AUTOPROBE: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis

Zhaoyan Xu
SUCCESS LAB
Texas A&M University
College Station, TX, US
z0x0427@cse.tamu.edu

Antonio Nappa
IMDEA Software Institute
Madrid, Spain
antonio.nappa@imdea.org

Robert Baykov
SUCCESS LAB
Texas A&M University
College Station, TX, US
baykovr@cse.tamu.edu

Guangliang Yang
SUCCESS LAB
Texas A&M University
College Station, TX, US
glyang@cse.tamu.edu

Juan Caballero
IMDEA Software Institute
Madrid, Spain
juan.caballero@imdea.org

Guofei Gu
SUCCESS LAB
Texas A&M University
College Station, TX, US
guofei@cse.tamu.edu

ABSTRACT

Malware continues to be one of the major threats to Internet security. In the battle against cybercriminals, accurately identifying the underlying malicious server infrastructure (e.g., malicious remote hosts used as exploit servers to distribute malware through drive-by downloads, C&C servers for botnet command and control) is of vital importance. Most existing passive monitoring approaches cannot keep up with the highly dynamic, ever-evolving malware server infrastructure. As an effective complementary technique, active probing has recently attracted attention due to its high accuracy, efficiency, and scalability (even to the Internet level).

In this paper, we propose AUTOPROBE, a novel system to automatically generate effective and efficient fingerprints of remote malicious servers. AUTOPROBE addresses two fundamental limitations of existing active probing approaches: it supports pull-based C&C protocols, used by the majority of malware, and it generates fingerprints even in the common case when C&C servers are not alive during fingerprint generation.

Using real-world malware samples we show that AUTOPROBE can successfully generate accurate C&C server fingerprints through novel applications of dynamic binary analysis techniques. By conducting Internet-scale active probing, we show that AUTOPROBE can successfully uncover hundreds of malicious servers on the Internet, many of them unknown to existing blacklists. We believe AUTOPROBE is a great complement to existing defenses, and can play a unique role in the battle against cybercriminals.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security

Keywords

Malware analysis, Malware detection

1. INTRODUCTION

Internet is an essential part of our life. However, malware poses a serious threat to Internet security. Millions of computers have been compromised by various malware families, and they are used to launch all kinds of attacks and illicit activities such as spam, clickfraud, DDoS attacks, and information theft. Such malicious activities are normally initiated, managed, facilitated, and coordinated through remotely accessible servers, such as exploit servers for malware's distribution through drive-by downloads, C&C servers for malware's command and control, redirection servers for anonymity, and payment servers for monetization. These malicious servers act as the critical infrastructure for cybercrime operations and are a core component of the malware underground economy. Undoubtedly, identifying malware's server infrastructure is of vital importance to defeat cybercrime.

Traditional approaches for detecting malicious servers mostly rely on passive monitoring of host and network behaviors in home/enterprise/ISP networks. However, such passive approaches are typically slow, incomplete and inefficient because miscreants use dynamic infrastructures and frequently move their servers (e.g., for evasion or as a reaction to takedowns). To solve this issue, active probing techniques have been proposed to detect malicious servers and compromised hosts in an active, fast, and efficient way [25, 41]. The basic idea is to use a *network fingerprinting* approach that sends specially crafted packets (i.e., *probes*) to remote hosts and examines their responses to determine whether they are malicious or not. Since probes are sent from a small set of scanner hosts, active probing is scalable, even for the entire Internet.

In this work we describe AUTOPROBE, which implements a novel approach to the problem of automatically building network fingerprints that can be used for (actively) detecting malware servers on the Internet. Our goal is similar to the recently proposed CYBERPROBE [25], which demonstrated how active probing can successfully detect malicious servers at Internet scale. However, our approach to fingerprint generation radically differs from the one used by CYBERPROBE. While CYBERPROBE takes as input network traces and leverages machine learning techniques on network traffic to generate the fingerprints, AUTOPROBE assumes the availability of a sample of the target malware family and applies dynamic binary analysis on the malware executable. AutoProbe addresses fundamental limitations in CyberProbe. First, CyberProbe is not able to generate fingerprints for many malware families that contain replay protection. In addition, the lack of semantics available in network traffic and the noise in the input network traces limit the quality of CyberProbe's fingerprints. Furthermore, CyberProbe cannot generate fingerprints when there is no known live C&C server to experiment with (thus no network interactions can be observed) or when the known C&C servers are only alive for a very short time (thus not enough traffic for building reliable fingerprints).

Dynamic binary analysis has been previously used by PeerPress to generate fingerprints for P2P malware [41]. However, PeerPress cannot be used to detect remote malicious servers. It can only generate fingerprints for malware that embeds some server-side logic and listens on the network for incoming requests such as P2P bots. Instead, the majority of malware families use a pull-based C&C protocol, where bots contain only client-side logic, send periodic requests for instructions to the remote C&C servers, and close the communication after the response from the C&C server is received. Pull-based C&C is the dominant choice because it avoids incoming probes being blocked by NAT gateways and firewalls. To build fingerprints for remote servers PeerPress would require the C&C server software, which is not available.

AutoProbe greatly complements PeerPress. It enables generating fingerprints for identifying C&C servers for malware that has only client-side logic, extending active probing beyond P2P bots to also include C&C servers.

AutoProbe applies dynamic binary analysis to achieve profound understanding on the packet semantics and deeper insight on the malware's logic for request generation (to remote servers) and response handling (back from the servers) in the following ways.

First, in analyzing (outgoing) request generation logic, AutoProbe focuses on two tasks: (1) It tracks the generation of variant bytes, whose value may change in a different environment, and their semantics. Through re-generating variant bytes in realistic environments, AutoProbe obtains a more accurate probe request. (2) It analyzes the logic to uncover as many request generation paths as possible. Thus, AutoProbe can generate more probing requests than existing approaches.

Second, in analyzing (incoming) response handling logic, AutoProbe employs a novel scheme for detection, i.e., AutoProbe identifies specific response bytes that can affect client-side malware's execution as the evidence to detect malicious servers. More specifically, AutoProbe applies dynamic symbolic execution to find a set of path constraints and generates light-weight network-level symbolic-constraint-based fingerprints for detection. Furthermore, AutoProbe can generate fingerprints even when a remote server is not alive thus no actual response can be received by the malware client, an unsolved challenge for existing approaches.

Our paper makes the following contributions:

- We propose a novel approach for automatically generating active probing fingerprints, which can detect remote malicious servers. Compared with prior work [25, 41], our approach leverages dynamic binary analysis and is able to generate fingerprints for the large number of malware families that use pull-based C&C protocols. Our approach works even when no live C&C server is available for training.
- We have implemented our approach into AutoProbe, a tool that uses a novel combination of dynamic analysis techniques including taint tracking, dynamic slicing, and symbolic exploration for producing accurate and high coverage probe generation, port selection, and classification functions.
- We conduct an extensive evaluation of AutoProbe with real-world malware families. We show that AutoProbe can successfully generate on average 2 fingerprints per malware family (regardless if the remote servers are alive). Furthermore, AutoProbe has successfully and quickly found hundreds of live malware servers on the Internet, most unknown to existing blacklists.

2. PROBLEM STATEMENT AND OVERVIEW

Active probing (or network fingerprinting) is a powerful approach for classifying hosts that listen for incoming network requests into a set of pre-defined classes based on the networking software they run. In a nutshell, active probing sends a probe to each host in a set of targets, and applies a classification function on the responses from each of those target hosts, assigning a class to each host. Given some target network software to detect, a *fingerprint* captures how to build the probe to be sent, how to choose the destination port to send the probe to, and how to classify the target host based on its response.

The problem of active probing comprises two steps: *fingerprint generation* and *scanning*. This paper focuses on the fingerprint generation step, proposing a novel approach to automatically build fingerprints for detecting malware servers. Our approach assumes the availability of a malware sample and applies dynamic binary analysis on the malware to build the fingerprint.

2.1 Motivation

Our program analysis approach to fingerprint generation addresses the following challenges that existing approaches suffer.

Produces valid C&C probes. In existing approaches, the candidate probes to be sent to the remote hosts are manually selected using protocol domain knowledge [7], generated randomly [7], or selected from prior messages the malware has been observed to send [25]. However, these three approaches are problematic. First, domain knowledge is not available for most C&C protocols. Second, randomly generated probes are most likely invalid because they do not satisfy the C&C protocol syntax and semantics. A remote C&C server is likely to refuse responding to invalid probes and the malware owners may be alerted by the invalid requests. Third, previously observed malware requests may be invalid when replayed at a different time and/or machine.

Figure 1 shows a Win32/Horst.Proxy malware request that includes the bot's IP address and an open port where it runs a Socks proxy. If the values of these fields do not match with the sender's, the C&C server can detect such inconsistency and refuse to respond.

```
GET /socks/proxy.php?ip=[redacted]&
    port=41080&os=XP&iso=USA&smtp=0 HTTP/1.1
User-Agent: Mozilla/5.0
Host: ldark.com
```

Figure 1: Request of Win32/Horst.Proxy

In another example, Win32/ZeroAccess [40] encodes the bot's IP address and OS information in an obfuscated URL (Figure 2). Identifying state-dependent fields, even when obfuscated, represents a great challenge for existing network-based approaches [7, 25].

Explores the space of valid C&C probes. CyberProbe is limited to using probes that have been previously observed being

```
GET /EA702A3CFE24B8EBEDAA47374020E6[REDUCTED]
    HTTP/1.1
User-Agent: Mozilla/5.0
```

Figure 2: Request of Win32/ZeroAccess

sent by the malware. However, those requests are often only a small subset of all probes the malware can generate.

For example Win32/Dirtjumper [1] uses a time-dependent algorithm to generate the filename. Without extracting the request generation logic from the malware, it is almost impossible for network-based approaches to produce all possible valid requests.

Minimizes false positives. One goal of *adversarial fingerprint generation* is to minimize the amount of traffic that needs to be sent to remote C&C servers during fingerprint generation. As a consequence, few responses might be available to build a signature on the response. When faced with insufficient training data, machine learning approaches can introduce false positives. Instead, AUTOPROBE leverages the intuition that the malware that produces the request knows how to check if the received response is valid. By examining the malware's request handling logic AUTOPROBE identifies the checks the malware performs to determine if the response is valid, which AUTOPROBE uses as a signature that minimizes false positives.

Does not require a live C&C server. Network-based approaches to fingerprint generation [7, 25] assume that at least one request-response interaction between malware and a C&C server has been captured on a network trace. However, an analyst often only has a malware sample that when executed no longer successfully connects to a live C&C server. That does not mean the operation to which the malware belongs no longer exists. Most often, the malware sample is simply old and tries to connect to dead C&C servers that have since been replaced with fresh ones. AUTOPROBE is able to generate fingerprints even when there is no known live C&C server from the malware family of interest to experiment with. The produced fingerprints can be used to scan for fresh servers that may have replaced the old ones.

2.2 Problem Definition

This paper addresses the problem of *automatic fingerprint generation*. Given a malware sample P from a malware family F the goal of automatic fingerprint generation is to automatically produce a *fingerprint* ϕ that can be used to scan for malicious servers belonging to family F located somewhere on the Internet. We assume the server-side code is not available in any form. The malware sample is provided in binary form with no source code or debugging symbols. We assume the malware sample initiates a set of requests S to contact its malicious servers.

A fingerprint comprises three elements: a *port selection function*, a *probe generation function*, and a *classification function*. AUTOPROBE builds these 3 functions using dynamic binary analysis on the malware sample.

The malware may select to which port to send a probe based on its local environment and the C&C server to be contacted, e.g., based on the time when the probe is sent and the C&C's IP address. Thus, the port selection function takes as input the local environment of the scanner host where it is executed and the target address to be probed. It returns the TCP or UDP port to which the probe should be sent.

The probe generation function takes as input the local environment and the target address to be probed and outputs the payload

```
if(InternetOpenUrl(handle, url_str) == VALID) {
  if(!HttpQueryInfo(handle, HTTP_QUERY_STATUS_CODE,
       &status)) {
      if (status != HTTP_STATUS_OK)
        return ERROR;
  }
  if(!HttpQueryInfo(handle, HTTP_QUERY_CONTENT_LENGTH,
       &length))
     return ERROR;
  while(length) {
     InternetReadFile(handle, lpBuffer, &bytes);
     sscanf(lpBuffer, "<a>%d</a>", &command);
     if (command <= 3 && command > 0) {
        ... //
     }
     length -= bytes;
  }
}
```

```
S1 = get_from_header(STATUS_CODE)
S2 = get_from_header(LENGTH_CODE)
S3 = get_payload()

S1 == 200 &          // Status code is 200
S2 >= 0   &          // Response has payload
(SEARCH(S3, "<a>1</a>") |
 SEARCH(S3, "<a>2</a>") |
 SEARCH(S3, "<a>3</a>") ) // Contains string
```

Figure 3: Classification function example.

of the probe to be sent to the target address. Building the probe generation function comprises two steps: (i) Identify the variant and invariant fields of each request r the malware sends. (ii) For each variant field, generate a re-generation logic which determines the value of the field based on the local environment of the scanner host and the target's address.

The classification function is a boolean function that takes as input the response from a target server, the local environment, and the target's IP addresses. It outputs true if the received response satisfies the checks that the malware performs on the response, which means that the target server belongs to family F. If it outputs false, the target server does not belong to family F. We verify that the malware sample performs checks on the response to determine that the response is valid. Otherwise, the probe is discarded as its response does not allow to classify target servers with certainty and would introduce false positives.

The classification function is a conjunction of boolean expressions corresponding to validation checks the malware performs on a received response. It can be expressed on the raw byte string or on the protocol fields if the C&C protocol is known, e.g., HTTP. In the latter case it is used with a protocol parser. An example classification function is shown in Figure 3. The malware checks that the response is successful (200 status code), that there is an HTTP body, and that the HTTP body contains one of three command strings.

2.3 Approach Overview

Figure 4 shows the architecture of AUTOPROBE. It comprises 4 phases: *malware execution*, *probe generation*, *classification function construction*, and *probing*.

Malware execution. AUTOPROBE first runs the malware executable inside an execution monitor that introspects the execution, monitors the system and API calls the malware uses, and produces an instruction-level trace of the execution. The execution monitor is implemented at the hypervisor-level so that the malware executing in the guest OS cannot interfere with it. The execution monitor is located inside a contained network environment that proxies

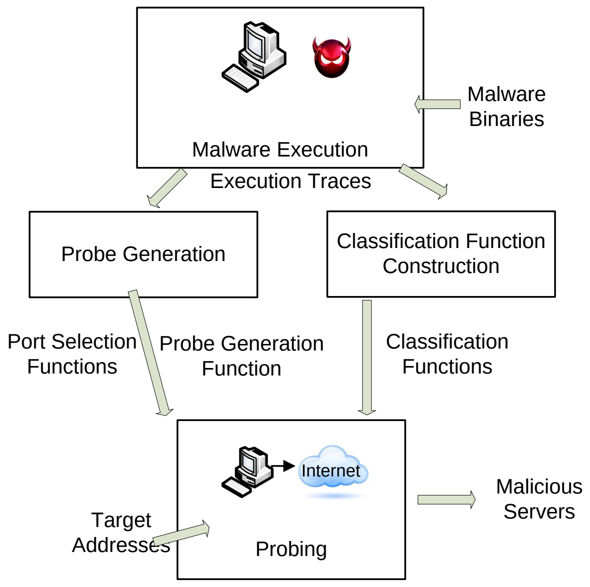

Figure 4: System architecture of AutoProbe.

communications to the Internet. The DNS proxy forwards DNS requests from the malware to the Internet. To incite the malware to start a C&C connection, if the DNS resolution fails, the DNS proxy creates a dummy response that points to a sinkhole server. For other TCP and UDP traffic AutoProbe uses whitelists to determine if the connection is considered benign and should not be analyzed (e.g., connection to top Alexa sites used by malware to check for connectivity) or if it is a C&C connection.

Probe generation. The probe generation phase analyzes the logic that the malware uses for (1) selecting the port to which the request is sent, and (2) generating the request. Both steps leverage backwards taint analysis, dynamic slicing, and symbolic execution techniques. Using these techniques AutoProbe identifies how the port parameter passed to the `socket` function and the buffer passed to the function that sends the request (e.g., `send`) are generated from the output of prior system calls. For each variant part in the request (or for the port number) the goal of this phase is to output a regeneration slice that can produce a new value based on the local host environment and the target's address. Since the malware may obtain the value of a variant field using some multi-path logic, not fully observable in a single execution, we develop a *control-flow-based exploration* technique that finds all paths that affect the generation of a variant field. We detail probe generation in Section 3.

Classification function construction. To build the classification function AutoProbe analyzes the logic the malware uses to validate the received response. Intuitively, invalid responses from target servers that do not belong to the malware family should fail the validation and force the malware to behave differently, e.g., close the connection or resend the request.

If during malware execution the C&C servers that the malware tries to connect were all down, AutoProbe uses a combination of two techniques: *response fuzzing* and *symbolic execution exploration*. The goal is to find the most effective symbolic equation for classifying server's response.

In the case when the malware execution phase captured at least one response from a remote server, AutoProbe tries to identify if the response is from a C&C server or other type of benign server, e.g., a sinkhole or a server that happens to be reusing the IP address previously assigned to a C&C server. For this it compares the malware's processing of the response from the remote server with the malware's processing of a random (i.e., invalid) response. If they are similar the collected response is likely invalid and can be ignored as it most likely comes from a benign server. Otherwise it is a valid C&C response and can be used to guide the symbolic execution exploration.

We detail the classification function construction in Section 4.

Probing. The probing phase takes as input the target IP ranges to probe (e.g., the currently advertised BGP ranges) and the fingerprint. It uses the port selection and probe generation functions to send the probe to a target, and applies the classification function on the response, determining if each target is a server of the malware family of interest. We detail the probing phase in Section 5.

3. PROBE GENERATION

The probe generation phase comprises 2 main steps: *control-flow-based exploration* and *trace analysis*. The control-flow-based exploration component executes multiple paths in the malware's request generation logic to identify different requests the malware may generate (Section 3.1). aThe trace analysis component identifies the variant parts of a request, identifies their semantics, and produces regeneration slices for them (Section 3.2). These two steps output the port selection function and a classification function that captures the valid requests the malware may generate.

3.1 Control-flow-based exploration

One limitation of dynamic analysis is that it only analyzes one execution path in the malware's request generation logic. The analysis of a single execution typically captures a large number of different requests that the malware can generate by modifying the values of variants fields in a request. However, it cannot capture different requests that the malware may generate depending on control-flow decisions on the running environment, i.e., on the output of system calls. Figure 5 illustrates this problem.

The malware checks the existence of a registry key using the `RegOpenKeyEx` function (line 3). If the call fails, the HTTP GET request sent by the malware contains a URL formatted according to line 2. But, if the call succeeds, the malware modifies the URL format by appending an additional parameter value to the end of the URL (lines 5-6). To understand that the malware can produce two different types of requests AutoProbe needs to explore the two execution paths introduced by the branch at line 3. For this, AutoProbe uses control-flow-based exploration, a technique that modifies the output of system calls that influence the request generation logic.

```
1  int rand_num = sub_0343(time);
2  sprintf(url, "/v1.0.1/?v=3.0&c=%ld", rand_num);
3  if (RegOpenKeyEx(PARA_KEY_PATH)) {
4     RegQueryValue(PARA_KEY_PATH, NULL, data, NULL);
5     sprintf(para, "PARAM=%s", data);
6     strcat (url,para);
7  }
8  if (InternetOpenUrl(handle, url) == VALID) {
9     // Handle Response.
10 }
```

Figure 5: Network request generation logic of Win32/LoadMoney.AF.

Control-flow-based exploration performs a backwards analysis on the execution trace starting at the function that sends the request, e.g., `InternetOpenUrl` on line 8 in Figure 5. For each branch it encounters, it performs a backward taint analysis on the CFLAG register to check if the CFLAGS has been influenced by the output of a system call. If it is not influenced then it keeps processing upwards until it finds the next branch. When it finds a branch that has been influenced by the output of a system call (line 3)[1] it forces the system call to generate an alternative result.

For *alternative result*, we mean forcing the conditional to take the other branch where the path is not explored in the first run. In our example, if in the original trace RegOpenKeyEx returned SUCCESS, it forces the function to return FAILURE so that the other execution branch is executed. This process stops when the beginning of the execution is reached or a configurable maximum number[2] of system-call-influenced branches has been found. The details of control-flow-based exploration is further illustrated in Algorithm 1.

```
$\Theta$: Trace
$ins$: instruction in trace
$\Phi$: Set of Instruction of Conditional Branches
$\Delta$: Set of Labeled System Call Output Memory/Register
$T$: Set of Tainted Memory/Register
$F$: Set of System Calls Affecting Control Flow
$req$: Request Sent by Malware
for $ins_i$ in $\Theta$ do
    if $ins_i$ in $\Phi$ then
        eflags $\rightarrow T$
        Backward Taint eflags
        if tainted $\in \Delta$ then
            Record System Call into $F$
            Clean eflags
        end
    end
end
for $fun$ in $F$ do
    for output:$o_i$ of $fun$'s outputs do
        if $o_i$ changes control flow then
            Rerun malware
            Enforce $o_i$ for $fun$ along execution
            Collect new trace $\Theta_i$ Collect new $req_i$
        end
    end
end
```

Algorithm 1: Algorithm for Control-flow-based Exploration

3.2 Trace Analysis

The analysis of an execution trace that produced a network request comprises 3 steps: identify the variant bytes in the request and the target port, recover the semantics of variant bytes in the request, and generate a regeneration slice for the variant bytes in the request and the port.

Identify variant parts and their semantics. The request is commonly a combination of invariant and variant bytes. To identify variant bytes in the request AUTOPROBE applies dynamic slicing to each of the bytes in the request starting from the function that sends the request. Note that while each byte slice is independent they can be performed in parallel on a single backwards pass on the trace for efficiency. If the slice ends in a fixed constant such as an immediate value or a constant in the data section then the byte is considered invariant. If the slice ends in the output of an API call with known semantics and whose output is influenced by a system call (e.g., rand), it is considered variant. In this case, AUTOPROBE clusters consecutive bytes influenced by the same API call (e.g., all consecutive bytes in the request influenced by rand()) into variant fields. Then it labels those variant fields using the semantic information on the API call collected from public repositories (e.g., MSDN). Some examples of semantic labels are *time*, *ip*, *random*, and *OS version*. In current version of AUTOPROBE, we have semantics information for over 200 Windows system and library calls. The handling of the port selection is similar but it starts at the function that selects the port (e.g., connect, sendto) and since the port is an integer value, AUTOPROBE slices for all bytes that form the integer simultaneously.

Reconstruction slices. For each variant field in the request the probe construction function needs to capture how the variant field needs to be updated as a function of the scanner's environment (e.g., the current time). For this, AUTOPROBE applies dynamic slicing on the previously identified variant bytes. The slice contains both data and control dependencies. For control dependencies AUTOPROBE conservatively includes in the slice the eflags register value for each branch instruction it encounters that may influence the generation of the variant bytes. The slice ends when all variant bytes are traced back to some semantic-known system calls or the trace start is reached. The slice is a program that can be re-executed using the current local environment (e.g., local IP, MAC address, or time) to reconstruct the field value.

4. CLASSIFICATION FUNCTION CONSTRUCTION

To build the classification function, AUTOPROBE conducts dynamic binary analysis on the malware's response handling to extract a set of symbolic equations. Figure 6 depicts the architecture of the classification function construction. The intuition behind this phase is that the malware's processing of a response typically comprises two widely different logic to handle valid and invalid responses (without differentiating them the malware could be controlled by arbitrary messages, which is certainly not desirable by the malware author).

For example, if the response is considered valid, the malware may continue its communication with the remote C&C server, but if considered invalid it may close the communication or re-send the previous request. To verify the validity of a response, the malware parses it and checks the values of some selected fields. Such validation checks are branches that depend on the content of the response. Each check can be captured as a symbolic formula and their conjunction can be used as a classification function.

Therefore, in this step, our workflow is first to generate different responses, such as the responses from live servers and arbitrary message generated by the fuzzing module (the right part of Figure 6). Secondly, based on execution of malware (red devil in Figure 6) with different responses, we apply symbolic execution and path exploration analysis to find what could be the valid response. Lastly, we generate the classification function for each valid request and response repair.

The remainder of this section describes the classification function construction when a C&C response was obtained during malware execution, which is illustrated in the left part of Figure 6 (Section 4.1) and when no response is available, which is illustrated in the right part of Figure 6(Section4.2).

4.1 With a C&C Response

To differentiate valid and invalid responses AUTOPROBE focuses on the differences between validation checks on invalid and valid responses. For example, a valid response will successfully go through all validation checks but an invalid response will fail at

[1]Or an API call known to perform a system call like RegOpenKeyEx
[2]In our implementation, we set 100 for the maximum system-calls influenced branches.

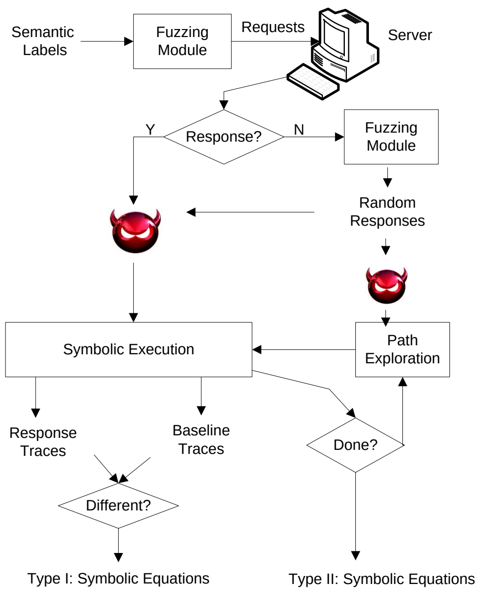

Figure 6: Classification function construction architecture.

least one of those checks producing an execution trace with a smaller number of content-dependent branches.

This case comprises 3 steps shown in the left part of Figure 6. First, AUTOPROBE marks as symbolic each byte in the response received from the server during the original malware execution and performs symbolic execution on those symbols along the execution. For each branch influenced by the input symbols (i.e., validation check), it produces a symbolic expression that summarizes the check. The symbolic execution stops when execution reaches some preselected calls such as `closesocket` and `exitprocess`, or when no validation check is found in the previous n branches, i.e., $n = 50$. In addition to the symbolic formula, AUTOPROBE also outputs a θ_1 forward slice containing all instructions that operate on symbolic inputs.

Second, AUTOPROBE repeats the previous step but this time on a randomly generated (i.e., invalid) response. If the C&C base protocol is known (e.g., HTTP) rather than a random response AUTOPROBE uses a generic error message (e.g., an HTTP 404 response). The outcome is another symbolic expression and a θ_2 forward slice.

Third, AUTOPROBE determines if the θ_1 and θ_2 slices capture the same logic. For this, it aligns them and produces a δ slice, which records the instruction differences. Then it computes the distance between both slices η as:

$$\eta = \frac{\theta_1}{\theta_2} = \frac{\omega_{bn}\Sigma_{bn_1} + \omega_{fn}\Sigma_{fn_1}}{\omega_{bn}\Sigma_{bn_2} + \omega_{fn}\Sigma_{fn_2}}$$

where bn and fn are respectively the number of unique code blocks and unique system calls in δ. Since malware mainly uses system calls to conduct malicious behaviors, we set different weights, such as higher ω_{fn} and lower ω_{bn}, to give preference to unique system calls.

If η is below a predefined threshold m, (experimentally set to 10), the response is discarded since it is handled similarly to the random response and thus is likely invalid. Otherwise, AUTOPROBE considers both executions different and extracts the symbolic execution results, which directly differentiate θ_1 and θ_2, as two sets of equations, S_t and S_n, representing the validation checks results for valid and invalid responses. This step discards unnecessary symbolic equations and reduces the classification overhead.

During probing, AUTOPROBE compares the response from a target server with these two sets of symbolic equations. It determines that the target server is malicious if the response satisfies all symbolic expressions in S_t and none in S_n.

4.2 Without a C&C Response

The malware may not receive any response from the C&C server during malware execution. In this case, AUTOPROBE uses the approach illustrated on the right part of Figure 6, which comprises two steps: *fuzzing responses* and *exposing possible malicious logic*.

The first step is to fuzz the malware with multiple responses. When the C&C protocol is unknown, the fuzzing uses random responses. If the C&C base protocol is known (e.g., HTTP), it starts with a successful response such as 200 OK and then continues with other message types. The payload of the message can be constructed based on some responses from some known benign servers or totally arbitrary bytes. Ideally, malware will not trigger its malicious logic for these responses because they can be considered as *invalid* server's response. Hence, we can select any of these execution trace with request-and-response repair as the baseline of analysis. Therefore, for each pair of responses AUTOPROBE calculates the distance η and finds the pair with the largest η as the baseline of the second step.

In the second step, AUTOPROBE conducts forced execution [41] on all response-sensitive branches. Forced execution is a binary analysis technique which *forces* the program to execute a specific path, exposing more behaviors. Two limitations of forced execution are inefficiency and that the forced execution may not be *reachable* because in a real execution environment the branch condition cannot be satisfied. To solve these issues, we combine symbolic execution with forced execution. In particular, we symbolize each byte in the response and continue online symbolic execution. If we find any branch that depends on the symbolic byte, we record the branch. Then we force execution of the unexplored branch. Next, we calculate the η of the original and forced paths, if η increases, we record the symbolic equation for the forced path. We iteratively continue the exploration finding all symbolic equations that increase η.

As a summary, Algorithm 2 details the process of path exploration.

5. PROBING

Once the fingerprints are generated by AUTOPROBE the next step is to scan networks (e.g., the Internet) looking for malicious servers. For TCP fingerprints, the scanner first performs a horizontal SYN scan to identify hosts with the target port open. For each target host listening on that port, the scanner uses the slices to regenerate the values of the state-dependent fields in the request, sends the updated request to the target, and records its response. UDP fingerprints are handled similarly except that horizontal scanning is not needed.

Our response classification module takes as input the symbolic equations in the fingerprint and the concrete target response, and conducts symbolic-equation-based matching. If the request is generated from our non-response analysis, the detection result is a suspicious score,

$$\lambda = \frac{\#\ of\ matched\ equations}{\#\ of\ equations}$$

The higher λ, the more likely the target server is malicious, and thus worth to conduct further investigation on these remote servers.

```
$\Theta$: Execution Trace Execution
$\Theta_0$: Execution Trace For Random Response
$P$: Malicious Program
$pc$: Instruction Pointer
$S$: Set of Symbolized Set for Response
$\Phi$: Set of Branches Instruction
$\Psi$: Output Symbolic Equations Set
Symbolize all bytes in Response
Running Malware $P$
for $eip$ do
    Enable Forward Symbolic Execution if $eip \in \Phi$ then
        if eflags symbolized then
            Save Execution Snapshot $i$
            Enable Enforced Execution
            Revert eflags
            Disable Enforced Execution
            Monitor Execution and Collect $\Theta_i$
            Calculate $\eta_i$
            if $\eta_i > \eta_0$ then
                Online solving symbols
                if Solvable then
                    Save Trace $\Theta_i$
                    Add Symbolic Equations for $\Theta_i$ to $\Psi$
                end
                else
                    Recover to Snapshot $i$ to $eip$
                    Continue Execution
                end
            end
        end
    end
end
```

Algorithm 2: Algorithm for Path Exploration

Otherwise, if the request is generated from concrete (live) server's response, we require the response to satisfy all the symbolic equations to declare detection.

6. EVALUATION

In this section, we first evaluate AUTOPROBE for generating fingerprints of real-world malware samples. Then, we use the fingerprints to scan for malicious servers.

Malware Collection. We collect recent malware from 56 families broken into two datasets. Dataset I contains 37 popular and notorious malware families including Sality [13], ZeroAccess [40], Ramnit [30], Bamital [4], Taidoor [34]. We are able to collect 10 different variants (with different MD5) for each family from some public malware repositories [22, 27], thus making a total of 370 malware binary samples in Dataset I. Dataset II contains 19 malware families, which have been kindly provided to us by the authors of CYBERPROBE. We use Dataset II to compare the accuracy of the fingerprints produced by AUTOPROBE with the ones produced by CYBERPROBE.

For the malware execution phase we run the malware for 5 minutes each on a virtual machine with Intel Core Duo 1.50GHz CPU and 8 GB memory. Each run outputs an execution trace that serves as the starting point for the fingerprint generation components.

6.1 Evaluation of Probe Generation

In Table 1, we summarize the results from probe generation. We collect malware's execution/network traces and conduct the analysis. First, AUTOPROBE analyzes the network traces, extracts all the malware's network requests, and filter out those requests sent to domains in the Alexa top 10,000 list [2]. The number of Remaining/Original requests are shown in Table 1 in the R/O column. Then, for each dataset, the table splits the malware into two groups corresponding to whether at least one request received a response from a remote server (ResponseSeen), or all requests failed to receive a response (NoResponse). For each group, the table shows the number of requests produced by the malware in the group during the executions and the number of probes produced by AUTOPROBE, split into probes that contain some variable parts and those that have only constant parts. The last column shows the maximum number of probes that CYBERPROBE can produce for the group.

All requests are HTTP and on average it takes AUTOPROBE 13.2 minutes to analyze/process one execution trace, relatively slow but a reasonable cost for off-line analysis tools.

AUTOPROBE generated a total of 105 fingerprints/probes for all 56 malware families in the two datasets. Since multiple requests may be generated by the same execution path, the total number of probes is smaller than the number of requests captured on the network. We also observe that the majority of generated probes contain some variable parts. This means dynamic binary analysis enables AUTOPROBE to extract more complete probe generation functions than network-based approaches, because the variable parts in the probe generation functions provide higher coverage.

Note that on both datasets, AUTOPROBE can generate fingerprints for all the malware, even those with no response, for which CYBERPROBE cannot. This demonstrates a clear advantage of AUTOPROBE. For the samples with a response in Dataset II, CYBERPROBE is able to generate a fingerprint similar to AUTOPROBE. However, for 57% of those, AUTOPROBE produces probes construction functions with variable fields rather than concrete probes in CYBERPROBE. Thus, AUTOPROBE probe construction functions are potentially more accurate. We also find 4 cases in which requests clustered together by CYBERPROBE are indeed generated by different logic in the malware. Thus, they should have been considered different as their responses are not guaranteed to have the same format.

6.2 Evaluation of Classification Function

In this section, we first verify how our heuristics of classification function work in the real world, i.e., whether malware behaves differently when fed with *valid* and *invalid* responses. To verify that, we extract all 76 probes that trigger responses from the live remote servers. We also generate 76 random responses, which comprise of HTTP 200 response code, a common HTTP header and some arbitrary bytes in the payload. We feed our generated responses to malware and compare malware execution with the cases when the valid response from live remote servers is received. Among all 76 test cases, we find that in 71 cases (93%) malware has noticeable behavior differences (malware will typically execute over 10 more system calls and over 50 more code blocks when receiving valid responses). Then we manually examine the remaining 5 exceptional cases. It turns out that all these remotes servers are not malicious any more: four of them are verified as sinkhole domains and the last one returns a 404 error response (possibly server already cleaned). From this experiment, we reasonably believe that our heuristics work well for most of malware communications.

In our evaluation, AUTOPROBE generates a total of 70 classification functions for all *ResponseSeen* cases and 31 for the 29 *NoResponse* cases. The reason why we have more classification functions than the number of cases is because some malware probes can generate different responses to trigger different malware behaviors. This further demonstrates the advantage of AUTOPROBE because existing work cannot generate such probing.

The matching efficiency is important for the classification function. For the *ResponseSeen* cases, the detection requires that all symbolic equations in the classification function match, so

			Probe Generation Functions				
Dataset	**Type**	**Malware Families #**	**R/O**	AutoProbe **Probes**	**Variable**	**Constant**	CyberProbe
I	ResponseSeen	24	45/74	39	22(56%)	9 (23%)	N/A
I	NoResponse	13	167/167	14	11(78%)	2 (14%)	0
II	ResponseSeen	9	113/183	37	21(57%)	16(43%)	37(100%)
II	NoResponse	10	121/121	15	8(54%)	7 (46%)	0

Table 1: Probe generation results.

AutoProbe can finish matching when any of the equations fails to match. For the *NoResponse* cases, it calculates the suspicious score based on the matching results for all equations. For efficiency, our scanner records the response traffic and conducts offline matching.

Table 2 summarizes the classification function efficiency. It shows the time consumed for classifying 1,000 responses. For the *ResponseSeen* cases, on average, the classification function consists of 17 equations and takes 251 ms to complete the matching. The worst case is one classification function that consists of 36 equation comparisons (CP) and takes 757 ms to parse 1,000 responses. For the best case, it takes 9 comparisons and 102ms to finish the matching. For the *NoResponse* cases, a classification function typically contains more equations than the *ResponseSeen* cases (50 on average) and takes 973 ms on average to complete the matching. For the best case, the matching takes 37 comparisons and 483ms to obtain the result. Overall, when classifying responses from Internet-wide scanning (Section 6.6), our classification component takes an average of 5 hours to analyze 71 million responses.

6.3 Case Studies

In this section, we study some probes generated by AutoProbe for real-world malware samples.

Bamital. Bamital is a malware family involved in click-fraud. The probe generation component identifies three variable parts in the initial C&C request (Figure 7): (1) requested file name: `m.php` (2) `os` field which is obtained from the system call `GetVersionEx` (3) `host` field which is the output of a customized domain generation algorithm (DGA).

```
GET/[%1]?subid=61&pr=1&os=20&id=8BBFF356C9BA
905540BBB48D98C90697&ver=[%2] HTTP/1.0
Host: [%3].info
User-Agent: Mozilla/4.0 (compatible; MSIE
7.0; Windows NT 5.1)
Pragma: no-cache

[%1] = slice_0(random)
[%2] = slice_1(os_version)
[%3] = slice_2(time)
```

Figure 7: Probe for Batimal Trojan

During malware execution no C&C server response was observed, as the C&C servers were no longer alive. However, by feeding the malware with a `HTTP/1.1 200 OK` response, AutoProbe is able to analyze the malware's logic, which searches for the strings `<a>` and `<b>` in the response and eventually constructs new requests to download binary files. The produced classification function requires a successful connection with `200` status code and the presence of the string `<a>[.*]</a>` and `<b>[.*]</b>`. If a response to a probe satisfies those constraints, the sender is classified as a Bamital C&C server.

Taidoor. Taidoor is a malware family that has been used in targeted attacks [34]. Its C&C is also built on top of HTTP. The first state-dependent field is the URL filename, which is randomly generated with its length limited to 5 characters. The `id` URL paramater value is built from the output of the `GetAdaptersInfo` library call, used to obtain the host's MAC address. When malware parses the response, the malware uses the value of the `id` field (the MAC address) as the key to decode the response, which introduces a strong correlation between the request and the response. The classification function comprises two steps: decode the data using the request's `id` as key, and check that the decoded data is a valid ASCII string.

Sality. For Sality, AutoProbe identifies 3 HTTP probes for files `spm/s_tasks.php`, `logos_s.gif` and `231013_d.exe`. For the request of the `231013_d.exe` executable, the downloaded file will be directly executed. The classification function considers the set of three file requests and responses. Any server hosting files at those URLs will be considered a Sality server.

Other Malware. For Xpaj.B, AutoProbe generates one HTTP `POST` request with an encoded string, such as `POST /tRHmgD?kjBQMgpwJFLP=QOrbhqDjVeJmN`. The classification function looks for the string `"filename="` at the beginning of the response. For ZeroAccess AutoProbe produces an HTTP probe for the `links.php` file. The malware visit all URLs in the response. The classification function flags the target host as a ZeroAccess server if the response contains a list of URLs.

6.4 Scanning Setup

We conduct network scans using 5 machines. All machines run GNU/Linux Ubuntu 12.1 LTS with dual core 2.2 GHz CPUs and the memory configuration ranges from 2 GB to 16 GB.

6.5 Localized Scanning

As mentioned earlier, AutoProbe generated totally 105 probes for 56 malware families. To test the effectiveness of these probes, we select 28 malware families for localized probing test.

Target network range. We first scan the network ranges that have been observed in the past to host some malicious servers. According to the provider locality property of malicious servers found in [25], these network ranges are more likely to find malicious servers than other regions on the Internet. We start with a seed set of 9,500 malware server IPs collected from MalwareDomainList.com as well as the IP address of the malicious servers detected in [25]. We then expand the IP list to include their network neighbors, i.e., those in the same /24 subnets and those from the BGP route information[3]. In this way, we have collected 2.6M IPs for our localized scanning.

Result. Table 3 details the 28 localized scans. The left part of the table shows the scan configuration: the scan date, the malware dataset, the target port, the number of hosts scanned, and the number of scanners used (SC). The middle part of Table 5 shows the results: the scan duration, the response rate (Resp., i.e., the percentage of targets that replied to the probe), the number of total malicious servers found, the number of found malicious servers already in the seed set, and the number of new malicious servers (not in the seed set). Through 28 scans, AutoProbe has identified

[3]We obtain the most specific BGP route that contains each seed IP address.

Matching Scheme	Worst (CP)	Worst (ms)	Best (CP)	Best (ms)	Avg. (CP)	Avg. (ms)
ResponseSeen	36	757	9	102	17	251
NoResponse	67	1,923	37	483	50	973

Table 2: Efficiency of Classification Functions (time measured when handling 1000 continuous responses). Here CP denotes the number of equation comparisons.

ID	Scan Date	DataSet	Port	# Scanners	Time	Resp.	Found	Known	New	VT	MD	UQ
1	2013-11-03	II	80	3	2.3h	64%	6	4	2	2	1	0
2	2013-11-03	II	80	3	2.4h	64%	4	3	1	0	0	0
3	2013-11-03	II	80	3	2.4h	64%	5	2	3	0	0	0
4	2013-11-03	II	80	3	2.3h	64%	4	2	2	0	0	0
5	2013-11-03	II	80	3	2.8h	64%	2	2	0	0	0	0
6	2013-11-03	II	80	3	3.2h	64%	9	4	5	1	0	0
7	2013-11-08	II	80	3	2.6h	63%	2	2	0	1	0	0
8	2013-11-08	II	80	3	2.7h	63%	1	1	0	1	1	0
9	2013-11-08	II	80	3	1.2h	63%	0	0	0	0	0	0
10	2013-11-08	II	80	3	1.8h	63%	0	0	0	0	0	0
11	2013-11-10	I	80	2	3.3h	64%	32	12	20	1	0	0
12	2013-11-10	I	80	2	3.8h	64%	12	3	9	1	1	0
13	2013-11-10	I	80	2	4.1h	64%	3	0	3	0	0	0
14	2013-11-10	I	80	2	3.2h	64%	3	1	2	1	0	0
15	2013-11-10	I	80	2	3.8h	64%	17	4	13	2	0	0
16	2013-11-10	I	80	2	3.9h	64%	5	4	1	0	0	0
17	2013-11-10	I	80	2	3.6h	64%	9	5	4	0	0	0
18	2013-11-10	I	80	2	3.2h	64%	11	4	7	1	1	1
19	2013-11-10	I	80	2	3.3h	64%	0	0	0	0	0	0
20	2013-11-10	I	80	2	3.5h	64%	4	2	2	0	0	0
21	2013-11-10	I	80	2	3.3h	64%	3	1	2	1	1	0
22	2013-11-10	I	80	2	3.7h	64%	0	0	0	1	0	0
23	2013-11-10	I	80	2	3.1h	64%	8	8	0	1	1	0
24	2013-11-10	I	80	2	3.0h	64%	1	1	0	0	0	0
25	2014-02-17	I	80	2	3.6h	60%	11	3	8	3	0	0
26	2014-02-17	I	80	2	3.9h	60%	7	5	2	1	2	1
27	2014-02-17	I	80	2	4.1h	60%	4	3	1	3	1	1
28	2014-02-17	I	80	2	3.8h	60%	9	5	4	3	2	0
				TOTALS:			172	81	91	24	11	3

Table 3: Localized Scanning Results of AUTOPROBE.

a total of 172 malicious servers among which 81 are known (in the seed set) and 91 are new (previously unknown) malicious servers. We compare our results with some existing malicious domain blacklists, namely VirusTotal [36] (VT), Malware Domain List [23](MD), and URLQuery [35](UQ). The best coverage is achieved by VirusTotal, which knows 14.1% of the servers found by AUTOPROBE (24/172). URL Query knows 11(6.39%) servers and Malware domain list knows only 3(0.02%) malicious servers. In this case, AUTOPROBE detects 6 times more malicious servers than the best of these blacklist services, clearly demonstrating that AUTOPROBE is an effective scheme for detecting malicious servers. On average, AUTOPROBE can efficiently scan 2.6 million IPs with two parallel scanners in 3 hours.

6.6 Internet-wide Scanning and Comparison with CYBERPROBE

We next conduct Internet-wide scanning and compare the results with CYBERPROBE. To minimize the impact to the whole Internet because of our scanning while still clearly verifying the effectiveness of AUTOPROBE, instead of scanning all fingerprints, we focus on three malware families (soft196, ironsource, optinstaller) also scanned by CYBERPROBE [25].

Since these 3 malware families use HTTP C&C, we first perform an Internet-wide horizontal scan of hosts listening on the target port 80. For the horizontal scan, we collect the BGP table from RouteViews and compute the total number of advertised IP addresses. We conducted two horizontal scans on November 4, 2013 and February 19th, 2014. Both are summarized in Table 4. We limit the scan rate to 60,000 packets per second (pps) for good citizenship. The scan takes 2.9 hours and we find over 71 million live hosts listening on port 80.

After obtaining this 71 million live HTTP server list, we performed 3 scanning using our AUTOPROBE and a copy of CYBERPROBE obtained from the authors (together with the fingerprints) for the three selected malware families. Table 5 summarizes the comparison. The top part of the table has the results for the CYBERPROBE scans and the bottom part the results for AUTOPROBE. Each row corresponds to one scan. The scan identifiers (CP-x for CYBERPROBE and AP-x for AUTOPROBE) imply different setup for this experiment. Similarly as in the localized scanning, we also compare our results with popular blacklist databases, VirusTotal (VT) [36], Malware Domain List (MD) [23] and URLQuery (UQ) [35] in the right part of Table.

The results show that for every malware family the fingerprints produced by AUTOPROBE find more servers than the one produced by CYBERPROBE. Overall, AUTOPROBE has found 54 malware servers, versus 40 malware servers found by CYBERPROBE, which represents a 35% improvement. Finally, we also conduct five additional Internet-wide scans for probes that cannot be generated by CYBERPROBE, i.e., those from the `NoResponse` malware server cases. The result is summarized in Table 6. As we can see, AUTOPROBE can detect 83 malware servers, with most of them (80%) are new servers. Compared with CYBERPROBE, which cannot generate any probe for *NoResponse* cases, AUTOPROBE clearly has unique advantage and complements existing work well.

HID	Type	Start Date	Port	Targets	# Scanners	Rate(pps)	Time	Live Hosts
1	I	2013-11-04	80	2,528,563,104	4	60,000	2.9h	71,068,585 (2.8%)
2	I	2014-02-19	80	2,659,029,804	4	50,000	3.5h	71,094,003(2.8%)

Table 4: Horizontal scanning results.

ID	Scan Date	Port	Fingerprint	SC	Time	Resp.	Found	Known	New	VT	MD	UQ
CP-1	2013-11-06	80	soft196	2	24.6h	91%	9	8	1	1	0	0
CP-2	2013-11-06	80	ironsource	2	24.6h	92%	11	7	4	4	1	0
CP-3	2013-11-08	80	optinstaller	2	24.6h	90%	20	4	16	6	0	0
				CYBERPROBE **TOTALS:**			40	19	21	11	1	0
AP-1	2013-11-08	80	soft196	2	25.3h	90%	13	8	1	3	1	0
AP-2	2013-11-08	80	ironsource	2	25.3h	92%	17	6	4	9	2	0
AP-3	2013-11-08	80	optinstaller	2	25.3h	90%	24	5	16	9	2	0
				AUTOPROBE **TOTALS:**			54	19	21	21	5	0

Table 5: Comparison of malware servers found using AUTOPROBE and CYBERPROBE for three malware families. Here CP-x denotes CYBERPROBE and AP-x denotes AUTOPROBE.

False positives and false negatives Given the lack of perfect ground truth, to measure our false positives we check whether the server can successfully trigger client-side malware's malicious logic and establish successful communication with the remote server. Hence, for each detected server, we conduct another round of verification by redirecting malware's request to the detected servers and monitor malware's execution afterwards. If the malware's execution goes into the behaviors we found in the analysis phase, we think it is true positive case. In our test, we do not find any false positive case. To measure false negatives, we use the detection result of CYBERPROBE as the ground truth.[4] The result shows that AUTOPROBE can correctly detect all the results in CYBERPROBE using different signatures for the same families. We further discuss potential false positives and false negatives in Section 7.

7. LIMITATION AND DISCUSSION

We now discuss limitations and possible evasions of AUTOPROBE.

Possible False Positive and False Negative. As discussed in Section 6, we do not find any false positive and false negative cases in our detection result. We think it is because we apply very strict criteria to determine whether it is a malicious server or not. For example, we ensure the response can indeed trigger malware to download malicious file or send some response. However, since our criteria of detecting malicious server purely depends on malware's behaviors, lacking of full and precise understanding of malware logic may mislead our detection. For example, malware may download one malicious file from the server and its continual logic may depends on the success of downloading. However, if our analysis tool cannot capture malware's behavior using such file in the limited monitoring time, AUTOPROBE may directly treat any server hosting this file as the malicious one. We think the root cause of such false positive/negative is because the limitation of dynamic analysis: we can only observe partial result of malware logic. To improve and provide more accurate result, we should provide more analysis time and more code coverage measurement in the real world deployment.

Malware checks on responses. Our classification function construction assumes that the malware will behave differently when receiving valid and invalid responses from remote servers. If the malware violates this assumption, i.e., performs no checks or only cursory checks on the responses, the generated fingerprints may produce false positives when probing benign servers. However, this situation does not arise in our examples and we believe it is unlikely as it would be extremely easy to infiltrate such C&C protocol.

Classification function through code reuse. The classification function produced by AUTOPROBE is a logic expression applied on the response or the output of a parser on the response. Those expressions are difficult to extract if the variables follow non-linear relations. In those cases we could apply binary code reuse techniques [5, 20] to directly (re)use the malware's reponse handling code. In the extreme case, AUTOPROBE could rerun the malware in the controlled environment on the responses received from target servers. Obviously, such approaches are expensive, so they are better used only when our current approach cannot determine a symbolic expression.

Semantics-guided fuzzing. The fingerprints produced by AUTOPROBE use valid probes that satisfy the C&C protocol grammar because the probe construction functions that generate them have been extracted from the malware's request generation logic. However, for some families it may be possible to generate additional fingerprints using invalid probes that do not satisfy the C&C grammar but still trigger a distinctive response from the C&C servers. Invalid probes are easier to be identified by the C&C server managers but may be useful when the C&C masks as a benign protocol. When a live C&C server is known, AUTOPROBE could be enhanced with a semantics-guided fuzzing approach that uses the semantic information extracted during probe generation to modify valid probes into invalid and test them against the C&C server.

Dynamic analysis limitations. The dynamic analysis techniques used by AUTOPROBE are known to have some limitations. For example, dynamic taint analysis is known to be vulnerable to over-tainting and under-tainting [33], which may introduce inaccuracies in our detection of variable parts during probe generation. Similarly, symbolic execution is challenging in the presence of complex loops [32] and implicit flows [18], and may explore unreachable paths [33]. We admit all these issues can affect the performance of AUTOPROBE. However, these issues are not specific to AUTOPROBE and affect in some degree all dynamic analysis solutions. More importantly, AUTOPROBE takes steps to minimize the effect of those challenges. For example, AUTOPROBE does not need to analyze the complete malware logic but only its request generation and response handlig logic. It can confirm that paths build requests by monitoring that indeed a request is observed on the network. Furthermore, even if dynamic analysis marks some request parts as variable, AUTOPROBE still

[4]Even though CYBERPROBE also cannot determine its correctness in some cases.

ID	Scan Date	Port	Fingerprint	SC	Time	Resp.	Found	Known	New	VT	MD	UQ
AP-1	2013-11-06	80	Sality	5	12.1h	90%	23	3	20	1	0	0
AP-2	2013-11-06	80	Taidoor	5	13.2h	91%	14	4	10	2	1	0
AP-3	2013-11-08	80	Bamital	5	12.6h	92%	11	1	10	2	0	0
AP-4	2014-02-23	80	Vidgrab	5	13.4h	94%	21	6	15	3	1	0
AP-5	2014-02-23	80	Horst	5	13.9h	94%	13	2	11	2	1	0
				AUTOPROBE **TOTALS:**			82	16	66	8	3	0

Table 6: Additional 5 scanning results of AUTOPROBE for `NoResponse` cases.

does backward slicing on those bytes verifying that they are indeed generated from the output of system/API calls. Clearly, any future advances in dynamic binary analysis will also benefit our approach.

Handling encrypted traffic. In the evaluation, we find around 30% malware samples use encoded packets to communicate with their remote servers. While in current AUTOPROBE we do no decode these encrypted traffic (a common research challenge in this area, and out of the scope of this paper), AUTOPROBE can observe malware's logic of handling *correctly-encoded* response and *incorrectly-encoded* response. In particular, we can generate some random response packet and record the malware execution path, which represents malware's logic of handling invalid packet. If any response packet deviates malware's execution from this path, we think the source of the packet is likely suspicious.

Other possible evasions. Among possible evasions, one is to use some existing exploits as the client request. AUTOPROBE needs to filter out all the requests that exploit remote servers and malware authors could use that to prevent being tampered by AUTOPROBE. However, using exploits for remote communication increases the probability of being detected by existing IDS systems. Another possible evasion is to use coordinated servers since AUTOPROBE does not correlate traffic to different servers. Malware authors may allow one server to receive a request, forward it to another server, and allow the other server to issue commands. This scheme definitely increases the maintenance cost for botmasters. Some existing IDS systems such as BotHunter [15] could complement AUTOPROBE in some situations.

8. RELATED WORK

Research on Internet-wide Probing. Scanning the internet is one way to find large-scale network-level vulnerabilities. Provos et al. scanned Internet to identify vulnerable SSH servers through vulnerability signatures [29]. Dagon et al. [11] scanned DNS servers on Internet to find those providing incorrect resolutions. Heninger et al. [16] scanned the Internet to find network devices with weak cryptographic keys. All these studies apply some widely-known signatures to achieve the purpose.

Different from them, active probing to detect network-based malware has been proposed in several previous work [3, 14, 25, 28, 41]. In [14], Gu et al. proposed to actively send probing packets through IRC channels. Zmap [12] is another internet-wide scanner which is efficient enough to scan the whole internet in less than 45 minutes. However, it targets to test the aliveness of remote hosts instead of detecting possible malicious servers.

PeerPress [41] is one related work that also adopts dynamic malware analysis to find P2P malware's network fingerprints. Nevertheless, as we have stated the difference earlier, the target of such probing is on the malware samples that actively open the port for communication, such as P2P malware and Trojan Horse. AUTOPROBE targets at remote malicious servers and we assume the server-side logic is not available for analysis in collected binaries, a different assumption from PeerPress.

Research on Network Fingerprint Generation. Fingerprinting network applications is a widely studied topic. Botzilla [31] is a method for detecting malware communication through repetitively recording network traffic of malware in a controlled environment and generating network signatures from invariant content patterns. AUTOPROBE has a different goal of fingerprinting malicious servers and adopts binary-level analysis to find the invariant part in packets.

FiG [7] proposed a framework for automatic fingerprint generation that produces OS and DNS fingerprints from network traffic. In contrast, AUTOPROBE applies a different approach for automatic fingerprint generation that takes as input a malware sample and applies dynamic binary analysis on the malware's execution.

Research on Malware Binary Analysis and Protocol Reverse Engineering. There are multiple existing studies that discuss effective and efficient techniques for malware analysis. Such techniques include taint analysis [19, 26], enforced execution [38], path exploration [24], program slicing [5], symbolic execution [37] and trace alignment [17]. AUTOPROBE applies many of these techniques in our new problem domain in a novel way to automatically generate network fingerprints.

Among all studies on binary analysis, protocol reverse engineering work, such as [8–10, 21, 39], is also closely related to AUTOPROBE. We adopt similar approach as in [6] to figure out the semantics meanings of malware's request. However, one difference between AUTOPROBE and existing work is that AUTOPROBE does not attempt to understand the complete protocol of malware's communication, and AUTOPROBE uses many other different techniques to aid the generation of fingerprints.

In short, the above studies are complementary to our work. AUTOPROBE will greatly benefit from the advances in these fields.

9. CONCLUSION

In this paper, we present AUTOPROBE as an automatic framework to generate active probing fingerprints for Internet-wide malicious server detection. Our approach employs dynamic malware analysis to improve the effectiveness and efficiency of existing work. The dynamic analysis can help expose more requests, identify the fingerprint response, and assist in efficient detection. Furthermore, AUTOPROBE proposes new solutions for some real-world challenges such as none-alive servers. We also show that AUTOPROBE can generate more accurate network fingerprints for malicious servers probing. In our extensive Internet-scale scanning, AUTOPROBE outperforms the existing state-of-the-art system in discovering more malicious servers.

10. ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation under Grant CNS-0954096 and the Air Force Office of Scientific Research under Grants FA9550-13-1-0077 and FA-9550-12-1-0077. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF and AFOSR.

This research is also partially supported by the Spanish Government through Grant TIN2012-39391-C04-01 and a Juan de la Cierva Fellowship for Juan Caballero.

The authors would like to thank the systems' administrator at their respective institutions and VirusTotal, Malware Domain List, VxVault for their services.

11. REFERENCES

[1] Dirtjumper. `http://www.infonomics-society.org/IJICR/DirtJumper`.

[2] Alexa Top Domains. `http://www.alexa.com/`.

[3] Ofir Arkin. A remote active os fingerprinting tool using icmp. *;login: The USENIX Magazine*, 27(2), November 2008.

[4] Bamital Malware. `https://now-static.norton.com/now/en/pu/images/Promotions/2013/Bamital/bamital.html`.

[5] Juan Caballero, Noah M. Johnson, Stephen McCamant, and Dawn Song. Binary code extraction and interface identification for security applications. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2010.

[6] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *ACM Conference on Computer and Communications Security*, Chicago, IL, November 2009.

[7] Juan Caballero, Shobha Venkataraman, Pongsin Poosankam, Min G. Kang, Dawn Song, and Avrim Blum. fig: Automatic fingerprint generation. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2007.

[8] Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn 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] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol specification extraction. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2009.

[10] Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang. Discoverer: Automatic protocol description generation from network traces. In *USENIX Security Symposium*, Boston, MA, August 2007.

[11] David Dagon, Chris Lee, Wenke Lee, and Niels 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] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Usenix Security Symposium*, August 2013.

[13] Nicolas Falliere. Sality: Story of a peer-to-peer viral network. Technical report, 2011.

[14] Guofei Gu, Vinod Yegneswaran, Phillip Porras, Jennifer Stoll, and Wenke Lee. Active botnet probing to identify obscure command and control channels. In *Proceedings of 2009 Annual Computer Security Applications Conference (ACSAC'09)*, December 2009.

[15] Guofei Gu, Junjie Zhang, and Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proceedings of USENIX Security'07*, 2007.

[16] Nadia Heninger, Zagir Durumeric, Eric Wustrow, and J.Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, 2012.

[17] Noah M. Johnson, Juan Caballero, Kevin Zhijie Chen, Stephen McCamant, Pongsin Poosankam, Daniel Reynaud, and Dawn Song. Differential slicing: Identifying causal execution differences for security applications. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, 2011.

[18] Min Gyung Kang, Stephen McCamant, Pongsin Poosankam, and Dawn Song. DTA++: Dynamic taint analysis with targeted control-flow propagation. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium*, San Diego, CA, February 2011.

[19] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and Xiaofeng Wang. Effective and efficient malware detection at the end host. In *USENIX Security Symposium*, Montréal, Canada, August 2009.

[20] Clemens Kolbitsch, Thorsten Holz, Christopher Kruegel, and Engin Kirda. Inspector gadget: Automated extraction of proprietary gadgets from malware binaries. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2010.

[21] Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2008.

[22] Malicia. http://malicia-project.com/. `http://malicia-project.com/`.

[23] Malware domain list. `http://malwaredomainlist.com/`.

[24] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring Multiple Execution Paths for Malware Analysis. In *Proceedings of IEEE Symposium on Security and Privacy*, 2007.

[25] Antonio Nappa, Zhaoyan Xu, M. Zubair Rafique, Juan Caballero, and Guofei Gu. Cyberprobe: Towards internet-scale active detection of malicious servers. In *Network and Distributed System Security Symposium*, 2014.

[26] James Newsome and Dawn Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2005.

[27] Offensive Computing. http://www.offensivecomputing.net/. `http://www.offensivecomputing.net/`.

[28] Jitendra Padhye and Sally Floyd. Identifying the tcp behavior of web servers. In *SIGCOMM Conference*, San Diego, CA, August 2001.

[29] Niels Provos and Peter Honeyman. Scanssh - scanning the internet for ssh servers. In *Technical Report CITI TR 01-13, University of Michigan*, October 2001.

[30] Ramnit Malware. `http://en.wikipedia.org/wiki/Ramnit`.

[31] Konrad Rieck, Guido Schwenk, Tobias Limmer, Thorsten Holz, and Pavel Laskov. Botzilla: Detecting the phoning home of malicious software. In *ACM Symposium on Applied Computing*, 2010.

[32] Prateek Saxena, Pongsin Poosankam, Stephen McCamant, and Dawn Song. Loop-extended symbolic execution on binary programs. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2009.

[33] Edward J. Schwartz, Thanassis Avgerinos, and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Proceedings of IEEE Symposium on Security and Privacy*, 2010.

[34] Taidoor Malware. Xpaj.b malware. `http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_the_taidoor_campaign.pdf`.

[35] Urlquery. `http://urlquery.net/`.

[36] Virustotal. `http://www.virustotal.com/`.

[37] Tielei Wang, Tao Wei, Guofei Gu, and Wei Zou. Taintscope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection. In *Proc. of IEEE S&P'10*, 2010.

[38] Jeffrey Wilhelm and Tzi cker Chiueh. A forced sampled execution approach to kernel rootkit identification. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, 2007.

[39] Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. Automatic network protocol analysis. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008.

[40] James 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`.

[41] Zhaoyan Xu, Lingfeng Chen, Guofei Gu, and Christopher Kruegel. Peerpress: Utilizing enemies' p2p strength against them. In *ACM Conference on Computer and Communications Security*, Raleigh, NC, October 2012.