CyberProbe: Towards Internet-Scale Active Detection of Malicious Servers

Antonio Nappa*‡, Zhaoyan Xu†, M. Zubair Rafique*, Juan Caballero*, Guofei Gu†
*IMDEA Software Institute ‡Universidad Politécnica de Madrid
{antonio.nappa, zubair.rafique, juan.caballero}@imdea.org
†SUCCESS Lab, Texas A&M University
{z0x0427, guofei}@cse.tamu.edu

***Abstract*—Cybercriminals use different types of geographically distributed servers to run their operations such as C&C servers for managing their malware, exploit servers to distribute the malware, payment servers for monetization, and redirectors for anonymity. Identifying the server infrastructure used by a cybercrime operation is fundamental for defenders, as it enables take-downs that can disrupt the operation and is a critical step towards identifying the criminals behind it.**

In this paper, we propose a novel active probing approach for detecting malicious servers and compromised hosts that listen for (and react to) incoming network requests. Our approach sends probes to remote hosts and examines their responses, determining whether the remote hosts are malicious or not. It identifies different malicious server types as well as malware that listens for incoming traffic such as P2P bots. Compared with existing defenses, our active probing approach is fast, cheap, easy to deploy, and achieves Internet scale.

We have implemented our active probing approach in a tool called CyberProbe. We have used CyberProbe to identify 151 malicious servers and 7,881 P2P bots through 24 localized and Internet-wide scans. Of those servers 75% are unknown to publicly available databases of malicious servers, indicating that CyberProbe can achieve up to 4 times better coverage than existing techniques. Our results reveal an important *provider locality* property: operations hosts an average of 3.2 servers on the same hosting provider to amortize the cost of setting up a relationship with the provider.

I. Introduction

Cybercrime is one of the largest threats to the Internet. At its core is the use of malware by miscreants to monetize infected computers through illicit activities such as spam, clickfraud, ransomware, and information theft. To distribute the malware, control it, and monetize it, miscreants leverage remotely-accessible servers distributed throughout the Internet. Such malicious servers include, among many others, exploit servers to distribute the malware through drive-by downloads, C&C servers to control the malware, web servers to monitor the operation, and redirectors for leading fake clicks to advertisements. Even P2P botnets require "server-like" remotely accessible peers for newly infected hosts to join the botnet.

Identifying the server infrastructure used by an operation is fundamental in the fight against cybercrime. It enables take-downs that can disrupt the operation [12], sinking C&C servers to identify the infected hosts controlled by the operation [52], and is a critical step to identify the miscreants running the operation, by following their money-trail [59].

Most current defenses identify malicious servers by passively monitoring for attacks launched against protected computers, either at the host (e.g., AV installations, HIDS) or at the network (e.g., NIDS, spamtraps, honeypots), or by running malware in a contained environment monitoring their network communication [22], [27]. These passive approaches achieve limited coverage, as they only observe servers involved in the attacks suffered by the protected hosts, or contacted by the malware samples run. To increase coverage, security companies aggregate information from multiple distributed sensors or execute more malware samples, but this requires a large investment or a large user base, and still does not achieve Internet-scale coverage. These approaches are also slow; malicious servers are detected asynchronously, when attacks happen to target the protected hosts. This is problematic because miscreants often use dynamic infrastructures, frequently moving their servers to make detection difficult, as well as in reaction to individual server takedowns [36]. By the time a new server is detected, a previously known one may already be dead.

A prevalent active approach for identifying malicious servers is using honeyclient farms, which visit URLs, typically found through crawling, looking for exploit servers performing drive-by downloads [42], [61]. Such farms are at the core of widely deployed browser defenses such as Google's SafeBrowsing and Microsoft's Forefront. However, honeyclients focus on exploit servers and do not cover other malicious server types. In addition, achieving coverage is expensive, requiring large investments in server farms to run the crawlers and honeyclients. Thus, they are often deployed only by large corporations.

In this paper, we propose a novel *active probing* approach for detecting malicious servers and compromised hosts that listen for (and react to) incoming network requests. Our approach sends probes to remote hosts and examines their responses, determining whether the remote hosts are malicious or not. The probes are sent from a small set of *scanner* hosts

NDSS '14, 23-26 February 2014, San Diego, CA, USA

http://dx.doi.org/10.14722/ndss.2014.23218

to a large set of *target* hosts. The targets may belong to the same network (e.g., a hosting facility), different networks across the Internet (e.g., all hosting facilities of the same provider), or correspond to all remotely accessible Internet hosts. Our approach is general and can identify different malicious server types including C&C servers, exploit servers, web front-ends, and redirect servers; as well as malware that listens for incoming traffic such as P2P bots.

Compared with existing defenses, our active probing approach is fast, cheap, easy to deploy, and achieves Internet scale. It does not require a sensor to be hosted in every network. Using 3 scanners, it can probe the Internet in 24 hours searching for a specific family of malicious servers, e.g., C&C servers of the same malware family or exploit servers of a specific operation. The scanners can be geographically distributed and rate-limited to respect bandwidth constraints on the networks hosting them. To reduce the probing time we can simply add more scanners. Given its speed, it can be used to understand the size of the server infrastructure used by an operation at a small window of time. Furthermore, it enables tracking (dynamic) malicious infrastructures over time, by periodically scanning for the servers of the same operation.

We have implemented our approach in a tool called CyberProbe, which comprises two components: *adversarial fingerprint generation* and *scanning*. CyberProbe implements a novel adversarial fingerprint generation technique, which assumes that the servers to be fingerprinted belong to an adversary who does not want them to be fingerprinted. Adversarial fingerprint generation takes as input network traces capturing dialogs with servers of a malicious *family of interest*, and builds a fingerprint, which captures what probes to send and how to determine from a target's response if it is malicious. The fingerprint generation process is designed to minimize the traffic sent to malicious servers and to produce inconspicuous probes to minimize the chance of detection by the adversary. The scanning component takes as input a fingerprint and a set of target ranges and probes those targets to check if they belong to the family of interest.

We have used CyberProbe to build 23 fingerprints for 13 malicious families (10 malware families and 3 drive-by download operations). Using CyberProbe and those fingerprints, we perform 24 scans (12 of them Internet-wide). The scans identify 7,881 P2P bots and 151 distinct malicious servers including C&C servers, exploit servers, payment servers, and click redirectors. Of those servers, 75% are unknown to 4 public databases of malicious infrastructure: VirusTotal [56], URLQuery [54], Malware Domain List [35], and VxVault [58]. This demonstrates that for some families CyberProbe can achieve up to 4 times better coverage than existing techniques. CyberProbe is also fast; in some cases it can even identify malicious servers *before* they start being used by the miscreants, when they are simply on stand-by.

Our results uncover an important *provider locality* property. A malicious operation hosts an average of 3.2 servers on the same provider to amortize the cost of setting up a relationship with the provider. As malicious servers are often hosted in cloud hosting providers [36], these providers need to be aware of provider locality. When they receive an abuse report for a malicious server, chances are more servers of the same family are being hosted on their networks.

This work makes the following contributions:

- We propose a novel active probing approach for Internet-scale detection of malicious servers. Our approach sends probes to remote target hosts and classifies those targets as belonging to a malicious family or not. Compared to current solutions our active probing approach is fast, scalable, easy to deploy, and achieves large coverage.
- We implement our approach into CyberProbe, a tool that implements a novel adversarial fingerprint generation technique, and three network scanners. CyberProbe builds fingerprints from a set of network traces for a malicious family, under the assumption that the adversary does not want its servers to be fingerprinted, and probes target networks or the Internet using those fingerprints.
- We use CyberProbe to conduct 24 localized and Internet-wide scans for malicious servers. CyberProbe identifies 151 malicious servers, 75% of them unknown to existing databases of malicious activity. It also uncovers an important provider locality property of the malicious servers hosting infrastructure.

II. Overview and Problem Definition

CyberProbe uses an active probing (or network fingerprinting) approach that sends probes to a set of remote hosts and examines their responses, determining whether each remote host belongs to a malicious family or not. Network fingerprinting has been a popular security tool for nearly two decades [9]. A fingerprint identifies the type, version, or configuration of some networking software installed on a remote host. It captures the differences in the responses to the same probes sent by hosts that have the target software installed and those that have not. A fingerprint can identify software at different layers of the networking stack. Tools like Nmap [39] use it to identify the OS version of remote hosts, and other tools like fpdns [16] or Nessus [37] use it for identifying application-layer software such as DNS or Web servers.

Our fingerprints target application-layer software and its configuration. Each fingerprint targets a specific malicious family. For C&C servers and P2P bots, a fingerprint identifies the C&C software used by a malware family. For exploit servers, a fingerprint can identify the exploit kit software or a specific configuration of the exploit kit. For example, a fingerprint could be used to identify all BlackHole exploit servers on the Internet, and a different fingerprint could be used to identify only BlackHole exploit servers belonging to a specific operation. For the latter, we leverage the intuition that exploit servers belonging to the same operation are managed by the same individuals, and therefore have similarities in their (exploit kit) configuration [36]. Since an exploit kit is typically a set of web pages and PHP scripts installed on an off-the-shelf web server (e.g., Apache or Nginx), the fingerprint needs to capture characteristics of the exploit kit independent of the underlying web server.

A malicious family may have multiple fingerprints. For example, a malware family may use different C&C protocols, or different messages in the same C&C protocol. A different fingerprint can be generated for each of those protocols or message types, but all of them identify the same family.

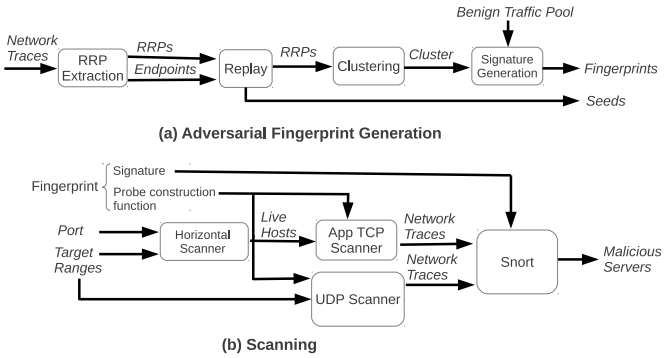

Fig. 1. Architecture overview.

Similarly, an exploit kit stores a number of files on a web server (e.g., PHP, PDF, JAR), and a fingerprint could capture a probe (and its corresponding response) for each of those files.

Our active probing approach takes as input network traces capturing traffic involving a few *seed servers* that belong to the family of interest, often only one. The fingerprints CyberProbe generates enable finding not only the seed servers, but also other previously unknown servers from the same family. Thus, active probing provides a way of amplifying the number of servers known to be part of the infrastructure of a malicious operation.

A. Problem Definition

The problem of *active probing* is to classify each host h in a set of remote target hosts H as belonging to a target family x or not. Active probing comprises two phases: *fingerprint generation* and *scanning*. The goal of fingerprint generation is to produce one or more fingerprints for a *family of interest* x, where each fingerprint $FG^x = \langle P, f_P \rangle$ comprises a *probe construction function* P and a *classification function* f_P. The probe construction function returns, for a given target host $h \in H$, the sequence of probes to be sent to the target host. The classification function is a boolean function such that when we send the probes $P(h)$ to host h and collect the responses R_P from h, $f_P(R_P)$ outputs true if h belongs to the family of interest and false otherwise. The goal of scanning is given a fingerprint, a port number, and a set of target hosts, to send the probes, collect the responses, and determine whether each target host belongs to the family of interest (i.e., matches the fingerprint).

B. Adversarial Fingerprint Generation Overview

In this work we introduce the concept of *adversarial fingerprint generation*, i.e., how to generate fingerprints for servers owned by an adversary who may not want them to be fingerprinted. The challenge in traditional fingerprint generation is to find probes that trigger *distinctive* responses from servers in the family of interest, i.e., responses that can be differentiated from those by servers not in the family. A general framework for fingerprint generation is proposed in FiG [7]. It generates candidate probes, sends them to a set of training hosts comprising hosts in the family of interest and outside of it, and applies learning algorithms on the responses to capture what makes the responses from hosts in the family of interest distinctive.

clickpayz1

probe: GET /td?aid=e9xmkgg5h6&said=26427
signature: alert tcp any any -> any any (msg:"clickpayz1"; content: "302"; http_stat_code; content: "|0d0a0d0a|Loading..."; sid:1; rev:1;)

bh2-ngen

probe: GET /ngen/shrift.php
signature: alert tcp any any -> any any (msg:"bh2-ngen"; content: "200"; http_stat_code; content: "|0d0a|Content-Disposition: attachment|3b| filename=font.eot|0d0a|"; sid: 2, rev:1;)

Fig. 2. Example fingerprints. Fingerprints are SNORT rules, ZOMG

Our adversarial fingerprint generation approach follows that framework, but has two important differences. First, we consider an adversarial scenario where the set of training hosts from the family of interest are malicious servers. We do not control them and they may be tightly monitored by their owners. In this scenario, it is critical to minimize the amount of traffic sent to those malicious seed servers and to produce probes that look inconspicuous, i.e. that resemble valid messages. As FiG generates random candidate probes, a huge number of such candidates needs to be sent before finding a distinctive response, as most random probes do not have proper protocol structure and will be ignored or incite a generic error response. Instead, CyberProbe replays previously observed requests to the seed servers. These requests come from valid interactions with the malicious servers and thus are well-formed and inconspicuous. We obtain such requests by executing malware in a contained environment (Section II-D), by monitoring a honeyclient as it is exploited in a drive-by download, or from external analysis [10].

Second, our approach differs in the process used to build the classification function. FiG's classification functions have two main problems: they operate on the raw response, ignoring any protocol structure, and they need a specific matching engine. Instead, a key intuition in this work is that the classification function can be implemented by using a network signature on the responses from the targets. Network signatures typically capture requests sent by malware infected hosts, but can similarly capture responses from remote endpoints. This relationship between fingerprint generation and signature generation enables prior and future advances on either field to be applied to the other. CyberProbe generates protocol-aware network signatures compatible with Snort [50] and Suricata [53], two efficient signature-matching open source IDSes. Figure 2 shows example fingerprints for a clickfraud operation and a drive-by download operation.

Figure 1a shows the adversarial fingerprint generation architecture. It takes as input a set of network traces capturing interactions with servers from the family of interest. First, it extracts the unique request-response pairs (RRPs) in the traces. Then, it replays the requests to the servers in the traces, keeping only replayed RRPs with distinctive responses. Next, it clusters similar requests. Finally, it generates signatures for the responses in a cluster. It outputs one or more fingerprints for the family of interest, each comprising a probe construction function and a signature.

C. Scanning Overview

We use two types of scans based on the target ranges: *Internet-wide* and *localized*. Internet-wide scans probe the entire IPv4 address space while localized scans probe selected ranges. Our localized scans explore the *provider locality* of the

malicious servers. That is, whether the managers of a malicious family select a small number of hosting and ISP providers and install multiple servers in each, to amortize the cost of setting up a relationship with the provider (e.g., registering with a fake identity, setting up working VMs). Using the seed servers as a starting point, a localized scan probes only the set of IP ranges belonging to the same providers that host the seed servers. Localized scans do not allow identifying the full infrastructure of a malicious family. However, they require sending only a very small number of probes, and quite frequently they still identify previously unknown servers.

We envision two different application scenarios for our active probing approach. Some entities like antivirus vendors, police, or national security agencies may want to use Internet-wide scans to identify all malicious servers of a family on the Internet. However, other entities like hosting providers or ISPs may want to simply scan their own IP ranges to identify malicious servers installed by their clients.

Scanners. Figure 1b shows the architecture of CyberProbe's scanning component. It comprises three scanners: a horizontal TCP scanner, a UDP scanner, and an application-layer TCP scanner (app-TCP). The horizontal TCP scanner performs a SYN scan on a given port, and outputs a list of hosts listening on that port. The UDP and app-TCP scanners send the fingerprint probes and collect or analyze the responses. For TCP fingerprints, CyberProbe first runs the horizontal scanner and then the app-TCP scanner on the live hosts found by the horizontal scanner. This allows reusing the results of the horizontal scanner for multiple scans on the same port. All 3 scanners can be distributed across multiple scanner hosts. The receiver component of the UDP and appTCP scanners can output a network trace containing all responses or run Snort on the received traffic to output the set of hosts matching the fingerprint. Saving the network trace requires significant disk space (e.g., 50 GB for an Internet-wide HTTP scan), but enables further analysis of the responses.

Scan scope. Currently, our UDP and appTCP scanners probe one fingerprint at a time since different fingerprints, even if for the same family, may use different transport protocols and require scanning on different ports. The scanners can be easily modified to scan with multiple fingerprints if the target port and target hosts are the same and the fingerprints use the same transport protocol. However, an important goal of the scanning is to spread the traffic received by a target over time and each additional fingerprint makes the scan more noisy.

D. Malware Execution

Executing malware in a contained environment is a widely studied problem [22], [27], [57]. For active probing, the main goals are acquiring the malicious endpoints known to the malware sample (e.g., C&C servers and P2P peers) and collecting instances of the network traffic between the sample and the malicious endpoints. Since C&C servers are highly dynamic it is important to run the malware soon after collection to maximize the probability that at least one of the C&C servers is alive.

We use two containment policies for running the malware: *endpoint failure* and *restricted access*. The endpoint failure policy aborts any outgoing communication from the malware by sending error responses to DNS requests, resets to SYN packets, and sinking outgoing UDP traffic. This policy is designed to trick the malware into revealing all endpoints it knows, as it tries to find a working endpoint. The *restricted access* policy allows C&C traffic to and from the Internet, but blocks other malicious activities such sending spam, launching attacks, or clickfraud. This policy also resets any connection with a payload larger than 4 KB to prevent the malware to download and install other executables.

The malware is first run with the endpoint failure containment policy and a default configuration. If it fails to send any traffic, it is rerun with different configurations. For example, it is queued to be rerun on a different VM (e.g., on QEMU if originally run on VMWare) and for an extended period of time (e.g., doubling the execution timer). This helps to address malware samples that use evasion techniques for specific VM platforms, and to account for malware samples that may take longer to start its network communication.

III. Adversarial Fingerprint Generation

This section explains the process of generating fingerprints for a malicious family of interest starting from a set of network traces. Fingerprint generation comprises 4 steps. First, it extracts from the network traces the set of request-response pairs (RRPs) (Section III-A). Then, it replays the requests to the live servers collecting their responses (Section III-B). Next, it clusters RRPs with similar requests (Section III-C). Finally, it generates signatures for each cluster (Section III-D).

Benign traffic pool. Adversarial fingerprint generation also takes as input a pool of benign traffic used to identify which parts of the responses from servers in the family are distinctive, i.e., do not appear in benign traffic. This pool comprises three traces: two of HTTP and HTTPS traffic produced by visiting the top Alexa sites [2] and a 2-day long trace comprising all external traffic from a network with 50 users, captured at the network's border. We scan the traces with two IDS signature sets, verifying that they do not contain malicious traffic.

A. RRP Feature Extraction

From the network traces, CyberProbe first extracts the RRPs, i.e., TCP connections and UDP flows initiated by the malware or honeyclient towards a remote responder, and for which some data is sent back by the responder. Here, a UDP flow is the sequence of UDP packets with the same endpoints and ports that times out if no communication is seen for a minute. For each RRP, CyberProbe extracts the following feature vector:

$\langle proto, sport, dport, sip, dip, endpoint, request, response \rangle$

where $proto$ is the protocol, $sport$, $dport$, sip, dip are the ports and IP addresses, and $endpoint$ is the domain name used to resolve the destination IP. The $request$ and $response$ features represent the raw content of the request and response.

To extract the protocol feature CyberProbe uses protocol signatures to identify standard protocols commonly used by malware such as HTTP. Protocol signatures capture keywords present in the early parts of a message (e.g., GET or POST in HTTP) [13], [19]. They are able to identify the protocol even if it uses a non-standard port, and can also identify non-standard protocols on standard ports. Both situations are common with malware. For unknown application protocols, the protocol feature is the transport protocol.

RRPs for which the request endpoint is one of the top 100,000 Alexa domains [2] are discarded. This removes traffic to benign sites, used by malware to test connectivity and by exploit servers to download vulnerable software or redirect the user after exploitation. In addition, it removes RRPs that have identical requests (excluding fields known to have dynamic data such as the HTTP Host header), to avoid replaying the same request many times. From the remaining RRPs CyberProbe builds an initial list of malicious endpoints. For this, it resolves each domain in the $endpoint$ feature to obtain the current IP addresses the domain resolves to. It returns the union of the destination IP addresses and the resolved IPs.

B. Replay

The next step is to replay the requests in the RRPs extracted from the network traces to the known malicious endpoints. The goal is to identify requests that lack replay protection, i.e., requests that if replayed to the same server at a later time or to another server of the family still incite a distinctive response. CyberProbe replays each unique request in the RRPs to every entry in the initial list of malicious endpoints, collecting the responses from endpoints that are alive.

The replay uses a commercial Virtual Private Network (VPN) that offers exit points in more than 50 countries, each with a pool of IP addresses, totaling more than 45,000 IPs. Using a VPN is important for two reasons. First, while the requests CyberProbe replays have a valid protocol syntax, there is still a small chance that they are replayed in an incorrect order or are no longer valid. If so, the managers of the malicious family could notice it and block the sender's IP address. In addition, we are interested in requests that generate a response without requiring any prior communication with the malicious server. Since CyberProbe replays all requests to each endpoint, it is important that the request being replayed is not influenced by any state that a previously replayed request may have set in the server. To achieve independence between replays, the replayer changes the VPN exit node (an thus its observable IP address) for each request sent to the same endpoint. Intuitively, a server keeps a separate state machine for each client that connects to it. Thus, by employing a previously unused IP address, the server will be in its initial state when it receives the replayed request.

Filtering benign servers. A common situation when replaying is that the IP address of a malicious server in the input network traces may have been reassigned to a benign server. Responses from benign servers need to be removed before building a signature to avoid false positives. To filter responses from benign servers, CyberProbe leverages the intuition that a benign server will not understand the replayed request and typically will ignore it (e.g., for a binary C&C request) or return an error (e.g., HTTP 404). Thus, as a first step CyberProbe removes from the replayed RRPs those with no response or where the response is an error (e.g., HTTP 4xx). However, a surprisingly large number of benign HTTP servers reply with a successful response (i.e., HTTP 200 OK) to *any* request, possibly including a custom error message in the body of the response. Thus, a technique is needed to identify custom error messages without a priori knowledge of how they may look. To address this challenge, CyberProbe also sends an HTTP request for a random resource to each potentially malicious HTTP server, leveraging the insight that if the responses from a server to the replayed request and to the random request are similar, most likely the server did not understand either request and the response is an error message.

CyberProbe considers two HTTP responses similar if they have the same result code, the same Content-Type header value, and similar content. Two non-HTML contents are similar if their MIME type as returned by the UNIX `file` tool is the same. For HTML documents, it uses an off-the-shelf similarity package [21], which serializes the HTML trees of the pages as arrays and finds the longest common sequence between the arrays. It measures similarity as:

$$d(a,b) = \frac{2 * length(LCS(array(a), array(b)))}{length(array(a)) + length(array(b))}$$

.

After removing errors and responses from benign servers the remaining RRPs are replayed twice more to the endpoints that responded, so that variations in the responses, e.g., changes in the HTTP Date and Cookie headers, are captured. The output of the replay phase are the remaining replayed RRPs. The original RRPs extracted from the network traces are not part of the output, i.e., only RRPs for which the request successfully replays are used to build the fingerprint. The unique endpoints in the output RRPs are the seed servers.

C. Clustering RRPs by Request Similarity

Next, CyberProbe clusters the RRPs by request similarity to identify instances of the same type of request across the network traces. This step prevents generating multiple fingerprints of the same type and enables producing more general fingerprints. We use two separate clusterings, a protocol-aware clustering for HTTP and a transport clustering for other protocols.

For HTTP, CyberProbe groups RRPs for which the requests have the same method (e.g., GET or POST) and satisfy the following conditions:

- **Same path.** The path in both URLs is the same and does not correspond to the root page.
- **Similar parameters.** The Jaccard index of the sets of URL parameters is larger than an experimentally selected threshold of 0.7. Parameter values are not included.

For other protocols, CyberProbe groups packets from the same transport protocol, with the same size and content, and sent to the same destination port. The output of the request clustering is the union of the traffic clusters output by the two clusterings. Each cluster contains the RRP feature vectors and the clusters do not overlap.

Probe construction function. From the requests in each cluster, CyberProbe produces a probe construction function. The probe construction function is basically one of the probes in the cluster where the value of a field may be replaced by the special TARGET and SET macros. The TARGET macro represents that the field needs to be updated with the value of the target endpoint during scanning, e.g., the HTTP Host header. The SET macro is used for fields that have different values in the cluster's requests. It represents that the value of the field can be chosen from this set when generating a new probe during scanning.

Algorithm 1 Signature Generation Algorithm

```
def tokenize_responses(cluster) {
  tokens_info = []
  # Get unique fields for responses in cluster
  unique_fields = get_distinct_fields(cluster)
  for field in unique_fields
    # Get unique values for field
    unique_values = get_distinct_fields_values(field)
    # Tokenize unique field values
    tokens = tokenize(unique_values)
    for token in tokens
      # Get feature vectors for responses with the token
      vectors = get_responses(token)
      # Add token
      tokens_info.add(field,token,vectors)
  return tokens_info

def refine_signature(tokens_info,curr_sig)
  tinfo,rem_tokens_info =
    get_token_max_overlap(tokens_info,curr_sig)
  rsig = add_token(curr_sig,tinfo)
  if cov(rsig) = cov(curr_sig)
    refine_signature(rem_tokens_info,rsig)
  if fp(curr_sig) < thres_fp
    return curr_sig
  refine_signature(rem_tokens_info,rsig)

def generate_signatures(cluster) {
  signatures = []
  tokens_info = tokenize_responses(cluster)
  while true
    # Find token that maximizes coverage
    tinfo,rem_tokens_info,cov_increase =
      get_max_coverage_token(signatures,tokens_info)
    if cov_increase < thres_cov break
    else
      initial_sig = add_token(∅,token_info)
      refined_sig = refine(rem_tokens_info,initial_sig)
      if refined_sig
        signatures.add(refined_sig)

  return signatures
}
```

D. Signature Generation

For each cluster, signature generation produces signatures that capture parts of the responses that are unique to the family of interest, i.e., that are uncommon in the benign traffic pool. CyberProbe builds token-set payload signatures, which are supported by both Snort and Suricata. A token set is an unordered set of binary strings (i.e., tokens) that matches the content of a buffer if all tokens in the signature appear in the buffer, in any order. The more tokens and the longer each token the more specific the signature.

Algorithm 1 describes the signature generation. Its salient characteristics are that when the protocol is known (e.g,. HTTP) the tokenization is performed on fields and that multiple signatures may be generated for each cluster. For each field in the responses in the cluster, it identifies distinctive tokens i.e., tokens with high coverage and low false positives. We define the *false positive rate* of a token in a field to be the fraction of responses in the benign pool that contain the token in the field, over the total number of responses in the benign pool. The *coverage* is the fraction of responses in the cluster with the token in that field, over the total number of responses in the cluster. A token is distinctive if it has a file coverage larger than 0.4 and a false positive rate below 10^{-9}.

Algorithm 1 can generate multiple signatures because distinctive tokens do not need to appear in all responses in the traffic cluster. This is important to handle noise in the cluster, e.g., from incorrectly labeled malware in the input traces. The get_distinct_fields function returns all fields in the response (or a single field if the protocol is unknown), except fields that contain dynamically generated data (e.g., the Date and Set-Cookie HTTP headers), as those fields should not be part of the signature. The tokenize function uses a suffix array [1] to extract tokens larger than 5 bytes that appear in the set of unique field values.

Full (F)	**Unreserved** (U)	**Allocated** (I)	**BGP** (B)
4.3B (100%)	3.7B (86%)	3.7B (86%)	2.6B (60%)

TABLE I. NUMBER OF IPV4 ADDRESSES (IN BILLIONS) FOR DIFFERENT INTERNET-WIDE TARGET SETS.

IV. SCANNING

This section first describes general characteristics of our scanning such as the target ranges to scan, scan rate, scan order, and scanner placement. Then, it details the implementation of our horizontal, UDP, and appTCP scanners.

A. General Scanning Characteristics

Scan ranges. We perform 3 types of scans based on the ranges to be probed: *localized-reduced*, *localized-extended*, and *Internet-wide*. For Internet-wide scans, prior work has used different ranges that qualify as "Internet-wide" [3], [11], [20], [28]. These studies do not scan the full Internet IPv4 space (F), but rather the non-reserved IPv4 ranges ($U \subseteq F$) [3], the IANA-allocated blocks ($I \subseteq U$) [11], [20], or the set of advertised BGP ranges ($B \subseteq I$) [28]. These ranges differ in their sizes, which are shown in billions of IP addresses in Table I. The U and I ranges are nowadays the same as all the non-reserved IPv4 space has been allocated by IANA. In this work, for Internet-wide horizontal and UDP scans, we first collect the BGP ranges advertised the day of the scan from the RouteViews site [46]. Then, we union those ranges removing any route overlaps. The table shows that using the BGP information to exclude non-routable ranges reduces the scan range up to 40%.

Localized scans focus on IP ranges belonging to providers that have been observed in the past to host a server of the malicious family. To select the target ranges for localized scans we use the IP addresses of the seed servers and the BGP route information. For localized-reduced scans, we obtain the most specific BGP route that contains each seed's IP address, and output the union of those routes. For localized-extended scans, for each seed server we first obtain the most specific route containing the seed's IP. From each of those routes, we extract the route description, which typically identifies the provider that the route belongs to. Then, we query again the BGP information for the list of all other routes with the same description (i.e., from the same provider) and make their union our target set.

Scan rate. Nowadays, a well-designed scanner running on commodity hardware can send fast enough to saturate a 1 Gbps link (i.e., 1.4 Mpps) [14] and some work enables commodity hardware to saturate even 10 Gbps links [45]. Thus, a scanner often needs to be rate-limited to avoid saturating its uplink, disconnecting other hosts in the same network. In this work, for good citizenship we limit each horizontal and UDP scanner

host to a maximum of 60,000 packets per second (26 Mbps), and each appTCP scanner host to a rate of 400 connections per second.

Scan order. Our horizontal and UDP scanners select which target to probe next using a random permutation of the target address space. Drawing targets uniformly at random from the target ranges mixes probes to different subnets over time, avoiding the overload of specific subnets [51]. To scan in random order, without needing to keep state about what addresses have already been scanned or are left to be scanned, our horizontal and UDP scanners use a linear congruential generator (LCG) [24]. Since the IP addresses output by the horizontal scanner are not sequential, the appTCP scanner does not use a LCG but simply randomizes the order of the target IP addresses.

Whitelisting. The LCG iterates over a single consecutive address range. However, the BGP ranges to be scanned may not be consecutive. Also, we may need to exclude certain ranges, e.g., those whose owners request so. To address these issues, before probing a target, the horizontal and UDP scanners check if the target's IP is in a whitelist of IP addresses to scan, otherwise they skip it. The whitelist is implemented using a 512 MB bit array, where each bit indicates if the corresponding IP address needs to be probed. This ensures that checks are done in $O(1)$. Since most commodity hardware has a few GBs of memory this is a good tradeoff of memory for efficiency. For the appTCP scanner, which does not use an LCG, we simply remove IP addresses that should not be probed from the input target list.

Scanner placement. Multiple scanners can be used to distribute a scan. Since a single scanner may be able to saturate its uplink it is typically not needed to use multiple scanners on the same network. It is preferable to add them in separate networks with independent uplinks. All scanners use the same generator for the LCG. To split the target hosts between scanners, we assign each scanner a unique index from zero to the number of scanners minus one. All scanners iterate over the targets in the same order, but at each iteration only the scanner whose index matches the target IP modulo the number of scanners sends the probe.

B. Horizontal Scanner

An efficient horizontal scanner is fundamental to perform fast and resource-efficient scans because the large majority of IP addresses (97%–99% depending on the port) do not send responses to probes. Two important characteristics of our horizontal scanner are the lack of scanning state and the asynchronous sending of probes and receiving of responses.

Our horizontal scanner performs TCP SYN (or half-open) scans. While there exists different types of TCP scans [39], TCP SYN scans are arguably the most popular one because they can efficiently determine if a target host is listening on a port. They are also called half-open scans because they never complete a full TCP handshake. A SYN packet is sent to a target and if a SYNACK response is received, the scanner marks the target as alive and sends it a RST packet, which avoids creating state on the scanner or the target. A single SYN packet is sent to each target without retransmissions, which prior work has shown as a good tradeoff between accuracy (low packet loss on the backbone) and efficiency (avoiding doubling or tripling the number of probes) [28]. The horizontal scanner is implemented using 1,200 lines of C code and runs on Linux. It comprises a sender and a receiver module. Both modules are independent and can be run on the same or different hosts. We describe them next.

Sender. The sender uses raw sockets to send the probes. Raw sockets bypass the kernel network stack so that no state is kept for a probe. They prevent the kernel from doing route and ARP lookups, and bypass the firewall. When a SYNACK packet is received, the kernel automatically sends a RST packet since it is unaware of the connection. On initialization the sender creates a buffer for a raw Ethernet request. It fills all fields in the Ethernet, IP, and TCP headers except the destination IP address, source port, sequence number, and TCP and IP checksums. Using a single buffer and caching most field values reduces memory accesses, increasing performance. The source IP is the IP address of the receiver. If the receiver runs on a separate host the sender spoofs the receiver's IP address. To enable the receiver to identify valid responses, the sequence number is filled with the XOR of the target IP and a secret shared between the sender and the receiver. The checksums can be computed on software or outsourced to the network card if it supports checksums on raw sockets.

The sender implements rate limiting by enforcing an inter-probe sleeping time. The Linux kernel does not provide fine-grained timers by default, so OS functions like *usleep* or *nanosleep* are too coarse for microsecond sleeps. Instead, the scanner deactivates CPU scaling, computes the sleeping time in ticks, and then busy-waits using the `rdtsc` instruction until it is time to send the next probe.

Receiver. The receiver is implemented using libpcap [31] and set to sniff all SYNACK packets. Note that the number of received packets is much smaller than the number of probes, e.g., only 2.6% of the advertised IPs listen on 80/tcp. Thus performance is less critical in the receiver than in the sender. Once the sender completes, the receiver keeps listening for a predefined time of 5 minutes to capture delayed responses. The receiver uses the shared secret, the acknowledgment number, and the source IP to check if the SYNACK corresponds to a valid probe. If so, it outputs the source IP to a log file of live hosts. There is no need to keep state about which IPs have already responded. Once the scan completes, duplicated entries due to multiple SYNACKs are removed from the log.

C. AppTCP & UDP Scanners

The appTCP and UDP scanners need to be able to send probes from different fingerprints, which may capture different application-layer protocols and message types. The probe construction function in a fingerprint abstracts the specificities of probe building from the scanner. Each probe construction function comprises two C functions. The first function is called during initialization and builds a default probe. Then, for each target host the appTCP or UDP scanner passes the target IP to the second function, which returns the TCP or UDP payload for the probe (e.g., updating the default probe with target-specific field values).

Both scanners can apply the fingerprint by running Snort on the received traffic. In addition, they can collect the responses into a network trace and then run Snort offline on the trace. In

our experiments we store the responses to enable post-mortem analysis and for collecting benign responses to enhance the benign traffic pool.

AppTCP scanner. The appTCP scanner is implemented using the libevent [30] library for asynchronous events, which is able to handle thousands of simultaneous non-blocking connections. It comprises 600 lines of C code plus the code that implements the probe construction functions for each fingerprint. It takes as input the list of live hosts identified by the horizontal scanner. To limit the connection rate the appTCP scanner operates on batches and the batch size limits the maximum number of simultaneously open connections. Reception is asynchronous, i.e., each received packet triggers a callback that reads the content from the socket. It sets a maximum size for a response since most classification functions operate on the early parts of a response. The default is 1MB but can be modified for any fingerprint. This limit is needed for servers that respond to any request with a large stream of data. For example, SHOUTCast [48] radio streaming servers may send a 1GB stream in response to an HTTP request for a random file.

UDP scanner. The UDP scanner uses the same architecture as the horizontal scanner, but builds instead UDP probes using the fingerprint's probe construction function. It comprises 800 lines of C code. The sender component also uses raw sockets, but embeds the secret in the source port instead of the sequence number. Similar to the appTCP scanner, the receiver component sets the maximum size of a response to 1MB.

V. Evaluation

This section presents the evaluation results for adversarial fingerprint generation (Section V-A), our scanning setup (Section V-B), scanning results (Sections V-C to V-E), and detailed analysis of selected operations (Section V-F).

A. Adversarial Fingerprint Generation Results

We obtain malware from two different sources: VirusShare [55] and the MALICIA dataset [33]. We run the malware on our infrastructure to produce the network traces used as input to the fingerprint generation. VirusShare malware is not classified, so we use a traffic clustering algorithm to split the executables into families [43]. The MALICIA dataset contains malware distributed through drive-by downloads, already classified into families, so clustering is not needed. For the exploit servers, we use network traces of the honeyclients collecting the malware in the MALICIA dataset. In addition, we add another exploit server family not present in MALICIA that we identify in URLQuery [54] and use a honeyclient to collect the network traces.

Table II summarizes the results of adversarial fingerprint generation. It shows the type and source of the network traces, the number of malicious families, the number of network traces processed, the RRPs in those traces, the RRPs replayed after filtering, and the number of seeds and fingerprints output. Overall, CyberProbe produces 23 fingerprints for 13 families: 3 exploit server families and 10 malware families. Of those, one fingerprint uses UDP and the rest use HTTP. The number of generated fingerprints is low compared to the number of network traces processed because some families have many traces (e.g., 700 for winwebsec) and because much malware connects to dead servers, which have likely been replaced by newer ones.

B. Scanning Setup

For the localized horizontal and UDP scans we use a single scanner at one of our institutions. This machine has 4 CPU cores running at 3.30GHz, a network connection at 1Gbps and 4GB of RAM. To distribute the Internet-wide horizontal and UDP scans across different hosts and locations we also rent 4 large instances in a cloud hosting provider. For the HTTP scans, we rent smaller virtual machines on virtual private server (VPS) providers and also use two dedicated hosts installed at one of our institutions. For the VPSes we select the cheapest instance type offered by each provider that satisfies the following minimum requirements: 1GHz, 512RAM, 100Mbps link, and 15GB hard drive.

Different providers may offer different virtualization technologies (e.g., XEN, OpenVZ, VMWare). The cheapest ones are often based on OpenVZ technology (starting at $4/month). In total we spent close to $600 on cloud hosting for the experiments in this paper. The selected instances on different providers have different resources and those resources are sometimes not well specified. For example, some providers only specify the maximum amount of network traffic the instance can send over the rental period (e.g., 1TB/month), but do not specify the network interface bandwidth and whether they perform some additional rate-limiting of the VMs. To address the resource differences across VMs we split the target addresses proportionally to the hard drive and network bandwidth (when known) of the rented instances. This may result in some scanners being assigned larger ranges than others, e.g., 3 scanner hosts being used one with 50% and each of the other two with 25% of the total target addresses.

C. Horizontal Scanning

For TCP fingerprints, CyberProbe first performs a horizontal scan of the desired target ranges to identify hosts listening on the target port. Table III summarizes our horizontal scans. It shows the scan type, i.e., localized-reduced (R), localized-extended (E), or Internet-wide (I); the date of the scan; the target port; the number of target IP addresses scanned; the number of scanners used (SC); the sending rate for each scanner; the duration of the scan; and the number (and percentage) of live hosts found.

The first 9 scans are localized scans, targeting small ranges from 4,096 to 19 million IP addresses, and performed at very low scan rates. The goal of these localized scans was to test our scanning infrastructure, and to perform an initial evaluation of whether our hosting provider locality hypothesis holds (next subsection). The last 4 are Internet-wide scans, three on 80/tcp and one on 8080/tcp. Using 5 scanner hosts at a rate of 50,000 packets per second (pps), or 4 scanners at 60,000pps, it takes less than 3 hours for CyberProbe to perform an Internet-wide horizontal scan.

The Internet-wide scans found 2.6% of all advertised IP addresses listening on 80/tcp and 0.01% on port 8080. The 80/tcp scan on April 30th found 67.7 million hosts, the scan on July 1st 65.5 million, and the August 5th scan 63.5 million. The 8080/tcp scan found 239K live hosts. The difference on live hosts found between the 80/tcp scans is due to changes on the total size of the BGP advertised routes on the scan days. The live hosts intersection between the April 30th and July 1st 80/tcp scans is 43.9 million IPs (67%). That is, two

Type	Source	Families	Pcaps	RRPs	RRPs Replayed	Seeds	Fingerprints
Malware	VirusShare	152	918	1,639	193	19	18
Malware	MALICIA	9	1,059	764	602	2	2
Honeyclient	MALICIA	6	1,400	42,160	9,497	5	2
Honeyclient	UrlQuery	1	4	11	11	1	1

TABLE II. ADVERSARIAL FINGERPRINT GENERATION RESULTS

HID	Type	Start Date	Port	Targets	SC	Rate(pps)	Time	Live Hosts
1	E	2013-03-12	8080	13,783,920	1	300	14.3h	193,667 (1.4%)
2	R	2013-03-26	80	4,096	1	60	1.2m	2,053 (50.1%)
3	E	2013-04-08	80	7,723,776	1	125	19.1h	316,935 (4.1%)
4	R	2013-04-14	80	24,576	1	200	2.4m	14,134 (57.5%)
5	E	2013-04-15	80	32,768	1	200	3.6m	14,869 (45.3%)
6	E	2013-04-17	80	1,779,965	1	125	3.9h	751,531 (42.2%)
7	E	2013-04-20	8080	19,018,496	1	900	6.0h	301,758 (1.6%)
8	R	2013-04-23	80	105,472	1	250	7.2m	8,269 (7.8%)
9	R	2013-04-28	80	668,672	1	5,000	2.4m	36,148 (5.4%)
10	I	2013-04-30	80	2,612,160,768	4	50,000	3.5h	67,727,671 (2.6%)
11	I	2013-04-30	8080	2,612,160,768	4	50,000	3.5h	239,517 (0.01%)
12	I	2013-07-01	80	2,510,340,631	5	50,000	2.9h	65,633,678 (2.6%)
13	I	2013-08-05	80	2,459,631,240	4	60,000	2.9h	63,534,118 (2.6%)

TABLE III. HORIZONTAL SCANNING RESULTS.

thirds of the 80/tcp live hosts are stable for over 2 months. The others change due to servers being added and removed, and IP assignments changing over time. This indicates that we can trade coverage for politeness by reusing the results of a horizontal scan for multiple application-layer scans. This slowly decreases coverage over time, but minimizes the number of horizontal scans needed.

The results show that the 80/tcp localized scans find from 4.1% up to 57.5% of live hosts on the targeted ranges, well above the 2.6% Internet average. This happens because most seeds are located on cloud hosting providers, which are being abused to install malicious servers. Thus, localized scans focus on hosting services that house significantly more web servers than other residential or enterprise networks.

D. HTTP scanning

Table IV summarizes our HTTP scans, which probe the set of live hosts found by the horizontal scans, identifying malicious servers matching a fingerprint. The left part of the table shows the scan configuration: the scan date, the target port, the fingerprint used, the number of hosts scanned, the horizontal scan that found them (HID), and the number of scanners used (SC). We have used CyberProbe to perform 22 scans using 14 fingerprints. Note that we have yet to scan for the remaining fingerprints.

The middle part of Table IV shows the results: the scan duration; the response rate (Resp.), i.e., the percentage of targets that replied to the probe; the number of malicious servers found; the number of malicious servers found previously known to us, i.e., seeds and servers found by a prior scan for the same family; and the number of found servers previously unknown. CyberProbe takes on average 14 hours to perform an Internet-wide HTTP scan using 4 scanners and 24 hours using 3 scanners.

The results show that the 22 scans identified 194 servers, of which 151 are unique. Starting from 15 seeds CyberProbe identified 151 unique malicious servers, achieving a 10x amplification factor. Of the 22 scans, 91% (20) find previously unknown malicious servers, the exception being two localized scans for winwebsec and blackrev. The 11 localized scans find 66 servers (34 new), an average of 6 servers found per scan. The 11 Internet-wide scans find 128 servers (72 new), an average of 11.6 servers found per scan. While Internet-wide scans find more servers per scan, if we normalize by the number of targets scanned, localized scans find an abnormally high number of malicious servers. This verifies our provider locality hypothesis: cybercriminals are installing multiple servers on the same providers. Once they establish a relationship with a hosting provider they are likely to reuse it, minimizing the effort for locating new providers, learn their procedure to install new servers, and create fake identities for registration (e.g., Paypal accounts).

Coverage. The right part of Table IV shows the number of servers found by CyberProbe that were already known to 4 popular anti-malware cloud services: VirusTotal (VT) [56], URLQuery (UQ) [54], Malware Domain List (MD) [35], and VxVault (VV) [58]. All these cloud services use crowdsourcing to collect potentially malicious executables and URLs. Their coverage depends on the number and volume of their contributors. Some of them have infrastructures to automatically visit submitted URLs (VirusTotal and URLQuery) and execute the submitted malware to collect behavioral information (VirusTotal). The collected information is dumped into databases and public interfaces are provided to query them. As far as we know, Malware Domain List and VxVault follow a similar process to populate their databases from submissions, but the process is manually performed by volunteers. We select these specific databases because they are popular and allow querying by IP address, while other public databases, e.g., Google Safe Browsing [47], only enable URL queries.

The best coverage is achieved by VirusTotal, which knows 25.7% of the servers found by CyberProbe (50/194), followed by URL Query with 15.5% (30/194). Malware Domain List

ID	Start Date	Port	Fingerprint	Targets	HID	SC	Time	Resp.	Found	Known	New	VT	UQ	MD	VV
1	2013-01-08	8080	doubleighty	4K	-	1	62h	92%	5	4	1	0	3	1	0
2	2013-03-03	8080	doubleighty	193K	1	1	79m	91%	11	2	9	0	1	0	0
3	2013-03-26	80	winwebsec	2K	2	1	3m	96%	2	1	1	0	1	0	0
4	2013-04-08	80	winwebsec	316K	3	1	5.3h	22%	2	2	0	0	1	0	0
5	2013-04-15	80	blackrev	14K	4	1	18m	94%	1	1	0	0	0	0	0
6	2013-04-16	80	blackrev	14K	5	1	19m	94%	2	1	1	0	0	0	0
7	2013-04-17	80	bh2-adobe	751K	6	1	9.9h	55%	3	1	2	1	1	0	0
8	2013-04-17	8080	doubleighty	301K	7	1	5.1h	22%	4	2	2	0	0	0	0
9	2013-04-23	80	kovter-links	8K	8	1	8m	36%	2	1	1	1	0	0	0
10	2013-04-23	80	clickpayz1	8K	8	1	8m	31%	17	2	15	0	0	0	0
11	2013-04-28	80	clickpayz1	36K	9	1	35m	38%	17	15	2	1	0	0	0
12	2013-07-06	80	bh2-adobe	65.6M	12	3	24.7h	75%	10	1	9	3	1	0	0
13	2013-07-11	80	clickpayz1	65.6M	12	3	26.5h	74%	22	17	5	7	0	0	0
14	2013-07-16	80	clickpayz2	65.6M	12	3	26.6h	76%	25	12	13	5	1	0	0
15	2013-07-20	80	kovter-pixel	65.6M	12	3	26.5h	72%	7	1	6	4	0	0	0
16	2013-07-22	80	bh2-ngen	65.6M	12	3	24.6h	72%	2	1	1	0	0	0	0
17	2013-07-25	80	optinstaller	65.6M	12	3	24.5h	71%	18	1	17	3	2	0	1
18	2013-07-27	80	bestav-pay	65.6M	12	4	15.6h	70%	16	2	14	6	5	0	0
19	2013-07-29	80	bestav-front	65.6M	12	4	13.2h	*62%	2	1	1	1	1	0	0
20	2013-07-31	80	ironsource	65.6M	12	4	13.1h	*59%	7	1	6	5	5	0	0
21	2013-08-05	80	soft196	65.6M	12	2	23.8h	71%	8	1	7	6	5	0	0
22	2013-08-06	80	winwebsec	63.5M	13	3	15.6h	85%	11	0	11	7	3	0	0
							TOTALS:		194	70	124	50	30	1	1

TABLE IV. HTTP SCAN RESULTS.

and VxVault only know one of the servers each, an abysmal 1.1%. Overall, CyberProbe finds 4 times more malicious servers than the best of these services. The best coverage among those 4 services is achieved by those using automatic processing (VirusTotal and URLQuery). Although those 2 services have huge URL and malware collections, they still achieve limited coverage. Those services could be combined with our active probing approach so that when they discover new seed servers and families, fingerprints are automatically generated and scanned to identify other family servers. This would significantly increase their coverage and bring them closer to Internet scale.

Our results show that CyberProbe achieves 4 times better coverage than current approaches for identifying some malicious server families. However, there exist some implementation and deployment trade-offs that limit CyberProbe's coverage, which could be even higher. For example, we reuse the results of horizontal scans over time to minimize the number of horizontal scans. In particular, scans 12–21 target the live hosts found by horizontal scan 12 in Table III. As expected, the response rate of these HTTP scans decreases over time as those results become stale. However, we find that the response rate decreases slowly, from 75% to 70% 3 weeks later. Scans 19–20 show a lower response rate because they include 2 instances that (unaware to us) were rate-limited by the provider. Removing the instances from that provider the response rate was 70% for scans 19–20. This slow decrease justifies the trade-off of coverage for politeness. However, in other situations it may be possible or better to perform more aggressive scanning. We further discuss scan completeness in Section VI.

False positives. The majority of the fingerprints do not produce false positives. However, the bh2-adobe fingerprint, which captures a fake Adobe webpage (further explained in the next section) produces one false positive. It corresponds to a web server with a page that contains license keys for popular software from Adobe. The authors seem to have copied parts of the Adobe webpage that are part of our signature. We have not verified if the license keys work.

E. UDP scans.

One of the fingerprints produced by CyberProbe was for the UDP-based P2P protocol used by the ZeroAccess botnet. According to an analysis by Sophos [62], this P2P protocol has two types of peers: remotely reachable supernodes with public IP addresses and normal nodes behind NATs. There are two distinct ZeroAccess botnets, each using two ports for the P2P C&C (for 32-bit and 64-bit infected hosts). The executed malware was from one of the 32-bit botnets operating on port 16471/udp. The fingerprint captures a `getL` command in which a peer requests from a supernode the list of other supernodes it knows about, and the corresponding `retL` response where the supernode returns a subset of its peers.

Table V summarizes the UDP scans. A localized-restricted scan on 40,488 IPs belonging to a residential US provider was first used to test the fingerprint. It identified 55 supernodes, a response rate of 0.13%. A later Internet-wide scan found 7,884 supernodes (0.0003% response rate). Since the response comprises a list of advertised supernodes, we extract their addresses from the responses and compute their union across all 7,884 responses. There were 15,943 supernodes advertised at the time of the Internet-wide scan. Of those, 6,257 (39%) were found by our scan and 9,686 (61%) were not reachable. The unreachable hosts could have been cleaned, be offline, or have changed IP address (e.g., mobile devices, DHCP). Our scan also finds 1,627 supernodes alive but not advertised. This could be due to supernodes only responding with a partial list of peers and due to nodes that have changed IP since advertised. One day after the Internet-wide scan only 19%

Type	Start Date	Port	Fingerprint	Targets	SC	Rate(pps)	Time	Found
R	2013-03-19	16471	zeroaccess	40,448	1	10	1.2h	55 (0.13%)
I	2013-05-03	16471	zeroaccess	2,612,160,768	4	50,000	3.6h	7,884 (0.0003%)

TABLE V. C&C UDP SCANNING RESULTS.

Operation	Finger prints	Seeds	Servers	Prov.	Provider Locality
bestav	3	4	23	7	3.3
bh2-adobe	1	1	13	7	1.8
bh2-ngen	1	1	2	2	1.0
blackrev	1	1	2	2	1.0
clickpayz	2	2	51	6	8.5
doubleighty	1	1	18	9	2.0
kovter	2	2	9	4	2.2
ironsource	1	1	7	4	1.7
optinstaller	1	1	18	2	9.0
soft196	1	1	8	4	2.0
TOTAL	14	15	151	47	3.2 (avg.)

TABLE VI. SERVER OPERATIONS SUMMARY.

of the 15,943 advertised supernodes were alive. This high variability has previously been observed to make it easy to overestimate the size of a botnet using IP addresses [52]. However, the speed of active probing makes IP variability a smaller issue, enabling an easy and quite accurate method for estimating the size of P2P botnets.

F. Server Operations

Table VI summarizes the 10 server operations analyzed. It shows the number of fingerprints from the operation used in the scans, the seeds used to generate the fingerprints, the number of unique servers found, the number of providers hosting the servers found, and the ratio of servers per provider of the operation. Overall, these operations host an average of 3.2 servers per provider. The remainder of this section details selected operations.

BestAV. Best AV is an affiliate pay-per-install program that has been operating since at least August 2010 distributing the `winwebsec` family, which encompasses multiple fake AV brands [4]. Nowadays, BestAV manages 3 programs: the winwebsec fake antivirus, the `Urausy` ransomware, and another unknown family [5]. We have 3 fingerprints related to the BestAV operation. Two of the fingerprints were generated by running winwebsec malware. They capture C&C servers (winwebsec) and payment servers (bestav-pay). The Internet-wide scans reveal 16 payment servers and 11 C&C servers. There is strong provider locality as they use 4 cloud hosting providers for the 27 servers. Provider A hosts 6 payment and 5 C&C servers, provider B 9 payment and 4 C&C servers, provider C 2 C&C servers, and provider D the remaining payment server. The 3 providers used for payment servers provide only dedicated server hosting, which indicates that the managers do not want external services colocated with their payment infrastructure. The third fingerprint captures web servers used by the affiliates for checking statistics and collecting their installers. We manually generated this fingerprint after reading an external analysis, which identified 2 live web servers [4]. One of them was alive and we use it as seed server. An Internet-wide scan reveals a second server for the affiliates that we have not seen mentioned anywhere else. This server does not show any domain associated on different passive DNS databases, so we believe it is kept as backup in case the main one is taken offline.

Blackhole2-adobe. The bh2-adobe fingerprint captures a malware distribution operation through drive-by downloads that has been ongoing since at least October 2012 [6]. This operation configures their Blackhole 2 exploit servers to redirect users to a fake Adobe webpage if exploitation fails, which prompts the user to install a malicious Flash player update. The webpage has been copied from Adobe but resides on a different resource. Our fingerprint captures that an Adobe server will reply to that resource with a 404 error, but the exploit servers will successfully return an Adobe download page. Our Internet-wide scan on July 6 found 10 live servers, all in cloud hosting services. This supports recent results that the majority of exploit servers are abusing cloud hosting services [36]. Of the 10 servers, 3 were already known to VirusTotal. Another 2 were identified by VirusTotal four days later, and a third one 13 days after CyberProbe detected it. This shows how CyberProbe can find servers earlier than other approaches.

Blackhole2-ngen. The bh2-ngen fingerprint captures another drive-by download operation, distinguishable because their URLs contain the `/ngen/` directory. The Internet-wide scan reveals only 2 servers. To verify that CyberProbe does not miss servers we examine the URLQuery database. It contains 10 exploit servers with the /ngen/ string since October 2012. Since then, the database contains at most three servers operating on the same period of time. None of those 10 servers are located in known hosting providers, which makes us think they are using their own hosting. The new server CyberProbe found on July 7 is still not in URLQuery. It is the only server hosted on a known dedicated server hosting provider. We hypothesize it is either a test server or has not yet been set to receive traffic.

Doubleighty. The doubleighty family uses an unknown exploit server with a fixed resource in the landing URL: `/forum/links/column.php`. CyberProbe identifies 18 distinct servers in 3 localized scans with strong provider locality as two cloud hosting providers host 61% of the servers. After the March 3 scan, we used a honeyclient to visit the 9 new servers found. Seven of them exploited the honeyclient but two did not. We set the honeyclient to periodically visit those 2 servers. One month later (April 4) one of them started distributing malware. This shows that the server was installed much earlier than it started being used. It also shows that active probing can sometimes identify stand-by servers, before they exhibit their malicious behavior.

Kovter. Kovter is a ransomware family that blocks the infected computer and displays a police warning on the screen telling the user it needs to pay a fine to have it unlocked. CyberProbe produced two fingerprints for different C&C messages used by the malware. We performed one localized scan using the kovter-links fingerprint that found 2 servers and an Internet-

wide scan 3 months later using the newer kovter-pixel fingerprint that found 7. Thus, the C&C infrastructure has a high level of redundancy. One of the servers appears in both scans so it has been alive for at least 3 months. It is located in a German cloud hosting provider. Overall, the 8 distinct servers are distributed among 4 cloud hosting providers.

Clickpayz Clickpayz[1] is an online service that sells clicks. Quoting them: "clickPAYZ has a network of search sites with 10s of millions of people searching for everything under the sun". Some files in our malware datasets send clicks to their servers and the two fingerprints produced by CyberProbe seem to correspond to traffic sent by two different affiliates. The 39 unique servers identified by both fingerprints are click redirectors belonging to Clickpayz. They are located on 6 cloud hosting providers. Clickpayz managers are either unaware that their affiliates send them clicks via files flagged as malicious by different antivirus, or simply do not care. However, their claim of having tens of millions of people searching their sites is dubious and their site only provides an email address as contact information, typically a sign of dark objectives.

VI. Discussion

A. Ethical Considerations

Internet scanning has been carried out many times for different research goals [11], [14], [20], [28], [41]. Still, the unsolicited nature of the probes makes some targets consider it offensive. We take ethical considerations seriously in our study. For our horizontal scanning, we follow the recommendations of prior work, notably those by Leonard and Loguinov [28] who study how to perform maximally polite horizontal scans. We adopt their proposals of mixing the scanner IP addresses, setting up forward and backward DNS records for the scanners, running a web server on the scanners with a page explaining that the probing is part of a research project, and removing from the scan whitelist the ranges of owners that complain about our probing and are willing to share their IP ranges. Overall, we have removed from the whitelist 106K IP addresses. In addition, we limit the probing rate of our horizontal scanners to 60,000pps, well below their maximum rate. We also manually vet the generated fingerprints before scanning to make sure they do not contain attacks and will not compromise any host. Furthermore, we work with our system administrators to minimize the impact on the local network (e.g., bypass the firewall / IDS) and to quickly answer any complaints.

No prior literature details how to perform application-layer probing of malicious servers. Our HTTP probing is not malicious, it simply sends a request, which we have manually vetted first, and collects a response from a target server. However, our requests mimic those of malicious families, and often request inexistent resources from web servers. Thus, they may be considered suspicious or malicious by server owners, or may trigger IDSes loosely configured to match traffic on both directions. Overall, out of 11 VMs that we use for HTTP probing, 2 of them got suspended for "malicious" behavior. We did not get a warning from those providers, but found out when trying to access the instances. In addition, we received 3 emails warning us that our VMs may have been compromised. The communications from the providers and those received by the system administrators of our institutions show that the majority of the complaints come from web honeypots that do not advertise their IP addresses and consider any received traffic malicious. A less frequent reason are owners thinking our scanner hosts have been infected or are attacking them. Similar to the horizontal scanning, when the providers let us know their IP ranges, we avoid further probing.

Finally, it is worth noting that our scanning does not collect and publicize any sensitive information on remote networks.

B. Future Improvements

Completeness. Our current implementation is not guaranteed to find all servers forming the infrastructure of a family. There are two reasons for this. First, there are some families for which we cannot generate a fingerprint (e.g., their traffic cannot be replayed) or for which we may only be able to generate fingerprints for some of the server types they use (e.g., for the C&C server but not for their web servers). Second, our implementation has some limitations that limit our coverage. In particular, we have limited scanning capacity and are not able to run all fingerprints for a family simultaneously. In addition, we reuse the results of horizontal scans. This makes our probing more polite but reduces coverage slowly over time.

Complex protocol semantics. One limitation of our fingerprint generation approach is that a replayed request may fail to incite a response from a remote server, e.g., if a field in the request should be a checksum of the sender's IP address or if the request is encrypted using the IP address as initialization vector. Such semantic information cannot be easily obtained from the network traffic, but prior work extracts it from a binary that implements the protocol [8], [32]. For cases where a binary is available, e.g., with malware, we plan to integrate binary analysis techniques into our approach.

Shared hosting. Some types of web hosting such as shared hosting and content delivery networks (CDNs) involve installing multiple domains on the same web server under the same IP. Here, the web server requires the presence of a domain name in the Host header to route the request, as two domains on the same server may define the same resource (e.g,. index.html). This is problematic for our scanning as we do not know the domains hosted on each probed IP address. However, malicious servers rarely use shared hosting services because those services are *managed*, i.e., the web server owner installs the content for the clients, which is problematic if the content is C&C software or an exploit kit. We could leverage passive DNS databases to identify domains hosted on an IP address to be probed. Some challenges that we foresee are the current limited coverage of such databases and the large amount of queries needed to complete a scan.

Making the probes identifiable to selected parties. Whenever we get a complaint on our probing we ask the reporters for the IP ranges they own and we remove them from the whitelist. However, some reporters may not want to disclose their IP ranges, e.g., if they run web honeypots whose addresses should remain secret. For those cases, we could embed a secret in our probes and disclose it to selected parties. For example, we could fix the secret used to compute the sequence number of our TCP probes and reveal it to reporters so that they can check if a received probe was sent by CyberProbe. Whenever the secret is updated we would need to notify all reporters.

[1] https://www.clickpayz.com/

VII. Related Work

Active probing. Active probing (or active network fingerprinting) has been proposed for a variety of goals. Comer and Lin first proposed active probing to identify differences between TCP implementations [9] and tools like Nmap popularized the approach to identify the OS of remote hosts [17]. It has also been used to identify the version of application-layer servers [16], [37] and for tracking specific devices based on device clock skews [25]. A variant of active probing identified web users that visit a web server by querying the browser [15].

Most related to our work are active probing techniques to detect network-based malware. BotProbe [18] actively injects commands into IRC channels to identify if the target is an IRC bot or a human. PeerPress [63] uses active probing to detect P2P malware in a monitored network. Two fundamental differences with these work are that CyberProbe can detect any type of application that listens on the network, and that it focuses on probing external networks, achieving Internet scale. CyberProbe does not need to inject traffic into existing connections as BotProbe. The fingerprint generation used by CyberProbe differs from the one in PeerPress in that it leverages network traffic rather than binary analysis. This makes it possible to scale to running large quantities of malware. In independent work, Marquis-Boire et al. [34] manually generated fingerprints to identify the servers used by FinSpy, a commercial software that governments employ to spy on activists. Our work differs, among others, in that we propose a novel adversarial fingerprint generation technique that automatically generates fingerprints for a large number of malicious server families.

Fingerprint/signature generation. FiG proposed to automatically generate OS and DNS fingerprints from network traffic [7]. CyberProbe follows the same high-level fingerprint generation approach as FiG, but proposes a novel adversarial fingerprint generation technique, with two fundamental differences. First, it does not randomly or manually generate candidate probes, rather it reuses previously observed requests. This greatly reduces the traffic that needs to be sent to the training servers for generating a fingerprint, and produces inconspicuous probes. Both properties are fundamental when fingerprinting malicious servers. In addition, CyberProbe uses network signatures to implement the classification function, so it does not require a specific fingerprint matching component. Furthermore, CyberProbe addresses the problem of Internet-wide scanning.

There is a wealth of prior work on automatically generating network signatures for worm detection. Honeycomb [26], Autograph [23], and EarlyBird [49] proposed signatures comprising a single contiguous token. Polygraph [38] proposed more expressive token set, token subsequence, and probabilistic Bayes signatures. Wang et al. extended PAYL [60] to generate token subsequence signatures for content common to ingress and egress traffic. Nemean [64] introduced semantics-aware signatures and Hamsa [29] generated token set signatures that can handle some noise in the input traffic pool. Beyond worms, Botzilla [44] generated signatures for the traffic produced by a malware binary run multiple times in a controlled environment, Perdisci et al. [40] clustered and generated signatures for malware with HTTP C&C protocols, and FIRMA [43] generalized the approach to handle any protocol. A fundamental difference is that these studies generate network signatures for requests sent by the malware, while CyberProbe generates them on the responses from remote servers.

Scanning. Prior work demonstrates the use of Internet-wide scanning for security applications. Provos and Honeyman used it for identifying vulnerable SSH servers [41], Dagon et al. for finding DNS servers that provide incorrect resolutions [11], and Heninger et al. for detecting weak cryptographic keys in network devices [20]. In this work we propose another security application for active probing: identifying malicious servers. Leonard et al. [28] described how to perform Internet-wide horizontal scans with the goal of maximizing politeness. The design of our scanning is greatly influenced by their work. Other studies are related to how to perform fast scanning. Staniford et al. described techniques that malware can use to quickly spread through scanning [51]. Netmap [45] proposed a framework for fast packet I/O in software, which enables a single core to generate 14.88 Mpps, enough to saturate a 10Gbps link. Recently, Durumeric et al. proposed Zmap [14], a fast Internet-wide scanner that can do a horizontal scan of the Internet in 45 minutes from a single host. Compared to these studies our goal is to identify malicious servers. CyberProbe could incorporate some of these techniques to speed up the scanning, but currently we cap the scan speed for good citizenship.

VIII. Conclusion

In this paper, we have proposed a novel active probing approach for detecting malicious servers and compromised hosts that listen for (and react to) incoming network requests. Our active probing approach sends probes to remote hosts and examines their responses, determining whether the remote hosts are malicious or not. Compared with existing defenses, it is fast, cheap, easy to deploy, and achieves Internet scale. It identifies different malicious server types such as C&C servers, exploit servers, payment servers, and click redirectors, as well as malware that listens for incoming traffic such as P2P bots.

We have implemented our active probing approach in a tool called CyberProbe, which implements a novel adversarial fingerprint generation technique, and 3 scanners. We have used CyberProbe to build fingerprints for 13 malicious families. Using those fingerprints, CyberProbe identifies 151 malicious servers and 7,881 P2P bots through 24 localized and Internet-wide scans. Of those servers 75% are unknown to 4 databases of malicious servers, indicating that for some families CyberProbe can achieve up to 4 times better coverage than existing techniques. Our results also reveal an important *provider locality* property: cybercriminals host an average of 3.2 servers on the same hosting provider to amortize the cost of setting up a relationship with a provider.

IX. Acknowledgements

The authors are especially grateful to the Network Security teams at Texas A&M University and the IMDEA Software Institute for helpful discussions and their professional handling of this work. We would also like to thank the people behind VirusTotal, URLQuery, VxVault, and Malware Domain List for making their information publicly available.

This research was partially supported by NSF (Grant No. CNS-0954096) and AFOSR (Grant No. FA9550-13-1-0077).

Partial support was also provided by the European Union through Grant FP7-ICT No. 256980 and the EIT-ICT Labs CADENCE project, by the SoftNet-CM project, and by the Spanish Government through Grant TIN2012-39391-C04-01 and a Juan de la Cierva Fellowship for Juan Caballero. All opinions, findings and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1), 2004.

[2] Alexa - the web information company. http://www.alexa.com/.

[3] D. Benoit and A. Trudel. World's first web census. *International Journal of Web Information System*, 3, 2007.

[4] Tracking cyber crime: Inside the fakeav business. http://www.xylibox.com/2011/06/tracking-cyber-crime-inside-fakeav.html.

[5] The missing link - some lights on urausy affiliate. http://malware.dontneedcoffee.com/2013/05/the-missing-link-some-lights-on-urausy.html.

[6] Blackhole exploit kit v2 on the rise. http://research.zscaler.com/2012/10/blackhole-exploit-kit-v2-on-rise.html.

[7] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. Song, and A. Blum. fig: Automatic fingerprint generation. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2007.

[8] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *ACM Conference on Computer and Communications Security*, Alexandria, VA, October 2007.

[9] D. E. Comer and J. C. Lin. Probing tcp implementations. In *USENIX Summer Technical Conference*, Boston, MA, June 1994.

[10] Collection of pcap files from malware analysis. http://contagiodump.blogspot.com.es/2013/04/collection-of-pcap-files-from-malware.html/.

[11] D. Dagon, C. Lee, W. Lee, and N. Provos. Corrupted dns resolution paths: The rise of a malicious resolution authority. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2008.

[12] D. Dittrich. So you want to take over a botnet... In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, San Jose, CA, April 2012.

[13] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *USENIX Security Symposium*, Vancouver, Canada, July 2006.

[14] Z. Durumeric, E. Wustrow, and J. A. Halderman. Zmap: Fast internet-wide scanning and its security applications. In *USENIX Security Symposium*, Washington, D.C., August 2013.

[15] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies Symposium*, Berlin, Germany, July 2010.

[16] Fpdns. http://www.rfc.se/fpdns/.

[17] Fyodor. Remote os detection via tcp/ip stack fingerprinting, December 1998. http://www.phrack.com/phrack/51/P51-11.

[18] G. Gu, V. Yegneswaran, P. Porras, J. Stoll, and W. Lee. Active botnet probing to identify obscure command and control channels. In *Annual Computer Security Applications Conference*, Honolulu, HI, December 2009.

[19] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. acas: Automated construction of application signatures. In *ACM Workshop on Mining network data*, Philadelphia, PA, October 2005.

[20] N. Heninger, Z. Durumeric, E. Wustrow, and J. Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, Bellevue, WA, August 2012.

[21] Html::similarity. http://search.cpan.org/~xern/HTML-Similarity-0.2.0/lib/HTML/Similarity.pm/.

[22] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying spamming botnets using botlab. In *Symposium on Networked System Design and Implementation*, Boston, MA, April 2009.

[23] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security Symposium*, San Diego, CA, August 2004.

[24] D. E. Knuth. *The Art Of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[25] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2005.

[26] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *Workshop on Hot Topics in Networks*, Boston, MA, November 2003.

[27] C. Kreibich, N. Weaver, C. Kanich, W. Cui, and V. Paxson. gq: Practical containment for measuring modern malware systems. In *Internet Measurement Conference*, Berlin, Germany, November 2011.

[28] D. Leonard and D. Loguinov. Demystifying service discovery: Implementing an internet-wide scanner. In *Internet Measurement Conference*, Melbourne, Victoria, Australia, November 2010.

[29] Z. Li, M. Sanghi, B. Chavez, Y. Chen, and M.-Y. Kao. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2006.

[30] Libevent. http://libevent.org/.

[31] Tcpdump/libpcap. http://www.tcpdump.org/.

[32] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2008.

[33] The malicia project. http://malicia-project.com/.

[34] M. Marquis-Boire, B. Marczak, C. Guarnieri, and J. Scott-Railton. For their eyes only: The commercialization of digital spying. https://citizenlab.org/2013/04/for-their-eyes-only-2/.

[35] Malware domain list. http://malwaredomainlist.com/.

[36] A. Nappa, M. Z. Rafique, and J. Caballero. Driving in the cloud: An analysis of drive-by download operations and abuse reporting. In *SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, Berlin, Germany, July 2013.

[37] Nessus. http://www.nessus.org/.

[38] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2005.

[39] Nmap. http://www.insecure.org.

[40] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *Symposium on Networked System Design and Implementation*, San Jose, CA, April 2010.

[41] N. Provos and P. Honeyman. Scanssh - scanning the internet for ssh servers. Technical Report CITI TR 01-13, University of Michigan, October 2001.

[42] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *USENIX Security Symposium*, San Jose, CA, July 2008.

[43] M. Z. Rafique and J. Caballero. Firma: Malware clustering and network signature generation with mixed network behaviors. In *International Symposium on Recent Advances in Intrusion Detection*, St. Lucia, October 2013.

[44] K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov. Botzilla: Detecting the phoning home of malicious software. In *ACM Symposium on Applied Computing*, 2010.

[45] L. Rizzo. Netmap: A novel framework for fast packet i/o. In *USENIX Annual Technical Conference*, Boston, MA, June 2012.

[46] University of oregon route views project. http://www.routeviews.org/.

[47] Google safe browsing. https://developers.google.com/safe-browsing/.

[48] Shoutcast. http://www.shoutcast.com/.

[49] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Symposium on Operating System Design and Implementation*, San Francisco, CA, December 2004.

[50] Snort. http://www.snort.org/.

[51] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the internet in your spare time. In *USENIX Security Symposium*, San Francisco, CA, August 2002.

[52] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *ACM Conference on Computer and Communications Security*, Chicago, IL, November 2009.

[53] Suricata. http://suricata-ids.org/.

[54] Urlquery. http://urlquery.net/.

[55] Virusshare. http://virusshare.com/.

[56] Virustotal. http://www.virustotal.com/.

[57] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *Symposium on Operating Systems Principles*, Brighton, United Kingdom, October 2005.

[58] Vxvault. http://vxvault.siri-urz.net.

[59] R. J. Walls, B. N. Levine, M. Liberatore, and C. Shields. Effective digital forensics research is investigator-centric. In *USENIX Workshop on Hot Topics in Security*, San Francisco, CA, August 2011.

[60] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *International Symposium on Recent Advances in Intrusion Detection*, Seattle, WA, September 2005.

[61] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2006.

[62] J. Wyke. The zeroaccess botnet: Mining and fraud for massive financial gain, September 2012. http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/zeroaccess-botnet.asp:x.

[63] Z. Xu, L. Chen, G. Gu, and C. Kruegel. Peerpress: Utilizing enemies' p2p strength against them. In *ACM Conference on Computer and Communications Security*, Raleigh, NC, October 2012.

[64] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha. An architecture for generating semantics-aware signatures. In *USENIX Security Symposium*, Baltimore, MD, July 2005.