Chapter 13

IDENTIFYING AND ANALYZING WEB SERVER ATTACKS

Christian Seifert, Barbara Endicott-Popovsky, Deborah Frincke, Peter Komisarczuk, Radu Muschevici and Ian Welch

Abstract Client honeypots can be used to identify malicious web servers that attack web browsers and push malware to client machines. Merely recording network traffic is insufficient to perform comprehensive forensic analyses of such attacks. Custom tools are required to access and analyze network protocol data. Moreover, specialized methods are required to perform a behavioral analysis of an attack, which helps determine exactly what transpired on the attacked system. This paper proposes a record/replay mechanism that enables forensic investigators to extract application data from recorded network streams and allows applications to interact with this data in order to conduct behavioral analyses. Implementations for the HTTP and DNS protocols are presented and their utility in network forensic investigations is demonstrated.

Keywords: Network forensics, malicious web servers, client honeypots

1. Introduction

Network forensic readiness involves "maximizing the ability of an environment to collect credible digital evidence while minimizing the cost of incident response" [11]. The goal is to simplify network forensic tasks without sacrificing the quality of digital evidence. This can be achieved using specialized techniques and tools as well as by embedding forensic capabilities in networks, thus "operationalizing" network forensic readiness [1].

This paper examines network forensic readiness in the context of malicious web servers. Malicious web servers push malware to client machines – so called "drive-by-downloads" – by exploiting web browsers. A previous study [10] used client honeypots to find malicious servers on

the Internet. However, once identified, the attack origin and mechanism, and the actions performed by the malware could not be explained.

A major challenge is to extract and interact with application data from recorded network streams. In particular, it is difficult to demonstrate and analyze attacks because the streams have to be piped via network channels through the client application to execute the identical code path that made an attack possible. Since the attack source code is not readily available, analyzing system behavior (referred to as "behavioral analysis") is the primary means to infer the inner workings of an attack.

Network and application protocols do not support the replay of network data to analyze how an attack impacted an application or system. This inability has contributed to inadequate network forensic readiness in the context of client-side attacks.

This paper presents a custom solution using web and DNS proxies and demonstrates its utility in network forensic investigations. The solution, however, is specific to the HTTP and DNS protocols [3, 7] and is not easily generalizable. This is why the paper also calls for the support and implementation of a record/replay mechanism in these protocols to provide a generic network forensic solution.

2. Background

This section discusses the problems posed by malicious web servers and the overall lack of forensic readiness to cope with attacks.

2.1 Malicious Web Servers

Our previous study [10] concerned itself with identifying "drive-by-downloads," an emerging type of attack executed by malicious servers on client machines. These attacks target vulnerabilities in client applications and usually alter the state of the client machine without user consent. Typically, the malicious server installs malware on the client machine without the user's knowledge.

Our work concentrated on identifying malicious web servers that attack web browsers. The mere retrieval of a malicious web page with a vulnerable browser results in a successful compromise of the client machine. The web environment was chosen because these attacks are currently the most common type of drive-by-downloads.

We identified malicious web servers using high-interaction client honeypots. Such a honeypot uses a dedicated operating system to drive a vulnerable browser to interact with a potentially malicious web server. After each interaction, the operating system is checked for unauthorized state changes, e.g., new executable files in the startup folder. If any

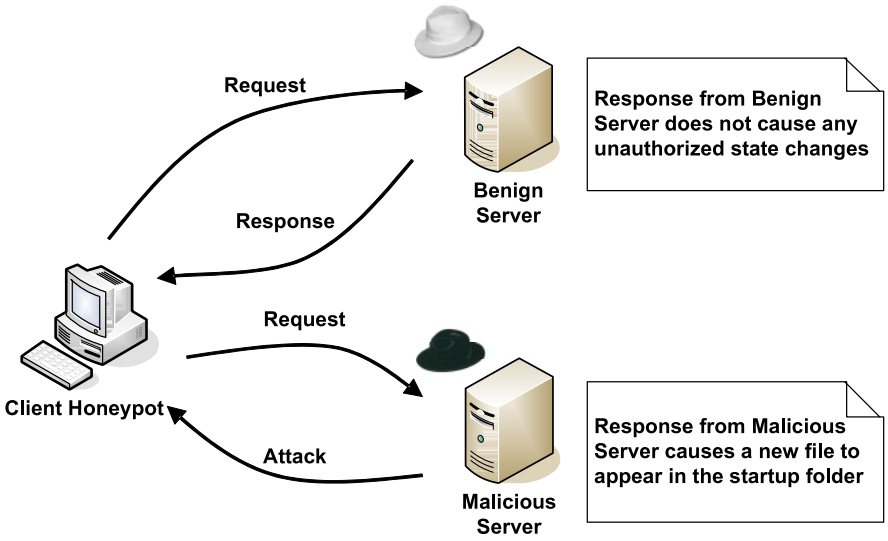

Figure 1. Client honeypot.

unauthorized state changes are detected, the server is classified as malicious (Figure 1).

Twelve instances of a high-interaction client honeypot were used to inspect about 300,000 web pages over a three-week period. A total of 306 malicious URLs were identified that successfully attacked a standard installation of Microsoft Windows XP SP2 with Internet Explorer 6.0 SP2. The malicious servers took control of the machine and primarily installed malware that attempted to defraud the user.

Unauthorized state changes to the client machines were recorded when the client honeypots identified malicious web servers. In addition, network data was collected using the `tcpdump` tool [6] and stored in `libpcap` data files. This data contained all the network traffic sent to and from a client honeypot, including HTTP and DNS requests and responses. Interested readers are referred to [10] for more details about the use of high-interaction client honeypots.

2.2 Forensic Analysis

Analysis of network and application data (DNS records, HTML pages and IP source addresses) helps identify the servers involved in attacks and their role in the attacks. Also, the inspection of HTML pages reveals embedded source code, which could provide information about attack mechanisms.

In general, an attack incorporates an exploit that targets a vulnerability and a payload that is executed after the vulnerability is exploited. Usually, the embedded source code only implements the initial exploit. The payload, which is typically in the form of a binary, requires behavioral analysis to determine how it operates. Behavioral analysis requires the attack code to be executed again on the client machine, but this is difficult to accomplish for several reasons, including server location, server domain name and security context. Opening a web page from a web server is quite different from opening it as a file. A page that is opened from a previously-saved file might not trigger. In fact, to trigger successfully, the attack code has to be sent to the client application via the network as if it originated from the malicious server.

Recorded network data does not lend itself to straightforward forensic analysis. Application data embedded in `libpcap` files has to be extracted using custom tools; but, even then, the data may not directly support behavioral analysis. For example, HTML pages extracted with these tools cannot be fed to a browser to provide information about if and how the attack occurred.

Note that it is difficult to develop an application that interacts with a malicious server in order to analyze an attack. This is because the dynamic nature of the network makes malicious web servers appear different over time. Also, it is often the case that hackers implement fluctuations to hide attack sources and hinder forensic investigations of attacks. As a result, forensic analyses of attacks must be based on the data recorded during the initial identification of malicious web servers.

2.2.1 Replaying Network Data. In order to replay network traffic at the transport layer, recorded packets must be placed back on the wire. The technique involves splitting the network flow into server traffic and client traffic. After one side of the flow, say client traffic, is selectively placed on the wire, the server would have to recognize a request as if it originated from a client and provide the normal response.

Separating the network flow into client and server packets is an easy task. It is done by filtering the network flow by client source or server IP address. However, having the client or server interactively respond to the replayed network traffic is not a capability that is normally supported by the transport layer of a network protocol such as TCP [5] (regardless of whether IPv4 [4] or IPv6 [2] is employed).

Several mechanisms of the TCP protocol are responsible for this. TCP is a stateful protocol that uses a three-way handshake to establish a connection between a client and server. First, the client sends a TCP packet to the server with the ACK flag set. The server acknowledges

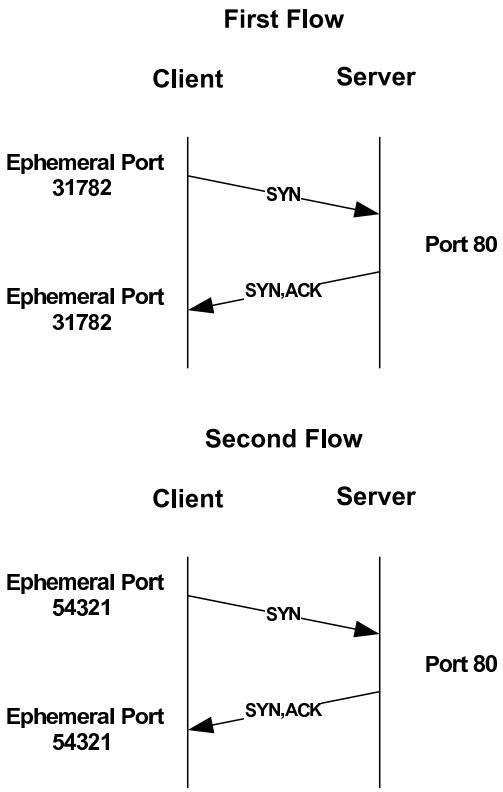

Figure 2. Ephemeral port assignment.

this connection request by sending a TCP packet with the ACK and SYN flags set. The connection is established when the client sends a TCP packet with the ACK flag set. Sequence numbers are exchanged during the handshake to identify the other party in each communication. Thus, connections cannot be established when TCP traffic is replayed by placing only one side of the network flow on the wire.

Another difficulty is posed by ephemeral ports that are created by the client to accept response packets from the server during the process of establishing a connection. In other words, the client application temporarily becomes a server. An ephemeral port is dynamically assigned in the high port range with each connection as shown in Figure 2. This port remains closed when no connection is being established. Replaying network traffic against the client requires matching the temporarily-opened ephemeral port with the destination port specified in TCP packets. Without this matching, the traffic would not reach the client application.

The `tcpreplay` tool [14] can place recorded packets back on the wire, but it does so in a passive manner without modifying the recorded packets to address the TCP handshake and ephemeral port assignment constraints. It places packets on the wire in their original form mainly for the purpose of testing network performance and inline security devices (e.g., firewalls and intrusion detection systems).

2.2.2 Network Fluctuations. The dynamic nature of networks prevents forensic investigators from retrieving the original content from malicious web servers. Having identified a malicious web server, subsequent attempts to interact with the server and retrieve information to support attack analysis are hindered by network fluctuations. In particular, the server may exhibit non-deterministic behavior, providing content that is different from what was originally sent to the client application.

The simplest technique used by hackers to implement network fluctuations is to remove the malicious content from the server. A second mechanism is to manipulate DNS (the service that maps host names to IP addresses of physical machines) to resolve to different physical machines whenever a host name lookup is performed. Such network structures are referred to as fast-flux networks [12]. This makes the attack infrastructure more resilient to failure and also hinders forensic investigations.

A third network fluctuation technique uses a mechanism known as "IP tracking." Exploitation kits deployed on web servers, such as Mpack v0.94 [9], can be configured to trigger only during the initial contact with a malicious web server. Subsequent interactions with the web server from the same IP address provide the identical, albeit benign, web pages. Thus, the malicious web server that launched an attack on the client honeypot appears to be benign to the forensic investigator.

3. Proposed Solution

Our solution engages a record/replay mechanism in which recorded data is played back through the client application (Figure 3). This makes it much easier to extract relevant information from the data. Instead of writing a custom forensic analysis application, the existing functionality of the client application can be used to extract information from network data. Moreover, replaying recorded data through the client application supports behavioral analysis.

As mentioned above, implementing a record/replay mechanism at the network transport layer poses several challenges. Therefore, our solution

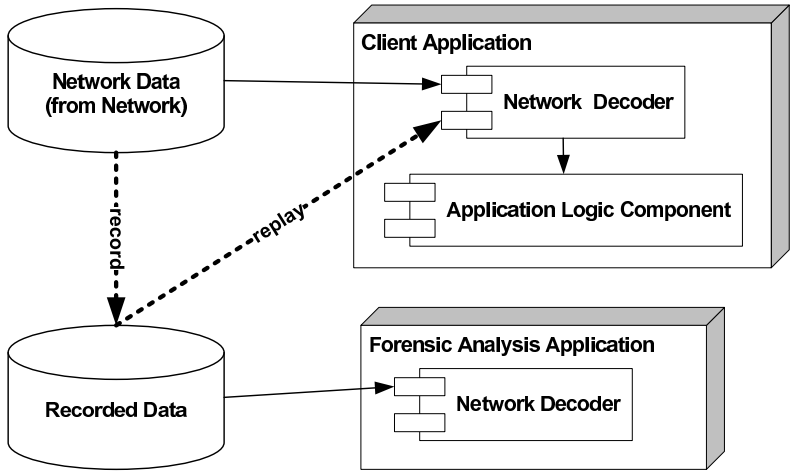

Figure 3. Record/replay mechanism.

uses the application layer to implement record/replay. Specifically, all client application and malicious web server communications are routed through a proxy that records all the application data. The proxy, if instructed to replay the stored data instead of fetching it from the actual server, can repeatedly replay the server responses to the client.

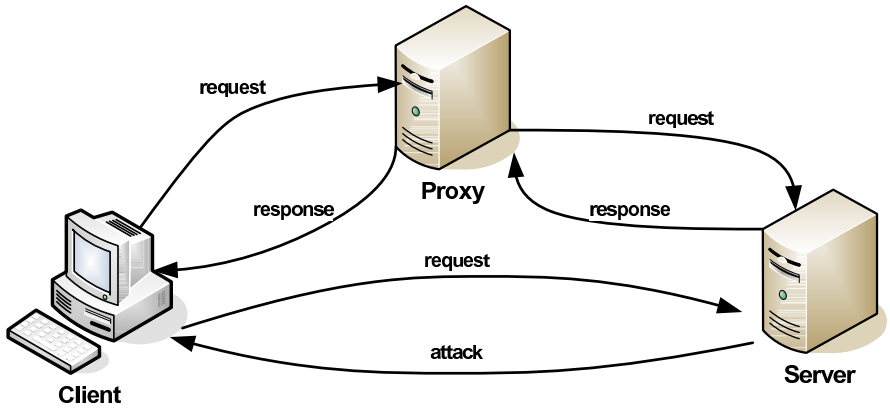

Figure 4. Proxy architecture.

The top portion of Figure 4 presents the proxy architecture. The architecture supports forensic analysis because the proxy server stores all the data during the initial operation of the client honeypot. In particular, it is possible to perform a behavioral analysis of the attack code using a browser. The browser makes the HTTP request, which is routed via the proxy. Since the proxy already knows the response, it returns the server

```
refresh_pattern . 999999 100\% 999999 override-expire
ignore-reload override-lastmode ignore-no-cache ignore-no-store
ignore-private ignore-auth
```

Figure 5. Squid configuration options.

response it has already saved without passing another request to the malicious server. Furthermore, the application data is easily retrieved even though it might have been stored by the proxy in a proprietary format. Browsers and DNS clients can obtain and decode the proxy data. For example, WGET could obtain the HTTP response, and the HOST tool could translate DNS responses stored on the proxy server. Thus, our solution reuses the code of existing tools, eliminating the need to procure custom tools for extracting and analyzing application data.

3.1 Proxy Solution

Web browsing uses two application protocols (HTTP and DNS). Consequently, two proxy solutions are implemented: a web proxy that routes and stores HTTP data and a DNS proxy that performs the same operations on DNS data.

The web proxy relays HTTP data and stores this data in its cache. The caching functionality, which is part of the HTTP/1.1 specification [3], improves response performance and availability, and permits disconnected operation (to some extent). The caching functionality was not designed for forensic purposes, rather to enhance performance and availability. Because of this focus, it is also concerned with data staleness and, therefore, defines a mechanism that checks whether a newer resource is available or whether the resource itself should never be cached. A proxy utilizes constraints on freshness and security/privacy as well as cache correctness.

If a web proxy adheres to these functional requirements strictly, a saved malicious web page might be invalidated by the freshness constraint and fetched again from the server upon a subsequent request. In contrast, our solution attempts to use the web proxy for storage rather than caching without applying the mechanisms defined in the HTTP/1.1 specification. In particular, it uses Squid [15], an open source web proxy implementation. Squid is highly configurable and permits deviations from the HTTP/1.1 specification. In fact, the forensic requirements for the web proxy can be achieved using the Squid configuration settings shown in Figure 5.

DNS proxies, similarly to web proxies, are designed to store DNS responses in their caches for a predefined period of time. Once the validity of a DNS response has expired, the DNS proxy must perform another DNS lookup on the actual DNS server. Again, an implementation is needed that can override this behavior. This can be done using `pdnsd` [8], a simple DNS daemon with permanent caching designed to deal with unreachable or down DNS servers (e.g., in dial-up networking). The purging of older cache entries can be prevented by setting the maximum cache size to a high value (e.g., using the configuration option `perm_cache=204800`).

3.2 Limitations

The proposed solution has some limitations. First and foremost, it is not easily applicable to other network data. The HTTP and DNS protocols are based on a simple request/response model. Since the protocols were designed at a time when dial-up networks dominated, caching proxies were incorporated to conserve resources and increase reliability. Proxy storage capabilities facilitate forensic data collection and analysis; however, they are unlikely to be provided by modern protocols. For example, peer-to-peer protocols and other popular protocols such as SSH do not have a simple request/replay structure, which makes it difficult to offer proxy record/replay capabilities.

Second, the proxy solution does not provide the same interactivity as a real server. State information (e.g., for authentication) is held by the client and is usually conveyed back to the server in the form of a cookie. While a proxy is able to store this information, a client would have to adhere to the same request sequence to solicit the same responses. For example, if a client accesses a web page after authentication, it would have to be re-authenticated before it could access the same web page from the server at a later time; this is because the required authentication information is missing from the request. Furthermore, the proxy solution will not work when encryption is used by the two communicating parties.

Finally, interacting with a server via a proxy might solicit different server responses. This is not a concern in a forensic setting. However, because the data sent to the client is recorded by the proxy, this might pose a problem when searches are performed using a client honeypot. Specifically, a server might check for the existence of a proxy and not behave maliciously if such a proxy is encountered as a precautionary measure. Figure 4 illustrates this situation. The top flow shows a client application interacting with a server via a proxy. The server detects the

proxy set-up and, therefore, delivers a benign web page. The bottom flow shows a client interacting directly with the server. The server does not detect a proxy that potentially records data and believes it is free to launch the attack.

4. Conclusions

Identifying and analyzing web server attacks are difficult tasks due to the lack of forensic readiness of network protocols. Our custom proxy-server-based record/replay solution adds network forensic readiness capabilities to client honeypots. The solution supports the examination of application data by reusing the capabilities of the clients that consume the data. It also permits the data to be sent interactively to client applications to perform behavioral analyses of attacks, which provide a more complete picture of attack mechanisms and impact.

While forensic capabilities have been implemented at the application layer using existing proxy solutions, we believe a generic solution could be implemented at the network transport layer. The difficulty in implementing such a solution is primarily due to the fact that existing protocols were not designed with network forensic readiness in mind. We believe that incorporating forensic requirements during protocol design is instrumental to achieving network forensic readiness.

References

[1] B. Endicott-Popovsky, D. Frincke and C. Taylor, A theoretical framework for organizational network forensic readiness, *Journal of Computers*, vol. 2(3), pp. 1–11, 2007.

[2] S. Deering and R. Hinden, RFC 2460: Internet Protocol Version 6 (IPv6) Specification (www.faqs.org/rfcs/rfc2460.html), 1998.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, RFC 2616: Hypertext Transfer Protocol – HTTP/1.1 (www.ietf.org/rfc/rfc2616.txt), 1999.

[4] Information Sciences Institute, RFC 791: Internet Protocol, University of Southern California, Los Angeles, California (www.faqs.org/rfcs/rfc791.html), 1981.

[5] Information Sciences Institute, RFC 793: Transmission Control Protocol, University of Southern California, Los Angeles, California (www.faqs.org/rfcs/rfc793.html), 1981.

[6] V. Jacobson, C. Leres and S. McCanne, `tcpdump` (www.tcpdump.org).

[7] P. Mockapetris, RFC 1035: Domain Names – Implementation and Specification (www.ietf.org/rfc/rfc1035.txt), 1987.

[8] T. Moestl and P. Rombouts, `pdnsd` – Proxy DNS server (www.phys.uu.nl/∼rombouts/pdnsd/index.html).

[9] C. Seifert, Know your enemy: Behind the scenes of malicious web servers (www.honeynet.org/papers/wek), 2007.

[10] C. Seifert, R. Steenson, T. Holz, Y. Bing and M. Davis, Know your enemy: Malicious web servers (www.honeynet.org/papers/mws), 2007.

[11] J. Tan, Forensic readiness (www.arcert.gov.ar/webs/textos/forensic_readiness.pdf), 2001.

[12] The Honeynet Project and Research Allicance, Know your enemy: Fast-flux service networks (www.honeynet.org/papers/ff/fast-flux.pdf), 2007.

[13] A. Turner, `flowreplay` design notes (synfin.net/papers/flowreplay.pdf), 2003.

[14] A. Turner, `tcpreplay` (tcpreplay.synfin.net/trac).

[15] D. Wessels, H. Nordstroem, A. Rousskov, A. Chadd, R. Collins, G. Serassio, S. Wilton and C. Francesco, Squid web proxy cache (www.squid-cache.org).