Identification of Possible Attack Attempts Against Web Applications Utilizing Collective Assessment of Suspicious Requests

Marek Zachara(✉)

AGH University of Science and Technology, Kraków, Poland
mzachara@agh.edu.pl

Abstract. The number of web-based activities and websites is growing every day. Unfortunately, so is cyber-crime. Every day, new vulnerabilities are reported and the number of automated attacks is constantly rising. In this article, a new method for detecting such attacks is proposed, whereas cooperating systems analyze incoming requests, identify potential threats and present them to other peers. Each host can then utilize the knowledge and findings of the other peers to identify harmful requests, making the whole system of cooperating servers "remember" and share information about the existing threats, effectively "immunizing" it against them.

The method was tested using data from seven different web servers, consisting of over three million of recorded requests. The paper also includes proposed means for maintaining the confidentiality of the exchanged data and analyzes impact of various parameters, including the number of peers participating in the exchange of data. Samples of identified attacks and most common attack vectors are also presented in the paper.

Keywords: Websites · Applications security · Threat detection · Collective decision

1 Introduction

According to a recent report by Netracft [14], the number of websites around the world is estimated to be almost 850 million, with 150 million of them considered "active". People rely on Internet and the websites in their daily activities, trusting them with their data and their money.

Unfortunately, with more and more data and resources handled by websites, they have become an attractive prey to criminals, both individuals and organized crime. It is very difficult to measure the scale of the cyber-threats, and their impact on the companies and the economy as a whole, since there is no commonly accepted methodology available yet. As an example, McAfee estimates that the cost of the cybercrime reaches 1.5 % of the GDP for the Netherlands and Germany [12]. There is also a more detailed study for the UK [2]. However, such estimations might be imprecise, because they are based on imperfect surveys, which may lead to a high estimation error, as explained in [4,8].

N.T. Nguyen and R. Kowalczyk (Eds.): TCCI XXII, LNCS 9655, pp. 45–59, 2016.
DOI: 10.1007/978-3-662-49619-0_3

1.1 Vulnerability of Web Applications

The initial web sites in the 1990 s were meant primarily for publishing and dissemination of information. Virtually all of their resources were meant for public access. The HTTP protocol developed then, and still used today for the transport of the web pages, does not even include any means of tracking or controlling user sessions. Today's web sites are, however, quite different. They often gather and control valuable data - including personal data, passwords or bank accounts.

Symantec claims that while running a thousand of vulnerability scans per day, they found approximately 76 % of the scanned websites to have at least one unpatched vulnerability, with 20 % of the servers having critical vulnerabilities [21]. In another report [22], WhiteHat Security stated that 86 % of the web applications they tested had at least one serious vulnerability, with an average of 56 vulnerabilities per web application.

It can be assumed, that one of the reasons for the low security of web applications is their uniqueness. While the underlying operating systems, web servers, firewalls, and databases are usually well known and tested products, that are subject to continuous scrutiny by thousands of users, a web application is often on its own, with its security depending primarily on the owner and developers' skills and will.

1.2 Malware and Automated Attacks

The massive amount of websites, and their availability over the Internet, led to a rise of automated methods and tools for scanning and possibly breaking into them. Bot-nets and other malware are often targeting websites for known vulnerabilities. For example, Symantec in one of their previous reports stated that in just the single month of May in 2012 the LizaMoon toolkit was responsible for at least a million successful SQL Injections attacks, and that approximately 63 % of websites used to distribute malware were actually legitimate websites compromised by attackers.

Easy access to information and ready-made tools for scanning and exploitation of websites' vulnerabilities resulted in a large number of individuals, collectively known as "script kiddies" attempting random break-in attempts against them, using the same tools downloaded from the Internet.

1.3 The Tools for Battling the Attacks

The reason why firewalls do not protect websites from harmful requests is that their ability is only to filter traffic at the lower layers of the OSI model (usually up to layer 5 for stateful firewalls), while the identification of harmful requests is only possible at layer 7. There are specialized firewalls, known as Web Application Firewalls (WAF) [15,16], but their adoption is limited, primarily because of the time and cost required to configure and maintain them.

There are two broad classes of methods employed for battling attack attempts and identification of the harmful data arriving at the server. The first group consists of various signature-related methods, similar to the popular anti-virus software. Some examples are provided in [7,19]. These methods try to identify known malicious attack patterns on the basis of their knowledge (provided a priori). The primary benefit of such methods is the low number of false-positive alarms. Their primary drawback is the inability to identify new threats and new attack vectors, until they are evaluated by some entity (e.g. a security expert), and introduced into the knowledge database. Within the fast-changing Internet threat environment, they provide very limited protection against attackers, as vulnerabilities are exploited often within hours of their disclosure (citing the Symantec report again [21]: "Within four hours of the Heartbleed vulnerability becoming public in 2014, Symantec saw a surge of attackers stepping up to exploit it").

The second group of methods relies on various types of heuristics to identify potential threats in real time. This approach has been employed for network traffic analysis and intrusion detection [5,17] with SNORT [18] being the well-known open source implementation of such Intrusion Detection Systems (IDS). Although there have been a number of attempts to utilize similar approach in order to secure websites [3,11], these methods usually rely on very simple heuristics and a simple decision tree. There is also a method developed by the author of this article that utilizes weighted graph for modeling and storage of users' typical page-traversing paths that has been presented in [23].

1.4 Rationale for the Collective Assessment

All these methods previously described rely on local evaluation and assessment of the requests, utilizing the knowledge either provided or acquired at a single web server. However, the rise of automated tools and malware led to a situation where the same attack vectors, or even the same requests are used to scan and attack various unrelated websites. Establishing a method of sharing the information between web servers about encountered malicious requests could therefore provide substantial benefits in protecting the websites against these attacks.

2 The Principles of the Method

The attacks on web applications/web servers are usually done either by manipulating parameters sent with a request (*parameter tampering*) or by requesting URLs different to these expected by the application (*forceful browsing*). A good account on specific attacks and their impact can be found in [6]. An example of such attacks are presented in Fig. 1, where the attacker is apparently trying to identify, at various possible locations, a presence of phpMyAdmin - a commonly used web-based database management module.

The web server's log files are usually the easiest way to retrieve information about the requests arriving at the server, allowing for easy parsing and identification of suspicious requests. They have been widely used for this purpose [1,9], and the reference implementation of the proposed method also utilizes the log files as the source of information.

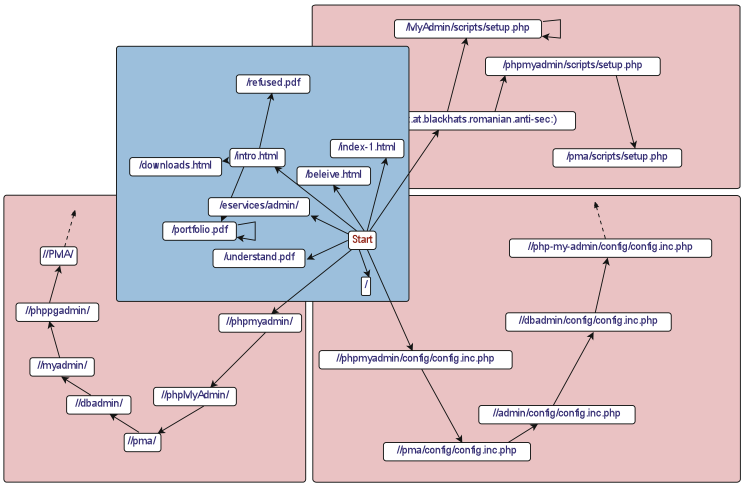
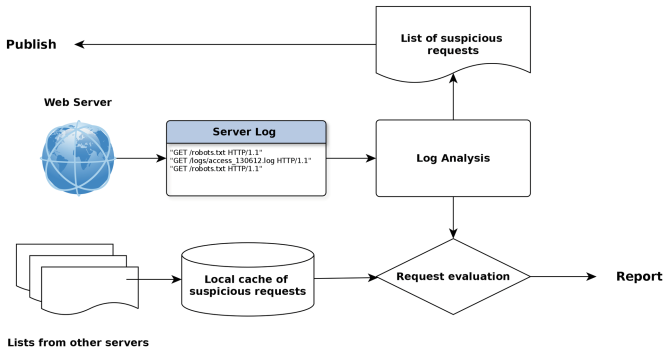

/MyAdmin/scripts/setup.php
/phpmyadmin/scripts/setup.php
.at.blackhats.romanian.anti-sec:)
/pma/scripts/setup.php
/refused.pdf
/intro.html
/downloads.html
/index-1.html
/beleive.html
/eservices/admin/
/portfolio.pdf
/understand.pdf
Start
/
//PMA/
//phppgadmin/
//myadmin/
//dbadmin/
//pma/
//phpmyadmin/
//phpMyAdmin/
//php-my-admin/config/config.inc.php
//dbadmin/config/config.inc.php
//phpmyadmin/config/config.inc.php
//admin/config/config.inc.php
//pma/config/config.inc.php

Fig. 1. Sample users' sessions extracted from a web server's log file, with attack attempts marked by red background (Color figure online)

A high-level overview of the proposed method is presented in Fig. 2. As can be seen there, each server maintains its own list of suspicious requests, which is published for other servers to download. On the other hand, the server retrieves similar lists from other peers and keep cached copies of them. Each new request arriving at the server is evaluated against these lists, which constitute the server's knowledge about the currently observed attack patterns. Requests that are considered as potentially harmful are then reported to the administrator.



```
"GET /robots.txt HTTP/1.1"
"GET /logs/access_130612.log HTTP/1.1"
"GET /robots.txt HTTP/1.1"
```

Fig. 2. Overview of the data flow for the collective identification process.

2.1 The Key Concepts of the Proposed Method

This method relies on the fact that most attacks are automated. They are performed either by malware (like LizaMoon mentioned before), or "script kiddies" who download and use ready-made attack tool-kits. These two groups have the ability to perform attacks attempts on a large scale. However, these attacks are usually identical, or very similar to each other. Since they are performed by programs and usually rely on one or just a few attack vectors, they generate similar requests to a large number of web servers.

For clarity reasons, let's reiterate the features of the attacks performed by malware and "script kiddies":

- **Distributed.** The attacks are usually targeted towards a number of web hosts, not at the same time, but within a limited time-frame.
- **Similar.** Malware and "script kiddies" both use single pieces of software with incorporated attack vectors. This software produces identical or very similar requests to various hosts.
- **Unusual.** The requests generated by the automated software are not tailored for the specific host, so they often do not match the host's application.

Fortunately, these features can be utilized to detect and neutralize a major part of the threats coming from these sources. These attacks target a large number of hosts (web applications) in hope of finding a few that will be vulnerable to the specific attack vector. In the process however, they send requests to hosts that are either not vulnerable, or in most cases - do not have the specific module installed. Such requests can easily be identified at these hosts as suspicious and presented to other hosts as an example of an abnormal behavior. Other hosts, which retrieve the information, are then aware that certain requests have been made to a number of other hosts and were considered suspicious by them. This knowledge, in turn, can be utilized to assess and possibly report an incoming requests. Specific methods of such assessment will be discussed later, but it is worth to note that a very similar mode of operation is used by our immune system, where antigens of an infection are presented by body cells to the T-cells, which in turn coordinate the response of the system by passing the 'knowledge' about the intruder to the other elements of the immune system.

2.2 The Exchange of Information

There are various means that can be employed to facilitate the exchange of information between web servers about suspicious attacks. Large scale service providers can opt for a private, encrypted channels of communication, but the general security of the web would benefit from a public dissemination of such information.

Certainly, publishing information about received requests could be a security issue in its own right. Fortunately, it is not necessary to disclose full requests' URLs to other peers. Since each host needs just to check if the URL it marked as suspicious has also been reported by other hosts, it is enough if the hosts compare the *hashes* (digests) of the requests' URLs. Hashes (like SHA1) are generated using one-way functions and are generally deemed irreversible. Even though findings are occasionally published about the weaknesses of certain hashing methods [20], there are always algorithms available that are considered safe, even for the storage of critical data (at the time of writing two examples are SHA-2 and MD6). If a host only publishes hashes of the requests that it deems suspicious, it allows other hosts to verify their findings, but at the same time protects its potentially confidential information.

For the proposed method to work, it is enough if hosts just publish a list of hashes of the received requests that they consider suspicious, but other data may improve the interoperability and possible future heuristics. Therefore, it is suggested that hosts use a structured document format, e.g. based on JSON [10]. A sample published document may look like the one in Listing 1.1. This will be referred to as a *List of Suspicious Requests*, abbreviated *LSR*. The list presented in Listing 1.1 is a part of a list taken from an actual running instance of the reference implementation (and includes the mentioned additional information).

The explanation of all the elements in this LSR is provided below:

– (C) denotes the class of the information. This can either Original or Forward from another peer.
– (A) is the age of the suspicious request, e.g. how many hours ago it was received.
– (MD5) indicates the algorithm used and is followed by the resultant hash of the request.
– (R) includes the actual request received and is presented here for informational/debug purposes only.

Listing 1.1. Sample LSR with additional debug information

```
{ C:O, A:57, MD5:2cf1d3c7fe2eadb66fb2ba6ad5864326, R:"/pacpdvlgj.html" }
{ C:O, A:53, MD5:2370f28edae0afcd8d3b8ce1d671a8ac, R:"/statsa/" }
{ C:F, A:32, MD5:2f42d9e09e724f40cdf28094d7beae0a }
{ C:F, A:31, MD5:8f86175acde590bf811541173125de71 }
{ C:F, A:24, MD5:eee5cd6e33d7d3deaf52cadeb590e642 }
{ C:O, A:17, MD5:bd9cdbfedca98427c80a41766f5a3783, R:"/Docs/ads3.html" }
```

2.3 Maintenance of the Lists

For the process to work as intended, each server must not only identify suspicious requests, but also generate and publish the list of them and retrieve similar lists from other servers. However, exchanging of the LRS leads to an issue of data retention, and two questions need to be answered:

– How long should an LSR contain an entry about a suspicious request after such a request was received.
– Should the hashes received from other peers be preserved locally if the originating server does not list them anymore, and if so, for how long?

Both issues are related to the load (i.e. number of requests per second) received either by the local or the remote server. Servers with very high loads and a high number of suspicious requests will likely prefer shorter retention times, while niche servers may only have a few suspicious requests per week and would prefer a longer retention period. The results of experiments presented later in this article illustrate how the retention period may impact the quality of detection, and more results may be obtained if the method becomes more widely adopted.

3 Implementation and Test Environment

A reference implementation has been prepared to verify the feasibility of the proposed method and to prove this concept. The application has been programmed in Java and have been tested using the data from a few real web servers. The architecture of the application is presented in Fig. 3.

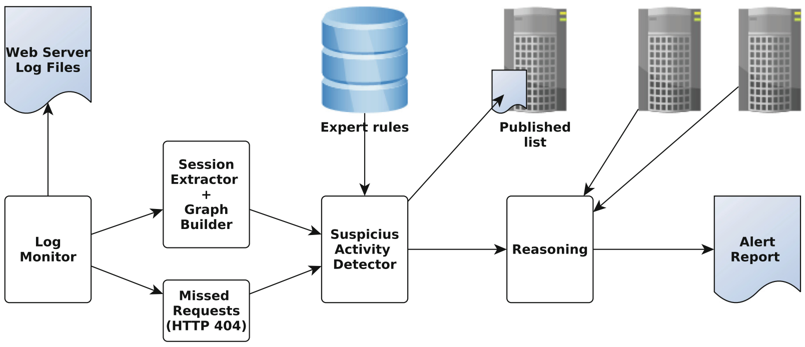

Web Server Log Files
Expert rules
Published list
Session Extractor + Graph Builder
Log Monitor
Suspicius Activity Detector
Reasoning
Alert Report
Missed Requests (HTTP 404)

Fig. 3. The architecture of the reference implementation

The requests for unavailable resources (e.g. these that result in HTTP 4xx response) are directly forwarded to the aggregation of suspicious requests. The original application also includes a behavior-based anomaly detection module, which is outside the scope of this article, but has been described in [23].

The *Suspicious Activity Detector* module aggregates all suspicious requests and matches them against some pattern-based rules, to eliminate requests that are well known and harmless but are often missing on servers. The list of requests, together with their occurrence frequency is presented in Table 1. They can safely be ignored and not reported to the administrator. This process is referred to as “white-listing”.

In addition to this general white-listing, applicable for all the web sites, a website may benefit from additional white-listing of specific pages, which could generate false-positives. During the tests, it was found out that three of the tested websites had a relatively large number of reports for just two URLs (see Table 2). It is likely that the websites’ structure might had changed during the testing

Table 1. A white-list of requests that are considered legitimate, yet often result in a "not found" (HTTP 4xx) response.

Request	Total number		Comment
	Not found	Present	
/ & index.html	1,342	161,542	Home page of a website
favicon.ico	24,899	4,067	The website's icon
apple-touch-icon*.png	2,255	112	Icons used by IOS-based devices
robots.txt	14,728	16,712	*robots.txt* and *sitemap.xml* are
sitemap.xml	111	493	file looked for by search engines

period as both web-pages are often present in commercial websites. Such change would also explain the number of request attempts for these missing resources.

Table 2. An additional, site-specific white-list of requests that generate a large number of "not found". The number of parenthesis indicate the number of affected websites from the tested set.

Request	Total number	
	Not found	Present
/services.[html|php]	153 (3)	4,219 (1)
/contact.[html|php]	1,230 (3)	9,810 (2)

Finally, the remaining suspicious requests are formatted according to the sample presented in Listing 1.1, and are stored in a document inside the web server directory to be accessible by other hosts.

3.1 Reasoning

The last module (named 'Reasoning') receives the current request (if it is considered suspicious) and, at the same time, periodically retrieves lists of such suspicious requests from other hosts. With each request, the module has to decide whether it should report it for human (e.g. administrator's) attention or not. There are various strategies that can be implemented here, depending on the type of application being protected and the number and the type of the peer servers it receives the data from. Some basic strategies are discussed in the next chapter, yet the system administrator may be willing to adjust the system's response depending on their own requirements.

4 Test Method and Achieved Results

For the purpose of evaluating the proposed algorithm, a number of log files have been acquired, as listed in Table 3. These log files came from a number of

Table 3. Number of lines per log file used for the test experiments. Only parseable, validated lines were counted. Bottom row lists the number of requests that resulted in error response 4xx or 5xx.

	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6	Site 7
All requests	311,530	1,030,186	108,859	53,271	418,361	254,638	886,233
4xx and 5xx	14,861	287,394	4,017	14,281	41,706	7,381	25,885

unrelated servers, located in several countries. Due to the nature of the method, the logs must cover the same period of time, in order to simulate a specific real time period. All these logs cover the same period of approximately 1 year.

Based on these logs, a simulation environment was prepared, that emulated these web servers over the specific time period. Each emulated server analyzed its log files and published the LSR for the other servers to download. To clearly identify the benefits of the collective detection, and eliminate the impact of other methods, all and only requests that resulted in HTTP response 4xx ("not available") or 5xx ("server error") were listed in the server's LSR. The overall number of such requests is presented in Table 3. Additionally, the changes in number of these requests over consecutive weeks is illustrated in Fig. 4. As can be seen there, these numbers can vary from week to week even by an order of a magnitude.

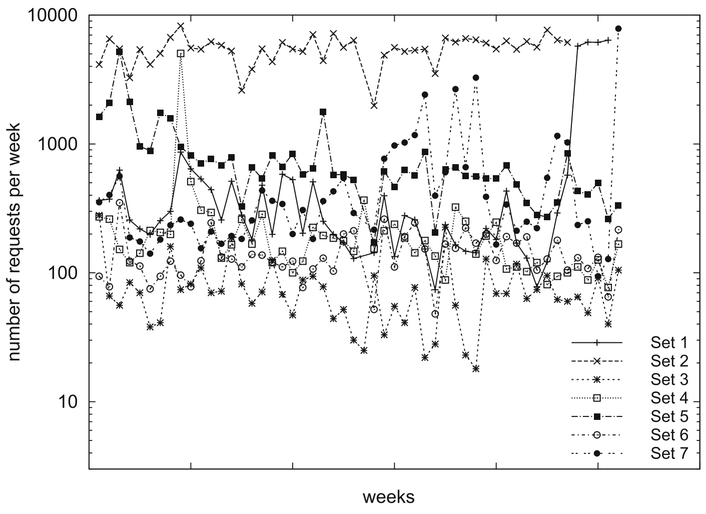


Fig. 4. Change in number of requests resulting in 4xx or 5xx error response over time. Aggregated per week for each of the server log sets.

The results achieved with the collective detection were also compared with the results of an analysis performed by LORG [13], an Open-Source project designed to identify malicious requests in a web server's log files.

Several scenarios were tested, with different configuration, in order to determine their impact on the quality of detection. These include evaluating the impact of the number of web servers participating in the exchange and changing how long the suspicious requests are kept in the LSRs.

4.1 The Baseline Scenario

In this scenario, all seven servers were emulated, and the LSRs kept records of suspicious requests for 7 days. The results of the collective detection for each server is presented in Table 4, together with the result of the LORG analysis.

Table 4. Results of the baseline scenario, number of reported suspicious requests by LORG and collective detection, split into these which ended up as "unavail." (4xx and 5xx response) and these that resulted in 2xx response ("present"). The number of reported incidents is also presented as a per mille (‰) of total log entries.

	LORG & CD		LORG only		Collective detection		
Response:	Present	Unavail.	Present	Unavail.	Present	Unavail.	Ratio
Site 1	0	3	0	0	10	1,030	3‰
Site 2	23	5	148	8	150	6,411	6‰
Site 3	0	0	0	0	1,430	165	15‰
Site 4	0	0	28	16	0	306	6‰
Site 5	0	0	19	3	259	3,143	8‰
Site 6	0	0	7	1	63	1,356	6‰
Site 7	0	4	43	0	47	2,000	2‰

As can be seen in this table, the collective analysis results in significantly more reported requests. The results are split into two groups - requests that resulted in a HTTP 2xx response ("present") and requests that did not process correctly, with the response 4xx or 5xx sent to the client. The later are labeled as'unavailable' and can be safely classified as scans or attack attempts.

Some of the results, especially these related to the requests reported as harmful by LORG were investigated manually, with the following findings:

- Most of the requests reported by both LORG and the collective detection were requests for a home page, but with additional parameters intended to alter its operation; like */?include=../../../etc/passwd*.
- Majority of the requests reported by LORG but not the collective detection were actually false positives. This includes almost all of the 148 requests resulting in "present" response from Site 2.

– The unusual number of "present" responses for reported requests in Site 3 is a result of the configuration of this website, which returned a page even for unexpected URLs, unless they were severely malformed.

Most common examples of requests reported by the collective detection and LORG are presented in Table 8. Since this table presents the most popular examples (every one occured several hundred times), the requests may appear quite ordinary. Collective detection is however able to also identify more sophisticated attacks, as illustrated in Table 5.

Table 5. Less common examples of malicious requests identified by collective detection.

Example URL
/includes/fckeditor/editor/filemanager/upload/php/upload.php
/includes/uploadify/uploadify.swf
/index.php?m=admin&c=index&a=login&pc_hash=
/wp-content/themes/felis/download.php?file=../../../wp-config.php
/?page_id=\"><script> alert(\"m3t4l&master\");</script>
/go?to=http://pastebin.com/raw.php?i=pA3y1PSN
/cgi-bin/php-cgi?%2D%64+%61 %6C%6C%6F%77 %5F%75 %72 %6C%5F%69*(...)*
/asp/freeupload/uploadTester.asp
/beheer/editor/assetmanager/assetmanager.asp

Statistical analysis of the requests reported by the collective detection resulted in identification of the most common attack vectors, which are presented in Table 6.

Table 6. Most common attack vectors identified by collective detection.

Occurences	Description
10,332	Attempts to locate or use basic WordPress URLs
2,049	Requests for various */admin/* URLs (excluding WordPress)
1,679	Attempts to locate CKEditor
265	Attempts to identify or exploit WordPress xml-rpc
163	Attempts to locate OpenFlashChart
150	Direct attempt to manipulate parameters of the primary web page; e.g. */?(...)params* or */index.php?(...)params*
107	Requests targeting Uploadify
43	Attempts to access PhpMyAdmin

4.2 Impact of the LSR History Length and the Number of Peers

It is rather obvious, that the effectiveness of the detection depends on the number of peers involved in the exchange and how long they keep the history of the suspicious requests. Table 7 presents the results of the collective assessment for smaller group of peers. Interestingly, the overall traffic processed by the participating websites has much more impact on the final results than the number of peers involved.

Table 7. Decrease in number of reported requests against the baseline scenario (percent of the baseline reported request) for decreased number of peers participating in the collective assessment.

	6 peers	5 peers	4 peers	3 peers	2 peers
High-traffic sites	95.8 %	94.3 %	85.8 %	86.7 %	61.1 %
Low-traffic sites	91.7 %	77.5 %	56.3 %	37.9 %	21.8 %

The impact of the history length is presented in Fig. 5. As can be seen there, there is very little gain in number of reported requests for history length longer than 10 days. It seems that 5–10 days is the maximum a host would need. High-traffic servers may need to settle for a 1–2 day history, but that still provides substantial benefits for the collective detection.

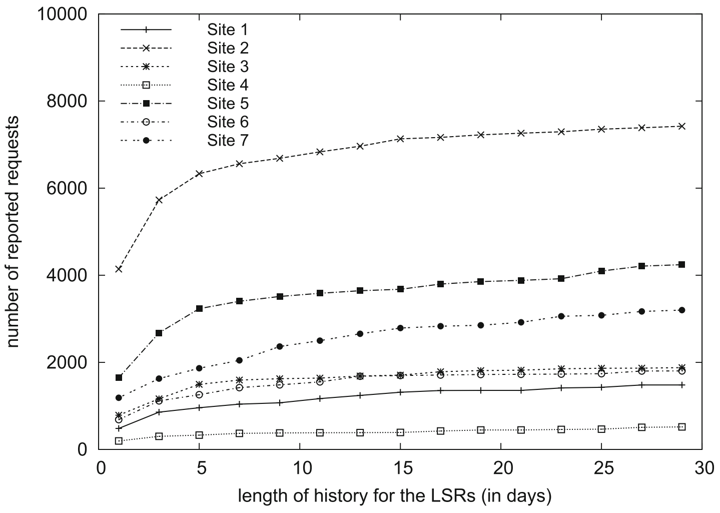


Fig. 5. Number of suspicious request reported by each site as the function of LSR history length.

Table 8. Sample most common findings of potentially malicious requests by LORG and collective detection.

Example URL
Detected by the collective assessment, but not by LORG
/administrator/index.php
/admin.php
/?q=user/register
/wp-admin/admin-ajax.php
/wp-login.php
/wp-login.php?action=register
/xmlrpc.php
Detected by LORG, but not the collective assessment
/pl/?page=http://www.lincos.eu/cache/mod_custom/ID-RFI.txt????
/wordpress/wp-admin/load-scripts.php?c=1&load%5B%5D=hoverIntent,common, admin-bar,svg-painter&ver=4.2.2
/?x=()
Detected by both LORG and the collective assessment
/wp-admin/admin-ajax.php?action=revslider_show_image&img=../wp-config.php
/?x=() { :; }; echo Content-type:text/plain;echo;echo;echo M'expr 1330 + 7'H;/bin/uname -a;echo @
/?page_id=../../../../../../../../../../../../etc/passwd
/upload.asp?action=save&type=IMAGE&style=standard'%20and%201=2 %20 union%20select%20S_ID,S_Name,S_Dir,*(...)*

5 Conclusions and Future Work

The method proposed in this paper aims at providing automated identification of potentially harmful requests with the minimum level of involvement from the system administrators. Suspicious requests are presented to other peers; i.e. web servers, which participate in information exchange. A report of a suspicious activity is produced when identical requests had been encountered and identified as suspicious by other peers.

The method may significantly hinder the *modus operandi* of most malware, due to the fact that after the few initial attempts to attack a number of servers, it will become increasingly difficult for a malware to successfully attack new ones. The servers will recognize the attacks because of the knowledge acquired from their peers. This way, the whole system will develop an immune response similar to the one observed in living organisms.

Emulation-based evaluation of the method, utilizing real data acquired from seven web servers showed that the median ratio of reports were 0,6 % of all the requests. This translates to an average of 5–10 suspicious requests per day brought to the administrators' attention, which is an acceptable level, making analysis of the log files feasible.

Analysis of the data also showed, that most of these request can be grouped into a few common attack vectors. For example, attempts to locate WordPress specific web pages accounted for almost $^2/_3$ of all the reported attempts. Another 10 % were attempts to locate CKEditor - a commonly used module, but having numerous vulnerabilities. An administrator can thus easily reduce the number of reported attempts, by an order of magnitude, if they know they do not have the specific module, nor care about such probing attempts.

Achieved results were compared to LORG, an open source software designed to identify attack attempts by analyzing the URLs of the incoming requests. Surprisingly, a major part of LORG's findings were identified as false positives. It was due to the fact, that two of the websites used large and complicated forms that resulted in a long array of parameters passed with the requests, which were considered suspicious by LORG. LORG also did not identify any scans/probes that did not have an arguments sent along with the URL. This is due to the nature of its analysis, yet they constituted the majority of the identified attempts.

In terms of the computational power required for the described method, the reference single-thread implementation (in Java) was able to process over 30,000 requests (log entries) per second on a typical PC (Intel, 4 GHz). It shall therefore not add a significant workload for the web server.

The method described in this paper can provide substantial benefits to the security of websites right now. It also opens new paths of research that could lead to its further refinement or specialized applications. The development of the decision algorithm will presumably provide the most benefits for the system, since improvement to the local reasoning and identification of the suspicious request will reduce the amount of data exchanged between the peers and will speed up each assessment. By introducing local user tracking and their behavior analysis, the system could also be able to distinguish "blind" scans (which are usually less harmful) from targeted attack attempts. This may be a benefit for high-traffic sites, however will likely result in a delayed detection of attack attempts for the group of servers participating in the data exchange network.

References

1. Agosti, M., Crivellari, F., Di Nunzio, G.: Web log analysis: a review of a decade of studies about information acquisition, inspection and interpretation of user interaction. Data Min. Knowl. Disc. **24**(3), 663–696 (2012)
2. Anderson, R., Barton, C., Böhme, R., Clayton, R., Van Eeten, M.J., Levi, M., Moore, T., Savage, S.: Measuring the cost of cybercrime. In: Böhme, R. (ed.) The Economics of Information Security and Privacy, pp. 265–300. Springer, Heidelberg (2013)
3. Auxilia, M., Tamilselvan, D.: Anomaly detection using negative security model in web application. In: 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM), pp. 481–486 (2010)

4. Florêncio, D., Herley, C.: Sex, lies and cyber-crime surveys. In: Schneier, B. (ed.) Economics of Information Security and Privacy III, pp. 35–53. Springer, Heidelberg (2013)
5. García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., Vázquez, E.: Anomaly-based network intrusion detection: techniques, systems and challenges. Comput. Secur. **28**(1–2), 18–28 (2009)
6. van Goethem, T., Chen, P., Nikiforakis, N., Desmet, L., Joosen, W.: Large-scale security analysis of the web: challenges and findings. In: Holz, T., Ioannidis, S. (eds.) Trust 2014. LNCS, vol. 8564, pp. 110–126. Springer, Heidelberg (2014)
7. Han, E.E.: Detection of web application attacks with request length module and regex pattern analysis. In: Genetic and Evolutionary Computing: Proceedings of the Ninth International Conference on Genetic and Evolutionary Computing, 26–28 August 2015, Yangon, Myanmar, vol. 2, pp. 157. Springer, Switzerland (2015)
8. Hyman, P.: Cybercrime: it's serious, but exactly how serious? Commun. ACM **56**(3), 18–20 (2013)
9. Iváncsy, R., Vajk, I.: Frequent pattern mining in web log data. Acta Polytechnica Hungarica **3**(1), 77–90 (2006)
10. JSON: a lightweight data-interchange format. http://www.json.org
11. Kruegel, C., Vigna, G., Robertson, W.: A multi-model approach to the detection of web-based attacks. Comput. Netw. **48**(5), 717–738 (2005)
12. McAfee: Net Losses: Estimating the Global Cost of Cybercrime (2014). http://www.mcafee.com/us/resources/reports/rp-economic-impact-cybercrime2.pdf
13. Muller, J.: Implementation of a Framework for Advanced HTTPD Logfile Security Analysis, Master's thesis (2012)
14. Netcraft: Web Server Survey (2015). http://news.netcraft.com/archives/2013/11/01/november-2013-web-server-survey.html
15. OWASP: Web Application Firewall. https://www.owasp.org/index.php/Web_Application_Firewall
16. Pałka, D., Zachara, M.: Learning web application firewall - benefits and caveats. In: Tjoa, A.M., Quirchmayr, G., You, I., Xu, L. (eds.) ARES 2011. LNCS, vol. 6908, pp. 295–308. Springer, Heidelberg (2011)
17. Rieck, K., Laskov, P.: Language models for detection of unknown attacks in network traffic. J. Comput. Virol. **2**(4), 243 (2007)
18. Roesch, M.: Snort: lightweight intrusion detection for networks. In: LISA, USENIX, pp. 229–238 (1999)
19. Salama, S.E., Marie, M.I., El-Fangary, L.M., Helmy, Y.K.: Web server logs preprocessing for web intrusion detection. Comput. Inf. Sci. **4**(4), p123 (2011)
20. Stevens, M.: Advances in hash function cryptanalysis. ERCIM News **2012**(90), 26–27 (2012)
21. Symantec: Internet Security Threat Report (2015). http://www.symantec.com/security_response/publications/threatreport.jsp
22. WhiteHat: Website Security Statistics Report (2013). http://info.whitehatsec.com/2013-website-security-report.html
23. Zachara, M.: Collective detection of potentially harmful requests directed at web sites. In: Hwang, D., Jung, J.J., Nguyen, N.-T. (eds.) ICCCI 2014. LNCS, vol. 8733, pp. 384–393. Springer, Heidelberg (2014)