understanding the design of third generation botnets,” *Computer Networks*, vol. 57, no. 2, 2013.

[65] J. Squire, “Universal Plug and Play IGD - A Fox in the Hen House,” in *Blackhat Briefings USA*, 2008. [Online]. Available: https://www.blackhat.com/presentations/bh-usa-08/Squire/BH_US_08_Squire_A_Fox_in_the_Hen_House%20White%20Paper.pdf

[66] K. Steding-Jessen, N. L. Vijaykumar, and A. Montes, “Using low-interaction honeypots to study the abuse of open proxies to send spam,” *INFOCOMP Journal of Computer Science*, vol. 7, no. 1, 2008.

[67] Symantec, “Inception framework: Alive and well, and hiding behind proxies,” Symantec, Tech. Rep., 2018. [Online]. Available: https://www.symantec.com/blogs/threat-intelligence/inception-framework-hiding-behind-proxies

[68] ”thecableguy“, “Security breached devices - port tcp 4145,” 2018. [Online]. Available: https://forum.mikrotik.com/viewtopic.php?t=137840

[69] G. Tsirantonakis, P. Ilia, S. Ioannidis, E. Athanasopoulos, and M. Polychronakis, “A Large-scale Analysis of Content Modification by Open HTTP Proxies,” in *Symposium on Network and Distributed System Security (NDSS)*, 2018.

[70] Z. Whittaker, “Hackers hijack thousands of Chromecasts to warn of latest security bug,” *TechCrunch*, 2019. [Online]. Available: http://social.techcrunch.com/2019/01/02/chromecast-bug-hackers-havoc/

[71] D. Wing, S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk, “RFC 6887: Port Control Protocol (PCP),” RFC Editor, Tech. Rep., 2013.

Appendix

A. *UPnP*

Listing 1 shows an example request for discovering all UPnP supporting devices, and a single response from a gateway device.

```
> M-SEARCH * HTTP/1.1
> HOST: 239.255.255.250:1900
> MAN: "ssdp:discover"
> MX: 2
> ST: "ssdp:all"

< HTTP/1.1 200 OK
< CACHE-CONTROL: max-age=1800
< DATE: Sun, 22 Jul 2018 00:28:45 GMT
< EXT:
< LOCATION: http://192.168.15.1:49152/gatedesc.xml
< SERVER: Linux/2.6.21.7-cig-65, UPnP/1.0, Portable SDK ...
< X-User-Agent: redsonic
< ST: upnp:rootdevice
< USN: uuid:7461726a-6170-6175-6c69-(...)::upnp:rootdevice
```

Listing 1. UPnP Service Discovery Request and Response. The request is send as UDP datagram to multicast group 239.255.255.250 (or alternatively directly to device's unicast address), the response is HTTP over UDP.

The device description file contains information about the device (including its name, manufacturer, serial number, etc.), a list of exposed devices and their respective services (such as our target `WANIPConnection:1`). Listing 2 shows a condensed example. The service description file (pointed by *SCPDURL*) contains definitions of available methods ("actions") and their parameters for introspection.

```
<root
   xmlns="urn:schemas-upnp-org:device-1-0" configId="1337">
<device>
 <deviceType>
   urn:schemas-upnp-org:device:InternetGatewayDevice:1
 </deviceType>
  ...
 <deviceList>
  <device>
   <deviceType>
     urn:schemas-upnp-org:device:WANConnectionDevice:1
   </deviceType>
   <friendlyName>WANConnectionDevice</friendlyName>
   <manufacturer>MiniUPnP</manufacturer>
   <manufacturerURL>
     http://miniupnp.free.fr/
   </manufacturerURL>
   <modelDescription>
     MiniUPnP daemon
   </modelDescription>
   <modelName>MiniUPnPd</modelName>
   <modelNumber>20170705</modelNumber>
   <modelURL>http://miniupnp.free.fr/</modelURL>
   <serialNumber>00000000</serialNumber>
   <UDN>uuid:7061756c-6974-6172-6a61-(...)</UDN>
   <UPC>000000000000</UPC>
   <serviceList>
    <service>
     <serviceType>
      urn:schemas-upnp-org:service:WANIPConnection:1
     </serviceType>
     <serviceId>
      urn:upnp-org:serviceId:WANIPConn1
     </serviceId>
     <SCPDURL>/WANIPCn.xml</SCPDURL>
     <controlURL>/ctl/IPConn</controlURL>
     <eventSubURL>/evt/IPConn</eventSubURL>
    </service>
   </serviceList>
  </device>
 </deviceList>
 <presentationURL>http://192.168.15.1/</presentationURL>
</device>
</root>
```

Listing 2. Condensed UPnP Device Description showing availability of version 1 of the WANIPConnection service of the Internet Gateway Device.

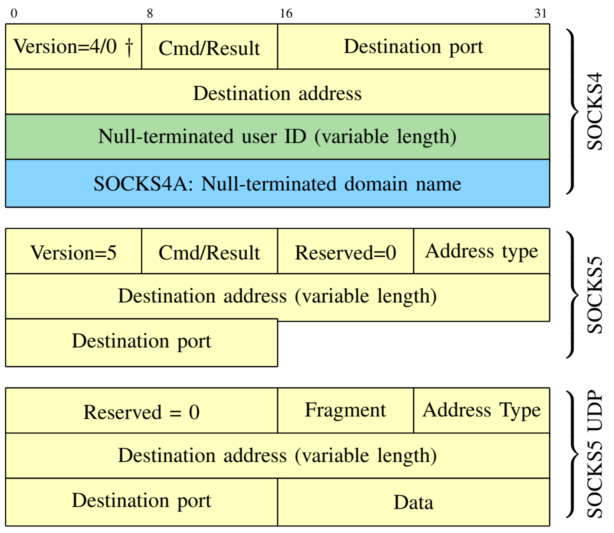


Fig. 4. SOCKS4(A) and SOCKS5 headers († for responses)

TABLE XIII. SOCKS COMMANDS AND ERROR CODES ([44], [38])

SOCKS4		**SOCKS5**	
Code	Command	Code	Command
0x01	Connect	0x01	Connect
0x02	Bind	0x02	Bind
		0x03	UDP associate
Code	Reason	Code	Reason
0x5A	Request granted	0x00	Succeeded
0x5B	Request rej./failed	0x01	General failure
0x5C	Failed: no identd	0x02	Not allowed
0x5D	Fail: identd confirm	0x03	Net unreachable
		0x04	Host unreachable
		0x05	Connection refused
		0x06	TTL expired
		0x07	Unsupp. command
		0x08	Unsupp. address type
		0xFF	No valid auth

B. *SOCKS*

Figure 4 visualizes the headers for both commonly used SOCKS versions to illustrate the lack of backward compatibility between the versions. SOCKS4A extension works by using a non-routable (first three bytes zeroed) IP address as destination and appending a null-terminated domain name at the end. SOCKS5 requires authentication method negotiation and authentication, which are omited in this presentation. SOCKS5 UDP datagrams are wrapped into a UDP preamble and sent to the server-given endpoint for delivery.

Table XIII shows the differences in commands and error codes, emphasizing the differences in the result codes.