ISSN: 2278-6244
Impact Factor: 6.690

OPEN SOURCE RULES FOR REAL-TIME PROTECTION OF WEB SERVER

Ekaterina Dudin, Asstt. Prof., PhD, University of Telecommunications and Post, Sofia, Bulgaria

Anna Otsetova, Asstt. Prof., PhD, University of Telecommunications and Post, Sofia, Bulgaria

Abstract: *Modern web-based applications often remain open for hacker attacks and vulnerabilities of the server operating system. This fact requires the additional protection of the web server and the software it uses. This paper presents the main types of hacker attacks and the ability to prevent them using ModSecurity-based protection rules provided by Open Web Application Security Project. The ModSecurity modul, applying a set of rules for protection by inspecting incoming traffic and the response to these requests by the server was discussed. Practical results of their impact on various attacks - HTTP (fingerprinting), DoS (Denial of Service), DDoS (Distributed Denial of Service), SQL injections, etc. were presented.*

Keywords: *ModSecurity, Rules, HTTP fingerprinting, DoS, DDoS, SQL injections*

INTRODUCTION

Attacks to modern web applications are characterized by a different approach, scope and purpose [3]. To achieve information security of modern web applications, the following options are use:

- Firewalls;
- Administrative accounts for access to databases;
- Terminate access by using the Internet Protocol Message Protocol (ICMP) and Simple Network Management Protocol (SNMP);
- Providing protection for both the operating system and the used applications;
- Regular updates and patches on servers,
- Checking and validating the input data in order to verify code.

In order to ensure stable protection of a web server, it is necessary to periodically assess the known vulnerabilities, in parallel with the updating of the software used and the regular updating of the used technologies [6].

This paper proposes the use of open source real-time server protection policies provided by the Open Web Application Security Project (OWASP).

The purpose of the report is to offer practical solutions for developing real and complex

rules for the protection of a real-time web server.

BASIC WEB SERVER ATTACKS

The most common hacker attacks on modern web servers are [10]:

1. *HTTP fingerprinting attacks*.

Each server is characterized by its unique profiling. This circumstance enables the identification of both the type and the version of the software that is used [7]. HTTP fingerprinting attacks are uniquely identifiable (on the fingerprint principle). To scan for HTTP attacks, programs such as Httprint (operating under Windows, Linux, and Mac OSX Httprecon) are used [2].

2. *DoS (Denial of Services).*

The purpose of a DoS attack is to block the server by filling its operating memory and fully occupying its resources. The sender sends a large number of immediate requests, thus making it difficult to service the real users [3, 9]. The fact that attacks on the server are multiple queries from one user makes it easier to protect.

3. *DDoS (Distributed Denial of Service).*

DDoS attacks aim to make the server resources inaccessible for a certain period of time or permanently. In these cases, the server does not distinguish the malicious from the legitimate request, since both types of requests use the same protocols and ports [2]. The main steps to prevent their action are:

- Ensuring bandwidth surplus for incoming traffic - this is one of the easiest ways to protect a server from lower-level DDoS attacks, but this approach is costly;
- Using an Intrusion Detection System (IDS);
- Using a product to protect against DDoS attacks. Several manufacturers offer devices designed specifically to detect and provide DDoS protection and prevention that are specifically designed to detect and frustrate the DDoS attacks;
- Back up Internet connection with a separate base with Internet addresses for critical users. This alternative is used in case the primary chain is overloaded with malicious queries.

Unlike DoS attacks, DDoS are sent from hundreds of sources simultaneously, this circumstance make protection complex and even in some cases impossible [7]. This paper does not offer comprehensive DDoS attack protection rules, as DDoS attacks are filtered at the internet provider level.

4. *SQL injections* - a code injection technique used to attack a web application without filtration or other protection methods [8]. This type of attack allows the use of the database information on the server [2].

5. *Shell command execution* - a combined technique for achieving maximum effect. The effects of these attacks consist of three basic steps [7, 12]:

First step:

Select applications vulnerable to SQL injection, and then create a *.php* extension file that records the desired content of the attacker.

Step Two:

Create a file containing the System command *($ _ REQUEST ['cmd']*).

Step Three:

Deleting server content - *www.site.com/exec.php?cmd=rm-rf/.*

6. *Attack Brute force attacks* - used to break passwords by using all possible combinations of letters, numbers and characters set in the attack algorithm. An effective protection measure for this type of attack is to limit the number of attempts from one source to identify a user [5].

OPEN SECURITY POLICIES USAGE

ModSecurity is a web-based firewall application [9]. It controls inbound and outbound data streams by applying a set of real-time protection rules. The principle of this module is presented in *Figure 1* [1].

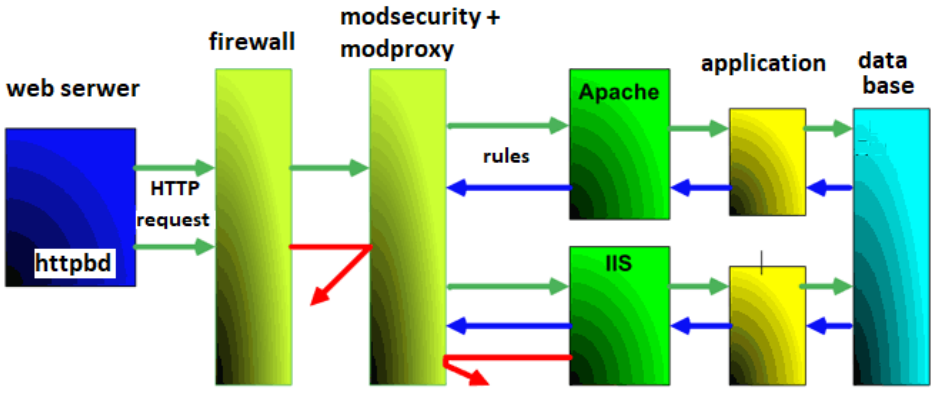

Fig. 1. *ModSecurity*

When detecting suspicious traffic or abnormal behavior on the part of the user, *ModSecurity* acts as a borderline that verifies malicious content by skipping or blocking requests received according to the result of the previous step. *ModSecurity* activates procedures and offers a number of options, including some injecting content into communication, as well as a full

HTTP transaction entry [1, 2].

The structure of the archive and the access to these rules are presented on the OWASP Foundation's official website: *https://github.com/SpiderLabs/owaspmodsecurity-crs* [11].

The main benefits of using these policies are:

Flexibility when creating and editing security rules - using text editors such as notepad, vi, nano, emacs, etc. [4, 6].

- Possibility to create own rules,
- An opportunity to develop comprehensive and strong protection rules.

When an attack is detected, the *ModSecurity* module provides the option of selecting a user action option [11]:

- recording the information in a log,
- blocking incoming traffic,
- sending an e-mail to the site administrator,
- sending commands to the server operating system,
- running a certain external script file.

PRACTICAL IMPLEMENTATION OF COMPLEX RULES FOR PROTECTION

ModSecurity is compatible with Apache Foundation, Nginx, and Microsoft Internet Information Services (IIS).

This paper proposes the use of a virtual container installed with OS Ubunto 14.04.3, Apache 2.4.27, PHP, MySQL and OWASP ModSecurity Core Rule Set (CRS) Version 3. In addition, a WordPress system is installed to provide additional protection from common attacks and specific vulnerabilities.

For the purpose of our paper, the following *ModSecurity* configuration was used:

root@example:/etc/modsecurity# tree

modsecurity_crs_10_setup.conf

modsecurity_crs_11_dos_protection.conf

server-baner-protection.conf

shell-exec-block.conf

sql-injection-block.conf

wp-brute-force-block.conf

1. *Brute Force Attack Protection*

The following logic is suggested - when entering a user and password, the server responses

have the following statuses:

- 200 for wrong password and return page to login,
- 302 Redirect - redirecting to the administrative part when correctly identified to the system,
- 401 - deny access and log into the log file of an "*ip address blocked*" message.

Applying this mechanism we check the status of the response to the client and provide the opportunity to count wrong attempts. Upon reaching a certain limit, the host is blocked for a certain time (for example, for 5 minutes - 10 attempts). When sending 10 POST queries to */wp-login.php*, the server responds to status 401, blocking the attacker and writing the log file data to the server.

==> /var/log/apache2/wp-access.log <==

192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 200 3601
192.168.254.6 - - [12/Une/2017:09:14:16 +0200] "POST /wp-login.php HTTP/1.1" 200 3601

==> /var/log/apache2/modsec_audit.log <==

blog.example.dev 192.168.254.6 - - [12/ Une/2017:09:14:15 +0200] "POST /wp-login.php HTTP/1.1" 401 458 "-" "-" VsAkrH8AAQEAABF@0vUAAAAB "-" /20160214/20160214-0854/20160214-085436-VsAkrH8AAQEAABF@0vUAAAAB 0 1110 md5:0e2b472e2caf97d6a685435ef42b058e

==> /var/log/apache2/wp-access.log <==

192.168.254.6 - - [12/Une/2017:09:14:16 +0200] "POST /wp-login.php HTTP/1.1" 401 458

2. *Protection from SQL ingection code* is expressed in the use of a set of OWASP rules. A

content file was used to demonstrate the effectiveness of the rules:

#OR 1

 # DROP sampletable

*# DROP/*comment*/sampletable*

*# DR/**/OP/*bypass blacklisting*/sampletable*

*# SELECT/*avoid-spaces*/password/**/FROM/**/Members*

 # SELECT /!32302 1/0, */ 1 FROM tablename*

 # ‘ or 1=1#

 # ‘ or 1=1-- -

*# ‘ or 1=1/**

 # ' or 1=1;\x00

When attempting to deliver a malicious code, the server response is instantaneous, with the first request:

==> /var/log/apache2/wp-error.log <==

[Wen Une 12 09:16:21.680851 2017] [:error] [pid 4477] [client 192.168.254.6] ModSecurity: Access denied with code 403 (phase 2). Pattern match "(/*!?|*/|[';]--|--[\\\\s\\\\r\\\\n\\\\v\\\\f]|(?:--[^-]*?-)|([^\\\\-&])#.*?[\\\\s\\\\r\\\\n\\\\v\\\\f]|;?\\\\x00)" at ARGS:test. [file "/etc/modsecurity/sql-injection-block.conf"] [line "18"] [id "981231"] [rev "2"] [msg "SQL Comment Sequence Detected."] [data "Matched Data: /* found within ARGS:test: DROP/*comment*/sampletable"] [severity "CRITICAL"] [ver "OWASP_CRS/2.2.9"] [maturity "8"] [accuracy "8"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"] [hostname "blog.example.dev"] [uri "/"] [unique_id "VsAnhX8AAQEAABF9zVgAAAAA"]

==> /var/log/apache2/modsec_audit.log <==

blog.example.dev 192.168.254.6 -- [12/Une/2017:09:16:21+0200] "GET /?test=DROP/*comment*/sampletable HTTP/1.1" 403 279 "-" "-" VsAnhX8AAQEAABF9zVgAAAAA "-" /20160214/20160214-0906/20160214-090645-VsAnhX8AAQEAABF9zVgAAAAA 0 1708 md5:e04d1076a86e8d56b7cd506d2581f14d

==> /var/log/apache2/wp-access.log <==

192.168.254.6 -- [12/Une/2017:09:16:21+0200] "GET /?test=DROP/*comment*/sampletable HTTP/1.1" 403 279

ISSN: 2278-6244
Impact Factor: 6.690

3. Shell command execution protection

To prevent this type of attack, a "*drop*" method is used instead of "*deny*," as a result of the applied rules, the attacker receives a blank response from the server [7]. In this way, no resources are assigned to respond to the interrogated request, but the attacker is deluded (*Figure 2*).

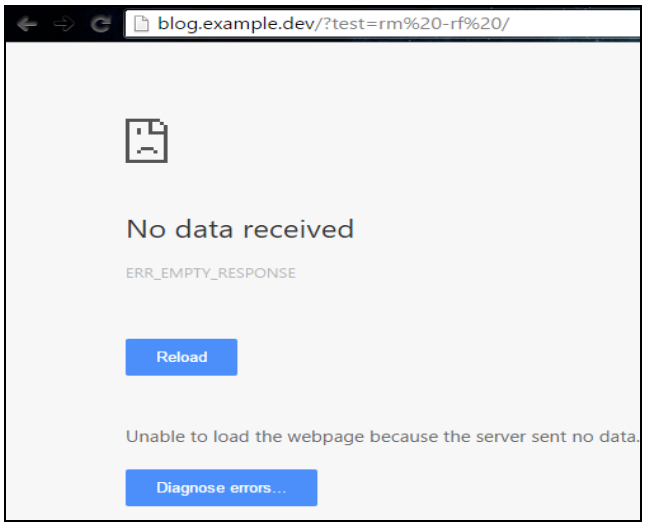

Fig. 2. Shell command execution protection

4. DoS attack protection is provided by experimental rules by setting the following configuration parameters for the system: 60 sec time interval, 100 requests in the activation time interval and 5 min blocking time.

The server response after application of those protection rules is presented in *Figure 3*.

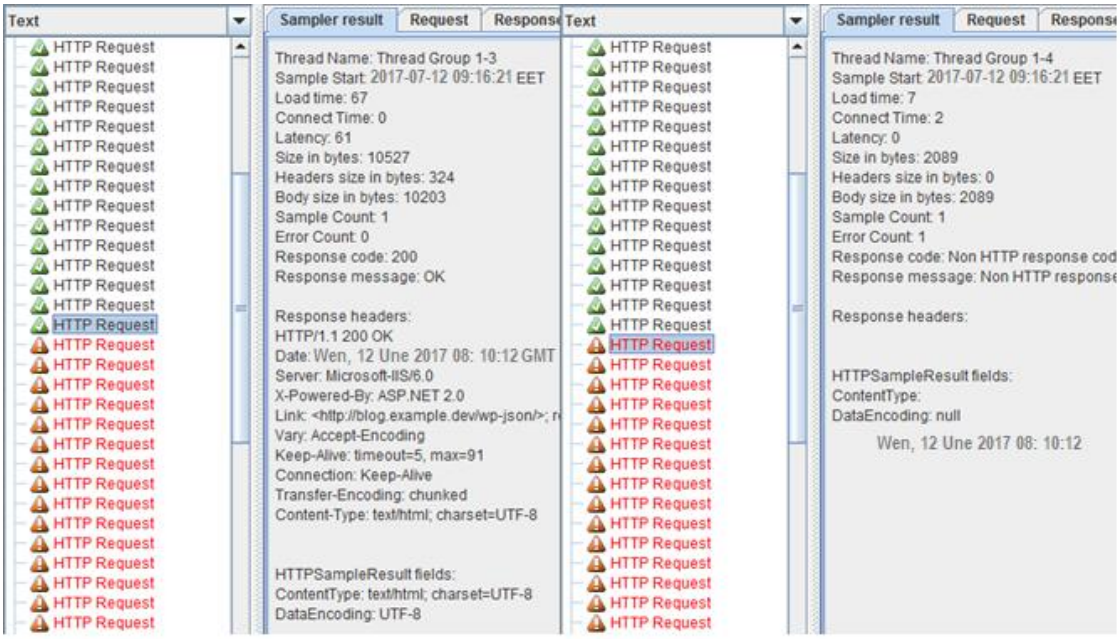

Fig. 3. DoS attack protection

5. HTTP fingerprinting protection with a rule:

SecServerSignature "Microsoft-IIS/6.0"

SecRule &REQUEST_HEADERS:Host "@eq 0" "id:1001,phase:1,deny"

SecRule &REQUEST_HEADERS:Accept "@eq 0" "id:1002,phase:1,deny"

SecRule REQUEST_METHOD !^(get|head|post)$ "id:1003,phase:1,t:lowerCase,deny"

SecRule REQUEST_PROTOCOL !^http/1\.(0|1)$ "id:1004,phase:1,t:lowercase,deny"

Header set X-Powered-By "ASP.NET 2.0"

Header unset Etag

The practical results of applying HTTP fingerprinting protection rules are presented in *Figure 4.*

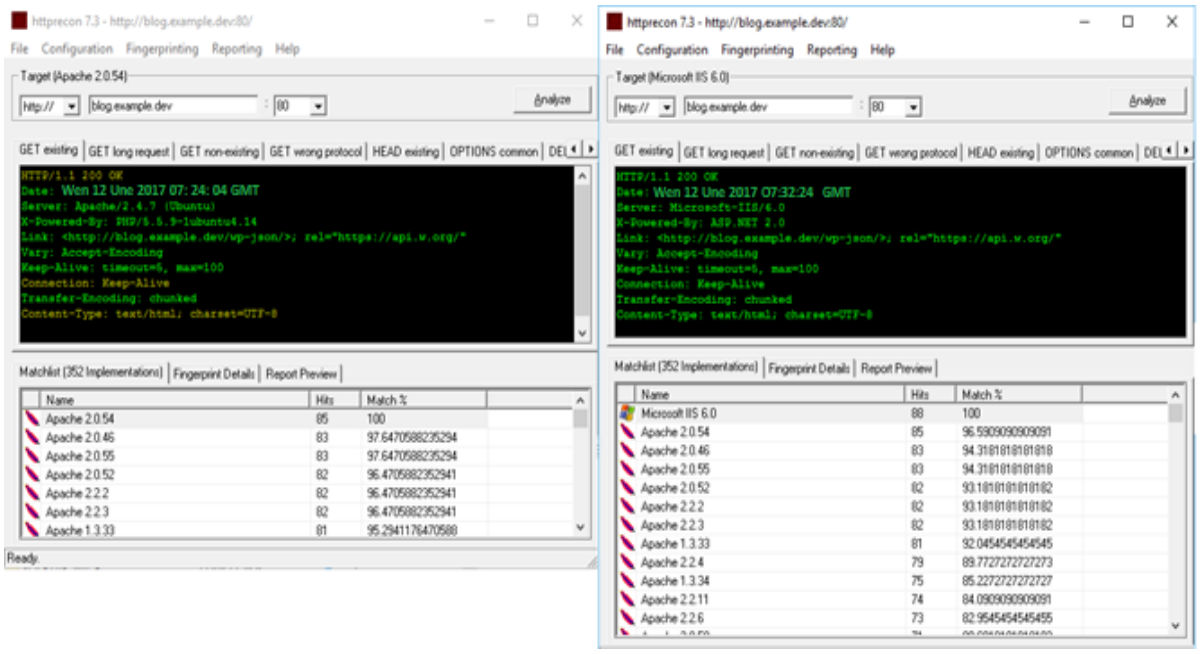

Fig. 4. HTTP fingerprinting protection

CONCLUSION

The paper proposes the use of open source security rules provided by the OWASP Foundation based on the *ModSecurity* module. Advantages of this system include: easy installation, compatibility with the most common web servers - Apache, Nginx and Microsoft IIS and numerous user-friendly rules. Based on the *ModSecurity* module, real, complex real-time web server protection rules are presented against breakthrough techniques and commonly used vulnerabilities in OS Linux. Practical results show that the selected security module provides Successful and secure server protection by neutralizing high- and medium-level threat attacks such as HTTP fingerprinting, Shell command, DoS

attack, SQL injection and Brute Force.

REFERENCES

1. Lochart, A., (2005), Network Security Hacks, O'Reilly, *https://books.google.bg/books?id=5AiAVtIAKsC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false*
2. Scambray, J., Kurtz, G., (2012), Hacking Exposed 7, Network Security Secrets and Solutions by Mcclure, Mc Graw Hill
3. McClure, St., Scambray, J., Kurtz, G., (2011), Hacking Exposed, Network Security Secrets and Solutions, Sixth Edition 6th Edition, Mc Graw Hill
4. Barnett, R., Grossman, J., (2001), Web Application Defender's Cookbook, Battling Hackers and Protecting Users
5. Magnus, M., (2009), *ModSecurity 2.5*, Rasct
6. Ristic, I., (2012), *ModSecurity Handbook*, Feisty Duck
7. Dudin, E., (2017), Web server protection by ModSecurity rules, (in Bulgarian)
8. Folini, Ch., Ristic, I., Modsecurity handbook, Fiesty Dusc, 2017
9. Mod Security 2.9, *https://www.modsecurity.org/,* 2017
10. Httprint, *http://www.net-square.com/httprint.html,* 2016
11. *https://www.owasp.org/,* 2001-2017
12. *https://malware.expert/signatures/,* 2017