Early Detection of Security Misconfiguration Vulnerabilities in Web Applications

Birhanu Eshete
Center For Information Technology
Fondazione Bruno Kessler (FBK-IRST)
Trento 38123, Italy
Email: eshete@fbk.eu

Adolfo Villafiorita
Center For Information Technology
Fondazione Bruno Kessler (FBK-IRST)
Trento 38123, Italy
Email: adolfo@fbk.eu

Komminist Weldemariam
Center for Information Technology
Fondazione Bruno Kessler (FBK-IRST)
Trento 38123, Italy
Email: sisai@fbk.eu

***Abstract*—In spite of the potential advantages that the World Wide Web offers to the way we interact with businesses and people via web applications, its advancement is paralleled with risks and vulnerabilities leading to attacks of diverse complexities and consequences. Among the pressing security risks in web applications, misconfiguration of security-related settings often leaves back doors open for attackers to exploit vulnerabilities and launch awful attacks. We have also seen that the default security configurations on most of the server package environments —where web applications are deployed— are usually relaxed to allow flexibility to developers and deployers and hence are not reasonably secure when the environment is targeted for real production.**

In this paper, we present SCAAMP (Security Configuration Assistant for Apache, MySQL and PHP), as an automation support to supplement defense against security misconfiguration vulnerabilities. SCAAMP automatically audits security configuration settings of server environments in web application development and deployment. Moreover, it offers features to automatically adjust security configuration settings and quantitatively rates level of safety for server environments before deploying web applications. Using SCAAMP, we were able to evaluate eleven server packages for Apache, PHP and MySQL across three operating system platforms. Our evaluation revealed that SCAAMP is able to audit current security configuration settings and alert users to fix the server environment to achieve the level of safety of security configuration with respect to recommended configurations for real-life web application deployment.

***Keywords*-Web Applications; Security; Configuration; Web Server Environments; Deployment;**

I. Introduction

Web application development has shown tremendous evolution in line with the world wide web and is continuously becoming simpler to realize applications: thanks to ready-made frameworks which often do not require to write even a single line of code [1], [2], [3]. Putting security in the equation, the path is not as smooth as one can imagine it because of the ever-rising security threats (in number and variety) and eventually leading to attacks on web applications [4], [5], [6], [7]. Without doubt, simplicity of developing and deploying web applications, of course, invites multitude of script-kiddies to the crowd of web application developers. Typically such a trend is more of a risk than an opportunity as beginners have very little (if not none) know-how of security risks and they tend to produce functioning but insecure web applications [8], [9], [10], [11]. Even if they have some knowledge and skills, they need to have understanding of how the ready-made source code works, which is time-consuming and practically infeasible in the absence of security experience. On a different scale, the situation also applies to experienced developers because they usually focus on other important matters (e.g., core functionality and complexity) of the application instead of memorizing detailed security requirements during application development or deployment.

One of the widely encountered vulnerabilities of web applications is security misconfiguration [12], which can happen at any level of an application stack, including the underlying platform, web server, database server, framework, and business logic code. According to the report by Open Web Application Security Project (OWASP) [13], security misconfiguration, the subject of this paper, is listed as one of the top ten risks (ranked number six in 2010) which could potentially lead to risks ranging from unauthorized access to some system data or functionality to a complete system compromise.

The situation with security misconfiguration is further aggravated by the fact that a single (and possibly insecure) environment is often used for hosting multiple web applications in shared technologies context (e.g., shared web server). Additionally, insecure instance of a configuration may be duplicated with potential risks. In this regard, a timely example is the emerging cloud service provision aiming at platform, infrastructure and software as a service which requires configuring multiple virtual machines in the cloud [14], [15]. Cloud service providers usually run array of virtual machines (and hence configurations) with multiple cloud users per each. If a reasonably acceptable security configuration is not in place, multiple cloud users would be vulnerable to security risks [16]. Therefore, proper security configuration is more of a necessity than a choice for cloud service providers to build trust among cloud service consumers.

Considering the proportion of security incidences reported to date, incidences which count for security misconfiguration are not only significant but also increasing. Consequently,

ringing the bell of demand for early detection and prevention mechanisms in automated way. For example, according to the Open Source Vulnerability Database (OSVDB) [17], about 42 public PHP configuration vulnerability incidents were documented till January 2011 most of which resulted in Cross-Site Scripting (XSS) attacks (also see in [18]). A query we made to the same source revealed that PHP configuration vulnerabilities increased from 26 in January 2007 to 42 in January 2011, an increase of 32%. Needless to mention here is the fact that other public vulnerability incident databases have similar trends of rising security configuration vulnerability incidences on web applications. Another query we issued on January 21, 2010 to the The National Vulnerability Database (NVD) of USA [19] returned 116 vulnerability incidents related to PHP configuration faults which allowed attackers to launch XSS, SQL injection and directory traversal attacks.

Expectedly, security is far more than ensuring the right configuration. But, even all other layers of security being in place, vulnerable configuration settings can leave the whole system completely insecure. Theoretically, one can imagine several attacks thereafter as each wrongly set security configuration is attributed to one or more attack vectors. Consequently, assisting web application developers and administrators in auditing security configuration, fixing configuration deficiencies and eventually preparing the application and the environment for deployment with reasonable security posture, turns out to fit in the multi-layer approach of defense against attacks on web applications.

However, there is no as such freely available tool that assists developers and administrators to automatically audit, fix and rate security configuration risks for web server environments on which web applications are tested (during development) and then deployed. This paper presents an automated tool —named, Security Configuration Assistant for Apache, MySQL and PHP (SCAAMP)— aimed at supplementing the mission of finding and fixing security misconfiguration vulnerabilities in web applications and web server environments. SCAAMP is evaluated in eleven widely used server environments across three popular operating system platforms. The results show that SCAAMP revealed security configuration vulnerabilities in default installations and detailed analysis of variations in security configurations from different perspectives are presented in this paper. Our tool is made freely available for public use[1].

In summary, our main contributions include:

- We performed an in-depth assessment of security misconfiguration vulnerabilities in web server environments;
- We designed a usable tool, called SCAAMP, to supplement automated security configuration vulnerability auditing, fixing and safety rating for Apache, MySQL and PHP by extending the functionality and usage scenario of PHPSecInfo [20];
- We conducted a detailed experimental evaluation of the tool on eleven real-life web application development and deployment environments with detailed analysis of configuration safety.

The paper is structured as follows. In Section II, we briefly describe the common misconfiguration vulnerability issues through figures from the real world and illustrative examples. Section III describes overview of SCAAMP architecture and its internal details. In Section IV, we discuss experimental setup and evaluation results. Related work is presented in Section V. Finally, conclusion and future work are discussed in Section VI.

II. Misconfiguration Vulnerabilities and Security Risks

There are many threats associated with web applications, including code injections, malicious e-mail attachments, drive-by downloads, account takeover, and click fraud. Although most of these threats revolve around exploiting implementation vulnerabilities, we focus, in this paper, on ways in which an attacker can abuse web functionality that exists by (mis)configuration of web server environments. For example, if a web application relies on *magic quotes* for input checking, the security of the application depends on the configuration of the PHP interpreter. Namely, if the *magic quotes* directive in the PHP configuration is turned off —i.e., *magic_quotes_gpc* = *off*, the web application will most probably have exploitable problems. In this case, for instance, an HTML form element can let a malicious web site generate GET and POST requests to arbitrary web sites, leading to security risks like code injection attack.

In what follows, we describe in detail about misconfiguration vulnerabilities and their consequences in web application security and illustrates attacks that exploit them, by providing evidences from various sources.

Apache, MySQL and PHP (AMP). AMP is arguably the most widely used web application server environment. One of the reasons is that the PHP interpreter, Apache HTTP sever and MySQL database server are all open source and they make a smooth blend leveraging ease of use, flexibility and portability across multiple platforms. According to the January 2011 NetCraft web server survey [21] on 273,301,445 websites, Apache remains dominant hosting 59.13% of the websites. The wide-spread use of PHP as a server-side web scripting language coupled with MySQL server as a backend combined with Apache is quite evident from the 75% share of live websites on the web written in PHP according to [22]. The popularity of AMP environment for web application development have unfortunately made the environment a day-to-day target for attackers who take advantage of potential vulnerabilities of

[1]SCAAMP: http://www.sourceforge.net/projects/scaamp/

which security misconfiguration is one. In what follows, we shade light on issues pertaining to security configuration of the AMP environment components with respect to defaults, official recommendations and opinions of security experts.

Security Configuration for AMP. PHP interpreter is shipped with the **php.ini** configuration file with official defaults set by the PHP group. The configuration settings of this file are read and used then after (if PHP is installed as a Apache module) or read on each invocation of the interpreter (if PHP is installed as CGI module) [23]. The configuration file contains several parameters (directives) used to control the behavior of PHP applications [24]. The directives are used to put constraints such as whether or not: to allow global variables (e.g., *register_globals*, to use certain functions (e.g., *disable_functions*), to reveal interpreter signature (e.g., *expose_php*) and many other settings. For example, *register_globals* controls if all CGI parameters passed will be registered as global variable automatically instead of being available through `$_GET`, `$_POST` or `$_REQUEST` only. Enabling this often gives the freedom of accessing even non-script parameters from other locations. More specifically, from a security perspective, one of the problems with the *register_globals* directive arises in combination with the fact that PHP allows usage of uninitialized variables. If a script accesses variables without initializing them first, an attacker may misuse this flexibility as an opportunity to submit crafted parameters (e.g., Injection attack) assigned to uninitialized parameter variables with his own values. The situation can be illustrated by the following example code from the PHP manual[2]:

```
<?php
 if (authenticated_user()) {
     $authorized = true;
 }
 if ($authorized) {
    include ``/highly/sensitive/data.php'';
 }
 ?>
```

If the user is not authenticated, the variable authorized is not initialized before it is read in the second *if-statement*. If *register_globals* is on, the problem with the above code is that if a user calls your script with the variable "authorized" set to 1, s/he can bypass your security checking routine authenticated_user. This is trivial to do. For example, if your script is called `myScript.php`, the attacker request to access the included page:

```
http://www.example.com/myScript.php?authorized=1
```

With this request the *register_globals* causes the PHP interpreter to create a global variable with the name *$authorized*, and set it to the value given, i.e, 1. That means, that the attacker will always be able to cause the script to include the *include statement* regardless of how s/he completed your form. Similar problems can arise with each variable that is accessed without initializing it first. Notice, however, that this feature has been deprecated as of PHP 5.3.0. Another interesting example in PHP is the *error_reporting* directive which could be set to display all errors, none or only warnings. During development, it is recommended that a developer sets *error_reporting* to all to facilitate error-debugging. But, if the same setting is used for deploying the application in the wild, the errors displayed may leak valuable information (e.g., column names in a table) giving piece of cake for an attentive attacker.

Like PHP, Apache has the **httpd.conf** file to manage global configuration details of the server and it also allows per-directory access control method via its **.htaccess** file [25]. There are directives which can be set in httpd.conf or php.ini. In this case, the value to be used at run time depends on whether or not Apache is configured to allow overriding of global configurations. One of the many examples in httpd.conf that we can control is whether or not to reveal server signature via the *server_signature* directive. By default, this directive is on and can give information such as server version as an insight to an attacker so as to launch attacks specific to that version.

The configuration of MySQL can be controlled either within the **my.conf** file or using MySQL-specific directives in php.ini. Among the potential security holes is the empty root password for most builds of MySQL which could lead to command injection attacks or denial of service attacks.

Recommended Security Configuration. Whether we are using pre-packaged or manually installed AMP server environment, there are default configuration settings for Apache, MySQL and PHP. The pitfall is that these defaults are not usually secure for deploying real web applications. In fact, official documentations of Apache, MySQL and PHP have recommended details and remarks for exceptions about security configuration. Furthermore, security experts also suggest configuration values and implications of security-critical directives taking differences in platforms, versions and usage scenarios as a foundation to warn web developers and administrators [25], [23]. However, there are still significant number of security misconfiguration incidents reported for the AMP environment [17], [19]. The trend shows that developers and administrators, specially inexperienced ones, have little know-how of the implied security vulnerabilities of configuration directives and tend not to stick to recommended security configuration settings.

Discussion. In fact, there are several ways in which web application can be vulnerable, including application logic faults due to poor coding practices; irrespective of the technology used for implementation, the security of the web server, as well as the back-end database. Protection mechanisms could be deployed at the host, network, application, and server

[2]http://www.php.net/manual/en/security.globals.php

levels. At all levels, configuration that reflects reasonable security is essential. The host, the server and the network are comparatively more secure because of matured protection solutions (e.g., Firewalls, Intrusion Detection Systems) which shifted the focus of attackers to public interfaces of web applications to sniff vulnerable spots. Therefore, it is critical to re-assess configuration vulnerabilities with respect to recommended configuration settings for web applications and develop a tool to be used by web application developers and/or administrators to evaluate the security posture of web applications and server environments from the standpoint of secure configuration. In line with this SCAAMP can benefit the users (i.e., administrators and developers) of a wide range of safe configurations, by giving users control over their backend configurations, by making it easier for developers to audit and fix misconfiguration vulnerabilities, and by reducing the need for follow-up extensive and expensive security testing for a given web application.

III. Architecture and System Details

In this section, we discuss the details of our system, called SCAAMP, to detect security configuration vulnerabilities on security critical directives, audit and fix those that are misused, and quantitatively measure the safety ratings of the environments —specifically, for the AMP server environment.

After assessing various recommendations and expert opinions on the security configuration of those directives based on official documentations, we identified certain values that help choose what constitutes safe recommend values for the directives. Our system shown in Figure 1 is a Web application and consists of three major components and one auxiliary component. The major components are the security configuration Auditor, Fixer and the Configuration Safety Rating Module. The auxiliary component contains the Utilities modules. To better describe internal details of SCAAMP, we may consider a typical workflow in the system.

Suppose that a user (web developer or administrator) wants to audit or fix security configuration of a certain AMP server environment. Using the Web UI and Initializer module, the user supplies credentials used to initialize the system (e.g., detecting platform and configuration file paths per each AMP component, permission tokens). Once the initialization is over, the user launchs the Auditor/Fixer of Apache/MySQL/PHP. Depending on the AMP component selected, the system conducts security configuration auditing or fixing by invoking the respective module.

The internal logic behind the Auditor module can be explained as follows. Given N number of security configuration directives and let D be the set of such directives denoted as $D = \{d_1, d_2, d_3, \ldots, d_{N-1}, d_N\}$ where each d_i is a tuple with attributes $<name, recommended\_value, description, remark>$. For each d_i, the configuration file is parsed to determine whether d_i is enabled or not. If it is enabled, then its value is retrieved and populated on audit result list. Finally, the audit result list is displayed per each active directive as $<name, recommended\_value, current\_value, description, remark, change\_menu>$.

For the Fixer module, a highlight of its internal operation is as follows. First, it collects possibly changed values of each d_i from $change\_menu$. Then it searches the configuration file using the $name$ of d_i as a search key. When it finds a match and $current\_value$ of d_i is different from the value of $change\_menu$, it assigns the value submitted via $change\_menu$ to $current\_value$ of d_i. This repeats as long as the changed d_i's are not exhausted.

At the end of either auditing or fixing, the system generates safety rating report (using the Configuration Safety Rating Module). Given N number of security-critical directives, the safety rating module quantifies the safety percentage as $([\sum_{i=1}^{m} f(d_i)]\ /\ N)\times 100$ where m is the number of security configuration directives currently in use, d_i is the security configuration directive being processed, $f(d_i)$ is a function that returns 1 if recommended value of d_i is equal to current value of d_i or returns 0 otherwise.

The HTTP server, the Interpreter and the database server are attached to their respective configuration files to be parsed by the Auditor for getting current values and modified by the Fixer to change configuration values. The Environment Resetter module is used to facilitate restarting the underlying server environment since most of the changes in configuration settings require resetting the server environment. The architecture of the system is reasonably generic in the sense that it could be easily adapted with little modifications to other server environments by replacing the actual instances of the web server, server-side script interpreter and database server.

IV. SCAAMP Experimental Evaluation and Results

We conducted an experiment to evaluate the effectiveness of our system in detecting misconfiguration vulnerabilities on Apache, MySQL and PHP on three operating system platforms.

A. Experimental Setup

For experimental evaluation of SCAAMP, we prepared eleven ready-to-use environments for AMP: eight real-world pre-packaged server environments —namely, `EasyPHP`, `Vertrigo`, `WampServer`, `XAMPP` (for Windows, Mac OS and Linux), `UniServer`, and `MAMP`— which we download from the SourceForge repository [26], and three manually installed packages by following their respected documentations that detail installation and configuration instructions. When choosing these pre-packages, we particularly looked

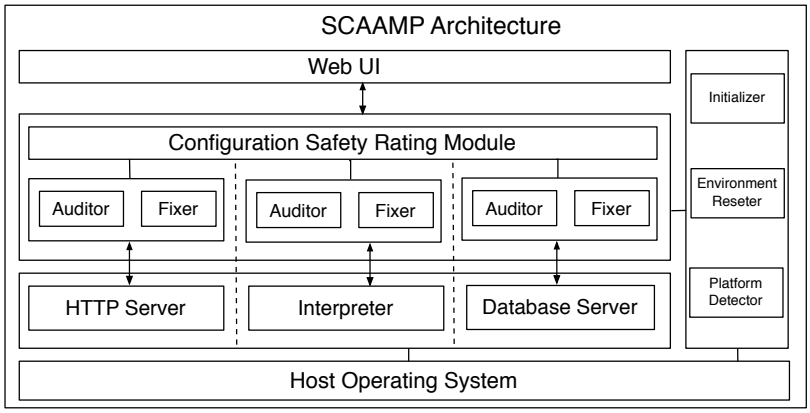

Figure 1. SCAAMP System Architecture.

Table I
AUDIT RESULT OF DEFAULT SECURITY CONFIGURATION SAFETY

Package	OS	PHP (%)	MySQL (%)	Apache (%)	Package Safety (%)
EasyPHP-5.3.3.1	Windows 7	50.00	12.50	0.00	20.83
Vertrigo_222	Windows 7	30.00	12.50	25.00	22.5
WampServer2.1	Windows 7	50.00	12.50	0.00	20.83
XAMPP for Win	Windows 7	28.57	25.00	0.00	17.86
XAMPP for Linux	Ubuntu10.10	48.39	12.50	0.00	20.3
XAMPP for OSX	OSX10.6.4	30.00	12.50	0.00	14.17
UniServer	Windows 7	44.12	0.53	37.50	27.38
MAMP	OSX10.6.4	32.26	12.50	25.00	23.25
Manual Linux	Ubuntu10.10	56.67	12.50	0.00	23.06
Manual Win	Windows 7	42.42	0.53	0.00	14.32
Manual OSX	OSX10.6.4	60.00	25.00	0.00	—
Overall Average Safety		**42.95**	**12.60**	**7.95**	—

for their popularity with respect to the number of total downloads and also verified that these packages are still actively used, from the weekly download statistics of January 19-26, 2011. For instance, `XAMPP` for Windows is downloaded 26,805,631 times and `WampServer` for 9,987,338 times. Roughly, this indicates that they are widely in use in real-life web applications development and deployment. Additionally, the distribution across operating system platforms is such that six packages for Windows 7, two packages for Linux-Ubuntu10.10 and three packages for MacOSX10.6.4. We then installed SCAAMP on the three platforms and run the audit for each server package (see Table I). After collecting the audit results of default security configuration safety ratings for all server packages using the Auditor module, we performed configuration fixing operations.

In particular, the Fixer module performs the security configuration fixing after the the developer or administrator has changed those directives with insecure configuration values to values which reflect a more secure configuration settings (Table II). For the purpose of the experimentation and to make the fixing uniform across the packages, we fixed five directives in PHP and one to three directives for Apache and MySQL taking recommended values as reference. Then after, we repeated the audit again. Finally, we compared the audit output of default configuration with the audit output of the enhanced configuration to rate the level of safety of each server environment. As an interesting side issue, we also analyzed the variation in the level of safety across the three operating systems, among packages within each operating system and between pre-packaged and manually installed server packages. Quantitative safety rating of a server environment is computed (see Section III) as percentage of directives for which values are set as per the recommended settings under the assumption that all security critical directives are of equal count in computing overall safety rating.

B. Evaluation Results

We examined the safety of the eleven packages that are widely used in web applications development and deployment as server environment and compared the safety results with each other, including across the different platforms. An important observation while evaluating SCAAMP is the fact that default configuration of security critical directives varies to a significant extent —be it across packages in the same operating system or among packages— in different operating systems. The automatic audit result (Table I) of default configuration revealed that all the packages are by far below the recommended security configuration settings. For instance, the default safety rate is in the range 14.17% (lowest) and 28.33% (highest) equating to an overall average package safety rate of 21.17%.

Splitting the safety rating to server components, the average safety rate of PHP, MySQL and Apache is 42.95%, 12.60% and 7.95% respectively. Of all the packages, manual installation of MacOSX has the highest PHP safety rate (about 60%) while `XAMPP` on Windows show the lowest PHP safety (about 28.57%). Apache seems comparatively safe (about 37.5%) in `Uniserver` package of Windows while it appears with safety rate of 0% in eight of the eleven packages used in the evaluation. Manual installation of MySQL on Windows and `Uniserver` on Windows have

Table II
AUDIT RESULT OF SECURITY CONFIGURATION SAFETY AFTER FIXING.

Package	OS	PHP (%)	MySQL (%)	Apache (%)	Package Safety (%)	Safety Improvement (%)
EasyPHP-5.3.3.1	Windows 7	66.67	50.00	25.00	47.22	26.39
Vertrigo_222	Windows 7	46.67	50.00	25.00	40.56	18.06
WampServer2.1	Windows 7	66.60	50.00	33.33	49.98	29.14
XAMPP for Win	Windows 7	39.39	50.00	25.00	38.13	20.27
XAMPP for Linux	Ubuntu 10.10	64.52	50.00	33.33	49.28	28.99
XAMPP for OSX	OSX 10.6.4	46.67	50.00	33.33	43.33	29.17
UniServer	Windows 7	58.88	50.00	75.00	61.29	33.91
MAMP	OSX 10.6.4	51.61	50.00	50.00	50.54	27.28
Manual Linux	Ubuntu 10.10	73.33	50.00	0.00	41.11	18.05
Manual Win	Windows 7	57.58	50.00	33.33	46.97	32.65
Manual OSX	OSX 10.6.4	76.67	50.00	33.33	53.33	25.00
Overall Average Safety		**58.96**	**50.00**	**33.33**	—	—

almost zero safety (about 0.53%) as opposed to the majority of packages with 12.5% safety rate.

From the above figures about the default security configuration, one can conclude that the default configurations of Apache, MySQL and PHP are not in a posture that a web developer may rely on. It is quite worrying that most of the packages used in our experiment are used for developing and deploying web applications being so insecure in terms of configuration and of course there is high probability that the server packages are used for real production with the default (insecure) configuration in place. As a consequence, an attacker can easily exploit such vulnerabilities, e.g., to inject code into a web application of the vulnerable site due to misconfiguration. See also the example we gave in Section II, the fact that PHP allows usage of uninitialized variables when the *register_globals* directive is turned *On*.

Table III
SECURITY CONFIGURATION SAFETY COMPARISON ACROSS AND WITHIN PLATFORMS.

OS	Average Safety (default)	Average Safety (after fixing) (%)	Highest Contributor (s) (%)	Lowest Contributor (s) (%)
Windows	20.62	47.36	WampServer2.1 EasyPHP5.3.3.1	XAMPP for Win
Linux	21.68	45.20	Manual Linux	XAMPP for Linux
MacOS	21.92	49.07	MAMP	XAMPP for OSX

Table III shows the comparison of audit results among Windows, Linux and MacOS. As you can see the difference is not that significant among the operating systems. However, within individual operating systems, there are packages contributing higher proportion (e.g., `WampServer` and `EasyPHP` in Windows) where as the contribution of some others is almost insignificant (e.g., `XAMPP` for MacOS).

Another dimension of comparison we made is whether there is significant variation between security configuration safety between pre-packaged and manually installed server environments, as shown in Table IV. To this end, the variation is as insignificant as 1% for audit results before and after fixing configuration settings. Therefore, the pre-packaged server environments do not introduce security configuration enhancements than simplifying ease of use. However, just comparing only manually installed packages, one noticeable difference is the relative safety PHP in MacOS (60% as opposed to 42% for Windows and 56% for Linux).

Table IV
SECURITY CONFIGURATION SAFETY COMPARISON: PRE-PACKAGED VERSUS MANUALLY INSTALLED

Package	Average Safety (default)	Average Safety (after fixing) (%)	Highest Contributor (s) (%)	Lowest Contributor (s) (%)
Pre-Packaged	20.89	47.54	Manual OSX	XAMPP for OSX
Manually Installed	21.90	47.14	UniServer	XAMPP for Win

V. RELATED WORK

As a remedy to the rising misconfiguration vulnerabilities of web applications, numerous techniques and tools, commercial and free, have been developed by security experts (see, e.g., in [8], [27], [28], [29]). We present a comparative discussion of the ones related to our work here.

Regarding security configuration vulnerability for the AMP environment, one of the closest tools we came across is PHPSecInfo [20]. PHPSecInfo automatically displays the current values and recommended values of PHP configuration directives from secure configuration perspective. It also summarizes the correctly and incorrectly set percentage of directives against the recommended values along with online links to descriptions of the conditions under which directives are used from security standpoint. Among the limitations of PHPSecInfo are: it is limited to PHP only, it misses some security critical configuration directives in PHP; it does not offer features to automatically fix security configuration values on the spot and re-auditing after fixing; and, it maintains recommendations about security configuration directives online. Our tool, however, includes the Apache HTTP server and the MySQL database server in the audit in addition to PHP. More directives (43 as compared to only 18 in PHPSecInfo) are covered by our tool. More importantly, our tool allows automatic fixing of security configuration values and technical details of recommendations about security configuration directives are shipped within the tool.

Another relevant security configuration auditing tool is PHP Security Audit [30]. This provides an audit script that checks core PHP configuration, available functions, and available classes to determine potential vulnerabilities and offer configuration suggestions. One good feature of this tool is that it generates recommendation report indicating what the user has to do (e.g., list of functions to disable) to fix vulnerabilities discovered after the audit. Like PHPSecInfo, this script is also limited only to PHP and has no features to automatically make the suggested changes to configurations.

A study on assessing and comparing the security of web servers, using a comprehensive survey of 400 security practices from several sources as base reference, is presented in [12]. More specifically, the authors examined the security features of web servers' installations and determined the security of a given web server configuration. They also defined a metrics based on the weighed percentage of security recommendations (i.e., the 400 security practices) to indicate the security level of the assessed web server. Differently from our work, this work has two main limitations. First, their assessment is limited only to Apache web server installation and configuration security features. Secondly, the assessment and comparison are done manually.

Although powered partly by commercial tools, Lin et al. [31] propose a tool that automatically assesses and detects vulnerability and misconfiguration severity. The system consists of two main components the Common Vulnerability Scoring System (CVSS) module and the Common Configuration Enumeration (CCE) module. The first component is designed to assess the vulnerabilities reported by the Nessus [32] —a network vulnerability scanner tool— according to system information. This allows to measure the vulnerability severity of an organization. The second component, whereas is used to scan the system and determine the actual presence of the misconfiguration in the system. More specifically, the CCE scanner helps administrators in checking whether their system configurations comply with their policy definitions. However, our work assumes that security policies should be in agreement with recommended configurations of web applications.

WAVES [33] is a black-box testing framework designed to conduct a wide variety of vulnerability assessment in web application, including cookie poisoning, parameter tampering, hidden field manipulation, input buffer overflow, session hijacking, and server misconfiguration. For example, using WAVES one can inject a properly crafted pattern into a web application under analysis to make it display database error messages. If these database error messages are displayed in response to the injection being made — say, the *error_reporting* directive in PHP configuration is set to display all errors— it is vulnerable to that pattern. SCAAMP complements this framework by detecting and fixing these kinds of vulnerabilities, at the earliest possible time and would otherwise be expensive and time-consuming task. Similarly, other tools (e.g., automated scanners [32], [34], [35]) are used to detect vulnerabilities such as missing patches, misconfigurations, use of default accounts and invocation of unnecessary services. However, the majority of these tools are either based on proprietary technologies or specific to platforms and above all are not freely available.

VI. Conclusion and Future Work

In this paper, we presented SCAAMP, a tool for auditing, fixing and safety rating security misconfiguration vulnerabilities in web applications and web server environments. We conducted an in-depth assessment of recommended security configuration settings for the AMP environment, taking official documentations and security experts' opinions as a basis. We designed a generic architecture, which can be customized to other target environments (e.g., Python, Ruby on Rails) with little modifications.

The system is instrumental in assisting web application developers and administrators by automatically providing security configuration vulnerability alerts at the early stage of development or right before deploying an application. Using the tool, we conducted detailed security configuration auditing of eleven popular AMP packages across three operating system platforms. Moreover, we also demonstrated how it can be employed to enhance security configuration safety by fixing insecure configuration directives to achieve an ideal level of security configuration. The experimental results indicate that the default security configuration details of the eleven packages are way too far from the recommended security configuration settings. This is in the contrary to the fact that most of them have been in use and are still in use for developing real-life web applications. Additionally, there exist observable variations among the eleven packages while the security configuration safety comparison across operating system platforms, and between manual and prepackaged server environments is not that significant.

SCAAMP has three major limitations to address in the future so as to give a more fine-grained security configuration vulnerability alert to users. First, it does not address overridden global configurations in situations where a certain web application can override global configurations and provide customized configuration settings per each directory of the application. Understanding the hierarchical structure of the directories in the application and recursively tracking the configuration settings against individual directories could be a way-out to deal with such a limitation.

The second limitation is the fact that SCAAMP does not systematically associate application specific configuration settings with global configuration settings. This could be incorporated in the automatic auditing by maintaining. We plan to explore extending SCAAMP to support the automatic auditing of application specific configuration by maintaining metadata (e.g., associating scripts with directives) of the application components with respect to security configuration.

Thirdly, the configuration safety rating assumes that all security configuration directives have equal weight in the quantitative safety rating, which may not be the case in general. This could be improved by undertaking further enhancements to safety metric computation either by assigning special weights to dependent or more critical set of security configuration directives. Therefore, future work moves along the lines outlined above.

REFERENCES

[1] A. Veglis, "PHP and SQL Made Simple," *IEEE Distributed Systems Online*, vol. 6, 2005.

[2] M. Jazayeri, "Some Trends in Web Application Development," in *2007 Future of Software Engineering*, ser. FOSE '07. IEEE Computer Society, 2007, pp. 199–213.

[3] A. Kumar, S. K. Agarwal, and P. Manwani, "The Spoken Web Application Framework: User Generated Content and Service Creation through Low-End Mobiles," in *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, ser. W4A '10. ACM, 2010, pp. 2:1–2:10.

[4] T. Holz, S. Marechal, and F. Raynal, "New Threats and Attacks on the World Wide Web," *IEEE Security and Privacy*, vol. 4, pp. 72–75, March 2006.

[5] P. Tramontana, T. Dean, and S. Tilley, "Research Directions in Web Site Evolution II: Web Application Security," in *Proceedings of the 2007 9th IEEE International Workshop on Web Site Evolution*. IEEE Computer Society, 2007, pp. 105–106.

[6] Jamie Riden, Ryan McGeehan, Brian Engert, and Michael Mueter, "Know your Enemy: Web Application Threats," http://www.honeynet.org/papers/webapp/, April 2008. Last checked Feb 2011.

[7] SANS Insitute, "The Top Cyber Security Risks," http://www.sans.org/top-cyber-security-risks/, September 2009.

[8] M. Cova, V. Felmetsger, and G. Vigna, "Vulnerability Analysis of Web Applications," in *Testing and Analysis of Web Services*, L. Baresi and E. Dinitto, Eds. Springer, 2007.

[9] D. Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna, "Multi-module Vulnerability Analysis of Web-based Applications," in *Proceedings of the 14th ACM conference on Computer and communications security*, ser. CCS '07. ACM, 2007, pp. 25–35.

[10] J. Fonseca, M. Vieira, and H. Madeira, "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks," in *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*. IEEE Computer Society, 2007, pp. 365–372.

[11] A. Barth, C. Jackson, and J. C. Mitchell, "Securing Frame Communication in Browsers," *Commun. ACM*, vol. 52, pp. 83–91, June 2009.

[12] N. Mendes, A. A. Neto, J. a. Durães, M. Vieira, and H. Madeira, "Assessing and Comparing Security of Web Servers," in *Proceedings of the 2008 14th IEEE Pacific Rim International Symposium on Dependable Computing*. IEEE Computer Society, 2008, pp. 313–322.

[13] OWASP. (January) OWASP Top Ten. [Online]. Available: http://www.owasp.org/index.php/Top_10

[14] Cloud Security Allianc, "Top Threats to Cloud Computing V1.0," http://www.cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf, March 2010.

[15] H. Liu, "A New Form of DOS Attack in a Cloud and Its Avoidance Mechanism," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, ser. CCSW '10. ACM, 2010, pp. 65–76.

[16] S. Subashini and V. Kavitha, "Review: A Survey on Security Issues in Service Delivery Models of Cloud Computing," *J. Netw. Comput. Appl.*, vol. 34, pp. 1–11, January 2011.

[17] OSVDB. The Open Source Vulnerability Database. [Online]. Available: http://osvdb.org

[18] NIST National Vulnerability Database, "CVE2008-0125: Cross-Site Scription (XSS) Vulnerability in phpstats.php," http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-0125, October 2008.

[19] NIST National Vulnerability Database. (2011). [Online]. Available: http://web.nvd.nist.gov/view/vuln/search

[20] PhpSecInfo. [Online]. Available: http://phpsec.org/projects/phpsecinfo/

[21] NetCraft. January 2011 Web Server Survey. [Online]. Available: http://news.netcraft.com/archives/2011/01/12/january-2011-web-server-survey-4.html

[22] W3Techs. Web Technology Surveys. [Online]. Available: http://w3techs.com/technologies/overview/programming_language/all

[23] PHP. Description of Core php.ini Directives. [Online]. Available: http://php.net/manual/en/ini.core.php

[24] P. Asadoorian, "Configuration Auditing To Help Prevent Web Application Attacks," http://blog.tenablesecurity.com/2009/08/configuration-auditing-phpini-to-help-prevent-web-application-attacks.html, 2009.

[25] Apache Software Foundation. Configuration Files- Apache HTTP Server. [Online]. Available: http://httpd.apache.org/docs/2.0/configuring.html

[26] Open Source Software, "Sourceforge." [Online]. Available: http://sourceforge.net/

[27] M. Curphey and R. Araujo, "Web Application Security Assessment Tools," *IEEE Security and Privacy*, vol. 4, pp. 32–41, July 2006.

[28] M. Monga, R. Paleari, and E. Passerini, "A Hybrid Analysis Framework for Detecting Web Application Vulnerabilities," in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*, ser. IWSESS '09. IEEE Computer Society, 2009, pp. 25–32.

[29] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna, "Toward Automated Detection of Logic Vulnerabilities in Web Applications," in *Proceedings of the USENIX Security Symposium*, August 2010.

[30] D. LeFree. PHP Security Audit. [Online]. Available: http://php-security-audit.com

[31] C.-H. Lin, C.-H. Chen, and C.-S. Laih, "A Study and Implementation of Vulnerability Assessment and Misconfiguration Detection," in *Proceedings of the 3rd IEEE Asia-Pacific Services Computing Conference (APSCC)*. IEEE, 2008, pp. 1252–1257.

[32] Nessus: The Network Vulnerability Scanner. [Online]. Available: http://www.nessus.org/nessus/

[33] Y.-W. Huang, C.-H. Tsai, T.-P. Lin, S.-K. Huang, D. T. Lee, and S.-Y. Kuo, "A Testing Framework for Web Application Security Assessment," *Computer Network*, vol. 48, pp. 739–761, August 2005.

[34] Acunetix Inc. (2010) Acunetix Web Security Scanner . [Online]. Available: http://www.acunetix.com

[35] Hewlett Packard, "HP WebInspect." [Online]. Available: www.hp.com/go/securitysoftware