AN ADVANCED WEB ATTACK DETECTION AND PREVENTION TOOL
Kapodistria Helen, Mitropoulos Sarandis, Douligeris Christos
Department of Informatics
University of Piraeus
80, Karaoli & Dimitriou Str.
Piraeus 185 34, Greece
ekapodistria@gmail.com, sarandis@unipi.gr, cdoulig@unipi.gr

Abstract

Web application security is critical in the design and evaluation of web applications, since most web applications increasingly store various types of sensitive data related to their users. Nowadays, sensitive data such as financial information, medical information or personal data, especially in social web applications, are exposed over the networks so it seems quite appealing to hack these applications and leak information to unauthorized users. This paper introduces a new tool which efficiently detects and prevents common web attacks based on input validation. This is accomplished through pattern recognition in a flexible and expandable way. Finally, a comparison to existing web application firewalls is presented showing the high applicability and usefulness of the new tool being introduced.

Keywords: web application security, web vulnerabilities, penetration testing tools, web application firewalls, cross-site scripting, path traversal, SQL injection, command injection, web attacks.

1. Introduction

Web application security is important since web applications store sensitive user information. For example, if a web application that stores credit card information is breached, sensitive information could leak and the application will be considered to have failed to secure its users’ sensitive data.

The scientific area related to web security and particularly to the security of the World Wide Web, the HTTP protocol and the web application software is called Web Application Security. This paper copes with the web application security, focusing on input validation vulnerabilities that, when exploited, could lead to a web application attack (Bertino E. et al., (2010)). A vulnerability is a system weakness which compromises the application and its data. A bug, meaning an error, flaw or failure, or a defect in the design and/or the architecture of the application could be a vulnerability. An attack is the exploitation of a vulnerability in order to access the application, to leak sensitive information or to gain more privileges.

A web application is a software application which is installed and running on a web server and which responds to requests for dynamic web pages via the HTTP protocol (Web Application Security Consortium, 2010). A web application can support various types of transactions such as financial, e-banking, consultation and real-time communication with other users, as well as functions and services within an organization. This wide variety of applications leads to the need of developing robust and reliable web applications that secure the associated information assets and the integrity of all the data involved.
Several tools have been proposed to help developers reduce application vulnerabilities through new APIs while, at the same time, various tools have been designed to help the applications test for the existence of such weaknesses. These tools, called penetration tools or web vulnerabilities scanners, search the web application for common vulnerabilities (Fong, E. et al, 2007). In most cases, it is impossible to identify all of the security weaknesses a web application may have. This is the reason why a large variety of tools exist, such as the intrusion detection systems and the web application firewalls, which audit the entire activity on the web server (Schultz E. et al, 2007). The latter are installed on the web server and watch the incoming and outgoing activity. Most of these tools are based on rule sets which define the web attack signature the firewall should detect (Gauci et al, 2010).

This paper introduces a new web server plug-in, which incorporates advanced web application firewall features. Specifically, it detects, prevents and records web attacks based on pattern recognition in an expandable and easily manageable way. The proposed system presents the following features that are innovative and new in comparison to other related tools (dotDefender, 2010), (ModSecurity, 2010):

- it is a cross-platform application, which means that it is OS-, as well as, web server-independent, hiding in this way the heterogeneity of servers and systems.
- it offers a flexible attack search engine, which scans http requests and responses on the server side in real time without affecting the web server performance. This feature can be considered very important as it does not add significant overhead to the overall attack detection process.
- the incorporation of new attack patterns can take place easily and in a manageable way, thus providing a flexible expansion capability.
- by scanning http responses, attacks such as stored XSSs can be detected, a feature that cannot be found in other web application firewalls.

In this paper, we first provide the necessary background information on the subject by explaining the way a web application is structured and operates (Section 2.1), and by presenting the HTTP messages involved (Section 2.2). Then, the most important web vulnerabilities, as they appear in the OWASP TOP 10 Project (OWASP, 2007), are presented (Section 2.3). A presentation of the web attacks that the proposed new software module detects follows (Section 2.4). Afterwards, the related work on web application security is analyzed (Section 3). Then, a thorough analysis of the features, i.e. how it works and what it detects, of the new software (Section 4) called WebDefender, namely is provided (Section 5). Finally, future improvements are discussed (Section 6) and concluding remarks (Section 7) are presented.

2. Background Information

2.1. Web applications

Web applications are three-tiered applications that mostly use the TCP/IP protocol and follow the client/server architecture (Shkla, 2003). The web browser lies on the first tier, the web server on the second while the database server on the third. The web browser requests web pages while the web server calls on intermediate software that uses appropriate technologies, such as PHP, ASP, ASP.NET,
CGI, JSP/Java etc, to create dynamic web pages. In order to create the webpage requested, the web server queries the database, which stores all the information of the web application and, then, fetches the information needed. Subsequently, the webpage is created and the web server serves the requested page to the browser. This functionality is depicted in figure 1.

![Figure 1 Web application architecture](image)

It is obvious from figure 1 that the web server plays both the role of the server, when receiving requests from a web browser, and the role of the client, when it sends queries to the database

### 2.2. HTTP Protocol

Web servers and web browsers communicate using the HTTP protocol. Web servers are often called HTTP servers while web browsers are called HTTP clients. The main idea is to allow the web browser to request and get a specific object from the web server (Comer, 2003). The basic HTTP methods are: GET, POST, HEAD and PUT.

The request to a web server contains in the first line the method used, the identity of the resource and the protocol version being used (request-line). The rest of the message includes headers and the message body.
The first line of the response contains the protocol version, a numeric status code and its explanation as a string (status-line). The rest of the message includes headers and the message body. Details about the syntax of http messages can be found in the HTTP specification (Fielding et al, 2009).

### 2.3. Web Application Vulnerabilities

Our web vulnerabilities study was based on OWASP’s top 10 project (OWASP, 2007). OWASP is a worldwide non-profit organization, whose purpose is the improvement of web application security. OWASP implements numerous projects to help software engineers increase the security of the web applications they develop (Fielding et al, 2009). Indicative projects include:

- OWASP WebScarab Project: a tool which detects security problems in web applications and web services.
- OWASP Java Project and OWASP PHP Project: for supporting Java and PHP developers.
- OWASP Top 10 Project: a project which presents the most critical vulnerabilities.

According to the latest Top 10 Web Vulnerabilities projects (OWASP, 2007), the most critical vulnerabilities are the ones shown in Table 1.

<table>
<thead>
<tr>
<th>OWASP Top 10 Web Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-site scripting (CSS) or XSS</td>
</tr>
<tr>
<td>Injection flaws</td>
</tr>
<tr>
<td>Malicious File Execution</td>
</tr>
<tr>
<td>Insecure Direct Object Reference</td>
</tr>
<tr>
<td>Cross Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>Information Leakage and Improper Error Handling</td>
</tr>
<tr>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>Insecure Cryptographic Storage</td>
</tr>
<tr>
<td>Insecure Communications</td>
</tr>
<tr>
<td>Failure to Restrict URL Access</td>
</tr>
</tbody>
</table>
2.4. Web Attacks

There are numerous ways to exploit a web vulnerability. Whenever someone exploits a web vulnerability, the web application is attacked. Some web vulnerabilities, such as Information Leakage and Improper Error Handling, are difficult to detect automatically or exploit in an automated way. There exist tools that may cause these errors to appear but none of them is capable of verifying the meaning of these messages and whether they are useful to attackers, which indicates the difficulty of detecting these attacks. On the other hand, attacks based on pattern recognition are easier to detect since there are specific pattern that can be transformed into rules to help in the identification process.

In this section, we describe several common web attacks (Watson D., 2007)(Shema M., 2010), whose detection is based on pattern recognition. It is easy to define rules to identify most of these attacks, while there still exist some others for which it is very complex to create logic rules to detect them (Johns M. et al, 2010). So, the attacks that are based on pattern recognition are detected in http messages in places that user input is expected. Such web attacks are detected and prevented by WebDefender.

2.5. XSS attack

During this attack, the attacker sends malicious material to the web application and retrieves data or he may add his own material on the websites of the application. It is also possible to retrieve session data, such as cookies and the sessionid. The source code of static web pages can be checked before being executed by the web browser but this cannot happen in a dynamic web page environment because the html code is dynamically generated. To perform this attack, we need a script written in JavaScript, VBScript, ActiveX, HTML or Flash which can be embedded or appear as a hyperlink (Di Lucca et al, 2004). Web pages that seem to be targeted for XSS attacks are either those that communicate with the database, login pages or the ones that remind the user of a forgotten password.

2.6. Injection Attacks

This attack category includes attacks based on input that may contain malicious context that could trick the code to be executed (Wassermann, G.et al, 2007). The most well-known attack of this category is the sql injection attack. The attacker inputs special strings that could modify the sql statement to run on the database server in a way that results in actions that the user could not have accomplished through any other means. Such actions include the modification of the database schema by deleting or altering tables, data modifications like deleting or updating records and the displaying of data that the attacker is not authorized to view. Input validation decreases the likelihood of sql injection exploitation. An input is valid when is does not contain dangerous characters like ‘, “, --, #, and words like union, delete, drop, insert, exec etc. Input validation should be located in HTML forms and placed where the code gets the query strings parameters. Dangerous characters and words should not be allowed to pass through the SQL parser and to be executed on the database server.

2.7. Command Injection
This attack transforms the web application into a pseudo system shell and executes the commands that the attacker has given. Then the attacker becomes a system user having the permissions of the web application on the system target. The commands can be injected wherever the attacker can input data such as forms, cookies, http headers etc. The attacker can inject commands, which are usually system commands like system(), is in Unix or ‘dir C:/’ in Windows. This is the way to run commands on the operating system without even being logged on. If the web application runs under root permissions, it becomes quite obvious that the attacker could be able to run any possible command.

2.8. Path Traversal

The attacker intends to access files and directories outside the web root directory. The web root directory is the place where the application source code and other files lie on the web server. This attack exploits the vulnerability 'Insecure Direct Object Reference'. While the attacker navigates, he detects absolute URLs that point to files. By modifying these variables, he can access files outside the web root directory. All he has to do is just to repeatedly add the characters '../' to be able to jump to other directories. For example, if he inserts the following string ‘../../../../etc/passwd’, he will be able to recover the passwd file system on Linux.

3. Related Work

As mentioned above, organizations such as OWASP and WebAppSec study and develop techniques to face web application attacks (Web Application Security Consortium Projects, 2010), (OWASP Projects, 2010). Many tools have been developed which either examine a web application for vulnerabilities, called penetration testing tools or web application vulnerability scanners, or detect any malicious activity in real time. The latter are called web application firewalls and intrusion detection systems.

Penetration testing tools are used during the application testing. Most of them are configured to work in the user’s environment by setting the web browser to use them as proxy servers. Known techniques are tested in an automated way. Some popular penetration testing tools are Paros (Paros, 2010), WebScarab (WebScarab, 2010) and Acunetix (Acunetix, 2010).

Web application firewalls (WAFs) protect web servers. Even though they often come with hardware support, there are also WAFs that are purely software-based and work as web server plug-ins. dotDefender (dotDefender, 2010) and ModSecurity (ModSecurity, 2010) are characteristic examples. WAFs detect and prevent attacks in real time based on rulesets and they are often compared to intrusion detection systems since they record malicious activity as well.

Apart from these tools, there also exist APIs for programming languages which increase the security of web applications (OWASP, 2010). Furthermore, there are projects that advise developers on how to design and develop their applications with security in mind. Many of these projects can be found at OWASP’s projects list (Web Application Security Consortium Projects, 2010).

Most of the web vulnerabilities that were mentioned in the previous section can be easily detected in the http messages. On the contrary, some others, such as Information Leakage and Improper Error
Handling, cannot be detected in an automated way since there are no criteria to determine when an error message provides many details or to evaluate a message context. Other attacks, like session management, can be detected using more complicated techniques such as intelligence or logic deduction; but these techniques decrease the web server’s performance and boost the execution time of a request.

As mentioned, WebDefender detects web attacks based on pattern recognition techniques. Although Web vulnerability scanners (Paros, 2010) (WebScarab, 2010) (Acunetix, 2010) work in the same way, they should be configured on a web client and they do not detect real time attacks. Furthermore, WAFs could be used to detect and prevent web attacks. The most reliable WAFs are based on hardware and not only software. More specifically, although software WAFs provide a lower cost solution, according to PCI Information Supplement (PCI Standards, 2008), hardware or appliance WAFs are suggested especially for high-volume sites or critical applications that need to support critical information. The disadvantage of these tools is that they check only the incoming traffic to the web server and not what the web server sends back.

WebDefender is a lightweight tool, which runs on a web server without deteriorating the web server’s performance having many web application firewall attributes. Some of its advantages compared to other tools include its simplicity to install, its easy parameterization, since the parameterization and administration are done through a simple web interface and its flexible search engine. WebDefender is easily expandable since other attacks which could be detected through pattern recognition can be added, while it can easily support modules that detect other attacks based on logic operations, such as session management. The system administrator can improve, add or temporarily disable attack patterns that WebDefender should search for. WebDefender can also detect web attacks stored in the database as it validates the http responses too. Thus, WebDefender is capable of detecting attacks that exist in the outgoing traffic of the web server, a feature not supported by other tools.

4. WebDefender

4.1. Requirements Analysis

WebDefender should control and validate http conversations between web browsers and the web server but clients should not be aware of the existence of the tool. Every attack detected should be recorded in the database in order to be able to review the necessary details later. WebDefender should validate all the http requests before they are sent to the web server, in order to detect web attacks that are the consequence of the lack of input validation. Meanwhile, http responses should be validated and searched for malicious context in order to prevent stored CSS.

Furthermore, a graphical user interface should give the system administrator the ability to view every suspect activity recorded, while he should also be able to parameterize WebDefender by setting basic parameters like the host name or the port the web server should listen to. Moreover, the system administrator should be able to set as inactive any attack categories he is not interested in having recorded or to modify the existing search patterns. Finally, he should be able to view statistics on the number of attacks recorded in every category.
4.2. System Overview

Figure 2 shows the communication flow whenever WebDefender is used.

![Figure 2 HTTP communication using WebDefender](image)

WebDefender runs as a separate process on the web server. It has been developed and tested on Linux and Apache but it can also be configured to run on IIS. The implementation language is PHP and the user interface is implemented in PHP-GTK2. All information about web attacks and all the results found are stored in a PostgreSQL database.

As already mentioned WebDefender works as a proxy server while it interferes in the web browser and web server communications. The web attacks identification is based on patterns and regular expressions. Web attacks to be detected are cross-site scripting (both stored and reflected), SQL injection, command injection and path traversal. For each one of them, regular expressions are defined, which can be modified and improved by the system administrator.

WebDefender accepts incoming messages from web browsers, checks their validity and sends them to the web server only if they are secure. If they are insecure, WebDefender records the attack and creates a new response to the web browser, which includes important information about the web attack. If the incoming message is found to be secure, WebDefender sends it to the web server and waits for the web server’s response. When the web server’s response is received, WebDefender validates the response before it is sent to the web browser. If the response is secure, then it is sent to the web browser. Otherwise, the malicious context is removed and the final message is sent to the web browser. In this case, an appropriate information message about the removal of the malicious context can be displayed to the final user.

Through the WebDefender interface, the administrator is able to be informed about every single web attack recorded. For each one of them, the information displayed includes the HTTP message, the web attack category and the IP address the attack came from. Furthermore, the administrator is able to modify the web attack patterns and the basic configuration settings, such as the web server name or the port the web server should listen to. Finally, graphs are provided that depict the percentages of the attack incidents.
4.3. Implementation Design

Three main UML diagrams, the state, the activity and the component diagrams, as well as the design of data stores are provided to describe the implementation design of WebDefender.

4.4. State Diagram

The state diagram (Figure 3) shows that the proxy server connects to port 80 when it receives the incoming data. If the port is already in use, the execution stops, otherwise the proxy server serves the clients (web browsers) by receiving incoming messages. When the proxy server receives a request, it checks if this message is secure. If it is, then the request is forwarded to the web server. If it is not, then the client is informed and the communication ends by letting the proxy server serve other requests. When the secure request is forwarded to the web server, the proxy server waits for the web server response. When the response is received, the proxy server validates this response. If it is secure, then it is sent to the web browser, the webpage serving is completed and the proxy server continues with new messages. If not, the response is ‘cleaned’ and the client is informed. The proxy server execution continues until there is a signal interrupt either by the operating system or by the system administrator.

Figure 3 State diagram of WebDefender

4.5. Activity diagram

The activity diagram provides a clear way to show how a proxy server processes an http conversation, such as a simple webpage serving. As discussed above, the web attacks can be detected either in the
request or in the response messages. This is the reason why WebDefender searches both requests and responses. The activity diagram of figure 4 depicts the process for a single http conversation.

Figure 4 Activity diagram of WebDefender

![Activity diagram of WebDefender]

4.6. Component diagram

The component diagram of figure 5 describes the system architecture, showing the relations between the nodes and the components of WebDefender.
Figure 5 Component diagram of the overall System

Observing the component diagram, it is obvious that the web browser and the web server do not communicate directly, but WebDefender interferes. WebDefender receives data on port 80 and communicates also with the database server.

WebDefender consists of software that implements the proxy server, which depends on php and a log file for auditing the run-time errors. WebDefender also consists of a graphical user interface which requires php-gtk2 to run.

The only restriction for a web server is to have been parameterized to accept incoming data on port 49152 or any other port the system administrator sets through the graphical user interface.

4.6.1. Data stores

WebDefender stores data in two places. The first one is a log file in the file system. By default, the path is "/var/log/apache2/webdefender.log". This log file stores error messages during the proxy server’s execution that are related to host connectivity, port binding etc. The file name and the path can be changed by the system administrator through the graphical interface.

The second place that data are stored is a database. The database named ProxyDB is a PostgreSQL database which stores information about the attacks, the proxy server’s configuration and the incidents detected and recorded. The database schema is shown in figure 6.
Table `configuration` stores the WebDefender configuration such as the ports and the host name. Table `attacks` stores the attacks WebDefender detects while their patterns are stored in table `attackpatterns`. Table `conversations` stores every incident recorded by the tool, which means every possible attack found. Finally, table `convattacks` stores the relationships between conversations and attacks.

### 4.7. Attack Patterns

As mentioned, WebDefender detects five different attack categories: reflected XSS, stored XSS, sql injection, path traversal and command injection. Each one of them has different search patterns which could be matched either in the http request or in the http response. These patterns are shown in table 2. Table 2 also shows where the attack could be found. REQ indicates that the attack should be searched in the request message while RESP that it should be searched in the response message.

<table>
<thead>
<tr>
<th>Attacks and Search Patterns</th>
<th>Search at:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cross-Site Scripting</strong></td>
<td><strong>REQ</strong></td>
</tr>
<tr>
<td>`((%3C)</td>
<td>&lt;(%69)</td>
</tr>
<tr>
<td>`((%3C)</td>
<td>&lt;(%69)</td>
</tr>
<tr>
<td><code>SQL Injection</code></td>
<td><strong>REQ</strong></td>
</tr>
<tr>
<td>`((%27)</td>
<td>(</td>
</tr>
</tbody>
</table>
These patterns are used as PERL-compatible regular expressions. WebDefender accepts a request and looks for these patterns in this request. If a pattern is matched, the request is considered insecure and the http communication is completed. If the request is found secure, then the web server’s response is searched for the patterns. In case the response is insecure, all the dangerous characters are removed and the ‘clean’ message is sent back to the web browser.

4.8. WebDefender operation examples

In this section we will see how WebDefender detects and prevents web attacks. We present an example of a cross-site scripting attack first without using WebDefender and then using WebDefender. We have to mention that for demonstration reasons here, when WebDefender detects an attack, the attack information is shown in the browser. Of course, in its production version, when WebDefender stops an attack and the information is not shown in the web browser and is only available to the system administrator.

In order to show the cross-site scripting attack we use a simple form that registers a new user. As username we enter a JavaScript command, <script>alert('XSS vuln!');</script>. This is shown in figure 7.
This value is stored in the database since the web application does not validate any input. When this value is fetched from the database, the command will execute resulting in a stored XSS.

The next step is to execute the stored XSS. To do so, we need to login on the application as administrator and try to view all the registered users of the application. At this time the stored XSS is executed. (Figure 8)
If WebDefender is used we will not be able to enter the malicious string in the database. In the case of a new user, WebDefender detects the XSS attack and shows a message as in Figure 9.

**Figure 9 XSS attack detection and prevention**

- Cross-Site Scripting: variable is name value '%|script%3E%3Al%27%27%3C%2Fscript%3E' pattern '%|script%3E'
WebDefender provides information about the name of the attack detected, the parameter in which the attack was attempted, its value passed and the search pattern matched. In the case of the stored XSS, WebDefender displays the page shown in figure 10. In this case, the JavaScript command is never executed since WebDefender has already removed the dangerous characters defined by the ‘clean’ pattern seen in the web attacks editor. The malicious context is highlighted and a warning appears at the top of the page.

**Figure 10 Stored XSS detection and prevention**

![Image of stored XSS detection](image)

4.9. **Administration GUI**

As mentioned before, WebDefender runs a graphical user interface to support the parameterization of the proxy server. Figure 11 shows the WebDefender’s GUI. In figure 11, we can see the table showing all the incidents recorded to date in descending order.
Figure 11 WebDefender main screen: Incidents recorded

Figure 12 shows the web attacks editor. For each web attack, we can define the http message WebDefender should search, the attack patterns to use and an optional pattern used when the http response should be cleaned of dangerous characters. In fact, this pattern indicates the dangerous characters WebDefender should remove when the attack pattern is detected. In the current tool version, this pattern is used only in the case of stored cross-site scripting to clean up the dangerous characters from the web response. Finally, we may also set a web attack as active or inactive.

Figure 12 Web attacks editor
Figure 13 shows the diagram that WebDefender exports displaying the percentage of web attacks recorded.

### Figure 13 Statistics diagram

![Diagram showing the percentage of web attacks recorded by different attack types.](image)

5. **Evaluation**

WebDefender was compared to dotDefender (dotDefender, 2010) and ModSecurity (ModSecurity, 2010). dotDefender is a commercial application firewall while ModSecurity is a widely used open source application firewall. Over 200 attack patterns were used to compare the effectiveness of these three tools. These patterns were based on common techniques from OWASP rules and from ways hackers use to exploit web applications vulnerabilities. The attack patterns used were categorized according to the attack they are used to exploit.

The number of the attack patterns detected by each tool is expressed as a percentage. WebDefender and ModSecurity have almost the same percentage of success, 8 percentage units ahead of dotDefender. Specifically, the percentages of success of each tool are: WebDefender 63.8%, dotDefender 55.7% and ModSecurity 64.3%. ModSecurity records the highest percentage, which was expected since this tool is one of the best and most ‘mature’ web application firewalls. However, a weakness on identifying sql injections came up. dotDefender showed a great weakness in detecting command injections and null-byte injections since almost all of them were recognized as bad url encoding. WebDefender had satisfying results for every attack category examined but lacks the ability to
check the parameters for encoding. For each attack category the results are shown in figures 14-17. The results were as expected since regular expressions cannot guarantee a 100% detection success. From our experience, if these checks become stricter then the possibilities to have false positives will increase.

Some indicative attack patterns are displayed in table 3 where w stands for WebDefender, d for dotDefender and m for ModSecurity. This table also presents the results for each tool, where 1 indicates that the attack was found and 0 otherwise.

### Table 3 Attack patterns examples

<table>
<thead>
<tr>
<th>Attack</th>
<th>w</th>
<th>d</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command Injection</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>;id;</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>;read;</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>;netstat -a;</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
| \
etstat -a\n                | 0 | 1 | 0 |
| ls –la                        | 1 | 0 | 1 |
| **Path Traversal**            |   |   |   |
| /etc/passwd                   | 1 | 1 | 1 |
| /etc/shadow                   | 1 | 0 | 1 |
| %00.../*/.../../etc/passwd    | 0 | 1 | 1 |
| /*.../*/.../.../..boot.ini    | 1 | 1 | 1 |
| ...%c0%af...%c0%af...%c0%af...%c0%af...%c0%af...%c0%af...boot.ini | 0 | 0 | 1 |
| **Cross-Site Scripting**      |   |   |   |
| `<script>alert(document.cookie)</script>` | 1 | 1 | 1 |
| `&lt;script&gt;alert(document.cookie)&lt;/script&gt;` | 1 | 1 | 1 |
| `<IMG%20SRC='javascript:alert(document.cookie)'>` | 1 | 1 | 1 |
| `&lt;script&gt;alert(document.cookie);&lt;script&gt;alert` | 0 | 0 | 1 |
| **SQL Injection**             |   |   |   |
| `;`                           | 1 | 1 | 0 |
| `#`                           | 0 | 0 | 0 |
| `--`;                         | 1 | 1 | 0 |
| `;`                           | 0 | 0 | 0 |
| or 'x'='x                     | 1 | 1 | 1 |
| ' or 0=0 `#                    | 1 | 1 | 1 |
| `; or a=a--`                   | 1 | 1 | 0 |
| `%20or%201=1`                  | 0 | 1 | 1 |

Figures 14 - 17 depict the results for every tool for every attack category examined.
During the SQL injection examination, WebDefender detected a large number of patterns having a success percentage of 55.24%. dotDefender came next with a percentage of 49.52%. ModSecurity was unable to successfully detect most of the attack patterns, as it had a success percentage of 36.19%.

Figure 14 Tools comparison for SQL Injection

In the command injection category, ModSecurity detected the 66.67% of the attack patterns and came first. WebDefender detected 60% of the patterns and dotDefender detected the least number of attacks.

Figure 15 Tools comparison for command injection

The third category examined was path traversal. ModSecurity detected almost all of the patterns since its success percentage was 95.83%. WebDefender was able to detect 72.22% of the patterns while dotDefender only 62.50% of them.
CSS was the last category to examined. ModSecurity detected all of the patterns. dotDefender was second by detecting 88,89% leaving WebDefender at the last place with a percentage of 83,33%.

The ability to identify stored XSS could not be evaluated since none of the other tools validate http responses while WebDefender does. A great advantage of WebDefender is that its rulesets can easily be modified. It may also run on any web server while the other tools have limitations.

6. Future work

Even though the current version of WebDefender, as presented in the previous sections, is quite powerful, there are still open issues that can enhance its performance and functionality. First of all, a connection to port 443 should be supported in order to validate messages through SSL. Secondly, web attacks, which are not based on pattern recognition, should be detected through methods based on more complicated actions. For example, session attacks should be detected and prevented by checking
that the session cookie comes from single IP address, its ID is not repeated or it expires as soon as possible.

Moreover, the web server’s response encoding should be supported, especially when the response is not encoded. Other work that should be done is to store the most regularly used pages temporarily in the proxy server so that the payload to the web server decreases.

WebDefender could also support more advanced configuration settings by giving the administrator the ability to choose how the proxy server should act when identifying a web attack. For example, the user could be allowed to continue to navigate and in parallel to record all of his movements. The navigation should be terminated after a specified number of recorded incidents. This was not implemented in this version of WebDefender since this tool is currently used only for academic purposes. This extra feature will give WebDefender the ability to act differently when a web attack is detected. This very useful feature is missing from most of the existing tools.

WebDefender could keep a list of all the files of a web application and the associated access rules in order to serve the browser requests based on these rules. This is a way to minimize problems such as path traversal, malicious files execution and failure to restrict URL access since it would be known if the access to that file is allowed or not. That means that the appropriate infrastructure should exist to support the above feature without decreasing the proxy server's efficiency and performance and without increasing the webpage execution time which could lead to delays in clients' serving.

7. Conclusions

This paper presented WebDefender, a cross-platform tool which aims to protect web applications that are installed on any web server. This tool has features of both penetration testing tools and web application firewalls providing great flexibility and expandability. A comparison to web application firewalls was performed, demonstrating a good level of performance. Preventing attacks on web applications based only on software does not seem to be 100% successful, an expected outcome when we talk about security. While efforts are made through the development of security APIs and new standards and guidelines for secure web applications development are investigated, the problem remains acute. However, the various security tools are continuously being improved and new techniques appear on the scene. The biggest problem can be attributed to the design of the applications, since many important security issues are not taken into account.

References


