Design & Develop Misconfiguration Vulnerabilities Scanner for Web Applications

Aidmar Wainakh
Syrian Virtual University
Damascus, Syria
aidmer_33602@svuonline.org

Dr. Ahmad Wabbi
Syrian Virtual University
Damascus, Syria
t_awabbi@svuonline.org

Dr. Bassel Alkhatib
Faculty of Information Technology Engineering- Damascus University
Damascus, Syria
t_balkhatib@svuonline.org

Abstract – *Misconfiguration is one of the most critical Web vulnerabilities, still it does not receive enough attention. Applying general security practices and general remediation proved inefficiency in dealing with this type of vulnerabilities. In this research, we discuss and highlight several issues in order to enhance misconfiguration detection, quantifying and fixing. Our approach detects misconfiguration based on extended set of security-related configurations, then quantify the vulnerabilities according to the environment characteristics, using the most recent scoring standard in this field and recommend customized secure remediation. We implemented our approach in a tool called MVS, and we were able to evaluate seven Apache-MySQL-PHP packages, ten open source Web applications and seven online websites. Our experiments revealed that the tool is able to detect misconfigurations at both the environment level and the application level, then recommend customized and secure remediation.*

Keywords: *Web applications, Web security, vulnerability, misconfiguration, CCSS, customized remediation.*

I. Introduction

Nowadays there is a significant increasing dependency on Web applications, more organizations rely on Web applications as a primary technology. Web applications can be personal websites, social networks, e-commerce applications, etc. The existence of Web applications in our life is so important that it makes them an attractive target for malicious users.

The security obsession of Web application administrators should be proportional to the magnitude of the assets they protect. The Web application market is growing so fast - According to Google, over 50 billion pages are on the Web in 2014 [1]- and that makes the Web security ever-moving target. 86% of all websites has at least one serious vulnerability according to WhiteHat [2].

There are many variant vulnerabilities types differ in exploitability, prevalence, detectability and impact. They were classified into different categories such as Information leakage and Server Misconfiguration. According to Context [3] the Server Misconfiguration is the most common vulnerability in Europe websites, as it's listed in the OWASP Top 10 constantly. Misconfiguration attacks exploit configuration weaknesses found in Web environment, as many servers come by default with unnecessary enabled features and services, such as remote administration functionality and content management. These flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise.

In spite of Misconfiguration's criticality little attention has been paid for it. Few researches and papers concern about it. The most recent researches in this topic tend to find exhaustive techniques to detect misconfiguration vulnerabilities at both the environment level and the application level. The most of approaches use general security recommendations to detect vulnerabilities, each researcher collects a set of recommendations using his own method, lacking of consensual secure configurations standard. Moreover, many of researches depend on experts' estimations to quantify the severity of vulnerabilities, instead of using modern scoring systems. In nutshell the contributions of this research are the following:

1. We presented automated scanner of Web applications configuration at the environment level and the application level, detect, quantify and fix misconfiguration vulnerabilities.

2. We put in use all the metrics of Common Configuration Scoring System (CCSS) in order to quantify the vulnerability severity accurately.

3. We improved the Gold Standard, which is a set of sensitive (security-related) configuration directives with security recommendations, for Apache-MySQL-PHP (AMP) environment.

4. We discussed the effect of strict security recommendations on the application performance.

5. We suggested a customized Gold Standard includes new approach to compute recommended values, taking into consideration CCSS score and performance issue.

6. We discussed the tuning configurations across AMP components.

7. We implemented our approach on seven AMP packages, ten open source Web applications and seven online websites.

II. Misconfigurations In Web Applications

Although the number of vulnerabilities is decreasing in general but the misconfiguration vulnerabilities are increasing in relative to others. For example, as noted by Context [3] the misconfiguration vulnerabilities number within a website increased from 2.6 in 2010 to 2.9 in 2012. According to the National Vulnerability Database (NVD) there are 24 vulnerabilities related to misconfiguration in 2010, increased to 31 vulnerabilities in 2012. The Open Source Vulnerability Database (OSVDB) indicates that PHP configuration vulnerabilities increased from 26 in January 2007 to 42 in January 2011, an increase of 32%.

Misconfiguration vulnerability is considered one of the high level vulnerabilities since it could lead to several risks such as gain information, denial of service, code execute and overflow. Misconfiguration attacks exploit configuration weaknesses found in Web environment. Many servers come by default with unnecessary features, sample files and modules. They may also enable unnecessary services, such as remote administration functionality and content management. Debugging functions may be enabled or administrative functions may be accessible to anonymous users. These features provide a great opportunity for a malicious user to bypass authentication methods and gain access to sensitive data. In the following, we present some examples of misconfiguration vulnerabilities in AMP environment.

Examples 1: In PHP the directive "*display_errors=On"* prints errors as part of the output. Such directive is considered a vulnerability that leads to gain information risk. A standard attack tactic would involve profiling a system by feeding it improper data, and checking for information in the returned errors.

Example 2: In Apache, the directive "*Options Indexes"* enables directory browsing. That means, if there is no index.html under a website directory, client will see all files and sub-directories listed in the browser which is considered a gain information risk.

Example 3: In Apache, the directive "*LimitRequestBody=0"* makes no restriction on the size of the HTTP request body sent from the client, which could lead to overflow risk, if a malicious user sends large requests.

III. Related Work

In this section we will present some papers and researches relate to our topic. We will mention their contributions and limitations in order to identify the gaps in the existing literature. "Automated Diagnosis of Software Configuration Errors" paper by S. Zhang and M.D. Ernst [4], presents a technique to identify the main cause of the configuration errors based on the behavior of the software system. In order to link the undesired behavior to specific configuration options, the technique uses static analysis, dynamic profiling, and statistical analysis. It differs from old approaches in two aspects: it is fully automated; and it can diagnose both crashing and non-crashing errors. The authors implemented their technique for Java software. Unlike ours, this approach focuses on identifying and monitoring configuration option-affected control flow rather than the values. Moreover, the paper was concerned about the diagnosis and did not discuss how to fix the localized errors.

In the paper "Quantitative Evaluation of Related Web-based Vulnerabilities" by D. Subramanian et al. [5], the authors propose a quantitative framework that combines degree of confidence reports pre-computed from various scanners. The output is evaluated and mapped based on derived metrics to appropriate remediation for the detected vulnerabilities. The authors show the relational mapping among a set of vulnerabilities. However, several remediation exist for a given vulnerability, it is necessary then to find the best possible vulnerability-remediation match (selective remediation). They suggest a mathematical model to select the suitable remediation for certain vulnerability based on many parameters related to the target system itself and that allows to select customized remediation. Although this approach is concerned about vulnerabilities in general, we share the same goal of providing a selective remediation that is appropriate to a particular system.

In the paper "Detection of configuration vulnerabilities in distributed (Web) environments" by M.M. Casalino et al. [6], the authors present a language-based approach to specify and execute declarative and unambiguous security checks for detecting vulnerabilities caused by system misconfigurations. The proposed language is based on the Security Content Automation Protocol (SCAP) specification and extends the Open Vulnerability Assessment Language (OVAL) configuration validation standard. The language allows the definition of configuration checks, the target software components as

well as the actual configurations by specifically separating the checking logic from the configuration retrieval. Unlike our research, this paper focuses on detecting system-level configuration vulnerabilities. The approach does not clarify the link between the vulnerability and the potential attacks it can lead to.

In "Early Detection of Security misconfiguration vulnerabilities in Web applications" paper by B. Eshete et al. [7], the authors present an automated tool to detect misconfiguration vulnerabilities in Web server environments. They extend the checked directives list taking into consideration the significant increase in misconfiguration. The authors implement their tool on eleven widely used AMP server environments across three popular operating systems. They came to two main conclusions. First, the default security configuration of these environments are way too far from the recommended security configuration settings. Second, the difference in configuration average safety is not that significant among the operating systems. This approach does not discuss the configuration vulnerabilities at application level. Moreover, the tool supposes that all security configuration directives have equal weight and does not use any quantitative standards.

Finally, the paper "Confeagle: Automated Analysis of Configuration Vulnerabilities in Web Applications" by B. Eshete et al. [4] is one of the main papers that inspired our research. The authors in this paper present an approach that combines hierarchical configuration scanning and preliminary source code analysis of Web applications to detect, quantify and fix the potential misconfiguration vulnerabilities. The values of each configuration directive were analyzed against a "Gold Standard" of configuration recommendations. Unlike generic Web vulnerability scanners, this approach proved high efficiency in detecting potential configuration vulnerabilities at the environment level in addition to the application level as well. In order to quantify the degree of vulnerabilities severity, they used CCSS base metrics. They implemented their approach in a tool called *Confeagle,* on AMP environment. They evaluated their approach on open source PHP applications, and compared its effectiveness with popular Web vulnerability scanners. However, they used only base metrics of CCSS and ignored the temporal and the environmental metrics, which leaves their quantitative results inaccurate. In addition, they used limited resources to combine the "Gold Standard". Our work widens the scope of this work and complements it by addressing the above limitations.

IV. Methodology

The main general goal of this research is to help administrators to avoid misconfiguration vulnerabilities as possible. In this section we will present the main methods we used through this research and our suggested approaches.

IV.1. Accurate Scoring Metrics

"A Problem Well-stated is Half-solved", Charles F. Kettering. In order to secure Web application we need to have good description of its security status and accurate measurement for severity of the existing vulnerabilities. That is the main aim of Common Configuration Scoring System (CCSS), which provides us with a set of metrics that measure the severity of software security configuration issues.

The CCSS metrics are organized into three groups: base, temporal, and environmental. Base metrics describe the characteristics of a configuration issue that are constant over time and across user environments. Temporal metrics describe the characteristics of configuration issues that can change over time but remain constant across user environments. Environmental metrics are used to customize the base and temporal scores based on the characteristics of a specific user environment. In our research we used all these groups of metrics in order to quantify vulnerability severity accurately.

IV.2. Improved Gold Standard

Gold Standard (GS) is a set of sensitive directives (security-related) in AMP environment. The concept of GS was suggested by [4]. We collected the GS using a collection of references which contains official documentations [1], [10], [11], [12], expert opinions [13], [14], [15] and configuration best practices [16], [17], [9], [19], as we used online CVSS calculators [20], [21] to generate CCSS base vectors. We extended the information about each directive in GS to include the following fields:
<directive_name, description, potential_risk, default_value, recommended_value, platform, component, component_version, base_vector, temp_vector>

The GS is the backbone of this research. In fact, collecting and analyzing the information about directives required a lot of efforts to do, because we had to deal with three different components with variant versions. Besides, there was not any trusted reference gathers all the sensitive directives in one list. So our GS comes as an attempt to build an initial core of secure configuration standard for AMP environment. Next, we present some examples of the directives in GS.

- Directive: *post_max_size*
Description: Sets max size of post data allowed.
Potential Risk: Overflow, malicious users may attempt to send oversized POST requests to eat the system resources.
Default Value: 8M
Recommended Value: 8M
Component: PHP
Version: Available since PHP 4.0.3.
Base Vector: AV:N/AC:M/Au:N/C:N/I:N/A:C

Temporal Vector: GEL:H/GRL:H

- Directive*: ServerTokens*

Description: This directive controls whether server response header field which is sent back to clients includes a description of the generic OS-type of the server as well as information about compiled-in modules and their versions.

Potential Risk: Gain Information, one crucial bit of information to hide is the version number. Hiding it keeps unwanted users from knowing how to quickly hack the Web server.

Default Value: Full.

Recommended Value: Prod.

Component: Apache.

Version: ALL.

Base Vector: AV:N/AC:L/Au:N/C:P/I:N/A:N.

Temporal Vector: GEL:H/GRL:ND.

- Directive*: skip-grant-tables*

Description: This option causes the server to start without using the privilege system at all, which gives anyone the access to all server databases.

Potential Risk: Bypass, enabling such option will permit to all users to access all databases on the server bypassing all privileges.

Default Value: FASLE.

Recommended Value: FASLE.

Component: MySQL.

Version: ALL.

Base Vector: AV:N/AC:L/Au:S/C:C/I:C/A:C.

Temporal Vector: GEL:L/GRL:L.

IV.3. Security vs. Performance

Many administrators prefer high performance over security because the majority of administrators do not have the required knowledge in security field. That makes them use the default configurations in the environment since the default configurations usually provide usability for users. Our concern was about providing the administrator with a solution that guarantees them secured applications without much loss in performance.

In the following we present examples about how the secure values of some directives could affect the performance.

- Directive: *memory_limit* (PHP)

This sets the maximum amount of memory -in bytes- that a script is allowed to allocate. The default value is 128M while security experts recommend the value 32M since high values could leave the application open to Overflow attack. On the other hand, small values will limit some scripts and make them use more time managing the small allocated memory. The experts recommend the value 256M for high performance [20].

- Directive: *max_input_time* (PHP)

This sets the maximum time -in seconds- a script is allowed to parse input data, like POST and GET. The default value is -1 (means no limit) while secure value is 30. Like previous directive the potential risks for high values is Overflow. The small value may cause to terminate some scripts during parsing data before it's done. The experts recommend the value 90 for high performance [20].

- Directive: *KeepAliveTimeout* (Apache)

This sets the number of seconds Apache will wait for a subsequent request before closing the connection. The default value is 5, and the security experts recommend the value 3. *KeepAlive* provides long-lived HTTP sessions which allow multiple requests to be sent over the same TCP connection. In some cases this has been shown to result in an almost 50% speedup in latency times for HTML documents with many images [20].

Administrators need to keep high performance with reasonable security, they do not have to trade performance for a slight attacks mitigation. Therefore we need to recommend values ranging between secure values and high performance values, taking into consideration the prevalence and the severity of the potential risks. In the next section we will suggest a formula that determines new recommended values taking these considerations into account.

IV.4. Customization

"Software vendors, issue an increasing number of security advisories, while users, on the other hand, struggle to understand if a given vulnerability is exploitable under their particular conditions" M.M. Casalino et al. [6]. This quote refers to a very important point, which is the difficulty of adjusting general security recommendations to meet certain system requirements.

Customization means to find the best security practices for particular system. In our case the customized secure configurations can be achieved by creating a customized Gold Standard that contains customized recommended values for certain website. In this research we focused on the directives which have numeric values only.

As we mentioned before the environmental metrics in CCSS are used to customize the base and the temporal scores based on the characteristics of a specific environment. Therefore we chose the environmental score to be one of the main parameters we use in computing the new customized values.

Unfortunately, many administrators are not able to generate the environmental metrics, since that requires good knowledge in security, so we need to create an adaptive layer between the administrator knowledge and the environmental metrics. We suggested a set of metrics called Admin metrics, which contains the main characteristics of the website to play the role of that layer. We tried to make these metrics as simple as possible. The Figure 1 demonstrates the main components of customization process.

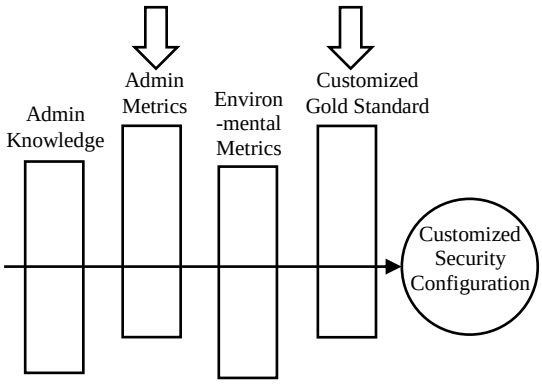

Figure 1: Main Components of Customization Process

Administrator metrics consist of the following:

1. Category (CAT): the type of the website such as personal, informational, e-commerce, etc.
   CAT values: according to TABLE 1.
2. Business Size (BS): the size of the business (revenue) that website presents.
   BS values: [*Low=0, Medium=0.5, High=1*].
3. Sensitive Data (SD): indicates if the website contains sensitive data or not, like credit cards or personal information for users, etc.
   SD values: [*Low=0, Medium=0.5, High=1*].
4. Web Oriented Business (WO): indicates if the business depends on Web market.
   WO values: [*Low=0, Medium=0.5, High=1*].
5. Profit (P): indicates if the website was made for profit purpose or not.
   P values: [*False=0, True=1*].

IV.4.1. Generate the Environmental Metrics

To generate the environmental metrics based on the admin metrics we need to find relations between these two sets of metrics. The environmental metrics basically contain the following:

- Local Vulnerability Prevalence (LVP): measures the prevalence of a vulnerability in a specific environment. So we will estimate this metric based on the number of repetitions for each vulnerable directive.
- Local Remediation Level (LRL): measures the level of protection against a vulnerability within the local environment. To simplify our problem we used same General Remediation in base metrics.
- Perceived Target Value (PTV): the motivation of an attacker to perform an attack relative to other environment. This metric relates to the category of the website, for example attackers are more interested in bank website than informational website. websites contain sensitive data are more attractive for attackers as well.
- Collateral Damage Potential (CDP): The economic loss of productivity or revenue through damage or theft of property or equipment. This metric relates to category, business size, Web oriented business and profit. For example e-commerce website for big company would lose much more than non-profit personal website.
- Confidentiality Requirements (CR): relates to the existence of sensitive data.
- Integrity Requirements (IR): relates to the existence of sensitive data and Web oriented.
- Availability Requirements (AR): relates to Web oriented and business size.

Based on the previous discussion and the sense experience, we suggest the following formulas to generate environmental metrics:

PTV=round_to_closest_metric_value[(CategoryPTV/2)+Sensitive Data+Access Complexity (AC)] **(1)**

Where *CategoryPTV* values: [*Low=0, Medium=0.5, High=1*], *Access Complexity* is base metric, *round_to_closer_metric_value* function rounds the result of calculation to closer value of *PTV* metric, knowing that *PTV* values are: [*Low=0.8, Medium=1.0, High=1.2, Not Defined=1.0*].

CDP=round_to_closest_metric_value[CategoryCDP+Web oriented*(Business size+ Profit)] **(2)**

Where *CDP*: [*None=1.0, Low=1.25, Low-Medium=1.5, Medium-High=1.75, High=2.0, Not Defined=1.0*]

CR=round_to_closest_metric_value(CategoryCR+2*Sensitive Data) **(3)**

IR=round_to_closest_metric_value(CategoryIR+Sensitive Data+Web oriented) **(4)**

AR=round_to_closest_metric_value[CategoryAR+Web oriented*(1+Business size)] **(5)**

Where *CR*, *IR* and *AR*: [*Low=0.5, Medium=1.0, High=1.51, Not Defined=1.0*]

The value of category metric is changed according to the environmental metric (CategoryPTV, CategortCDP…). The estimated values of category are shown in TABLE 1, where [*L:Low, M:Medium, H:High*].

TABLE 1: CATEGORY-ENVMETRICS TABLE

Category	PTV	CDP	CR	IR	AR
Personal	L	L	L	M	M
Sharing	L	L	L	L	L
Writers	L	L	L	M	L
Community	M	M	H	M	H
Blogs	L	L	L	M	L
Informational	L	M	L	H	M
Business Catalog	L	M	L	M	H
Directory	L	L	L	H	M
E-commerce	H	H	H	M	H

Example: let's calculate *AR* for Basmaty.com, which is a website belongs to the Business Catalog category, so

according to TABLE 1, category metric takes the following values:
CategoryPTV = Low, CategoryCDP = Medium,
CategoryCR = Low, CategoryIR = Medium, CategoryAR = High.
The administrator of Basmaty.com gave us the following values for admin metrics:
Business Size = *High*, Sensitive Data = *Low*, Web Oriented = *High*, Profit = *True*.
The Calculation as follows:
AR=round_to_closest_metric_value[CategoryAR+Web oriented(1+Business size)]*
*AR=round_to_closest_metric_value[High+High *(1+High)]*
AR=round_to_closest_metric_value[1 +1(1+1)]= round_to_closest_metric_value[3]*
Knowing that AR values are [*Low=0.5, Medium=1.0, High=1.51, Not Defined=1.0*]
⇒AR=1.51⇒AR= High, that means Basmaty.com requires high availability.

IV.4.2. Customized Recommended Value

As we mentioned in Security vs. Performance section we need to find a value ranging between secure and high performance value. To simplify the problem we will assumed that high performance value is the current value. If the environmental score has high value, that means the vulnerability is critical for that particular website, therefore the customized recommended value will be closer to the secure value. In the contrary, if the environmental score has low value, the customized recommended value will be closer to the current value in order to achieve high performance. The Figure 2 demonstrates the calculation process.

Based on previous discussion we suggest the following formula to compute the customized recommended value:

$$RecValue = d - (d - s) * (EnvScore/10) \quad (6)$$

Where *s*: secure value, *d*: default value, *EnvScore*: environmental score.

Example: The directive *memory_limit* in PHP has the current value 128M while the secure value is 32M, knowing that the environmental score is 7.0 for a website *x*, the recommended value will be:
*RecValue=128-(128-32)*7/10*
RecValue=60.8M

IV.5. Tuning

Apache, PHP and MySQL work together in perfect harmony to run Web applications. That harmony is based on data exchange between these components and the interferences between their configurations. There are relations between these components configurations directives, as well as between different directives in one component. Tuning configuration means to give directive a suitable value respecting the relations gathering this directive with other directives.

The tuning issue caught our attention because we work on checking directives and suggesting new secure values, so choosing harmonic values should be taken into consideration. In general declaring these relations will help administrators to avoid misconfiguration. When the administrator configure certain option in AMP environment, he needs to know all the directives across all components that could affect this option. In the following section we present examples of the relations between AMP directives:

- *expose_php* (PHP) & *ServerTokens* (Apache)
expose_php: Exposes to the world that PHP is installed on the server, which includes the PHP version within the HTTP header "X-Powered-By".
ServerTokens: controls whether "*Server"* HTTP header includes a description of the generic OS-type of the server, as well as the version of the interpreter. In case the administrator wants to hide the PHP version, he needs to disable both directives *expose_php* and *ServerTokens.*

- *upload_max_filesize* (PHP)<*post_max_size* (PHP)
upload_maz_filesize: sets the maximum size of an uploaded file. *post_max_size*: sets the maximum size of post data allowed. This setting also affects file upload. To upload large files, this value must be greater than *upload_max_filesize.*

- *post_max_size*(PHP)≤*LimitRequestBody* (Apache)
LimitRequestBody: restricts the total size of the HTTP request body sent from the client. Consequently, it could limit the size of post data *post_max_size* and the uploaded file *upload_max_filesize* as well, so it should be greater or equal *post_max_size.*

IV.6. Implementation

We implemented our tool Misconfiguration Vulnerabilities Scanner (MVS) basically using the system design suggested by [4] in addition to our new component the "Customized values generator". As shown in Figure 8 the tool workflow goes through three main phases: Parsing, Analysis and Fixing.

IV.6.1. Parsing

In Parsing phase we parse files and detect sensitive directives in both the environment and the application level. This phase contains three parsers: Environment, Runtime and PERDIR. All these parsers scan the respective locations to detect directives information (*name, value, section*), where *name* is the name of configuration directive, *value* is its value and *section* is the context that may indicate whether a directive is directory-level configuration, or script-level configuration.The environment parser searches for sensitive directives in php.ini, httpd.conf and my.ini files for PHP, Apache and

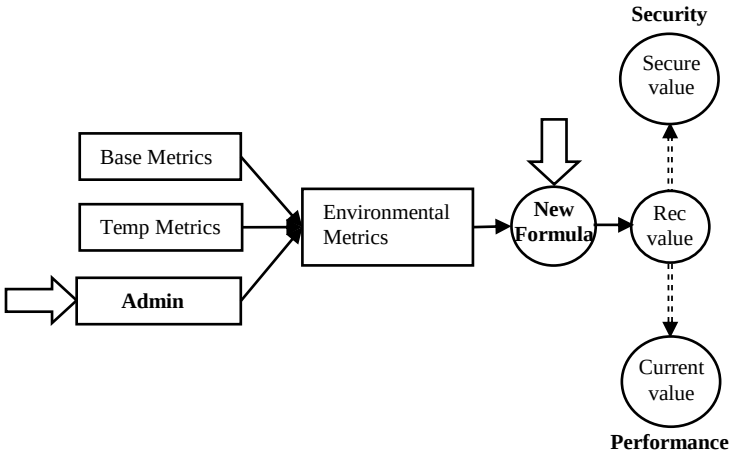

Figure 2: Customized Recommended Value

MySQL respectively. The runtime parser searches for sensitive directives in script files, knowing that PHP configurations could be changed by the following commands: `ini_set`, `ini_alter`, `ini_restore`, `session_save_path`, `error_reporting` and `set_time_limit`. While the PERDIR parser searches for sensitive directives in .htaccess files, the detected directives by this parser could be Apache or PHP directives as well, since there are Apache directives that could change PHP configuration from within .htaccess files: `php_value`, `php_flag`, `php_admin_value` and `php_admin_flag`.

IV.6.2. Analysis

In analysis phase we analyze the values of the detected directives against the Gold Standard (GS) recommended values and quantify vulnerable ones based on CCSS. As shown in Figure 3 this phase contains three components: Analysis Engine, Score Generator and Customized Values Generator. The analysis engine classify the directives values as *safety set* or *unsafety set*. While the score generator computes the CCSS scores based on the base, temporal vectors. The customized values generator computes the customized value of certain directive according to equation (6). As it generates the environmental vector based on admin metrics.

IV.6.3. Fixing

This phase contains one component: Fixing Engine which changes the directives values according to GS recommendations and generate new configuration files.

V. Experiments

We implemented our tool MVS on seven AMP packages, ten open source Web applications and seven online websites.

V.1. AMP Packages

First we installed the latest versions of the most common AMP packages. XAMPP 3.1.0, WAMP 2.4, AMPPS 2.4, UniServerZ 11.3.2, uwAMP 3.0.2 and EasyPHP 14.1. In addition, we installed manually AMP server environment that includes Apache 2.4.7, MySQL 5.6.20 and PHP 5.5.15. We implemented MVS on these packages at the environment level and we got the results as shown in TABLE 2. We noticed that the vulnerabilities number are greater than detected directives and that means the most of vulnerabilities are caused by default values, because in case MVS could not find a directive in the configurations files, it considers the default value.

TABLE 2: VULNERABILITIES NUMBERS IN AMP PACKAGES

Package	PHP		Apache		MySQL	
	DD	V	DD	V	DD	V
Manual	13	26	4	16	0	8
XAMPP	19	28	4	18	1	8
WAMP	20	25	4	16	1	8
AMPPS	19	28	12	33	0	8
UniServerZ	15	24	15	38	1	8
uwAMP	14	27	10	25	0	8
EasyPHP	0	26	5	18	2	7

Where *DD*: detected directives, *V*: vulnerabilities.

The Table 3 presents some samples of vulnerabilities in XAMPP package with details. As we notice the *expose_php* is enabled by default and that would lead to gain information risk, 55% of websites have at least one gain information vulnerability according to WhiteHat [2].

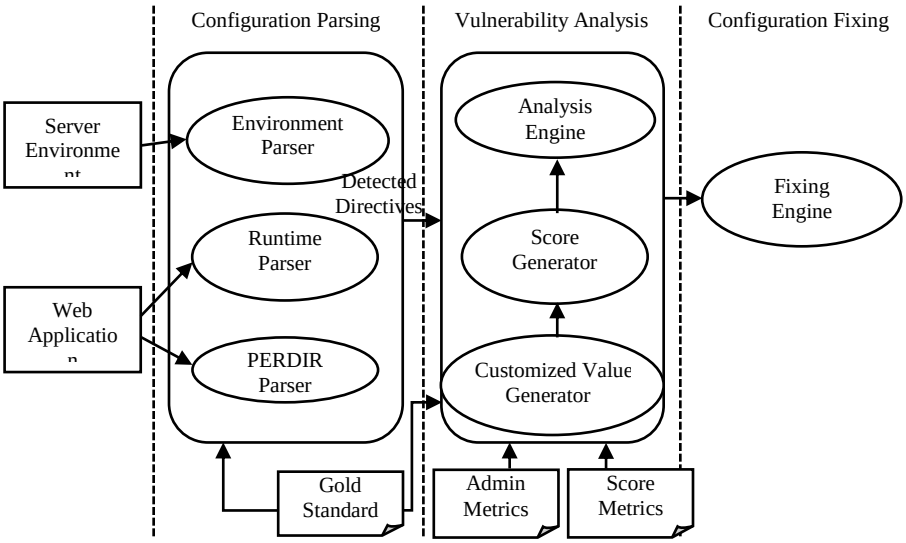

Figure 3: Workflow of the proposed tool

TABLE 3: SAMPLES OF VULNERABILITIES IN XAMPP

Directive	Val	Rec	S	BS	TS
expose_php	On	Off	F	5	5
memory_limit	128M	32M	F	7	5
ServerSignature	Off	Off	T	-	-
TraceEnable	On	Off	F	6	6
old_passwords	0	2	F	6	6
skip-federate	T	T	T	-	-

Where *Val*: value, *Rec*: recommended value, *S*:safely set, *BS*: base score, *TS*: temporal score.

The highest base score is 7.0 for the vulnerability of directive *memory_limit.* This vulnerability leads to DoS and overflow attacks. Such attacks cause a complete loss in availability, which is a main parameter in the base score equation. During the exploration of the packages vulnerabilities we noticed some values relate to tuning issue. For example: In AMPPS package, PHP component, MVS found *upload_max_filesize=32M* and *post_max_size=8M*, that does not meet the tuning rule: *upload_max_filesize<post_max_size.*

V.2. Open Source Web Applications

We installed the latest versions of the most popular open source Web applications. We implemented MVS on these applications at the application level (Runtime and PERDIR). The TABLE 4 shows the results.

In most cases the number of the detected commands are greater than the number of vulnerabilities and that's because of two reasons. First, there are directives classified as safely set. Second, there are directives that take values from PHP variables in the script, which makes it difficult to judge these directives, therefore we ignore them. We noticed that the most frequent vulnerable directives are: *error_reporting*, *display_errors* and *max_execution_time* which result in gain information and DoS attacks on the application.

TABLE 4: VULNERABILITIES NUMBERS IN OPEN SOURCE WEB APPLICATIONS

App	PHP		Apache	
	DC	V	DD	V
Joomla	31	22	1	1
Drupal	2	1	2	1
Wordpress	22	6	0	0
Moodle	93	23	0	0
DVWA	1	1	0	0
phpBB	2	1	0	0
myBB	1	0	0	0
osCommerce	6	3	3	1
SugarCRM	56	41	0	0
FluxBB	3	3	0	0

Where *DC*: Detected Command which refers to the number of commands that change the configuration settings in scripts.

In order to compare our tool with other scanners we used the comparison in *Confeagle* paper [4], since *Confeagle* is similar to our tool. The comparison is between *Confeagle*, *w3af*, *skipfish* and *Websecurify*. Except *confeagle*, the scanners are all generic vulnerabilities scanners. We installed the versions of the applications they used in their experiment, and we used XAMPP package as an environment. We implemented MVS on these applications at the environment and the application levels and we got the the results in Table 5. As we can see, configuration vulnerabilities at the application level are not reported by *w3af*, *skipfish* or *websecuritfy* scanners.

TABLE 5: COMPARING MVS'S VULNERABILITIES DETECTION RESULTS WITH GENERIC VULNERABILITY SCANNERS

Applic-ation	MVS		Con		W3		skip		sec	
	E	A	E	A	E	A	E	A	E	A
Joomla	28	9	28	4	2	0	3	0	1	0
Drupal	28	2	28	2	2	0	3	0	1	0
Word-press	28	7	28	2	2	0	3	0	2	0
Moodle	28	34	28	8	2	0	3	0	2	0

Where *E*: environment, *A*: application, Con: *Confeagle*, *W3*: *w3af*, *skip*: *skipfish*, *sec*: *websecuritfy*.

Although the generic scanners seem to focus on environment configuration vulnerabilities, not much of these vulnerabilities are reported either. While we notice similar results between MVS and *Confeagle* at environment level. But at application level, MVS detected more vulnerabilities than *Confeagle*. In Joomla 2.5 *Confeagle* detected *display_errors* vulnerability in four different files. While MVS detected these vulnerabilities and more five ones of *error_reporting* and *max_execution_time* directives. *Confeagle* ignored *error_reporting* directive and did not consider it as a vulnerability. Knowing that *error_reporting* could has the value *0* which means to turn off all error reporting and in that case the vulnerability of *display_errors* directive is meaningless. Our MVS detects both *error_reporting* and *display_errors* directives and shows their values so the administrator can estimate the actual severity of the combination of both directives. However, this example goes along with the tuning issue we tackled in a previous section.

V.3. Online websites

The websites we used in the experiments are shown in TABLE 6.

TABLE 6. ONLINE WEBSITES USED IN EXPERIMENTS

website	Category	# PHP files	# htaccess files
Usefulbooks.co.uk	E-commerce	252	1
Basmaty.com	Business Catalog	6288	10
Qcs-co.com	Business Catalog	2730	7
Gic-me.com	Business Catalog	2212	2
Doctorwainakh.com	Personal	1214	1
Mambo House	E-commerce	273	1
On Lib Arc	Business Catalog	256	0

We performed experiments on these websites at application level. For example, TABLE 7 shows some vulnerabilities in **usefulbooks.co.uk** which has the following admin metrics: Category = E-Commerce, Business Size = Medium, Sensitive Data = Medium, Web Oriented = High, Profit = True. As we notice, the environmental scores for Usefulbooks.co.uk are high values, since it is e-commerce website and high Web oriented.

TABLE 7. SAMPLES OF VULNERABILITIES IN USEFULBOOKS.CO.UK

Directive	Val	Rec	S	BS	TS	ES
error_reporting	E_ALL	NULL	F	5	5	9
memory_limit	160M	32M	F	7	5	7
max_execution_time	30000	30	F	7	5	7

Where *ES*: environmental score.

Finally we used the suggested formula (6) to compute the customized recommended value for some vulnerable directives and got the results in TABLE 8. As we see in the first two cases, the customized recommended values are closer to the secure value than to the current value, since the environmental score is high 7.0. While in the last case, the environmental score is low 3.0, which means that this vulnerability is not critical for gic-me website, so the customized recommended value is closer to current value.

TABLE 8. CUSTOMIZED RECOMMENDED VALUES

Website	Directive	Val	Sec	ES	CR
Basmaty	max_execution_time	100	30	7	51
usefulbooks	memory_limit	160	32	7	70.4
Gic-me	max_execution_time	5000	30	3	3509

Where *Sec*: secure value, *CR*: customized recommended value.

VI. Conclusion

In this research we have investigated misconfiguration vulnerabilities in Web applications and implemented our approaches in AMP environment. We presented several ideas to enhance the methods of detecting and quantifying the vulnerabilities. A comprehensive checking was applied at both the environment and the application level using extended Gold Standard. We studied the conflicts between performance and security, as we discussed the tuning configuration issue. We used the most recent scoring system CCSS to measure the severity of detected vulnerabilities. We used environmental metrics to compute the customized recommended values for vulnerable directives. The customized recommended values are generated to be proper for particular website according to its characteristics along with taking into consideration the performance issue. We implemented our tool MVS on seven AMP packages, ten open source Web applications and seven online websites.

We can summarize the limitations of this research in four points as follows: we customized only the directives which have numeric values. We assumed that the current value of directive is the high performance value. In the script parsing, we ignored the commands which include PHP variables. In addition, we could not evaluate the accuracy of our approach that computes the environmental metrics using

admin metrics, since there are no similar experiences or available data to compare with.

In future, we wish to work more on customization concept and extend it to all directives in Gold Standard, not just the numeric ones.

References

[1] World Wide Web Size, "The size of the World Wide Web," http://www.worldwideWebsize.com

[2] WhiteHat. 2013. "Website Security Statistics Report,".

[3] J. Tudor. 2013. "Web Application Vulnerability Statistics 2013," Context.

[4] S. Zhang and M.D. Ernst. 2013. "Automated Diagnosis of Software Configuration Errors," in *ICSE*.IEEE.

[5] D. Subramanian, H.T. Le, P.K.K. Loh and A.B. Premkumar. 2010. "Quantitative Evaluation of Related Web-based Vulnerabilities," in *SSIRI-C*.IEEE.

[6] M.M. Casalino, M. Mangili, H. Plate, and S. E. Ponta. 2012. "Detection of Configuration Vulnerabilities in Distributed (Web) Environments," CoRR, vol. abs/1206.6757.

[7] B. Eshete, A. Villafiorita, and K. Weldemariam. 2011. "Early Detection of Security Misconfiguration Vulnerabilities in Web Applications," in *ARES*. IEEE.

[8] B. Eshete, A. Villafiorita, K. Weldemariam, and M. Zulkernine. 2013. "Confeagle: Automated Analysis of Configuration Vulnerabilities in Web Applications," in *SERE*.IEEE.

[9] OWASP. 2013. "OWASP Top 10 – 2013".

[10] PHP. 2013. "PHP Security Manual," http://php.net/manual/en/security.php

[11] MySQL. 2013. "MySQL Secure Installation," http://dev.mysql.com/doc/refman/5.0/en/mysql-secure-installation.html

[12] MySQL, "Security-Related mysqld Options and Variables" http://dev.mysql.com/doc/refman/5.0/en/security-options.html

[13] Cyberciti. 2013. "Linux: 25 PHP Security Best Practices for Sys Admins," http://www.cyberciti.biz/tips/php-security-best-practices-tutorial.html

[14] TechRepublic. 2013. "10 things you should do to secure Apache," http://www.techrepublic.com/blog/10things/10-things-you-should-do-to-secure-apache/477

[15] Tecmint, "13 Apache Web Server Security and Hardening Tips," http://www.tecmint.com/apache-security-tips/

[16] OWASP, "PHP Configuration Cheat Sheet" https://www.owasp.org/index.php/PHP_Configuration_Cheat_Sheet

[17] Ch. Kumar. 2013. "10 Best Practices To Secure and Harden Your Apache Web Server," http://chandank.com/security/10-best-practices-to-secure-and-harden-your-apache-Web-server

[18] OWASP. 2013. "OWASP Configuration Guide," https://www.owasp.org/index.php/Configuration.

[19] Oracle. 2013. "Web Application Security Configuration Guide," http://docs.oracle.com/cd/E28595 01/Web App Security Guide.pdf

[20] High-Tech Bridge, "Web Applications Vulnerabilities CVSSv2 Calculator" https://www.htbridge.com/cvss_web_calculator/

[21] NVD, "Common Vulnerability Scoring System Version 2 Calculator" http://nvd.nist.gov/cvss.cfm?calculator&version=2

[22] S. Wieczorek. 2012. "Best Practice for Highest Performance," http://www.mgt-commerce.com/blog/magento-on-steroids-best-practice-for-highest-performance/