Justifying the need for forensically ready protocols: a case study of identifying malicious web servers using client honeypots

Chapter 1

JUSTIFYING THE NEED FOR FORENSICALLY READY PROTOCOLS: A CASE STUDY OF IDENTIFYING MALICIOUS WEB SERVERS USING CLIENT HONEYPOTS

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

Abstract Client honeypot technology can find malicious web servers that attack web browsers and push malware, so called drive-by-downloads, to the client machine. Merely recording the network traffic is insufficient to perform an efficient forensic analysis of the attack. Custom tools need to be developed to access and examine the embedded data of the network protocols. Once the information is extracted from the network data, it cannot be used to perform a behavioral analysis on the attack, therefore limiting the ability to answer what exactly happened on the attacked system. Implementation of a record/ replay mechanism is proposed that allows the forensic examiner to easily extract application data from recorded network streams and allows applications to interact with such data for behavioral analysis purposes. A concrete implementation of such a setup for HTTP and DNS protocols using the HTTP proxy Squid and DNS proxy pdnsd is presented and its effect on digital forensic analysis demonstrated.

Keywords: Security, Digital Forensics, Client Honeypots, Network Forensics

1. Introduction

Network forensic readiness (NFR) is a concept that is targeted at "maximizing the ability of an environment to collect credible digital evidence while minimizing the cost of an incident response" [10]. Digital network forensics should become easier without sacrificing the quality of the digital evidence. The effort to perform a forensic analysis should decrease while at the same time maintaining the level of credibility in

the digital evidence collected. This can be achieved through specific tools, checklists, etc., but also through embedding forensic capabilities into networks, thus "operationalizing" NFR [1].

This paper examines NFR in the context of malicious web servers. Malicious web servers are servers that push malware, so called drive-by-downloads, to a client machine by exploiting vulnerabilities of web browsers that access them. In a previous study [14], client honeypots were used to find these malicious servers on the Internet. However, once identified, the attack origin and mechanism as well as the actions performed by the malware could not be explained. The forensic tasks to answer these questions will be the focus of this paper.

The forensic task highlighted several issues in NFR, specifically the ability to extract and interact with application data from the network streams recorded. In particular, one cannot demonstrate and analyze the attack, which needs to be piped via network channels through the client application in order to execute the identical code path that made the attack possible. This is necessary because source code of the attack is not readily available, and observing system behavior, so called behavioral analysis, is the primary means to infer the inner workings of the attack.

Recorded network data does not allow replaying the attack against the application. Replaying recorded network data isn't functionality that is supported by the network and application protocols. The authors view the lack of this functionality as the root cause of the missing network forensic readiness in the context of client-side attacks. This paper presents a custom solution using web and DNS proxies and illustrates its effects on the forensic analysis. The proposed solution, however, is specific to the HTTP and DNS protocol [3, 8] and not easily generalizable. That is why the authors call for the support and implementation of a record/ replay mechanism in these protocols to provide a generic solution to the problem.

This paper is structured as follows. Section 1.2 recaps the previous study on malicious web servers and the issues identified in more detail. Section 1.3 introduces the solution and section 1.4 presents the conclusion.

2. Case Study

This section summarizes the previous study on malicious web servers followed by a description of the issues encountered that demonstrate the lack of forensic readiness.

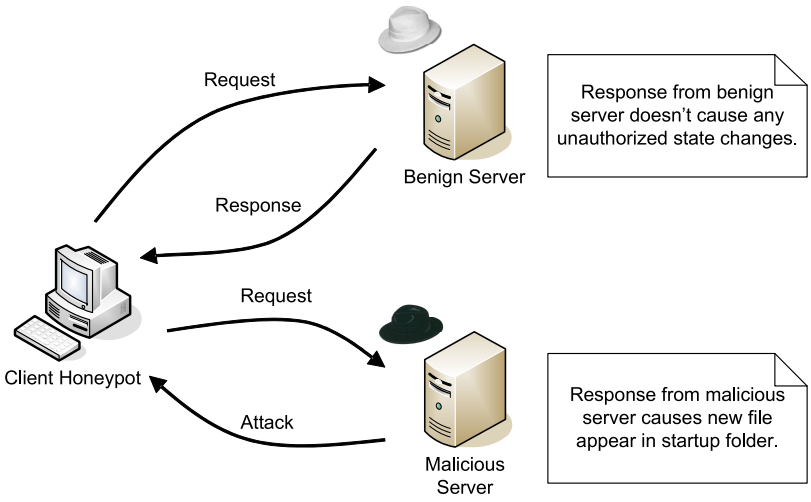

Figure 1. Client Honeypot

2.1 Overview

The previous study [14] concerned itself with identifying an emerging type of attack that occurs on the Internet: drive-by-downloads. Drive-by-downloads are client-side attacks that originate on malicious servers. These attacks target vulnerabilities of client applications and usually can alter the state of the client machine without user consent. Typically this means that the server is able to control and install malware on the client machine without the user noticing such actions.

This study concentrated on finding malicious web servers that attack web browsers. A mere retrieval of a malicious web page with a vulnerable browser would result in a successful compromise of the client machine. A web setting was chosen because those types of attacks seem to be the most prominent drive-by-download attack type today.

Malicious web servers were identified using high interaction client honeypot technology. Using a dedicated operating system, a high interaction client honeypot drives an actual vulnerable browser to interact with potentially malicious web servers. After each interaction, it checks the operating system for unauthorized state changes, such as a new executable file in the startup folder. If any unauthorized state changes are detected, the server is classified as malicious as shown in the example in Figure 1.

Using 12 instances of a high interaction client honeypot, about 300,000 web pages were inspected over a three-week period. 306 malicious URLs were identified that successfully attacked a standard installation of Mi-

crosoft Windows XP SP2 with Internet Explorer 6.0 SP2. The malicious servers took complete control of the machine and primarily installed malware that was targeted at defrauding the victim.

As client honeypots identified malicious web servers, unauthorized state changes were recorded that occurred on the client machine. In addition, all the network data with the tool Tcpdump [7] was recorded that resulted in network libpcap data files. This data contained all the network traffic that was sent to, and originated from, the client honeypot including the HTTP and DNS request/ responses. More information about the study can be found in the research paper [14].

2.2 Forensic Analysis Issues

As part of the forensics analysis, the data that was sent to the client was to be analyzed. The network and application data (DNS records, HTML pages, IP source addresses) would allow the identification of the servers involved in the attack and their role in the attack. Potentially by inspecting the HTML page, one could obtain information about the mechanism of the attack if the source code was embedded in the page.

An attack is comprised of an exploit that attacks the vulnerability and the payload that is executed upon successful exploitation. Usually source code is only available for the initial exploitation code. The payload is usually not available in source code form and requires behavioral analysis to determine how it operates. Behavioral analysis would require the attack code to execute once again on the client machine. Because of various factors, such as server location, the server domain name, and security context, execution of the attack code is not easy to do. Opening a web page from the web server, compared to opening it as a file, is quite different. The attack code has to be sent to the client application via the network as if it originates from the malicious server itself for it to successfully operate. If a page is opened from a previously saved file, it might not trigger.

Collected network data does not lend itself to a straightforward forensic analysis. The application data is embedded in the libpcap files and needs to be extracted via development of custom tools. Once extracted, however, the data does not allow for a behavioral analysis to take place. For instance, while it is possible to extract HTML pages with the custom tools, it is not possible to feed these pages into the browser in a way to analyze whether and why the attack took place. A description on why the network data could not simply be replayed can be found below in section 1.2.2.1.

The reader might question why the application could not simply interact with the actual attack server to analyze the attack. The answer lies in the dynamic nature of the network that makes malicious web servers appear different over time. Attackers might implement malicious fluctuations to make forensic analysis of their attacks more difficult. These are explained in more detail in section 1.2.2.2. As a result, the forensic analysis needs to be based on the data recorded during the initial identification of the malicious web servers.

2.2.1 Issues In Replaying Network Data. In order to replay network data on the transport layer, one would have to place recorded network packets back on the wire. A network flow would have to be split into network traffic of the server and the client. Once one side of the network flow is selectively placed on the wire, for example the client side, the server would recognize this request as if it originated from an actual client and serve a server response as it would do normally.

Separation of a network flow into client and server network packets is an easy task. It can be done through filtering the network flow by source of the client or server IP address. However, record/ replay is not a capability that the transport layer of the network protocol, such as TCP [5] (independent of whether IPv4 [6] or IPv6 [2] is used), supports out of the box. There are several challenges that hinder replaying TCP network traffic against applications. The main ones are reviewed next:

First, TCP is a stateful protocol that establishes a connection between a client and server via a three-way handshake as shown in Figure 2. Using this mechanism to establish the TCP connection, the client sends a TCP packet with the ACK flag set to the server. The server acknowledges this initial connection request with a TCP packet with the ACK and SYN flag set. The connection is fully established by the subsequent TCP packet from the client with the ACK flag. During this process, sequence numbers are exchanged that identify the other party. If such data is recorded and one side of the conversation placed on the communication link, no successful connections can be established.

Second, during the establishment of the connection, ephemeral ports are created by the client to accept the response packets by the server. In other words, the client application temporarily becomes a server itself. This ephemeral port is dynamically assigned in the high port range with each connection as shown in Figure 3. If no connection is being established, the port remains closed. Replaying network traffic against the client would necessitate matching the temporarily opened ephemeral port with the destination port specified in the TCP packets of the net-

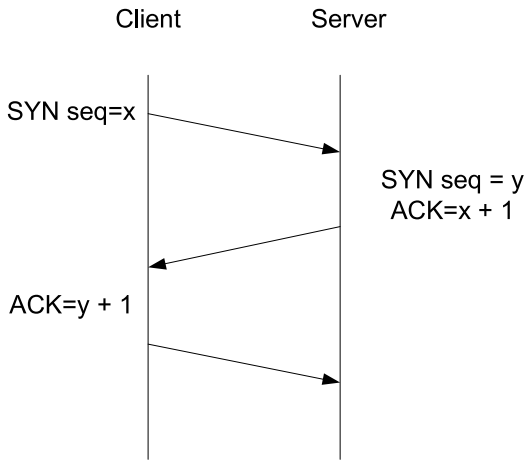

Figure 2. TCP Handshake

work flow. Otherwise, the traffic would not reach its desired target of the client application.

The freely available Tcpreplay tool [11] suggests that it is able to perform a replay functionality by its name. Indeed, this tool is able to place recorded packets back on the wire, but does so in a passive way without modifying the recorded packets to address the TCP handshakes and ephemeral port assignment issues identified above. Rather, it places packets on the wire in its original form, which can be used to test network performance and inline security devices, such as firewalls and intrusion detection systems.

2.2.2 Network Fluctuations. The dynamic nature of the network prevents the forensic examiner from simply retrieving malicious content from the identified malicious web server once again for analysis purposes. The content retrieved might be different from the content that was initially sent to the client application. If one identifies a malicious web server and subsequently tries to analyze the attack by repeatedly interacting with this server, they will run into malicious network fluctuations. The purpose of these malicious fluctuations is to frustrate the effects of forensic investigations through non-deterministic behavior. There are three main techniques attackers apply.

First, there is the simple technique of removing malicious content from the server. A malicious web server is identified by a client honeypot. On

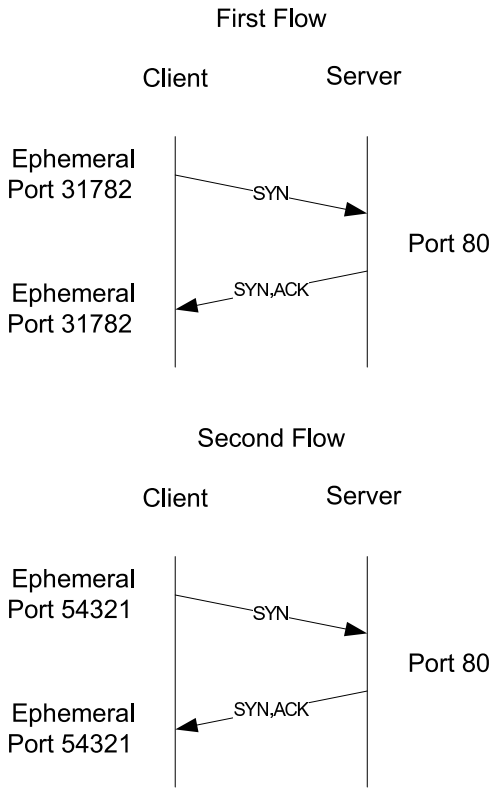

Figure 3. Ephemeral Port Assignment

subsequent interactions, even when interacting with the same web server, the malicious content is not contained on the page anymore. The content has simply been taken offline between the initial contact and the time the analysis took place.

Second, subsequent interactions with the malicious server might be routed to a different physical machine. The explanation for this behavior lies in the domain name system (DNS). DNS translates domain names into IP addresses, so the request targeted for the particular domain can be routed to the appropriate physical machine on the network. It has been observed that attackers make use of fast-flux service networks in which public DNS records are constantly changing [4]. This practice makes the attack infrastructure more resilient against failure, but also makes tracking of the malicious code more difficult.

The following example illustrates how attackers utilize these fast-flux networks. First, the attacker needs to be in control of a DNS server and a host name, such as myattackserver.com. The actual web page is hosted on multiple physical machines that all have different IP addresses. The attacker could then configure the DNS to resolve the myattackserver.com host name to one of these physical machines. On repeated requests, the DNS could resolve to a different physical machine. If not all physical machines host the attack code, one might be faced with a malicious page initially, but with a benign page on subsequent requests.

Third, attacks apply a mechanism called "IP Tracking". In-depth analysis of exploitation frameworks that can be deployed on web servers, such as Mpack v0.94, revealed that attacks make use of server side technologies that provide powerful mechanisms to the attack code [13]. An exploitation kit, for instance, can be configured to only trigger on the initial contact with the web server. A subsequent interaction with the web server from the same IP address would not solicit an attack, but rather a benign web page. As a result, a web server that launched an attack on the client honeypot would appear benign during the analysis phase.

3. Solution

In this section, the proposed solution is presented. It is based on the concept of record/ replay mechanism in which recorded data is played back through the client application as shown Figure 4. Previously recorded network data is played back to the client application. As a result, it is easier to extract relevant information from the data. Instead of writing a custom forensic analysis application that extracts the information from the network data, the existing functionality of the

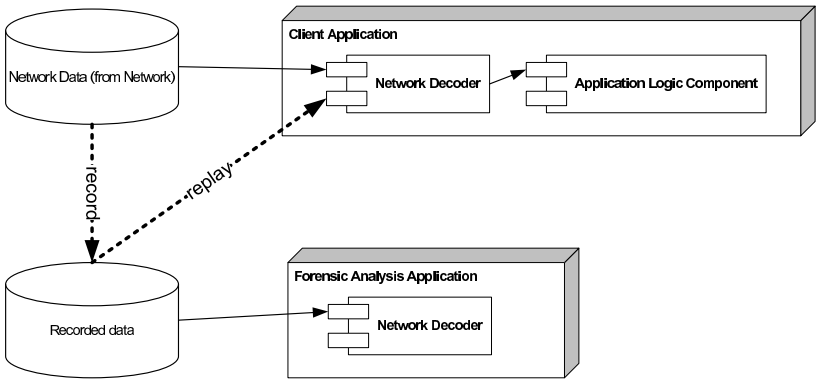

Figure 4. Record/ Replay

client application can be reused. In addition, replaying recorded data through the client application opens the door for behavioral analysis.

As mentioned above, a record/ replay mechanism on the network transport layer comes with many technical issues. As a result, the application layer to implement a record/ replay is the focus of the proposed solution in which the communication of the client application and malicious web server is routed through a proxy that records all the application data. The proxy, if instructed to always replay the stored data, instead of fetching it from the actual server, can repeatedly replay the server responses to the client. The architecture is shown in the top portion of Figure 7.

As the proxy server stores the data during the initial operation of the client honeypot, it can be used for forensic analysis at a later point in time. First, it is possible to perform a behavioral analysis of the attack code by using the actual browser. The browser makes the HTTP request, which is routed via the proxy. Since the proxy already knows about the response, it will return the server response without requesting it from the malicious server once again. Rather, the proxy returns the response from its internal storage. Second, even though the storage of the proxy might be in a proprietary binary format, the application data can be easily retrieved. Browsers and DNS clients can obtain and decode the proxy data. For example, WGET could easily obtain the HTTP response; the host tool could translate DNS responses stored on the proxy server. No custom tools have to be written by the forensic examiner to extract and analyze the application data. The code of existing tools is reused.

Next, the specifics of the proxy solution with a description of specific proxy configurations is presented followed by a discussion of its limitation.

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

Figure 5. Squid Configuration Options

3.1 Proxy Solution

Since two application protocols are used when web browsing, two proxy solutions need to be created: a web proxy that is able to route and store HTTP data and a DNS proxy that is able to route and store DNS data.

A web proxy relays HTTP data and is designed to store this data in its cache. Caching is a functionality that is part of the HTTP/1.1 specification in RFC 2612 [3] to improve response performance, availability and to some extent allow disconnected operation. However, the caching functionality is not designed for the purposes of forensics analysis. The HTTP/1.1 specification primarily contains caching for performance improvement and increased availability. With the focus on these two factors, the specification is also concerned with staleness of the data and therefore defines a mechanism that checks whether a newer resource is available or whether the resource itself should never be cached. In particular there are the freshness and privacy/ security constraints and cache correctness (Detailed description of these directives is available in the Section 14.9 of RFC 2612.)

If a web proxy strictly adheres to these functional requirements, a saved malicious web page might be invalidated by the freshness constraint and fetched once again from the server on a subsequent request. In contrast, the proposed solution aims at using the web proxy for storage rather than caching without application of these mechanisms defined by the specification. Squid [15], an open-source web proxy implementation, is highly configurable and allows for the deviation from the specification via the following configuration options as shown in Figure 5. With this implementation and these configuration settings, the web proxy will meet the above mentioned requirements in a forensic setting.

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 make another DNS lookup on the actual DNS server. Again, an implementation is needed that can overwrite this behavior. Pdnsd is a simple caching DNS daemon that permits this. It is a proxy DNS server with permanent caching

```
perm_cache=204800;
```

Figure 6. Pdnsd Configuration Options

[9] that is designed to cope with unreachable or down DNS servers (for example in dial-in networking). Purging of the older cache entries can be prevented by setting the maximum cache size to a high value as shown in Figure 6.

3.2 Limitations

The proposed solution has a few limitations. First and foremost, it is not easily applicable to other network data. HTTP and DNS are protocols that follow a simple request/ response model. These protocols have existed for a long time and during which Internet access was dominated by dial-up networks. Caching proxies were popular to save money and increase reliability, which was likely the main driver for the development of these tools. The existence of proxy storage capabilities and the flexibility of these tools permitrepurposing of the proxies for forensic data collection and analysis; however, this is not likely to be the case for other protocols. If one considers peer-to-peer protocols, or more interactive applications (such as SSH), the proxy record and replay capabilities are not likely to exist. They don't follow the simple request/ response structure of DNS and HTTP, which makes record and replay functionality harder to implement.

Second, the proxy solution does not provide the same interactivity as a real server interaction. State, for example authentication, is held by the client and usually conveyed back to the server in the form of a cookie. While the proxy will be able to store this information, a client would have to adhere to the same request sequence in order to solicit the same responses. For example, if a client accesses a web page after authentication, a client would again have to authenticate before accessing this web page when interacting with the server at a later time. A client cannot merely access the web page on the proxy without authentication first as important information, such as the cookie, would be missing in the request.

Further, the proxy solution is not able to handle encryption, because encryption is designed to ensure the confidentiality between two communicating parties. If a proxy could be placed in the middle and record all communication, it would effectively circumvent encryption. Of course

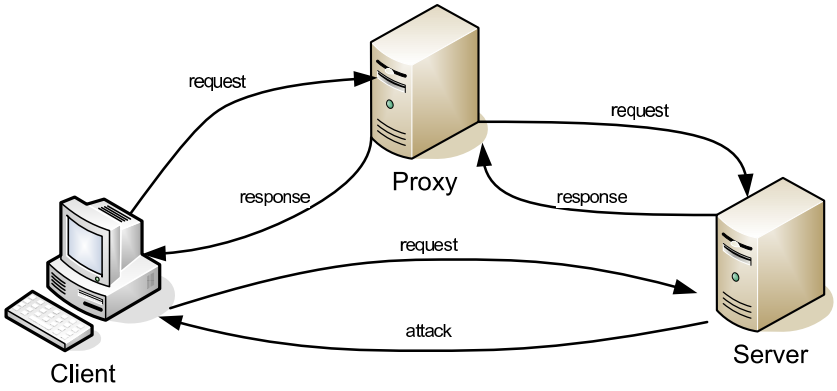

Figure 7. Proxy Architecture

this privacy constraint could not be disabled and therefore the system is not capable of handling encrypted data.

Finally, interacting with a server via a proxy might solicit different server responses. While this is not a concern in the forensic setting, as the data sent to the client will be the data recorded by the proxy, it might represent a problem when performing searches with client honeypot technology. A server might check for the existence of a proxy and not behave maliciously if such a proxy is encountered as a precaution against the recording measures. This is shown in Figure 7. The top flow shows a client application interacting with a server via a proxy. The server detects this setup and therefore delivers a benign web page; the bottom flow shows a client interacting with the server directly. Because no proxy is recording the data in between, the server is free to deliver the attack.

4. Conclusion

In a previous study, client honeypot technology identified numerous malicious web servers. Merely recording the network traffic was insufficient to perform an efficient forensic analysis. Custom tools had to be developed to access and examine the embedded data of the network protocols. Once the information was extracted from the network data and was accessible, it did not permit performing a behavioral analysis on the attack, therefore limiting the ability to answer what exactly happened on the attacked system.

This situation makes digital forensic investigations extremely difficult and time consuming. The effort to launch an attack from web servers is miniscule compared to the effort of analyzing the attack. Network foren-

sic readiness tries to address this imbalance by making digital forensic analysis easier.

This paper presents a step in this direction by introducing a record/ replay solution utilizing proxy servers. This setup permits easy examination of the application data by reusing the capability of the clients that consume such data. It also permits interactively sending this data to the client application in order to perform behavioral analysis on the attack. The resulting application behavior provides a more complete picture of what happened on the system. Unfortunately, the proposed usage of proxy servers was limited to a few protocols (HTTP and DNS) and tools (Squid and pdns proxy).

While these capabilities were implemented on the application layer using existing proxy solutions, the authors believe a generic solution on the network transport layer could be implemented as well. Such a solution is likely to remove the constraint of the proposed application layer solution that currently is only suitable to these few application protocols. A solution on the network transport layer is likely to address record/ replay functionality in a generic way. Several technical aspects were identified that hinder such an implementation. While these issues might be overcome technically, they are not trivial. A tool that attempts to implement such functionality is the flowreplay tool. *"flowreplay has the simple goal of reading a pcap file, taking the client side of the connection(s) and replaying that data using a standard TCP/ UDP socket to connect to a server."* [12]. However, as of September 2007 no functional version of the tool exists indicating that this task may be even more complex than described above.

The difficulties encountered are primarily sourced in the fact that network forensic readiness is an afterthought. One is faced with existing network and application protocols and tools that might not support the goals pursued in a forensic analysis. The authors call for including requirements within network and application protocols from the start that are designed to achieve network forensic readiness. A record/ replay requirement was identified that would ease efforts in the situation of analyzing malicious web server; however, additional requirements might be included and are left for future work.

References

[1] B. Endicott-Popovsky, D. A. Frincke and C. A. Taylor, A Theoretical Framework for Organizational Network Forensic Readiness, *Journal of Computers*, 2(3), May 2007

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

[3] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, RFC2616 - Hypertext Transfer Protocol – HTTP/1.1 (www.ietf.org/rfc/rfc2616.txt).

[4] The Honeynet Project, Know Your Enemy: Fast-Flux Service Networks (www.honeynet.org/papers/ff/).

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

[6] Information Sciences Institute, University of Southern California, RFC 791 - Internet Protocol, Version 4 Specification (www.faqs.org/rfcs/rfc791.html).

[7] V. Jacobson, Tcpdump (www.tcpdump.org).

[8] P. Mockapetris, RFC1035 - Domain Names - Implementation and Specification (www.ietf.org/rfc/rfc1035.txt).

[9] T. Moestl and P. Rombouts, Pdnsd - Proxy DNS Server (www.phys.uu.nl/ rombouts/pdnsd/index.html).

[10] J. Tan, Forensic Readiness (web.archive.org/web/20031203010126 /http://atstake.com/research/reports/acrobat/ atstake_forensic_readiness.pdf).

[11] A. Turner, Tcpreplay (tcpreplay.synfin.net/trac/).

[12] A. Turner, Flowreplay Design Notes (synfin.net/papers/flowreplay.pdf).

[13] C. Seifert, Know Your Enemy: Behind the Scenes of Malicious Web Servers (www.honeynet.org/papers/wek).

[14] C. Seifert, R. Steenson, T. Holz, Y. Bing and M. A. Davis, Know Your Enemy: Malicious Web Servers (www.honeynet.org/papers/mws/).

[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).