RevProbe: Detecting Silent Reverse Proxies in Malicious Server Infrastructures

Antonio Nappa
IMDEA Software Institute
antonio.nappa@imdea.org

Rana Faisal Munir
U. Politécnica de Cataluña
fmunir@lsi.upc.edu

Irfan Khan Tanoli
Gran Sasso Science Institute
irfankhan.tanoli@gssi.infn.it

Christian Kreibich
ICSI & Lastline
christian@icir.org

Juan Caballero
IMDEA Software Institute
juan.caballero@imdea.org

ABSTRACT

Web service operators set up reverse proxies to interpose the communication between clients and origin servers for load-balancing traffic across servers, caching content, and filtering attacks. Silent reverse proxies, which do not reveal their proxy role to the client, are of particular interest since malicious infrastructures can use them to hide the existence of the origin servers, adding an indirection layer that helps protecting origin servers from identification and take-downs.

We present RevProbe, a state-of-the-art tool for automatically detecting silent reverse proxies and identifying the server infrastructure behind them. RevProbe uses active probing to send requests to a target IP address and analyzes the responses looking for discrepancies indicating that the IP address corresponds to a reverse proxy. We extensively test RevProbe showing that it significantly outperforms existing tools. Then, we apply RevProbe to perform the first study on the usage of silent reverse proxies in both benign and malicious Web services. RevProbe identifies that 12% of malicious IP addresses correspond to reverse proxies, furthermore 85% of those are silent (compared to 52% for benign reverse proxies).

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection;
H.3.5 [**Online Information Services**]: Web-based services

Keywords

Reverse Proxies; Active Probing; Web Load Balancers

1. INTRODUCTION

Attackers constantly look for mechanisms to protect their server infrastructure. A popular approach introduces indirection by adding intermediate layers of servers that forward traffic between clients (e.g., bots, victims) and *origin servers* (e.g., C&C, exploit servers). Those intermediate servers stay exposed to clients but hide origin servers that attackers closely manage and that store the most sensitive data. Such intermediaries can serve as disposable resources since they do not store essential information and can run on infected machines or cheap cloud VMs. Furthermore, they can distribute traffic across multiple origin servers, making the malicious server infrastructure more resilient to take-downs. In contrast, origin servers are more valuable, should remain hidden, and may use more expensive "bullet-proof" hosting.

Miscreants can employ different types of intermediate servers to introduce indirection in malicious infrastructures. Botnet operators leverage publicly reachable computers in the botnet to serve as proxies [35, 36], HTTP redirectors are widely used in drive-by downloads [43], and traffic delivery systems (TDSes) aggregate traffic towards exploit servers [39]. *Reverse proxies* form another type of intermediate servers, set up by Web service operators to interpose the communication between clients and origin Web servers. They split the communication into two TCP connections: client to reverse proxy and reverse proxy to origin server. They differ from *forward proxies* (that reside in a local network and mediate communication of local clients to any Web service) and from *ISP proxies* (that intercept all HTTP traffic from ISP clients to any Web service) in their specificity to one Web service and their role as Web service endpoints. Benign Web services and content delivery networks (CDNs) employ reverse proxies for tasks such as load balancing [25], caching [25], and filtering attacks [50].

Silent reverse proxies are characterized by the fact that they do not reveal their proxy role (e.g., by introducing a Via header in the HTTP response [32]) and thus hide the existence of origin servers behind them. HTTP redirectors and TDSes differ from reverse proxies in that they do not hide the IP address of the origin servers, and also in that once the redirection happens the rest of the communication does not go through them. While reports of silent reverse proxies in malicious server infrastructures exist [21], their prevalence remains unknown. Currently, no effective tool exists to identify silent reverse proxies. Existing tools (e.g., [5, 7, 40]) fail to detect silent reverse proxies in common configurations. Furthermore, no tool details the hierarchy of servers hiding behind a silent reverse proxy, e.g., the number of servers behind a load balancer.

In this work we present RevProbe, a state-of-the-art tool for automatically detecting silent reverse proxies and identifying the server infrastructure behind them. RevProbe uses active probing to send requests to a remote target IP address and analyzes the responses looking for discrepancies and leaks indicating that the IP address does not correspond to a single server but to a reverse proxy masking other servers. When it detects a reverse proxy, it outputs a *reverse proxy tree* capturing the hierarchy of servers detected behind the reverse proxy. When possible, the identified servers are tagged with their software package, version, and IP address.

ACSAC '16, December 05 - 09, 2016, Los Angeles, CA, USA

ISBN 978-1-4503-4771-6/16/12...$15.00

DOI: http://dx.doi.org/10.1145/2991079.2991093

We design two novel techniques for detecting silent reverse proxies based on discrepancies they introduce in the traffic: extracting time sequences from the HTTP Date header and using the structure of default error pages to identify discrepancies in Web server software. We have also performed a comprehensive study of existing tools for reverse proxy detection (and Web server fingerprinting) and the techniques they use. We incorporate those techniques as well as our two novel techniques into RevProbe.

Security analysts can use RevProbe to assist malicious infrastructure take-downs, counterintelligence, and attribution. For example, a common form of malicious server take-down is performed voluntarily by ISPs and hosting providers, e.g., when they receive an abuse notification (from security companies, analysts, or affected users) about a server that violates their terms of use [29,41]. Such take-downs typically simply disconnect the server and perhaps store the server contents in case they are later required by law enforcement. However, if the reported server is a reverse proxy, the origin servers are not affected and the confiscated data is not sensitive. We argue that analysts should use RevProbe to identify if an IP address to report is a reverse proxy and that providers should modify take-down procedures for reverse proxies in two ways. First, the provider should take a network trace of the traffic of the reverse proxy before disconnecting it, so that the origin servers can be identified. Second, once the origin servers are identified, the provider should report them to the upstream providers or law enforcement. RevProbe can also be combined with recent active probing tools for detecting malicious servers [42, 51] to determine if identified servers constitute reverse proxies or origin servers.

RevProbe also has important applications in benign server infrastructures. It can be used during penetration testing to identify vulnerable servers hiding behind a reverse proxy, for exposing vulnerabilities introduced by the reverse proxy, for asset management, for auditing Web application firewall rules, for security compliance testing (as the hosting network policy may prohibit reverse proxies), for measuring and optimizing performance, and for cloud cartography [44].

We evaluate RevProbe in controlled configurations to establish ground truth, before unleashing it upon both benign and malicious live websites. To measure its accuracy compared to other reverse proxy detection tools, we first apply RevProbe alongside 6 other tools on 44 silent reverse proxy configurations (corresponding to over 99% of all reverse proxy configurations we observe in the wild, as described below). RevProbe perfectly recovers the reverse proxy tree in 77% of the configurations, and the presence of a reverse proxy in the remaining 23% configurations, significantly outperforming all competing tools. lbmap [40] comes closest, recovering the correct reverse proxy tree in 33% of the configurations and a reverse proxy in another 17%, but it only detects the less popular HAProxy [6] and Pound [20] reverse proxies, missing Apache [2] and Nginx [14] acting as reverse proxies.

Then, we use RevProbe to perform the first study of silent reverse proxies in both benign and malicious Web services. We apply RevProbe on the top 100,000 Alexa domains [1] and 46,731 malicious domains from different sources [3, 12, 22]. Our results show that 12% of active IP addresses in malicious Web infrastructures and 13% in benign infrastructures correspond to reverse proxies. The vast majority of malicious reverse proxies (85%) silently mask origin servers, compared to 52% of benign reverse proxies. Reverse proxies are predominantly used to load-balance traffic among multiple servers in benign infrastructures (85%) while in malicious site the dominant reverse proxy tree configuration (49%) is a reverse proxy with one server behind. More complex hierarchies also exist that represent a small percentage of the total.

In summary, we make the following contributions:

- We present RevProbe, a state-of-the-art active probing tool for detecting silent reverse proxies. RevProbe is the first tool for identifying the server infrastructure hiding behind a reverse proxy.
- We design two novel techniques for detecting reverse proxies based on discrepancies they introduce in the traffic: extracting time sequences from the HTTP Date header and using the structure of default error pages to identify discrepancies in Web server software.
- We perform a comprehensive study of existing tools for reverse proxy detection (and Web server fingerprinting) and the techniques they use. We incorporate those techniques as well as our two novel techniques into RevProbe.
- We compare the accuracy of RevProbe with 6 other reverse proxy detection tools on 44 silent reverse proxy configurations, showing that RevProbe outperforms existing tools.
- We apply RevProbe to perform the first study on the use of silent reverse proxies in malicious (and benign) infrastructures. Our results shows that 12% of malicious active IP addresses host reverse proxies and that 85% of those are silent.

2. OVERVIEW

A proxy is *explicit* if it requires the client application to specify its IP address and port, and *transparent* otherwise. Reverse proxies by definition function transparently because they do not require client configuration. To clients, reverse proxies act as the service's endpoints—for example, if a domain name is used to advertise the service, the domain will resolve to the reverse proxy's IP address. Thus, a reverse proxy looks like the origin server and requires the collaboration of the service owner to be installed.

There exist specialized types of reverse proxies. A *Web load balancer* (WLB) is a reverse proxy that load-balances requests across multiple servers according to some policy (e.g., round-robin, least-loaded) while a *Web application firewall* (WAF) filters requests to remove potential attacks.

Figure 1 (left and center) shows two usage scenarios for reverse proxies. On the left, one reverse proxy is used as a WLB to distribute connections across 3 origin servers on the same local network. The center figure shows a more robust server infrastructure with a line of reverse proxies and WLBs forwarding traffic to origin servers that may be hosted somewhere else on the Internet. Some reports (e.g., [21]) link this kind of infrastructure to exploitation-as-a-service models [34] where the origin servers would be exploit servers and the reverse proxies would hide them from the victims.

2.1 Problem Definition

In this work we develop a tool for detecting silent reverse proxies. We consider two alternative definitions of our problem: strict (simpler) and generalized (harder).

Strict problem definition. Given the IP address and port of a remote Web server $\langle ip, port \rangle$, output true if $\langle ip, port \rangle$ corresponds to a reverse proxy, false otherwise. This definition only identifies the existence of a reverse proxy; it does not attempt to identify the server infrastructure hiding behind it.

Generalized problem definition. Given the IP address and port of a remote Web server $\langle ip, port \rangle$, determine the server infrastructure behind it. The goal is to output a *reverse proxy tree* where each

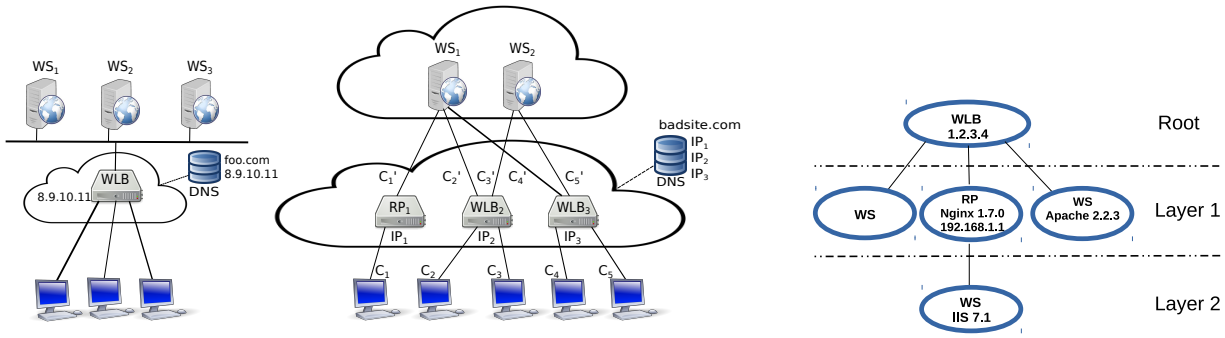

Figure 1: Reverse proxy usage examples (left and center) and a reverse proxy tree example (right).

node corresponds to a Web server. The root node corresponds to the input pair $\langle ip, port \rangle$, internal nodes correspond to other reverse proxies, and leaf nodes correspond to origin servers. A node with children is a reverse proxy and a node with multiple children a Web load balancer. Each node is annotated with its type: reverse proxy (RP), Web load balancer (WLB), or origin Web server (WS). Nodes can have 3 optional attributes: Web server software (e.g., Apache, Nginx), Web server version (e.g., 2.2.3 for Apache), and IP address.

The right-hand side of Figure 1 shows an example reverse proxy tree. It shows that the target IP address (1.2.3.4) corresponds to a WLB that load balances traffic across 3 servers in layer 1. Of those, two are origin servers and another is a reverse proxy to another origin server at layer 2. Some nodes have the optional attributes software package, software version, and IP address.

The goal of RevProbe is identifying whether a given $\langle ip, port \rangle$ corresponds to a reverse proxy (strict definition) and, if so, recover the reverse proxy tree (generalized definition). More specifically, the goal is to recover the shape of the reverse proxy tree. However, RevProbe will annotate the tree nodes with software, version, and IP address if it recovers that information during its processing.

Both problem definitions take as input an IP address. If the input is instead a domain, our approach first resolves the domain into a list of IP addresses. If the input is a URL, it follows all HTTP redirections and then resolves the final domain in the redirection chain to obtain a list of IP addresses. Our reverse proxy detection is then applied separately for each IP address.

A related problem is Web server fingerprinting (WSF) [8, 9], which aims to recover the software package and version running at a given $\langle ip, port \rangle$ endpoint. Current WSF approaches assume the endpoint corresponds to a single server, which is not true with reverse proxies. For example, existing WSF tools will incorrectly label endpoints that correspond to WLBs where responses may come from multiple servers potentially running different software. RevProbe can enable WSF on such cases.

2.2 Approach Overview

We assume only black-box access to the remote Web service identified by $\langle ip, port \rangle$. The code of the Web service is not available in any form. Thus, we use active probing techniques that send requests to $\langle ip, port \rangle$, collect the responses, and infer from those responses the presence of a reverse proxy and the server infrastructure behind it. RevProbe acts as a client that interacts with the Web service over the network.

There exist two general approaches to identify proxies through active probing: *timing-based* and *discrepancy-based*. Timing-based approaches leverage the property that every reverse proxy introduces an additional hop in the communication, which in turn introduces unnatural additional latency from the client's perspective [48]. Discrepancy-based approaches focus on identifying changes that the proxies may introduce in the traffic [49]. We use (mostly) a discrepancy-based approach because timing-based approaches cannot identify multiple servers hiding behind the reverse proxy, e.g., a WLB. They also have problems detecting reverse proxies when the delay they introduce is too small to be detectable across the Internet, e.g., when the reverse proxy sits in the same subnet as the origin server.

In contrast, discrepancy-based approaches can identify multiple servers behind a reverse proxy. However, they may miss a reverse proxy if it does not introduce any changes in the traffic, and miss some origin servers behind a WLB if they are all configured identically. To address some of these cases, RevProbe includes detection modules (e.g., Max-Forwards and phpinfo, detailed in Section 4) that are not based on discrepancies and thus can identify difficult configurations such as a reverse proxy running the same software as the origin server behind.

With discrepancy-based approaches, the more servers hide behind a reverse proxy the easier the strict problem definition gets, since each new server may introduce discrepancies, but the generalized problem definition gets harder as more information needs to be recovered.

3. STATE OF THE ART

We split the state of the art discussion into related published work (Section 3.1) and prior tools for reverse proxy detection, better described separately (Section 3.2).

3.1 Related Work

Weaver et al. [49] propose a technique to detect ISP and forward proxies by installing an application in client hosts and have it communicate with a Web server under their control. Discrepancies between the response sent by their server and the response received by the client application indicate the presence of a proxy. Our work focuses instead on detecting reverse proxies that serve as endpoints of a Web service. In this scenario we do not control the server the client connects to.

In his M.Sc. thesis, Weant [48] proposes detecting reverse proxies through timing analysis of TCP and HTTP round-trip times. Timing-based approaches focus on the strict problem definition and cannot identify multiple servers hiding behind the reverse proxy. They also fail to detect reverse proxies when the delay they introduce remains too small to be detectable across the Internet. Furthermore, Weant evaluates his technique on a single ground truth configuration and does not measure the prevalence of reverse proxies in malicious and benign infrastructures.

	Reverse Proxy Detection									
	Explicit			Silent			WSF			
Tool	**RP**	**WLB**	**WAF**	**RP**	**WLB**	**WAF**	**Exp.**	**Imp.**	**Classes**	**# Req.**
RevProbe	✓	✓	-	✓	✓	-	✓	✓	*	45
Halberd [5]	-	✓	-	-	✓	-	✓	-	*	25,570
Htrosbif [7]	-	✓	-	-	✓	-	✓	✓	75	20
http_trace.nasl [10]	✓	-	-	-	-	-	-	-	-	1
lbmap [40]	✓	✓	✓	✓	✓	✓	✓	✓	6	37
TLHS [33]	✓	-	-	✓	-	-	-	-	-	3
WAFW00f [24]	-	-	✓	-	-	✓	-	-	-	13
ErrorMint [4]	-	-	-	-	-	-	✓	-	*	9
HMAP [38]	-	-	-	-	-	-	✓	✓	28	178
HTTPRecon [8]	-	-	-	-	-	-	✓	✓	460	9
HTTPrint [9]	-	-	-	-	-	-	✓	✓	118	22
http_version.nasl [11]	-	-	-	-	-	-	✓	-	*	2
nikto.nasl [15]	-	-	-	-	-	-	✓	✓	1,250	6,297
Nmap [16]	-	-	-	-	-	-	✓	-	*	1

Table 1: Summary of existing tools for RP detection and WSF fingerprinting.

Feature	**Type**
Header name hash	string
Accept-Ranges	string
Allow	string
Connection	string
Content-Disposition	string
Content-Encoding	string
Content-Language	string
Content-Type	string
Date	datetime
P3P	string
Server	string
Status-Line	string
Transfer-Encoding	string
Upgrade	string
X-Powered-By	string

Table 2: WLB detection features.

Web server fingerprinting. A number of tools exist to fingerprint the program and version run by a remote Web server [8, 9, 26, 38, 45]. Among these, tools like Nmap [16] and ErrorMint [4] fingerprint the program version exclusively by examining explicit program version information provided by the server, e.g., in the Server header and error pages. Other tools like HMAP [38], HTTPrint [9], and HTTPRecon [8] use fingerprints that capture differences between how different Web server versions construct their responses. These type of fingerprints do not rely on the program version information explicitly provided by the server.

Vulnerability scanners such as Nessus [13] and OpenVAS [17] detect vulnerabilities in a variety of software, including Web servers. They first fingerprint the software running at a given endpoint and then look up those software versions in vulnerability databases. Both Nessus and OpenVAS run NASL scripts [31], which exist for WSF [11, 15].

A common limitation of all WSF tools is that they assume the fingerprinted $\langle ip, port \rangle$ endpoint corresponds to a single Web server, which proves incorrect with reverse proxies. When faced with a reverse proxy, they will often recover the software version of the origin server behind the reverse proxy, but they get confused if the reverse proxy manipulates the responses or acts as a load balancer to servers with different software. One of our conclusions in this work is that reverse proxy detection and Web server fingerprinting are best done together.

Automatic fingerprint generation. There exist approaches to automatically build program version fingerprints [26, 27]. Currently, RevProbe uses manually generated fingerprints and could benefit from such approaches. Recent work builds fingerprints for malicious server programs (e.g., C&C, exploit kits) scanning the Internet to locate servers that run them [42, 51]. These tools cannot distinguish between reverse proxies and origin servers in their results and could leverage our approach to this end.

IP disclosure. Recent work by Vissers et al. [46] proposes CloudPiercer, a tool that combines novel and known techniques to disclose the public IP address of DDoS-protected websites. RevProbe also uses some of these techniques (e.g., the phpinfo [19] analysis) to disclose the IP address of servers behind a silent reverse proxy. While for CloudPiercer it remains necessary to disclose a public IP address in order to show that a particular website is vulnerable to DDoS, RevProbe can discover the existence of other IP addresses behind the contacted server to flag the server as a silent reverse proxy. RevProbe could be used to extend CloudPiercer to support cases where the cloud-based DDoS protection services load-balances across multiple origin servers.

3.2 Reverse Proxy Detection Tools

Table 1 summarizes prior reverse proxy detection tools. For completeness it also includes Web server fingerprinting tools described in the previous section. We break the table into 3 blocks: reverse proxy detection, Web server fingerprinting, and number of requests sent by default. Tools with suffix $.nasl$ are NASL scripts [31] for the Nessus [13] and OpenVAS [17] vulnerability scanners. All tools take as input a target IP address or domain, and focus on the strict problem definition. They do not output a reverse proxy tree or recover the number of servers behind a WLB, but in some cases may identify a reverse proxy and its origin server.

The reverse proxy detection block distinguishes between detection of explicit and silent reverse proxies and also separates generic reverse proxy (RP), WLB, and WAF detection.

The WSF block captures if the software information comes exclusively from explicit version information provided by the Web server (e.g., Server header and versions in error pages) or if it is detected without trusting the explicit version information (e.g., using fingerprints). The final column in this block captures the number of classes (i.e., program versions) the tool can identify. An asterisk indicates the tool has no predefined classes, but rather outputs any explicit program version information. The rightmost column captures the average number of requests that the tool sends in default configuration to a target IP.

Next we detail each of the tools, the detection approaches they employ, and a comparison to RevProbe's approach.

Halberd. This tool focuses exclusively on detecting Web load balancers. It sends the same request for a period of 15 seconds to the target IP. Differences in some response headers (e.g., E-Tag, Server) indicate a load balancer. On our tests, it sends on average 25,570 requests, the most of all tools. RevProbe incorporates this technique but it also adds a novel second technique, based on extracting time sequences from the HTTP Date header, to detect WLBs and estimate the number of servers they mask.

Htrosbif and lbmap. These tools are similar. They send abnormal requests (20 and 37, respectively) to the target, trying to elicit a response from any reverse proxies. They use a database of signatures on the responses in order to identify reverse proxy software (i.e., HAProxy, Pound, Vanquish). They differ in the database of requests and signatures. Both fail to detect generic Web server software (e.g., Apache, Nginx) running as reverse proxy. RevProbe also incorporates a module for forcing proxy responses, but uses a novel method to extract fine-grained information from the error pages that also works with Apache and Nginx.

TLHS. Gregoire [33] presents the HTTP traceroute tool (TLHS), which leverages the Max-Forwards HTTP header that limits the maximum number of times a request is forwarded. RevProbe incorporates this technique, which cannot be used in isolation because popular reverse proxy software (e.g., Nginx) ignores the header. In our experiments it only works well with Apache reverse proxies.

http_trace.nasl. This NASL plugin only detects reverse proxies by examining the HTTP Via header, thus it cannot detect silent reverse proxies. RevProbe also includes an explicit RP detection module for completeness, but its goal is detecting silent reverse proxies.

WAFW00f. This tool exclusively detects WAFs. It sends a benign and a malicious request to the same URL. Discrepancies between both responses flag a WAF. This tool incorrectly flags any RP as a WAF and produces false negatives if the WAF does not return an error, but a "200 OK" response with an accompanying error message in the response body. Currently, RevProbe does not differentiate WAFs from other RPs because their fine-grained classification requires sending attacks (or at least attack-resembling requests) to third parties, an act easily perceived as offensive.

In addition to the methods used by the tools above, RevProbe also implements two novel detection techniques, and provides another module that implements a previously known technique not implemented by these tools (i.e., phpinfo [19]).

4. APPROACH

Figure 2 provides an overview of our approach. The preparation module first resolves domains and URLs into IP addresses. For each IP address, RevProbe determines if it corresponds to a reverse proxy by sending probes and examining the responses. If it finds a reverse proxy, it outputs a reverse proxy tree for that IP address.

For each IP address, RevProbe runs a number of detection modules. Each module may send requests to the IP address or simply examine the responses to requests sent by other modules. Each module outputs a, possibly partial, reverse proxy tree. Those trees are combined at the end into a reverse proxy tree for each target IP address. The rest of this section details the modules in Figure 2.

4.1 Preparation

A user may want to run RevProbe on a DNS domain or URL, rather than an IP address. The preparation module obtains a set of IP addresses from those domains and URLs. Then, each IP address is examined independently.

For domains, the preparation module resolves the domain and extracts the list of IP addresses it points to. For URLs, the preparation module fetches the URL and follows all HTTP redirections the URL may produce. From the final URL in the redirection chain, it extracts the domain name and resolves it as above.

After the preparation step, any HTTP connection from RevProbe uses the IP address to connect (i.e., without resolving the domain again) and provides the domain name if known in the Host header, otherwise the IP address.

4.2 Web Load Balancer Detection

The goal of the WLB detector is to detect the presence of a WLB and to provide a *lower bound* in the number of servers that hide behind it. The WLB detector uses three different tests: *same request*, *datetime sequences*, and *load balancer cookie*. These tests are detailed next.

Same request test. The first test is to send the same request multiple times to a given target IP address, monitoring for changes in the responses (e.g., different HTTP Server header values), which may indicate the existence of multiple Web servers and thus the presence of a WLB that forwards requests across them. The test checks for differences in the responses related to a server's configuration. As such, it will fail to detect a WLB or provide a lower bound on the number of origin servers if the origin servers are configured identically. In practice, running a perfectly homogeneous server infrastructure and updating the servers' content simultaneously proves challenging and such differences in configuration abound.

The test sends the same request n times to the target IP address and collects the sequence of responses $\{r_1, \ldots, r_n\}$. It also records the time of each request and response. The larger n, the more confidence in the test results, but the noisier the test results become.

To select the default value of n we test the WLB detection on two different reverse proxy configurations shown in Figure 4: type 2 (center) and type 3 (right). Figure 3 summarizes the results. For trees of type 2, detection is perfect starting with only 2 requests (solid line) because this type of tree has 2 web servers behind the WLB. For trees of type 3 (dashed line), the detection rate increases with n but reaches a plateau between 30 and 60 requests. After $n = 30$ doubling n increases the detection only by 4.1%. Since less queries make the tool faster and less noisy, we select $n = 30$ as default value.

The challenge in building this test is that not all changes in the responses indicate the presence of a WLB as some parts of the response are volatile and change regardless of the request being kept constant, without this indicating the presence of a WLB.

For each response, it first extracts the features in Table 2. The first feature is the hash of the sequence of header names (without value) in the order they appear in the response. All other features correspond to the value of an HTTP header. These features have been selected because their content is related to the server's configuration. The features do not include headers that are not standard [32] (except the popular X-Powered-By), volatile headers that change often (e.g., Set-Cookie), content-related headers (e.g., Content-Length, Etag, Last-Modified) since the content can be dynamic, caching-related headers (e.g., Cache-Control), and proxy-related headers (e.g., Via) that are examined in the explicit detection module (Section 4.3). These features can be of two types: string or datetime. This test focuses on the string features, the second test handles the Date header.

The intuition behind the string features is that their value should be deterministic for the same request and same server. Observing different values in responses to the same request indicates the presence of multiple servers, and thus a WLB. This test outputs the maximum number of distinct values for a feature c. If $c = 1$, no WLB was observed for this test. If $c > 1$, there is a WLB with c servers behind it. Note that if the configuration of a server is changed during the test RevProbe will count the server as two differently-configured servers. The probability of this event can be minimized by reducing the time between requests, at the expense of increasing the load of the target IP, or by repeating the test at a later time.

Datetime sequences test. The second test proposes a novel technique to extract time sequences from the HTTP Date header of the received responses. It does not require sending additional requests, as it examines responses to the former test. The motivation behind this test is that timestamps are a good source of information for identifying servers behind a WLB whose clocks are not synchronized. Multiple interleaved time sequences manifest the presence of a WLB and the number of servers behind can be approximated by the number of distinct interleaved sequences observed.

Algorithm 1 describes the datetime sequence identification algorithm. It iterates over the sequence of responses (line 3). For each

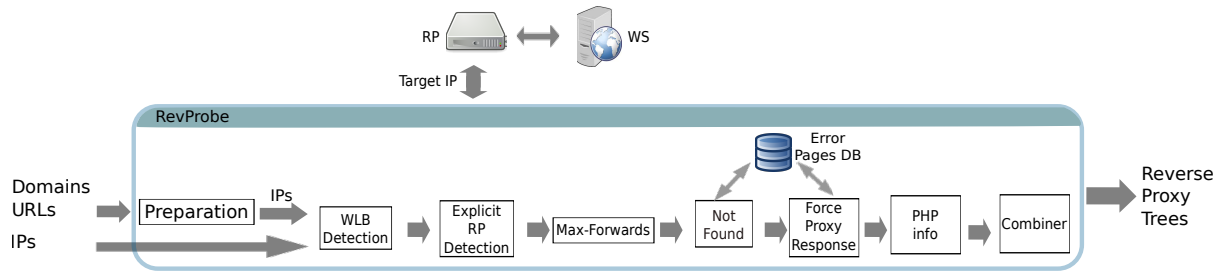

Figure 2: Approach overview.

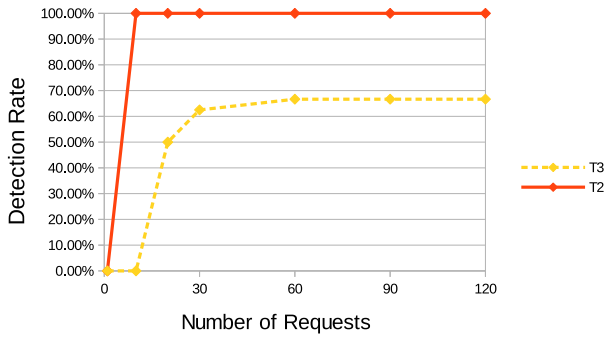

Figure 3: Optimal number of requests for WLB detection. Examples of T2 and T3 trees are depicted in Figure 4.

response, it first obtains the datetime in the Date header (line 4). Then, it checks if the datetime is past, but within a threshold th of the end of any existing sequence (lines 6–12). If so, it adds the response to the current sequence (line 8). Otherwise, it creates a new sequence for the response (lines 13–14). Note that if the date in the current request is in the past of the examined sequence, it cannot belong to that sequence as the Date header should monotonically increase for the same server. The threshold value th is dynamically updated (line 7) based on the time difference between this request and the last entry in the sequence that updated the Date header. The new threshold value is $th = \lceil (\sum_i^j RTT_i) * c \rceil$ where i is the index of the last response in the sequence that updated the Date header value, j the index of the current request, RTT_i the round trip time of request-response pair i, and c a constant factor (by default 2).

Load balancer cookie test. The third test observes that some Web load balancer software introduces cookies in the responses sent by the server for persistence, i.e., to make sure that different HTTP connections of the same Web session are forwarded by the WLB to the same server. The Set-Cookie header from each response obtained from the prior test is checked against the subset of the OWASP cookie database [18]. The OWASP database contains signatures for the cookies produced by different Web applications, including 5 signatures for cookies introduced by commercial load balancers. This test outputs 1 if no WLB is detected and 2 if a WLB is detected indicating that at least two origin servers are identified behind a WLB.

The WLB detection module outputs a tree with a root WLB node and in layer 1 the maximum of the number of servers found by the same request, date sequences, and load balancer cookie tests. If no WLB is found, it outputs a singleton tree with a root WS node.

Algorithm 1 Datetime sequence identification

Input: response sequence, R = $[r_1, \ldots, r_n]$
Output: num_servers $\in [1, \infty)$

```
1:  procedure DATETIME–SEQUENCE
2:      S = {}
3:      for i ∈ [1, n] do
4:          currTime = r_i.getDatetime()
5:          Found = False
6:          for s ∈ S do
7:              th = updateThreshold(s, currTime)
8:              if 0 ≤ (currTime − s.lastTime()) ≤ th then
9:                  s.append(currTime)
10:                 found = True
11:                 break
12:             end if
13:         end for
14:         if Found = False then
15:             S.newSeq(currTime)
16:         end if
17:     end for
18: end procedure
```

4.3 Explicit Reverse Proxy Detection

While detecting silent reverse proxies is its main goal, RevProbe also detects explicit reverse proxies that announce their presence. The explicit RP module does not produce traffic, examining a number of headers in all responses received from a target IP address.

The Via header must be used by proxies to indicate their presence[1] [32]. Each proxy that forwards an HTTP message (i.e., request or response) must append a comma-separated entry to the Via header specifying the protocol version of the message received and the proxy's hostname (or pseudonym). Some explicit RPs use the X-Via header instead. The explicit RP module parses the Via and X-Via headers if they exist to retrieve the list of explicit proxies. In practice, many reverse proxies are silent; they do not append themselves to the Via header and strip these headers. This module also examines the following cache-related headers, useful for detecting caching RPs: X-Served-By, X-Cache, X-Cache-Lookup, X-Varnish, X-Cache-Hits.

This module outputs a tree with a node for each explicit proxy identified, or a singleton WS node if no RP is found.

4.4 Max-Forwards

The Max-Forwards header in an HTTP request can be used to limit the number of proxies that can forward the request [32]. It is set by the source to a maximum number of hops. Each proxy must check its value: if the value is zero the proxy must not forward the request, but must respond as the final recipient; if greater than zero

[1]The X-Forwarded-For header is analogous but only included in requests to track the client's IP address.

it must decrease it and forward the request with the updated Max-Forwards value. Max-Forwards is only required to be supported with the TRACE and OPTIONS methods, but may be supported with other methods such as GET.

This module sends HTTP requests to the target IP each time increasing the value in the Max-Forwards header, from zero to a maximum number of hops (by default 3). We compare each response (starting with value 1) with the prior response. If the two responses have identical values in the string headers in Table 2, no reverse proxy is found and the test exits. If there is a difference, then a reverse proxy is found, e.g., value zero returns a 400 proxy error and value one returns 200 OK. In this case, the Max-Forward value is incremented and the test repeated to check if there may be multiple reverse proxies chained. A limitation of this technique is that some Web servers such as Nginx [14], HAProxy [6], and Pound [20], do not support the Max-Forwards header and always forward the request. Our test uses the GET method as we have experimentally observed that TRACE is often not supported.

This module outputs a tree with a node for each RP identified, or a singleton WS node if no RP is found.

4.5 Error Pages Database

This section describes the error pages database, which is an auxiliary module used by other detection modules, rather than a detection module itself.

The HTML error page in an HTTP error response may explicitly leak information about the server version and even the server hostname (or IP address). For example, some default pages for Apache report the Web server version in the $< address >$ tag. Furthermore, if the server uses the default error page from the Web server software (rather than a customized error page), the structure of the error page implicitly leaks the server's software.

We have built a database of 51 default error pages for popular Web servers. Each entry in the database is indexed by the hash of the sequence of HTML tag and attribute names in the error page. Each entry contains information about the server software the error page corresponds to, and whether some tag in the HTML content stores version or endpoint information.

Every time an HTTP error response is received by any module, RevProbe hashes the error page structure and looks up the hash in the database. If found, it tags the error page with the server software and extracts the explicit version and endpoint information, if any.

4.6 Not Found Module

This module sends a request for a non-existing resource, which triggers a corresponding error. The request will typically be forwarded by the RP and answered by the origin server since only the server knows what content exists. Caching RPs will not find the content in the cache and forward the request as well.

This module identifies a reverse proxy using two methods. The first method uses the error page database to extract (explicit or implicit) software information about the server returning the error page. Then, it compares this information with the Server header in the HTTP response. If it finds a discrepancy it flags a reverse proxy. For example, if the error page contains an $< address >$ tag with an Apache version while the Server header corresponds to Nginx, this indicates the presence of an Nginx RP in front of an Apache server. This method works because some Web proxies (like Nginx) overwrite the Server header of a response, even if the HTTP specification mandates they should not do so [32]. The second method checks if the returned error page contains an explicit hostname (or public IP address), which does not correspond to the target IP. If so, it also flags a reverse proxy and the hostname (or IP address) in the error page identifies the origin server.

This module outputs a tree with a root RP node and one WS node at layer 1 if a RP is found, otherwise a singleton WS node.

4.7 Force Proxy Response

A perfectly silent reverse proxy would forward all requests to the origin server(s). In reality, RP implementations will often parse the requests and do some basic checks on them. Based on those checks they will forward the request or reply to it themselves with an error. When an RP is present, most requests will be answered by the origin server, but the reverse proxy may answer incorrect requests itself.

This module sends a number of requests to the target IP address. We format some of the requests innocuously and some to trigger an error from popular Web servers used as RP. Then, it uses two methods to check for discrepancies between the responses to both types of requests.

The first method uses a fingerprint on the response to an incorrect request. In some cases that response is so specific to a particular Web server that it can be used as a fingerprint for that Web server. RevProbe has such fingerprints for 3 programs that can only act as RP but not as WS: HAProxy [6], Pound [20], and Varnish [23].

The second method looks for Web server software discrepancies. It compares the Web server software information extracted from the response to a proper request, with the same information extracted from the error response to an incorrect request. For the latter, it leverages the error page database. If it finds a discrepancy it flags a reverse proxy. This method is similar to the first method of the not found test in Section 4.5. The difference is that here it operates on software discrepancies found across multiple responses, while in Section 4.5 it focuses on discrepancies within the same response (Server header and HTML content).

This module outputs a tree with a root RP node and one WS node at layer 1 if a RP is found, otherwise a singleton WS node.

4.8 PHPinfo

Web servers that support PHP may serve a phpinfo.php file that the administrators forgot to remove. This file executes the phpinfo function [19], returning a wealth of data about the server that the server thus renders into the response page. The returned data may include the server's IP address in the SERVER_ADDR field. For each target IP, RevProbe tries to fetch this file from 4 common paths. If it finds it, then it checks if the SERVER_ADDR field contains a public IP address that differs from the target IP. If so, this reveals the presence of a reverse proxy, and also deanonymizes the server behind it.

This module outputs a tree with a root RP node and one WS node at layer 1 if a RP is found, otherwise a singleton WS node.

4.9 Combiner

The combiner takes as input the possibly partial reverse proxy trees produced by each of the detection modules and merges them to produce the final tree for each target IP address. In addition to merging the tree nodes, it annotates the nodes with the software package, version, and IP information that may have been recovered by the different tests.

The output of RevProbe is a reverse proxy tree for each target IP that has been flagged as a RP, as well as the list of target IP addresses for which RevProbe found no RP.

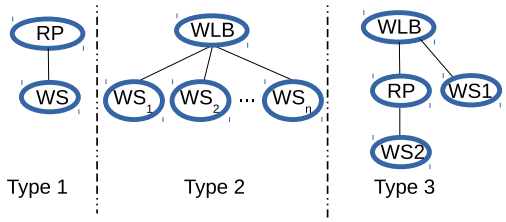

Figure 4: Server trees used for tool comparison. These 3 trees capture 99% of all configurations observed in the wild.

Program	RP	WS	Versions
Apache	✓	✓	2.2.22, 2.4.7
Nginx	✓	✓	1.1.19, 1.6.2
IIS	-	✓	7.5
HAProxy	✓	-	1.4.24
Pound	✓	-	2.6

Table 3: Programs used in tool comparison.

5. EVALUATION

This section evaluates our approach. First, we compare the accuracy of RevProbe with other prior RP detection tools on silent proxy configurations for which we have ground truth (Section 5.1). Then, we test RevProbe on live websites, namely on the top 100,000 Alexa domains and on 46,731 malicious domains (Section 5.2).

5.1 Tool Comparison

To compare the accuracy of RevProbe to other tools, we test them on 44 silent reverse proxy configurations for which we have ground truth. These 44 configurations correspond to variations of the 3 reverse proxy trees depicted in Figure 4. Note that RevProbe can detect arbitrary trees, not only different configurations of these 3 trees. But, we cannot obviously test every possible configuration. We evaluate on these 3 trees because these are the only trees existing tools can detect, and also because our measurements with live sites in Section 5.2 show that these 3 trees capture over 99% of all RP configurations in the wild.

In Figure 4, Type 1 (T1) corresponds to a reverse proxy with one origin server behind, Type 2 (T2) is a WLB that distributes connections to 2 or more origin servers, and Type 3 (T3) is a WLB balancing between one origin server and a reverse proxy that hides another origin server. For each tree, we test multiple software configurations. We use 7 versions of 5 programs, summarized in Table 3. Apache and Nginx can operate as origin server or reverse proxy, IIS only as origin server, and HAProxy and Pound only as reverse proxies. We configure all RPs and WLBs silently. The WLBs use a round-robin policy.

Table 4 summarizes the results. For each of the 44 configurations, it shows the type of tree tested, the server versions at each layer, and the test results for each tool. For T2 trees we use two servers in layer 1. In Section 5.2 we evaluate RevProbe on real services that include T2 trees with more backend servers. For T3 trees the server used as RP in layer 1 is marked with an asterisk.

RevProbe results. In all 32 T1 and T2 configurations RevProbe perfectly recovers the reverse proxy tree. In addition, it also recovers the software package and version of all servers in those trees.

In the more challenging T1 and T2 configurations where the same software is used at both layers, RevProbe successfully recovers the tree leveraging the HTTP Max-Forward technique in configurations with Apache as RP and the date sequencing technique in configurations using when Nginx is used as RP.

TT	Root		Layer 1	Layer 2		RevProbe	Halberd	Htrosbif	http_trace.nasl	lbmap	TLHS	WAFW00f
T1	Nginx	1.6.2	Apache/2.4.7			✓	-	-	-	-	-	-
T1	Nginx	1.6.2	Nginx/1.1.19			✓	-	-	-	-	-	-
T1	Nginx	1.6.2	IIS/7.5			✓	-	-	-	-	-	-
T1	Apache	2.4.7	Apache/2.2.22			✓	-	-	-	-	R	R
T1	Apache	2.4.7	Nginx/1.6.2			✓	-	-	-	-	R	R
T1	Apache	2.4.7	IIS/7.5			✓	-	-	-	-	R	R
T1	HAProxy	1.4.24	Apache/2.4.7			✓	-	✓	-	✓	-	-
T1	HAProxy	1.4.24	Nginx/1.6.2			✓	-	✓	-	✓	-	-
T1	HAProxy	1.4.24	IIS/7.5			✓	-	R	-	✓	-	R
T1	Pound	2.6	Apache/2.4.7			✓	-	✓	-	✓	-	-
T1	Pound	2.6	Nginx/1.6.2			✓	-	✓	-	✓	-	-
T1	Pound	2.6	IIS/7.5			✓	-	R	-	✓	-	R
T1	Apache	2.4.7	Apache/2.4.7			✓	-	-	-	-	R	R
T1	Apache	2.2.22	Apache/2.2.22			✓	-	-	-	-	R	R
T1	Nginx	1.6.2	Nginx/1.6.2			✓	-	-	-	-	-	-
T1	Nginx	1.1.19	Nginx/1.1.19			✓	-	-	-	-	-	-
T2	Nginx	1.6.2	Apache/2.4.7	Apache/2.2.22		✓	W	-	-	-	-	-
T2	Nginx	1.6.2	Nginx/1.1.19	Nginx/1.6.2		✓	W	-	-	-	-	-
T2	Nginx	1.6.2	IIS/7.5	Apache/2.4.7		✓	W	-	-	-	-	-
T2	Nginx	1.6.2	Nginx/1.6.2	Nginx/1.6.2		W	W	-	-	-	-	-
T2	Nginx	1.1.19	Nginx/1.1.19	Nginx/1.1.19		W	W	-	-	-	-	-
T2	Apache	2.4.7	Apache/2.4.7	Apache/2.2.22		✓	W	-	-	-	R	R
T2	Apache	2.4.7	Nginx/1.1.19	Nginx/1.6.2		✓	W	-	-	-	R	R
T2	Apache	2.4.7	IIS/7.5	Apache/2.4.7		✓	W	-	-	-	R	R
T2	Apache	2.4.7	Apache/2.4.7	Apache/2.4.7		✓	W	-	-	-	R	R
T2	Apache	2.4.7	Apache/2.2.22	Apache/2.2.22		✓	W	-	-	-	R	R
T2	HAProxy	1.4.24	Apache/2.4.7	Apache/2.2.22		✓	W	R	-	✓	R	R
T2	HAProxy	1.4.24	Nginx/1.1.19	Nginx/1.6.2		✓	W	R	-	✓	R	R
T2	HAProxy	1.4.24	IIS/7.5	Apache/2.4.7		✓	W	R	-	✓	R	R
T2	Pound	2.6	Apache/2.4.7	Apache/2.2.22		✓	W	R	-	✓	R	R
T2	Pound	2.6	Nginx/1.1.19	Nginx/1.6.2		✓	W	R	-	✓	R	R
T2	Pound	2.6	IIS/7.5	Apache/2.4.7		✓	W	R	-	✓	R	R
T3	Nginx	1.6.2	*Apache/2.4.7	IIS/7.5	IIS/7.5	W	W	-	-	-	-	-
T3	Nginx	1.6.2	*Apache/2.4.7	Apache/2.4.7	Nginx/1.6.2	✓	W	-	-	-	R	-
T3	Nginx	1.6.2	*Nginx/1.6.2	Apache/2.4.7	Apache/2.4.7	✓	W	-	-	-	-	-
T3	Apache	2.4.7	*Apache/2.4.7	Apache/2.2.22	Nginx/1.6.2	✓	W	-	-	-	R	R
T3	Apache	2.4.7	*Apache/2.4.7	IIS/7.5	Nginx/1.6.2	W	W	-	-	-	R	R
T3	Apache	2.4.7	*Apache/2.4.7	Apache/2.4.7	Nginx/1.6.2	✓	W	-	-	-	R	R
T3	HAProxy	1.4.24	*Apache/2.4.7	Apache/2.4.7	Nginx/1.6.2	W	W	R	-	W	R	R
T3	HAProxy	1.4.24	*Nginx/1.6.2	Nginx/1.6.2	Apache/2.4.7	W	W	R	-	W	R	-
T3	HAProxy	1.4.24	*Nginx/1.6.2	IIS/7.5	IIS/7.5	W	W	R	-	W	-	-
T3	Pound	2.6	*Nginx/1.6.2	IIS/7.5	IIS/7.5	W	W	R	-	W	R	R
T3	Pound	2.6	*Nginx/1.6.2	Nginx/1.6.2	Apache/2.4.7	W	W	R	-	W	R	-
T3	Pound	2.6	*Nginx/1.6.2	Apache/2.4.7	Apache/2.4.7	W	W	R	-	W	R	R

Table 4: Tool comparison. For each tool test, a ✓symbol means the reverse proxy tree was perfectly recovered, W that a Web load balancer was detected but not the rest of the tree, R that a reverse proxy was detected but not the rest of the tree, and a - symbol that no reverse proxy was detected. An asterisk before a server at layer 1 means that server was used as a reverse proxy.

Trees of type 3 have mixed results. In 4 out of 12 T3 configurations our tool recovers the perfect reverse proxy tree. For the remaining 8 T3 configurations it identifies that the target IP address corresponds to a WLB (strict problem definition) but it is not able to recover the internal tree structure. The main issue in recovering the internal structure of T3 trees is that only the Max-Forwards and Explicit RP detection modules can recover multiple RP layers. And, some programs ignore the Max-Forwards header completely, so silent RPs at intermediate layers are challenging to recover.

Other tools. Among the other tools, Halberd identifies a WLB in the 24 T2 and T3 trees. However, it fails to identify a RP in the remaining 8 T1 trees because there is only a Web server and, similarly to our WLB detection module, it requires more than one WS to identify a RP. Htrosbif fails to completely detect Apache and Nginx as RP, and it only detects HAProxy and Pound for which it has fingerprints. It does not detect any WLB and fully recovers only 4 T1 trees. http_trace.nasl fails all tests because it only detects explicit reverse proxies and our configurations only use silent reverse

	Domains		IPs		Root			
Source	**All**	**Active**	**All**	**Active**	**RP**	**Explicit**	**Silent**	**WLB**
Alexa Top 100K	100,000	95,713	124,923	97,531	12,522 (12.8%)	5,947 (47.5%)	6,565(52.4%)	10,695 (85.5%)
Malicious	46,731	41,553	26,556	22,529	3,158 (11.9%)	469 (14.1%)	2,689 (85.1%)	1,434 (45.4%)

Table 5: RevProbe results on benign and malicious domains.

proxies. lbmap also fails to detect Apache and Nginx as a reverse proxy in all configurations. It perfectly recovers the reverse proxy tree for T1 and T2 trees with HAProxy and Pound at the root, and the presence of a WLB in T3 trees with HAProxy and Pound at the root. TLHS uses the Max-Forwards header for detection, which works well with Apache acting as RP. Surprisingly, it also detects a RP in some T2 and T3 configurations where Apache is not used as RP/WLB (or not even used at all). This happens because the error page titles are different. WAFW00f focuses on detecting WAFs. None of our configurations has a WAF installed, so these detections could be considered false positives. WAFW00f uses discrepancies to identify WAFs but the technique it uses identifies any RP as a WAF.

Summary. The tool comparison shows that RevProbe outperforms prior RP detection tools. RevProbe perfectly recovers the reverse proxy tree in 77% of the configurations and the presence of a reverse proxy (strict problem definition) in the remaining 23%. None of the prior tools is a close competitor or a clear winner over the others. lbmap perfectly recovers 27% of the configurations and a WLB in another 14%, but it only detects HAProxy and Pound reverse proxies. Htrosbif perfectly recovers 9% T1 trees and a RP in 32% other configurations, but it has no support for WLBs. And, Halberd only detects WLBs. Furthermore, in Section 5.2 we show that RevProbe can detect other rare (<1%) configurations, beyond the 3 tree types in Figure 4, which prior tools do not support.

5.2 Live Websites

We test RevProbe on both benign and malicious live websites to compare how they use reverse proxies. As representative of likely benign websites we use the Alexa top 100,000 domains [1]. For malicious websites we use different sources [3, 12, 22]. For malicious websites, RevProbe uses a virtual private network (VPN) for anonymity. The VPN has a large pool of exit IP addresses, making it difficult to identify RevProbe's probing.

Table 5 summarizes the results. For each dataset, it first shows the number of domains tested and those that resolved to at least one IP address. Then, the number of distinct IP addresses that those live domains resolved to (possibly a larger number) and the number of those IP addresses that were active, i.e., responded to at least one probe. Next, it shows the number of active IP addresses where RevProbe detected a reverse proxy, how many of those root RPs were explicit and silent, and how many were WLBs.

Overall, RevProbe tests 97,531 distinct active IP addresses from Alexa and 22,529 from malicious domains. Among the active IP addresses, RevProbe identifies 12,522 reverse proxies in Alexa and 3,158 in malicious domains. Thus, 12% of malicious and 13% of benign active IP addresses correspond to a reverse proxy. Of the malicious reverse proxies, 85% are silent, compared to 52% for benign RPs. This shows how the vast majority of malicious reverse proxies are silent and used to hide the servers behind them. The fraction is significantly smaller among benign RPs, but still more than half of benign reverse proxies are silent. This may be due to benign services also wanting to hide their server infrastructure and to popular Web server software (e.g., Apache, Nginx) to be silent by default when running as RP.

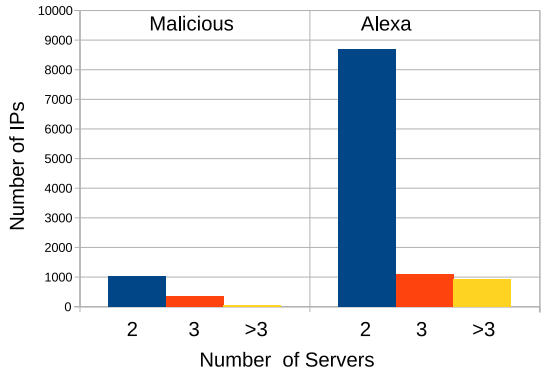

Figure 5: Number of origin servers found behind WLBs.

Of the malicious RPs, 45% are load balancers, compared to 85% of benign RPs. Thus, the vast majority of RPs in benign infrastructures are used to distribute traffic among multiple servers. Malicious infrastructures seem to use RPs for hiding origin servers more than for simple load-balancing. Figure 5 shows the distribution of the number of servers that RevProbe identifies behind WLBs. Of the malicious WLBs 73% have two servers behind, 24% have three servers, and 3% have more than 3. The maximum number of servers behind a malicious WLB is 6. For benign WLBs those percentages are 81% (2), 10% (3), 9% (>3), and the maximum 30. As expected, the WLBs of highly ranked benign Websites distribute their traffic among a larger number of servers than WLBs in malicious infrastructures.

Tree types. Table 7 summarizes the type of reverse proxy trees RevProbe identifies behind the active IP addresses. The most common tree is a WLB with a variable number of servers behind (type 2 in Figure 4 with varying number of servers in layer 1) detected for 45% of the malicious and 85% of the benign active IP addresses. The type 1 tree in Figure 4 (one reverse proxy with a single server) is detected for 49% of the malicious and 14% of the benign active IP addresses. The type 3 tree in Figure 4 (one WLB and a RP at layer 1) occurs in 3% of malicious and less than 1% of benign active IP addresses. These results point to most RPs being the root of simple hierarchies, predominantly T1 and T2 trees. The remaining trees have a root RP, a WLB in layer 1, and 2 servers in layer 2. This shows that other configurations also exist in the wild and are detected by RevProbe.

Web server software. Table 6 summarizes the Web server software that RevProbe identifies in the nodes of the recovered trees. Overall, RevProbe recovers the Web server type for 45% of the nodes in benign trees and 60% of nodes in malicious trees. The most common Web server software in malicious infrastructures is Squid found on 11% of the nodes tagged with Web server software, followed by Nginx (6%), and Varnish (5%). In Alexa domains, Varnish (15%) and Squid (15%) are most common, followed by Nginx (7%), and Apache (5%).

Program	Alexa	Malicious
Apache	1,249 (5%)	181 (6%)
IIS	317 (1%)	47 (2%)
Nginx	1,758 (7%)	195 (6%)
Squid	3,888 (15%)	348 (11%)
Varnish	4,070 (15%)	143 (5%)
Others	452 (2%)	958 (30%)
Null	14,768 (55%)	1,286 (40%)
Total	26,502	3,158

Table 6: Software identified running on the nodes in the reverse proxy trees recovered by RevProbe.

Source	RPs	Type 1	Type 2	Type 3	Oth.
Alexa	12,522	1,731 (14%)	10,695 (85%)	34 (0%)	62 (0%)
Malicious	3,158	1,555 (49%)	1,434 (45%)	107 (3%)	62 (0%)

Table 7: Reverse proxy tree types identified in benign and malicious domains.

The vast majority of servers use open source software. The most popular reverse proxy software correspond to caching proxies (Squid and Varnish). This is expected in benign infrastructures where performance is an important reason of using reverse proxies, but surprising in malicious infrastructures where the RP could cache sensitive information. Different types of malicious domains may behave differently in this regard, e.g., phishing domains are cached but exploit kit data is not.

Deanonymizations. The phpinfo module recovers either public or private IP address of a origin Web server. Using this approach we successfully recover 12 public IP addresses of malicious servers. For the benign domains, we recover the public IP address for 132 public IP addresses and 63 private IP address. To confirm the deanonymizations we connect to the 132 public IP addresses and fetch the root page using the corresponding domain in the HTTP Host header. If the content served is similar to the content served through the RP, the deanonymization is confirmed. Of the 132 benign IP addresses, 41 are confirmed, 62 did not respond, 23 show an error page, and 6 show different content. Overall, RevProbe deanonymizes 12 malicious servers and 41 Alexa servers hiding behind silent reverse proxies.

Summary. Our results show that reverse proxies are common in malicious Web infrastructures (12% of active IP addresses). Those reverse proxies are predominantly silent to hide the existence of servers behind (85%). Reverse proxies are also common in benign infrastructures (13% of active IP addresses) but are less often silent (52%). In benign infrastructures RPs are predominantly used to load balance traffic among multiple servers (85%), while in malicious infrastructures RPs are most often used to protect a single server (49%). The vast majority of RPs are root to simple server hierarchies, predominantly T1 and T2 trees (96%–85%).

6. DISCUSSION

Ethical considerations. Active probing sends traffic to targets that have not solicited it. Thus, some targets may consider the probes undesirable or even offensive. In addition, the probes can potentially place a burden on the target. We take this issue seriously and carefully seek balance between the amount of interaction and the merits of the results. Since we have no visibility into Web services' internals, we believe active probing approaches are required to detect reverse proxies and the server infrastructure behind them.

We have designed RevProbe to send a small number of requests (45 by default) to a remote target, which we believe to be a manageable load even for small websites. The requests sent by RevProbe are all well-formed HTTP requests. Only the "not found" and force proxy response modules send requests designed to trigger an error. For our experiments with third-party Web services we limit the force proxy error module to use a single type of incorrect request, an incorrect HTTP method such as "GOT / HTTP/1.1" rather than "GET / HTTP/1.1".

Modified take-down procedures for proxies. RevProbe can be used during abuse reporting and take-down of malicious Web servers to determine if the reported Web server is a reverse proxy. We argue that take-down procedures by ISPs and hosting providers should be modified to take into account whether the reported abusive server is a proxy. Simply taking down a proxy means little harm to the attacker because the origin servers are not affected, the proxy does not typically store sensitive data that may lead to the owner, and proxies often use cloud hosting services with cheap short-term leases (e.g., one month for $15), so that the attacker loses little money when the server is taken down. We propose that take-down procedures for proxies are modified in two ways. First, the provider should take a network trace of the proxy traffic before disconnecting it, so that the origin servers can be identified. Second, once the origin servers are identified, the provider should report them to the upstream providers, national CERTs, or law enforcement.

Shared servers. RevProbe identifies the server infrastructure hiding behind an IP address. In some scenarios multiple reverse proxies at different IP addresses may proxy to the same origin servers, e.g., with domains that resolve to multiple IP addresses and with Web services pointed at by multiple domains. Currently, RevProbe only detects whether servers identified behind different reverse proxies (i.e., target IP addresses) are the same if it recovers their public IP address. We leave as future work exploring other avenues to combine reverse proxy trees from different target IP addresses.

Incomplete trees. Our evaluation on controlled silent reverse proxy configurations shows that type 3 configurations with 2 layers of proxy servers are challenging to fully recover. Only two of our modules specifically recover sequences of reverse proxies. As next step, we plan to explore other techniques that may be able to detect sequences of reverse proxies, including timing-based approaches. In general, RevProbe cannot always recover a perfect reverse proxy tree, but it is a significant step forward from prior tools that do not address the generalized problem definition.

Combining fingerprinting. Current Web server fingerprinting tools have problems with reverse proxies as they assume an IP address corresponds to a single server. One conclusion of this work is that Web server fingerprinting and reverse proxy detection are best combined together. RevProbe takes a step towards this goal, being able to recover software package information for 40% of all servers. However, we have not yet built a large database of program version fingerprints. We plan to further explore this combination next.

Other protocols. In this work we have focused on HTTP communication, but our techniques should be equally applicable to HTTPS. Active probing approaches that look at discrepancies in the traffic are also applicable to other protocols, but require the protocol grammar. For proprietary protocols (e.g., C&C) the protocol grammar can be recovered by analyzing the network traffic [30] or program executions [28].

Evasion. A malicious Web service can change its response based on parameters of the client such as geographical location, User-Agent, and Referer [37, 47]. To address such cloaking, RevProbe uses a VPN when connecting to malicious Web services, and can be configured to change its HTTP parameters including User-Agent and Referer. To avoid detection, attackers may attempt to remove discrepancies introduced by the reverse proxy. However, complete removal requires deep understanding of the reverse proxy code and configurable options, as well as careful configuration of the origin servers. The difficulty to get all of these perfectly right is a key component of RevProbe's detection.

7. CONCLUSION

In this paper we have presented RevProbe, a state-of-the-art detector for silent reverse proxies that maps the server infrastructure they mask. RevProbe uses active probing to send requests to a target IP address and analyzes the responses looking for indications that this address corresponds to a reverse proxy. When it detects such a proxy it outputs a *reverse proxy tree*, capturing the hierarchy of servers it identified behind the reverse proxy. When possible, it tags the identified servers with their software distribution, version, and IP address.

We have compared RevProbe with prior tools on 44 silent reverse proxy configurations, showing that RevProbe consistently outperforms them. We have also employed RevProbe to perform the first study of the usage of silent reverse proxies in both benign and malicious Web services. Using RevProbe, we find that 12% of malicious and 13% of benign active IP addresses correspond to reverse proxies, that 85% of those are silent compared to 52% for benign reverse proxies, and that reverse proxies in benign server infrastructures load-balance traffic more frequently that those in malicious ones. Finally, we have shown that reverse proxy detection and Web server fingerprinting are best done together to handle cases where an endpoint load balances to multiple origin servers.

Acknowledgments

This work was largely done while Rana Faisal Munir and Irfan Khan Tanoli were interns at the IMDEA Software Institute.

This research was partially supported by the Regional Government of Madrid through the N-GREENS Software-CM S2013/ICE-2731 project, by the Spanish Government through the DEDETIS Grant TIN2015-7013-R, and by NSF grants CNS-1213157 and CNS-1237265. All opinions, findings and conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsors.

8. REFERENCES

[1] Alexa. http://www.alexa.com/.
[2] Apache. http://httpd.apache.org.
[3] Dns-bh malware domain blocklist. http://www.malwaredomains.com/.
[4] Errormint. http://sourceforge.net/projects/errormint/.
[5] Halberd. https://github.com/jmbr/halberd.
[6] Haproxy. http://www.haproxy.org/.
[7] Htrosbif. http://mac.freecode.com/projects/htrosbif.
[8] Httprecon. https://w3dt.net/tools/httprecon.
[9] Httprint. http://www.net-square.com/httprint_paper.html.
[10] Http_trace.nasl. http://plugins.openvas.org/nasl.php?oid=11040.
[11] Http_version.nasl. http://plugins.openvas.org/nasl.php?oid=10107.
[12] Malware domain list. http://www.malwaredomainlist.com/.
[13] Nessus. http://www.tenable.com/products/nessus.
[14] Nginx. http://nginx.org/.
[15] Nikto2. https://cirt.net/Nikto2.
[16] Nmap. http://nmap.org/.
[17] OpenVAS. http://www.openvas.org.
[18] Owasp cookies database. https://www.owasp.org/index.php/Category:OWASP_Cookies_Database.
[19] Phpinfo manual. http://php.net/manual/en/function.phpinfo.php.
[20] Pound. http://www.apsis.ch/pound/.
[21] RIG exploit kit. https://www.trustwave.com/Resources/SpiderLabs-Blog/RIG-Exploit-Kit-%E2%80%93-Diving-Deeper-into-the-Infrastructure/.
[22] Scumware. http://www.scumware.org/.
[23] Varnish. https://www.varnish-cache.org/.
[24] Wafw00f. https://github.com/sandrogauci/wafw00f.
[25] G. Barish and K. Obraczke. World Wide Web Caching: Trends and Techniques. *IEEE Communications magazine*, 38(5):178–184, 2000.
[26] T. Book, M. Witick, and D. S.Wallach. Automated Generation of Web Server Fingerprints, 2013. http://arxiv.org/abs/1305.0245.
[27] J. Caballero, M. G. Kang, S. Venkataraman, D. Song, P. Poosankam, and A. Blum. FiG: Automatic Fingerprint Generation. In *Network and Distributed Systems Security Symposium*, 2007.
[28] 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*, 2007.
[29] D. Canali, D. Balzarotti, and A. Francillon. The Role of Web Hosting Providers in Detecting Compromised Websites. In *International World Wide Web Conference*, 2013.
[30] W. Cui, J. Kannan, and H. J. Wang. Discoverer: Automatic Protocol Description Generation from Network Traces. In *USENIX Security Symposium*, 2007.
[31] R. Deraison. The Nessus Attack Scripting Language Reference Guide, 2000. http://virtualblueness.net/nasl.html.
[32] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999.
[33] N. Gregoire. Traceroute-like http scanner, 2011. http://www.agarri.fr/kom/archives/2011/11/12/traceroute-like_http_scanner/index.html.
[34] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and G. M. Voelker. Manufacturing Compromise: The Emergence Of Exploit-as-a-service. In *ACM Conference on Computer and Communications Security*, 2012.
[35] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *Network and Distributed Systems Security Symposium*, 2008.
[36] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: An Empirical Analysis of Spam Marketing Conversion. In *ACM Conference on Computer and Communications Security*, 2008.
[37] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert. Rozzle: De-Cloaking Internet Malware. In *IEEE Symposium on Security and Privacy*, 2012.
[38] D. W. Lee. HMAP: A Technique and Tool For Remote Identification of HTTP Servers. Master's thesis, University of California at Davis, 2001.
[39] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang. Finding the Linchpins of the Dark Web: A Study on Topologically Dedicated Hosts on Malicious Web Infrastructures. In *IEEE Symposium on Security and Privacy*, 2013.
[40] E. Marcussen. HTTP Fingerprinting - The Next Generation. In OWASP AppSec, 2012.
[41] 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*, 2013.
[42] A. Nappa, Z. Xu, J. Caballero, and G. Gu. CyberProbe: Towards Internet-Scale Active Detection of Malicious Servers. In *Network and Distributed System Security Symposium*, 2014.
[43] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All Your iFRAMEs Point to Us. In *USENIX Security Symposium*, 2008.

[44] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *ACM Conference on Computer and Communications Security*, 2009.

[45] S. Shah. HTTP Fingerprinting and Advanced Assessment Techniques. In *BlackHat Asia*, 2003.

[46] T. Vissers, T. V. Goethem, W. Joosen, and N. Nikiforakis. Maneuvering Around Clouds: Bypassing Cloud-based Security Providers. In *ACM Conference on Computer and Communications Security*, 2015.

[47] D. Y. Wang, S. Savage, and G. M. Voelker. Cloak and Dagger: Dynamics of Web Search Cloaking. In *ACM Conference on Computer and Communications Security*, 2011.

[48] M. S. Weant. Fingerprinting Reverse Proxies using Timing Analysis of TCP Flows. Master's thesis, Computer Science Department, Naval Postgraduate School, Monterey, CA, 2013.

[49] N. Weaver, C. Kreibich, M. Dam, and V. Paxson. Here Be Web Proxies. In *Passive and Active Measurement Conference*, 2014.

[50] P. Wurzinger, C. Platzer, C. Ludl, E. Kirda, and C. Kruegel. SWAP: Mitigating XSS Attacks Using a Reverse Proxy. In *Workshop on Software Engineering for Secure Systems*, 2009.

[51] Z. Xu, A. Nappa, R. Baykov, G. Yang, J. Caballero, and G. Gu. AutoProbe: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis. In *ACM Conference on Computer and Communications Security*, 2014.