International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)

http://ijcjournal.org/

An Architecture for Misconfiguration Patching of Web Services: A Case Study of Apache Server

Franklin Tchakounté[a*], Jean Marie Kuate Fotso[b], Vivient Corneille Kamla[c]

[a,b]*Department of Mathematics and Computer Science, Faculty of Science, University of Ngaoundéré, Cameroon*
[c]*Department of Mathematics and Computer Science, National School of Agro-Industrial Sciences, University of Ngaoundéré, Cameroon*
[a]*Email: tchafros@gmail.com*
[b]*Email: jeanmarie.kuatefotso@gmail.com*
[c]*Email: vckamla@gmail.com*

Abstract

Services are usually left configured by default and therefore subjects to vulnerabilities because they are not security enforced. Web services are so popular that they are targets of attacks to intrusions related to vulnerabilities discovered by attackers. This work proposes an architecture for patching Web service misconfigurations related to existing vulnerabilities. The approach underlying this architecture first retrieves and structures anti-vulnerability measures published by the official service manufacturers. Second, it evaluates the risk level using Common Vulnerability Scoring System (CVSS) on the current state of configurations. The proposed approach has been applied on Apache server on four vulnerabilities: version discovery, XSS, SQL injection and deny of service. Experimental results on a vulnerable environment demonstrate that the proposed approach considerably reduces vulnerabilities compared to similar solutions.

Keywords: CVSS; misconfiguration; patching; risk; vulnerabilities; Web Received on DD MM YYYY; accepted on DD MM YYYY; published on DD MM YYYY.

1. Introduction

The service Web is the most popular on Internet. It has revolutionized how people communicate and how companies operate [1]. It allows exchange of information but e-services such as e-commerce, e-learning etc [2]. Its popularity attracts malicious people who take advantage of various vulnerabilities to infiltrate. According to Marconato [3], vulnerabilities related to misconfiguration remain a serious problem since administrators mainly take default parameters. Existing works to that problem have two orientations [4].

---

* Corresponding author.

The first one consists to passively scan for the presence of vulnerabilities to inform users [7], the second one consists to check some subsidiary files according to pre-defined security directives and to make recommendations according to the actual status of configuration [4]. Existing solutions are more informative than curative. Moreover, security directives are not related to known vulnerabilities for the assessment of risk levels. This work proposes an architecture to identify vulnerabilities in main configuration files, to structure security directives according to the rules retrieved from official editor sites, to compute risk levels associated to the status of service configuration, and to automatically apply security directives on behalf of the user. Unlike existing work which only consider the status of service configuration at the installation, this work consider that this status can evolve over the time. A simulation of this architecture on the service Apache has provided satisfying results. It is able to reduce considerably vulnerabilities. The document is organized in four sections. The first section reviews existing solutions against misconfiguration vulnerabilities. The second section presents concepts about service and management of vulnerabilities. The third section describes the architecture to reduce misconfiguration vulnerabilities. The fourth section is dedicated to the implementation of the solution for Apache and the tests on vulnerable environments. The document ends with conclusion and perspectives.

2. Solutions against misconfiguration vulnerabilities

2.1. Motivation

The default configuration is mainly chosen during installation of services. This method is not controlled by administrators who do not enforced the service with security guards. We aim to propose a solution which automatically fetches and structures security rules from official service manufacturers, then compute the risk level based on the current configuration of security enforcement and takes decision accordingly. This work is the first step which proposes and simulates an architecture to achieve these goals. Simulations is made on Apache server. Authors have proposed some works related to misconfiguring services. They operate on the client side (application and browser) and on the server side.

2.2. Client-side patching

Zoummouri and his colleagues [17] proposed a model of prevention against browser settings configuration attacks to facilitate access to the user's machine. Their system aimed to improve the level of browser prevention against Web attacks. OWASP describes several scanners against Web vulnerabilities [18]. Those which are open source and freeware include OWASP Zed Attack Proxy, OWASP Xenotix XSS Exploit Framework, Vega, Wapiti, Grendel-Scan, and Grabber. They essentially perform a static analysis of the application code and a dynamic analysis during execution of the application, to detect vulnerabilities such as cross-site scripting, remote file inclusion, local file inclusion, HTTP response splitting, SQL injection, command execution and injection XML External Entity (XXE).

2.3. Server-side patching

Some works have been realized to analyze server-side configurations for the detection of hidden vulnerabilities. Nikto [5] is a hybrid open source scanner that performs two activities: static code analysis of applications and

server-side configurations. In the second case activity, it goes through various configuration directories looking for obsolete version files (indexes), and malicious scripts. In addition, it derives potentially harmful elements based on its default path rule base. SCAAMP [4] is a system built to detect vulnerabilities related to the default configurations of AMP (Apache, MySql and PHP) in terms of security in general. It lets the readjustment the configuration parameters of the Web server. This system evaluates the level of risk specific to the current state of configuration to propose recommendations to the user.

2.4. Limitations

Nikto and SCAAMP are found closer to this work. However, they face some limitations. Nikto aims to statically scan vulnerabilities and inform users. It does not deal with core configuration files and does not propose rules against vulnerabilities but relies on information on obsolete versions, access paths, and malicious scripts. It could be subject to false positive because it relies on a priori signature. SCAAMP deals with specific services only during the first installation. The generation of rules is independent of known vulnerability impacts. It computes the risk level for the actual configuration files states and recommend measures based on fixed rules. Nikto is more informative and SCAAMP's evaluation is not related to known vulnerabilities. This work complements Nikto but identifies vulnerabilities in central configuration files to design directives according to the rules retrieved from official editor sites. This work proposes evaluation of risk levels based on status of service configurations and applies directives on behalf of users.

3. Preliminaries

3.1. Services

In the client server architecture, the client is an application requesting resources held by the server application, through messages called requests. These queries are sent from a random source port on the client machine to a known destination port on the server machine [6]. The latter serves these resources through a service that represents the instance in the background of the running application. Some services are file sharing (FTP, SFTP), Web (HTTP), name resolution (DNS). To better ensure the security of these services they must be correctly configured. Configuring a service consists to technically manage its various components, as well as all updates made during its evolution. This configuration is deployed in the form of files and data in three steps:

- Identification of configuration items: Activities here include organization of information, type of information (file formats), naming rules, and identification of relationships between data.
- Control of the configuration: It includes activities confirming the consistency of all the configuration data of the service.
- Storing configuration status: To manage a configuration, it is necessary to be able to control its evolution over time. The evolution concerns the modification of the physical, functional and performance characteristics of the service. This work deals with Web services and consistency of configuration files in the security point of view.

3.2. Vulnerabilities

According to Avizienis *a*nd his colleagues [7], a vulnerability is an internal fault that allows an external fault to damage the system. This definition can be supplemented by the one elaborated in the framework of the MAFTIA project ([8] according to [7]): a vulnerability is an accidental, or intentional, malicious or not, fault in the specifications, the design or the configuration of the system, or in the way it is used, which can be exploited to create an intrusion. Exploitations of vulnerabilities can cause significant losses. They are:

- Direct financial loss: destruction of crucial data, decommissioning of the entire computer system;
- Loss of repudiation: questioning of credibility in the case of disclosure of highly confidential information;
- Loss of time: detection of security vulnerabilities, installation of security patches, efforts to restore destroyed data.

OWASP maintains the TOP 10 list of vulnerabilities [9, 10]. Four vulnerabilities such as version discovery, XSS, SQL injection and deny of service are within the scope of this work.

3.3. Management of vulnerabilities

As shown in Figure 1, the management of vulnerabilities starts with risk analysis, followed by the detection of malicious traces and ending with the application of corrective measures [11]. The analysis of vulnerability identifies risk factors to target and monitor. Considering sophisticated attacker techniques, it is important to first analyzing critical processes especially those that require high levels of compliance and confidentiality. Once critical processes are targeted, an inventory of computer hardware can be made to obtain an overview of the network. It is also important to identify security measures already in place to protect important equipment: internal policies, firewalls, intrusion detection and prevention systems, virtual private networks and information leak prevention systems. The detection of vulnerabilities is usually done by tools called scanners. These tools indicate potential flaws of the scanned network equipment. They list open ports, type and version of operating system and applications running on the server. Other scanners test the presence of the various known faults on services. Some examples are OpenVAS [12] and Nmap [13]. Vulnerabilities can be published to the general public as soon as they are detected by

- International organizations. Open Web Application Security Project (OWASP), National Vulnerability Database (NVD), Common Vulnerabilities and Exposures (CVE), and Forum of Incident Response and Security Teams (FIRST),
- individuals specialized in finding solutions for the detection of vulnerabilities (researchers)
- and interested individuals devoted to fight against attackers.

Application publishers are responsible for continually improving the quality of their product. Once they detect a defect, they put together an anti-defect called patch. A patch is a section of code that is added to software, to make changes. The security patch is nowadays a key element to protect information systems. It is sufficient to

regularly install patches released by the publishers to correct vulnerabilities of host operating systems, products and applications [14]. The integration of these patches can be done using tools such as Microsoft Baseline Security Analyzer (MBSA) or Windows Server Update Services (WSUS) [15]. This patch management is based on a process that goes through several steps [16].

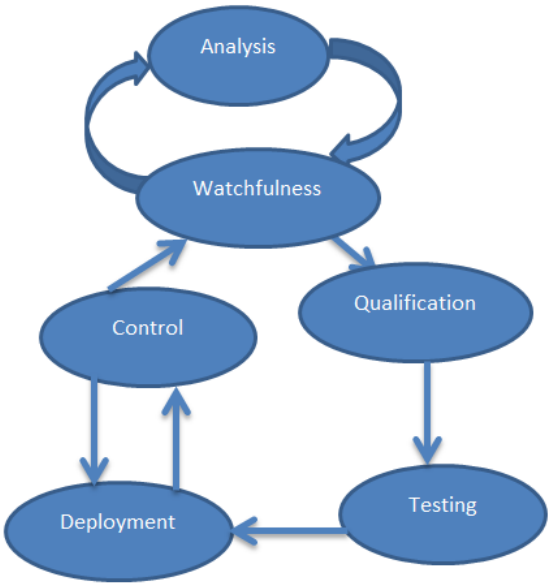

Figure 1: Patch management

4. Architecture for misconfiguration of services

This work proposes an architecture to assist an administrator to reliably apply security rules of the service to be installed. This section starts by making some assumptions, then defines the risk assessment model and describes components of the overall architecture.

4.1. Considerations

We assume that

- The current state of the system in terms of security is correct;
- The proposed approach applies to one service at a time and on a server independently;
- The proposed approach applies to a service that has been successfully started.

We define some concepts for this work.

- S_i is the service running on the process i;
- P_i is the path for the parent folder of the service configuration files i;

$$AC(P_i, U) = \begin{cases} 1 \; if \; U \; has \; all \; the \; permissions \; on \; P_i \\ 0 \; otherwise \end{cases}$$

- $Reload(S_i)$ is the primitive to reload the service S_i

4.2. Risk assessment

Let be,

$V = \{v_1, v_2, v_3 \ldots v_n\}$: the set of possible vulnerabilities on a service

$R = \{r_1, r_2, r_3 \ldots r_n\}$: the set of rules to apply to reduce the impact of a vulnerability

Let consider the event "*Apply rule r for vulnerabilities in V*". The contribution of each rule r_j on the impact α_i of the vulnerability *i* is defined in Equation 1.

$$\alpha_{ij} = P_{r_{ij}} \beta_{r_{ij}} \qquad (1)$$

where *:*

- $1 \leq i \leq cardV$, $1 \leq j \leq cardR$;
- $\alpha_i \epsilon [0, 10]$, the impact of vulnerability i;
- $n_i = card v_i \geq 1$, the number of security rules for the vulnerability i;
- It is assumed that rules have the same probability of apparition. $P_{r_i} = \frac{1}{n_i}$ is the probability that the rule r_i is not applied and 0 otherwise;
- $\beta_{r_{ij}} = \begin{cases} 0 \; if \; the \; rule \; r_j \; is \; applied \\ \frac{\alpha_i}{n_i} \; otherwise \end{cases}$;
- β_r, the level of risk for all the rules of r;

$$\beta_r = \sum_{j=1, i \in N^+}^{n} \alpha_{ij} \qquad (2)$$

Equation 2 represents the model of assessment of the level of risk generated according to the degree of application of anti-vulnerability rules. In this equation, α_i .is unknown. Its determination is made in the following part.

Evaluation of $\boldsymbol{\alpha_i}$

It is adopted the model known Common Vulnerability Scoring System (CVSS) [2] to evaluate α_i. The choice is justified by the fact that it is used by outstanding organizations governing the domain vulnerabilities such as CVE, NVD, and FIRST. Its risk level classification has also been exploited. The assessment of risk level with CVSS depends on Equation (3), which then depends to Equations (4), (5) and (6).

$$BaseScore = (0.6 * Impact + .4 * Exploitability - 1.5) * f(Impact) \quad (3)$$

$$Impact = 10.41 * (1-(1-ConfImpact)*(1-IntegImpact)*(1-AvailImpact)) \quad (4)$$

$$Exploitability = 20 * AccessComplexity * Authentication * AccessVector \quad (5)$$

$$f(Impact) = 0 \text{ if } Impact = 0;\ 1.176 \text{ otherwise} \quad (6)$$

- ComfImpact: the impact of the vulnerability on confidentiality;
- InteImpact: the impact of vulnerability on integrity;
- AvailImpact: the impact of vulnerability on availability;
- AccessComplexity: defines the complexity of access;
- Authentication: defines complexity on authentication;
- AccessVector: defines the access vector;

CVSS evaluates the risk level of the system according to the numerical scores defined by range to which it is added corresponding flags.

	Levels	Values
	null	0.0
	low	0.1-3.9
	medium	4.0-6.9
	high	7.0-8.9
	critical	9.0-10.0

Figure 2: Risk levels

4.3. Architecture

The final system is structured in four modules as shown in Figure 3.

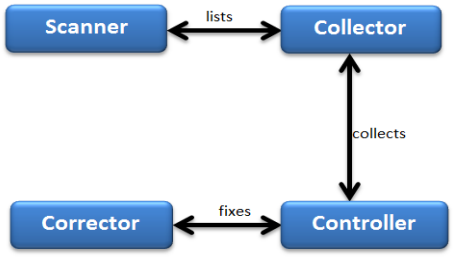

Figure 3: Architecture of the system

First, we monitor characteristics of the service (scanner). Second, we retrieve and structure security rules over the Web through official website of the service's owner (collector). Third, we check the state of the installed

service to see whether some rules have been already applied (controller). Fourth, we evaluate the security risk based on security rules appliance. The system patches according to this risk level (corrector).

Scanner

This module is responsible for scanning the active Web services (S_i) of the system to determine for each service, its name, its version, its active port, its manipulated files, its user *owner (u)*, and its rights *AC* (S_i, *u)*. To optimize operations, the scanner has a database which includes the list of Web services on the host system associated with characteristics. However, there will be a situation of additive (installation) of applications. In this vision, the refresher module is added. The refresher helps the scanner to:

a) take into account newly installed applications;
b) check the identifier (example: version) of the installed application with that of the database to limit the scan. This activity has a big impact on the availability of the entire system.

Collector

As the name suggests, it collects vulnerabilities and rules. It retrieves a report (Web services detected) from the scanner and performs a search for vulnerabilities and fix information on official publisher sites and other credible sources such as CVE. Vulnerabilities on applications are generally known to the public. Publishers sometimes leave in comment in the code of the application the various limits indicating vulnerable functions, updates to apply after installation, configurations to avoid, files to be renamed to adapt with the current configurations. This information is usually more detailed in the user manual submitted by the publishers. The CVE provides in this direction vulnerabilities related to application versions.

After this search, the collector transforms the unstructured information into rules while checking the syntax. It then groups these rules by category of vulnerabilities among which: version disclosure, Injection vulnerability, XSS, and Deny of service.

The rules are in the form of vector *R* with *R = (L, A, O)* where,

- *L* is the location. It gives the exact information about the file and the corresponding directory. In short, it defines the path for the rule to be made.
- *A* defines the action. An action could be to apply a modification on a specific line in configuration files or to enable or disable an option on configuration files.
- *O* denotes operands. In the modification modes, it refers to the instruction on the indicated line. It is usually in the form *(x, y)* where *x* is the variable and *y* is its state ((*x*, *on*) means that *x* is enabled).

Controller

The controller of the system marks the rules. This module allows avoiding configuration repetitions. The user *(u)* applies only once a configuration rule for the service (S_i). To mark the rules, this module negotiates with the host system for the acquisition of *AC* (L_{P_i} , *u)*. The controller completes reports on the state of the rules. These rules are labelled with a Boolean function (0, 1). The meaning of these two values is as follows:

- 0: The rule is not applied yet. However, it may be present but poorly applied;
- 1: The rule is correctly applied by the user.

Corrector

The corrector performs two specific tasks, risk assessment and adequate application of configuration patches. The risk assessment allows the user *(u)* to decide for the application of the service configuration patch (S_i). The higher the level of risk, the more the user will have to patch. The corrector negotiates with the system for the acquisition of *AC* (L_{P_i}, *u)* before applying patches. *Reload*(S_i) is performed to take in account changes on the configuration files. The configuration test is done by restarting the system to obtain a security report later reassuring or not the actual presence of the configuration patch. Figure 4 summarizes activities of this module.

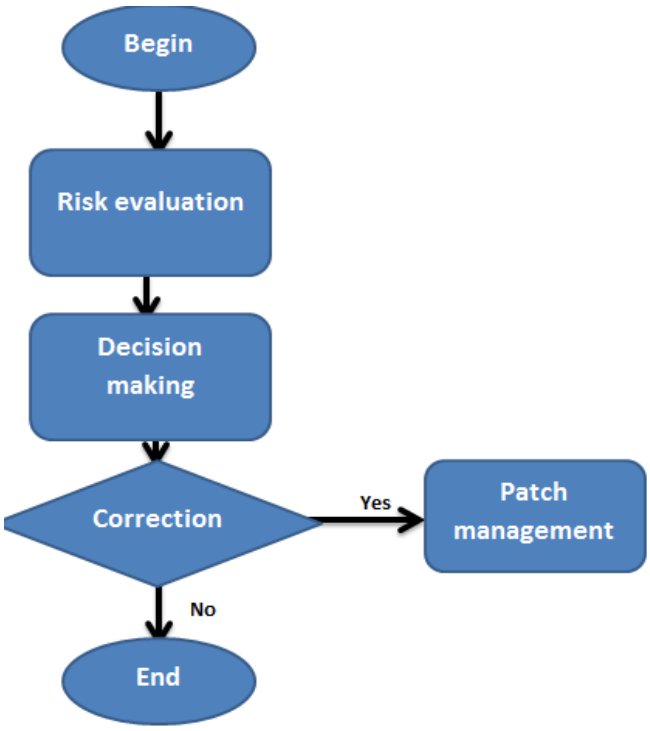

Figure 4: Corrector diagram

System interactions

The different modules are interconnected to enforce security configuration for a given service. Different

communications are carried out sequentially, each of them, labeled with a message stereotype.

The message <<Lists>>

This message allows the collector to retrieve the application report from the scanner. The message is structured as shown in Figure 5.

Services	Versions
Apache	2.4.7
CUPS	1.7
X	---
Y	---

Figure 5: Message's structure of scanner

The transmission is done sequentially and by blocks. For a service, it is sent the name (e.g. Apache), then the version (e.g. 2.4.7).

The message <<Collects>>

The collector in turn encapsulates information on the vulnerability and rules to the previous message. It associates with each service, all the collected vulnerabilities and the set of rules as provided by publishers. The message is transmitted to the verifier. It follows the structure shown in Figure 6.

Services	Versions	Vulnerabilities	Rules
Apache	2.4.7	Version, injection, XSS, deny of service	**Version:** -Serversignaturer(r_1) off -rule_x(r_2) on **Injection:** -rule_y (r_1) 201
CUPS	1.7	Version	----------
X	---	----------	----------
Y	---	----------	----------

Figure 6: Message's structure of the collector

The message <<Fixes>>

The controller marks all the rules of the message sent by the collector. Only rules labeled with 0 will be applied. Those marked with 1 mean that security rules are already effective on the system. Information passing from the verifier to the corrector is shown in Figure 7. They encapsulate the information regarding the state of applying rules.

Services	Versions	Vulnerabilities	Rules	states
Apache	2.4.7	Version, injection, XSS, deny of service	**Version:** -Serversignature(r_1) off -rule_x(r_2) on **Injection:** -rule_y (r_1) 201	**Version:** r_1 1 r_2 0 **Injection:** r_1 1
CUPS	1.7	----------	----------	----------
X	---	----------	----------	----------
Y	---	----------	----------	----------

Figure 7: Message's structure of the Controller

5. Implementation of modules

5.1. Environment of tests

We have used a computer with the following characteristics

- Constructor: COMPAQ;
- Operating system: Ubuntu version 14.01 LTS 64-bits
- Processor: Genuine Intel (R) processor 575 @ 2.00GHz;
- Memory: 2.00 GB;
- Hard disk : 50GB.

Apache version 2.4.7 is the Web service to be enforced which is the most popular according to Netcraft report [19].

5.2. Definitions

A **chatty service** is a service that provides to the client more information than is needed.

An **open service** is a service opened to any client requests.

An **injectable service** is a service that is vulnerable to injection vulnerabilities.

We consider a service as vulnerable if it is **chatty**, **open** or **injectable**.

5.3. Implementation

The implementation is summarized in three major steps as shown in Figure 8. The first phase consists to protect against version disclosure. The second one consists to configure the security module against injection attacks (Injection and XSS) and the third one defines parameters to limit deny of service attacks.

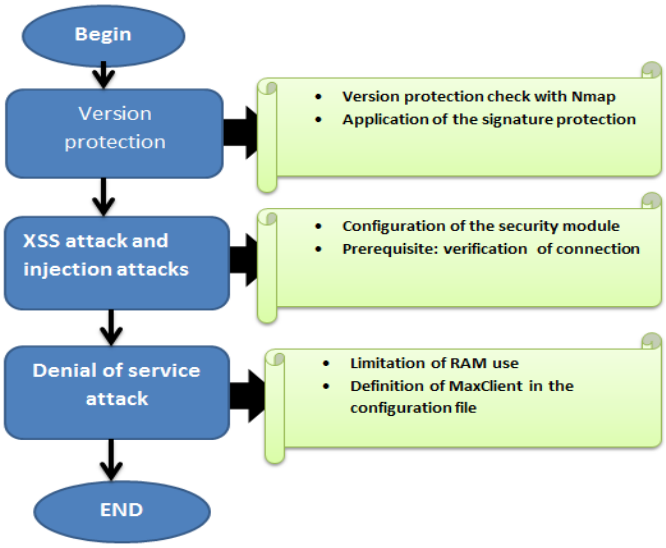

Figure 8: Implementation steps

The different artefacts developed with Bash and Zenity (for graphical aspects), are available in the Github page dedicated to this project [20].

5.4. Module interfaces

The home screen is illustrated in Figure 9.

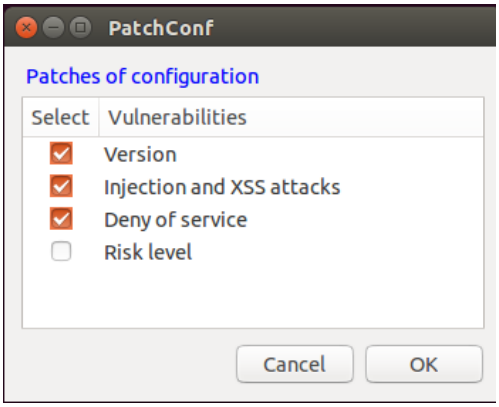

Figure 9: Home screen

Version protection

This option makes a service less chatty while responding to requests. Typically, attackers use scanner tools such as Nikto to detect service versions, and once the version is obtained, they consult editor or CVE sites to check its corresponding vulnerability information. Another scenario comes from the fact that Apache may not be available when it is solicited by the client. But instead of just returning an error message, the service displays other sensitive information such as the version. This situation can also be justified due to the fact that the

service is open thus responding to several requests.

Protection of XSS and injection

XSS and Injection attacks can occur when the appropriate protection module is not correctly configured on Apache or **injectable**. The proposed system relies on the module *Modsecurity* and checks that it is considered by other packages of the Web server. *Modsecurity* is the application firewall (WAF) that protects Web services such as Apache from application attacks. This module is responsible to filter requests and responses generated between the clients (visitors) and the Web server. It verifies that they comply with the established security policies. It is based on signatures of Web attacks, compliance with Web standards and protocols (HTTP, HTML, XML) as well as understanding of Web application.

Protection of deny of service

Deny of service aims to overload resources such as memory, processor and disk. Apache suffers greatly from this attack and sometimes becomes temporarily out of service because it is **open**. The service is present but cannot answer because the solicitation is huge. Behind that, it is possible to have an attacker but not always. It is due to the default configuration. The number of requests to receive (*MaxClients*), at a given moment, must be set according to the RAM size on the host system.

Risk calculation

Rules concerning version disclosure and deny of service vulnerabilities come from official Apache Web site [21] whereas those of injection vulnerabilities include steps for deploying *Modsecurity*.

a) Injection vulnerabilities

This vulnerability is planned to be banned with seven security rules

- Install the ModSecurity module (r_{11});
- Restart apache (r_{12});
- Rename the file modsecurity.conf-recommended (r_{13});
- Install *git-hub* (r_{14});
- Download the rules OWASP (r_{15});
- Include the rules of OWASP in Apache2 (r_{16});
- Enable the modsecurity module (r_{17}).

The values of known variables of risk levels are as follows:

$$\alpha_1 = 8.8, n_1 = 7 \; et \; \beta_{r_{1i}} = \begin{cases} 0 \; if \; the \; rule \; r_{1i} \; is \; applied \\ 1 \; otherwise \end{cases}$$

Calculation

$\beta_{r_1} = 1.257 \times 7 = 8.799$. According to CVSS, it is a high level of risk.

b) Deny of service

This vulnerability is planned to be banned five security rules

- Calculation of the *Maxclients* according to RAM (r_{21});
- Calculation of *Startservers* which is equal to $\frac{3}{10}$MaxClients (r_{22});
- Calculation of the *MinSpareserver* which is equal to $\frac{5}{10}$MaxClients (r_{23});
- Calculation of *MaxSpareserver* which is equal to $\frac{1}{10}$MaxClients (r_{24});
- Calculation of *ServerLimit* which is equal to *MaxClients* (r_{25}).

Calculation

Values: $\alpha_2 = 5.5, n_2 = 5$.

$\beta_{r_2} = 1.1 \times 5 = 5.5$. According to CVSS, the level of risk for this vulnerability is average.

c) Version disclosure attack

Its security rules are

- Activate the *ServerSignature* option. The server no longer displays any information on the error pages (r_{31});

Set the ServerTokens to “prod”. It allows to modify the contents of the field "Server" which is in the header of each answer sent by the server. It will

only contain the name of the Web server used (r_{32}).

Calculation

Values: $\alpha_3 = 8.4, n_3 = 2$.

$\beta_{r_3} = 4.2 \times 2 = 8.4$. According to CVSS, the level of risk for this vulnerability is high.

d) XSS attack

This vulnerability has the same security rules as the injection vulnerability. Thus, $n_4 = 7$ and since the test environment does not have any of these rules, its evaluation gives the same risk level result as that of injection. Thus $\beta_{r_4} = 8.799$.

6. Evaluation and discussions

This section evaluates the application of the system on a vulnerable Apache. The approach includes two steps: it presents first the security status before the application of the approach (pre-test) and secondly shows the improvement brought after the application of the approach (post-test).

6.1. Version Disclosure

The system works as follows concerning this vulnerability: The scanner module detects and presents a report mentioning Apache vulnerable to version attacks. Once this report is received by the collector module, it conducts a search of publishers and official sites for the collection of security rules. After checking the syntax of these rules, it sends to the controller two protection signatures in the structure of rules (*ServerSignature* and *ServerTokens*). At the same time, it indicates the *apache2.conf* file to apply these two security rules. The vector sent is the following: *('apache2.conf, write, ServerSignature on', 'apache2.conf, write, ServerTokens prod')*. The controller first checks for the presence of these rules on the *apache2.conf file*. In the actual case, the two rules are missing in this file at start. Thus, they are all numbered to 0 to be considered by the corrector as rules to be applied. This report is sent to corrector to assess the risk, required to apply configuration patches.

Scenario

The command "*Nmap -sV IP_address*" is launched on Zenmap [22] by specifying the IP address of the target. Zenmap is the graphical version of nmap. The command is performed locally on a machine where Apache is vulnerable. The output is illustrated in Figure 10

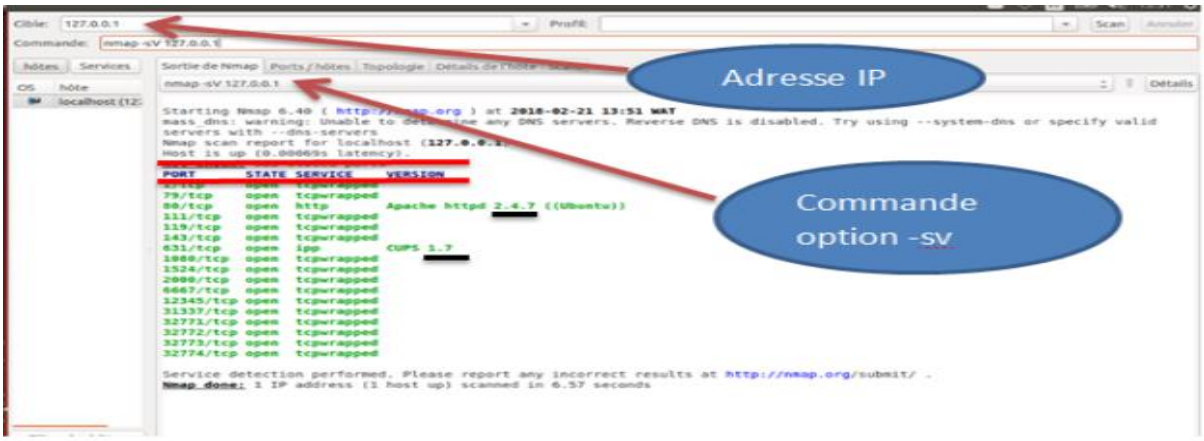

Figure 10: Vulnerable Apache

It is observed underlined in black, Apache and printing services with their versions. The box in red gives other information. These are: response port, service status, service name, and service version. The attacker can therefore consult the CVE Website, for the list of vulnerabilities for this version of Apache. Once the system is launched and the protection option chosen, rules are applied to hide the version of Apache as shown in Figure 11.

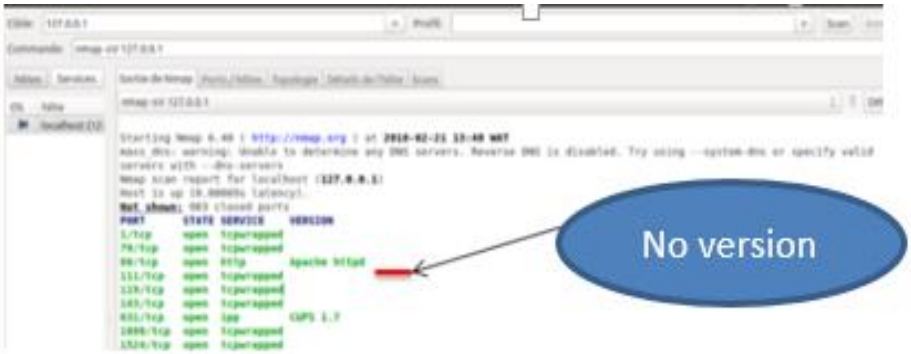

Figure 11: Secure Apache

It should be noted that the protection has been carried out and considered by the system. Thus, during a next scan, the system informs that it is protected. The improvement is verifiable with Nikto.

6.2. Injection and XSS attack

The collector detected these two types of vulnerabilities. Correction information was retrieved by the Collector from the Apache official Web site. The system automatically installs the security module called Modsecurity. The Collector retrieves security rules defined by OWASP on Github [23] concerning the deployment of Modsecurity. In this test case, the Controller notices the absence of Modsecurity. The test of Nikto on unsecure Apache is shown on Figure 12.

```
+ Server: Apache/2.4.10 (Debian)
+ Server leaks inodes via ETags, header found with file /, fields: 0x2b60 0x5
1d0512ae2d40
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the use
r agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user age
nt to render the content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Apache/2.4.10 appears to be outdated (current is at least Apache/2.4.12). A
pache 2.0.65 (final release) and 2.2.29 are also current.
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS
+ ///etc/hosts: The server install allows reading of any system file by addin
g an extra '/' to the URL.
+ OSVDB-561: /server-status: This reveals Apache information. Comment out app
ropriate line in the Apache conf file or restrict access to allowed sources.
+ OSVDB-3233: /icons/README: Apache default file found.
+ /server-status: Apache server-status interface found (pass protected)
+ 7517 requests: 0 error(s) and 10 item(s) reported on remote host
+ End Time:           2018-03-21 11:04:01 (GMT1) (24 seconds)
```

Figure 12: Nikto test on vulnerable Apache

It is observed that Nikto listed eleven vulnerabilities on the Apache Web service in 24 seconds. Among them, the XSS flaw, version disclosure and many more. The Corrector includes some rules recommended by OWASP. The well integration of rules is confirmed after performing the Bash script "*curl localhost/index.html?exec=/bin/bash*", by the message illustrated in Figure 13.

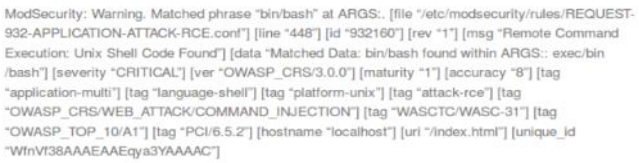

Figure 13: Attack's test

The system is applied to reduce vulnerabilities identified by Nikto.

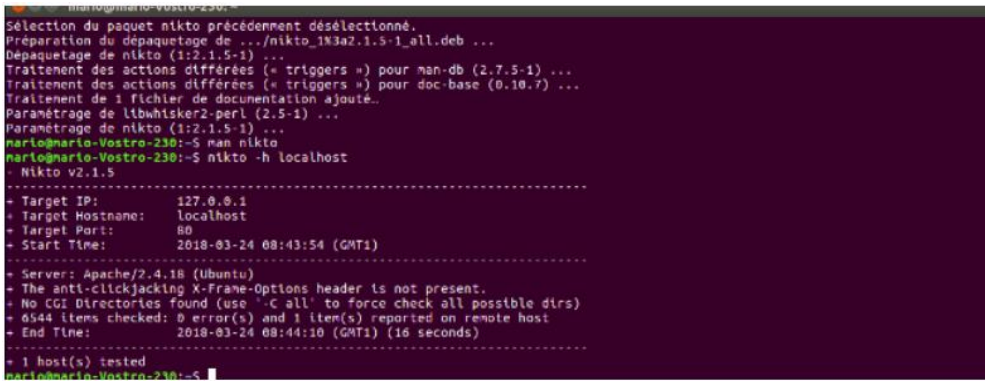

Figure 14: Post-test of Nikto

The vulnerability detection has been significantly reduced by 8 seconds. The number of vulnerabilities in this progression has been reduced to 2. It reveals the efficiency of the system.

6.3. Deny of service

Vulnerabilities detected by the Collector in this category of attacks are collected from official Apache editors. The correction solutions are transformed by the Collector into rules structured as follows:

R_1= (apache2, writes, MaxClient 1288), and so on. The controller marks all these rules to 0 because Apache is at the default configuration stage

It is defined the *MaxClients* based on the size of current RAM and the average size of Apache processes. It implies that the heavier the process, the less the *Maxclient* is. The part of code dealing with this part is also available to the Github page of this project.

Due to the limited number of processes during the test, it has not been easy to check improvements.

6.4. Comparison with SCAAMP

This section compares the proposed system to SCAAMP. Nikto reveals several vulnerabilities on the unsecure Apache as shown in Figure 15.

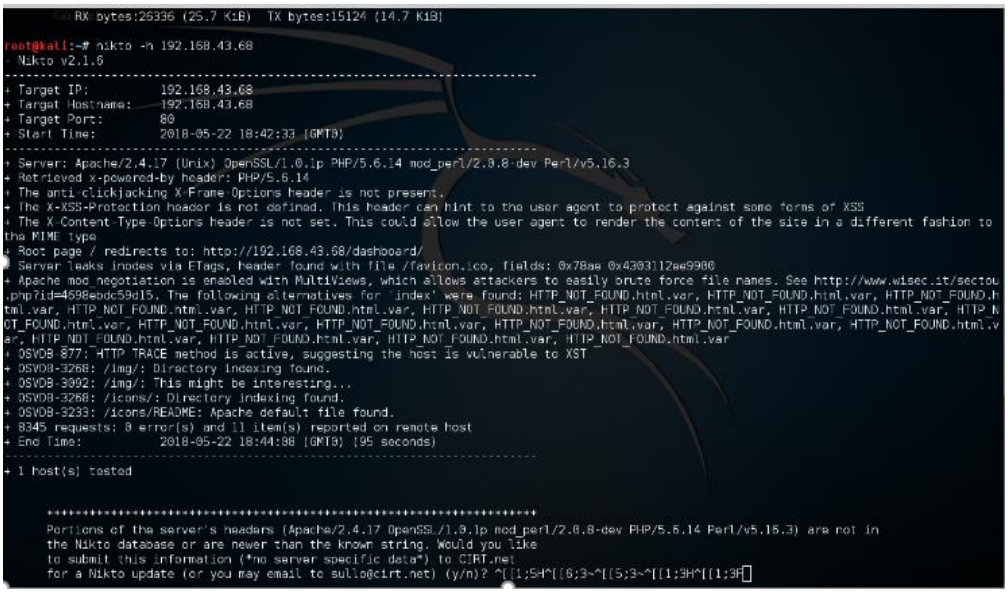

Figure 15: Test of SCAAMP

After the application of SCAAMP on the same unsecure Apache, the security level is estimated to 33% as shown in Figure 16. Therefore, the risk level is 77%. Within the same conditions, the proposed system reduces the longer list of vulnerabilities to only two as shown in Figure 14. SCAAMP does not inform about risk level on vulnerabilities, to which users are exposed. The determination of rules in this work is based on up-to-date retrieved recommendations from official editors; making the system flexible and scalable. SCAAMP works unlike with fixed rules.

Apache Security Configuration Safety Report

Total # of Directives (100%)	Safely Set	Unsafely Set	Not Used/No Value	Safety (out of 100%)
3	1 out of 3 directives	2 out of 3 directives	13 out of 16 directives	33.33

Figure 16: Summary of SCAAMP configuration

6.5. Limitations and recommendations

The proposed solution includes Nikto tools for static scanning and evaluates risk level and efficiently reduce vulnerabilities. However, this work builds the system only for Apache. It is recommended to make it generic to all services. The computation of risk level supposes that each rule has the same contribution to a vulnerability type. However, in real cases it cannot be the case. It is recommended to consider that rules have different impacts on the computation of misconfiguration risk level. The system should also integrate automatic retrieval and mining of rules and Nikto part to make the scanning fully complete.

7. Conclusion

This work proposes an architecture to identify vulnerabilities in core configuration files, to design directives according to the rules retrieved from official editor sites, to compute risk levels associated to the status of

service configuration, and to automatically apply directives on behalf of the user. Unlike SCAAMP, the status of service configuration is not the one at the installation but can evolve over the time. Experimentations on real environments demonstrate that the system reduce considerably vulnerabilities.

Future works will be oriented to

- consider of several services at once with possibilities of local or remote intercommunication (via a network);

- automate information retrieval and mining by the collector;
- assign different weights to the rules for the assessment of risk levels;
- automate update of CVSS values for risk level assessment in the corrector.

References

[1] R. Newmana, V. Chang, J. W. Walters, G. B. Wills. "Web 2.0—The past and the future". International Journal of Information Management, 36, pp. 591–598, 2016.

[2] G. Harry. Principales failles de sécurité des applications Web : principes, parades et bonnes pratiques de développement. CNRS, 2012

[3] G. V. Marconato. "Evaluation quantitative de la sécurité informatique : approche par les vulnérabilités". Ph.D thesis, INSA, 2009

[4] B. Eshete, A. Villafiorita, K. Weldemariam. "Early Detection of Security Misconfiguration Vulnerabilities in Web Applications," in Proc. Int. on Availability, Reliability and Security, 2011, pp. 169-174.

[5] 'Nikto2', https://cirt.net/Nikto2, accessed 05 January 2018

[6] M. van Steen, A. S. Tanenbaum: Distributed Systems: Principles and Paradigms. CreateSpace Independent Publishing Platform, 3rd edn, 2017

[7] A. Avizienis, J-C. Laprie, B. Randell, C. Landwehr. "Basic Concepts and Taxonomy of Dependable and Secure Computing". IEEE Transactions on Dependable and Secure Computing, 1(1), pp. 11-34, 2004.

[8] A. Adelsbach, D. Alessandri, C. Cachin. "Conceptual Model and Architecture of MAFTIA". University of Newcastle upon Tyne, pp 30-31, 2003.

[9] Vaadata. "Comprendre les vulnérabilités web en 5 min – episode #1 : Injections!". Internet: https://www.vaadata.com/blog/fr/comprendre-les-vulnerabilites-Web, March. 21, 2014 [Jul. 29, 2019].

[10] OWASP. "Top 10-2017 Top 10". Internet: https://www.owasp.org/index.php/Top_10-2017_Top_10, March, 27, 2018 [Jul. 29, 2019].

[11] A. Viardin. “Un petit guide pour la sécurité”. Internet : https://www.inetdoc.net/guides/tutoriel-secu/, 2003, [Jul. 29, 2019].

[12] OpenVAS. “The world's most advanced Open Source vulnerability scanner and manager”. Internet: http://www.openvas.org/, [Jul. 29, 2019].

[13] Nmap. “Chapter 2. Obtaining, Compiling, Installing, and Removing Nmap”. Internet: https://nmap.org/book/install.html, [Jul. 29, 2019].

[14] P. Chambet. “La gestion des correctifs de sécurité dans un parc Windows : des solutions techniques à la mise en œuvre pratique en entreprise”. Internet : http://www.chambet.com/publications/Correctifs_securite.pdf , [Jul. 29, 2019].

[15] A. Taylor. “Guide détaillé pour Microsoft Windows Server Update Services 3.0 SP2”. Internet : http://www.labreux.fr/tssi/ms/10%20-%20WSUS30SP2StepbyStep.pdf , [Jul. 29, 2019].

[16] S. Murugiah, S. Karen. Guide to Enterprise Patch Management Technologies. National Institute of Standards and Technology (NIST), 2013

[17] A. Zammouri, A. A. Moussa. “SafeBrowse: a New Tool for Strengthening and Monitoring the Security Configuration of Web Browsers’. in Proc. Int. Conf. Information Technology for Organizations Development (IT4OD), Fez, Morocco, May 2016, pp. 1-5.

[18] OWASP. “Category:Vulnerability Scanning Tools”. Internet:

https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools, Apr. 26, 2018 [Jul. 29, 2019].

[19] Netcraft. “February 2018 Web Server Survey”, Internet: https://news.netcraft.com/archives/2018/02/13/february-2018-web-server-survey.html, Feb. 13, 2018 [Jul. 29, 2019].

[20] Tchafros. “Misconfiguration Patching of Web Services”. Internet: https://github.com/tchafros/PatchConf, Jun. 12, 2018 [Jul. 29, 2019].

[21] Apache. “Apache HTTP Server Project”.Internet: https://httpd.apache.org/security/vulnerabilities_24.html, [Jul. 29, 2019].

[22] Nmap. “Chapter 12. Zenmap GUI Users’ Guide”. Internet : https://nmap.org/book/zenmap.html, [Jul. 29, 2019].

[23] SpiderLabs. “Modsecurity rules”. Internet: https://github.com/SpiderLabs/owasp-modsecurity-crs/tree/v3.0/master/rules, Dec. 25, 2017 [Jul. 29, 2019].