Web Vulnerability Measures for SMEs

Prosper K. Yeng[1], Peter Nimbe[2], Benjamin A. Weyori[2], Terje Solvoll[3], Bian Yang[1]
Norwegian University of Science and Technology, Norway[1]
University of Energy and Natural Resources, Ghana[2]
Nowegian Center for E-health Research University Hospital of North Norway, Norway[3]
{prosper.yeng, bian.yang}@ntnu.no[1]
{peter.nimbe,benjamin.weyori}@uenr.edu.gh[2]
Terje.Solvoll@ehealthresearch.no[3]

Abstract

An investigation was conducted into web vulnerabilities in commonly used web application templates and frameworks (WAFs) systems such as Joomla, WordPress, Moodle and C #.Net framework. A web vulnerability scoring scheme was developed and used to record metrics of the vulnerabilities associated with the web application templates and frameworks. A custom web application was also developed purported to demonstrate how the vulnerabilities could be shielded in web application frameworks (WAFs). The investigations and implementations were guided by Open Web Application Security Project. The study found some of the most common vulnerabilities in the frameworks and templates at different levels. The choice of Content Management Systems (CMS) templates and WAFs for web application systems development can then be guided by this study.

1 Introduction

Web-based systems consistently remains the top most frequent mode of ingress into systems and data breaches in the world[1, 2] despite their indispensable services[3, 4]. According to Semantic report, there was a global increase of 56 percent on web related attacks on endpoints in 2018[1]. In spite of the high increment in attack, Bitnami reported that over one million web applications are being deployed on its web store in every month[5]. This indicates the necessity of web-based systems in this era. Bitnami is just one of the web stores with various web application frameworks, content management systems (CMS) and templates for constructing web-based systems[5]. Bitnami makes it very easy for users to deploy web applications. It has over 130 most popular web application templates which can be used for web application development for vital and sensitive business areas such as e-commence solutions, corporate websites, university systems management and many more [6]. Web template Or Content Management System (CMS) is known as an already designed web application system which allows users to “plug-in” or substitute contents such as text, images and video into the web system[7-9]. A web application framework (WAF) is a library that makes it possible to develop a web application using pre-written code[7, 8]. WAFs provide mechanisms to enable typical work flows in web application development. The essence of the frameworks, templates and CMS (known in this study as WAFs) are to prevent repetitive and complex tasks, providing professional non-functional requirements such as navigations and look and feel. WAFs were also developed to address some challenges associated with web application developments such as need for integration of unstructured information with structured information, management of web pages and to ease web development challenges and difficulties often faced by non-technical, web application developers[7-10]. Additionally, other web application frameworks such as model, view and controller in C#.Net were developed to ease the web application development burden for technical IT experts. These WAFs helps the developer to concentrate on their core scenarios[4]. Web application frameworks have gained wider adoptions by users who have no professional IT training, SMEs and those who lack some kind of skills such as interface designs or coding which are often offered by these template systems[11] [4]. The materials which were investigated in this study include Content Management, Templates and frameworks. Therefore, Web Templates, CMS or WAF were used interchangeably in this study.

Considering the higher rate that web applications are being published, versus the higher rate of ingress into systems through web application systems, it triggers an alternate thinking that web-based template systems are associated with the most common vulnerabilities and hence driving the rise in attack. It is also considered that web templates are largely used particularly by SMEs who lack financial capabilities and IT experts to develop web systems from scratch[7-10]. Medium sized enterprises are companies with less than 250 employees, less than 50 million annual turnover and a balance sheet total of less than 43 million euros[12]. Small enterprises include companies with less than 50 employees, a turnover of less than 10 million and a balance sheet total of less than 10 million[12]. SMEs accounts for 99% of all enterprises in the European Union while providing millions of jobs[12]. SMEs rely on internet resources, such as web related systems, to gain the muscles and agility of larger organizations such as omnipresence, just in time delivery and faster rate of data processing capabilities. However due to the use of such resources in their operations, SMEs are more exposed to cyber threats in recent times and account for 58% [13]in cyber-attack in the past 12 months with an average cost of an attack estimated to be around $3 million, which includes cost of downtime during a breach, loss of profits, loss of clients and partners, loss of trustiness and cost incurred through mitigations. The main reasons why SMEs heavily rely on web application frameworks and templates are fewer personnel, low financial resources, less IT expertise and reliance on external expertise[14, 15]. Based on these reasons, the design and implementation of these WAFs were possibly aimed at large companies and did not consider the challenges and security needs of SMEs. Therefore, the contribution of this study was to investigate into the most commonly used WAFs for the most common web vulnerabilities and to develop a framework for proactive defenses. SMEs for instance, would therefore know the vulnerabilities associated with the various templates and their mitigation strategies and this can guide them in their decision of WAFs usage.

2 State-of-the-art

Web application templates, frameworks or content management systems are pre-designed, coded and implemented web-based systems with the aim to ease some kind of difficulties often faced by their targeted users. CSM such as Joomela!, Moodle and WordPress were developed to meet usability and non-technical users who desire to create professional web sites and web applications. Microsoft .Net framework has model, view and controller components to meet separation of concern requirements and to ease the repetitive burden from developers. Due to the needs being addressed by these WAFs, the usage rate is quite high[5]. But the question is that how does these WAFs meet security requirement of confidentiality, integrity and availability (CIA)? Are they not contributing to the increasing rate of ingress through the web systems if security is traded for usability and ease of systems development for low skilled IT persons and novices?

In addressing these questions, De Meo, et al, studied into SQL Injection (SQLi) security flaws in web application frameworks and developed SQLfast which is used as a formal approach for exploiting SQLi attacks. ASLan++ was used at the input as the modeling language with the AVANTSSAR Platform for security protocol analysis[16]. ASLan++ is for formal specification of dynamically composed security-sensitive web services and service-oriented architectures, their related security policies, and security properties, at the communication and application level[17]. AVANTSSAR is a framework for specification and Validation of trust and security of service-oriented architectures[18]. In SQLiFast, the ASLan++ translator, generates a transition system in the low-level language which is used to call a model checker and generates an Abstract Attack Trace (AAT) in attack scenarios. SQLfast automatically detects which type of SQLi has been exploited and, in an interactive way and generates the curl or sqlmap commands to support the attack reported. A proof-of-concept was conducted on WebGoat, Damn Vulnerable Web Application (DVWA), Joomla! 3.4.4, and Yet Another Vulnerable Web Application (YAVWA). De Meo, F. et al. study revealed possible SQLi vulnerability associated with an older version of Joomla! and known vulnerable websites but the vulnerability studied did not cover a lot of the top ten OWASPs outlined vulnerabilities[19] and the WAFs covered in this study were not among the most commonly used WAFs[5].

In defending against web vulnerabilities related intrusions, Saxena et al [20] proposed "SCRIPTGARD", which is a system that provides guarantees against code injection attacks and can be implemented on legacy applications. Saxena et al., analyzed sanitization defenses in a web application with over 400,00 lines of code. The system was widely used and was potentially processing untrusted data from users. Sanitization is the mechanism of using filters on user inputs to prevent the compromise of CIA from untrusted data[20]. The study revealed inconsistent sanitization and inconsistent multiple sanitization flaws. Application developers were inadvertently mismatching their choice of sanitizers with the browser's parsing context. The vulnerabilities were not lack of validation issues but rather due to indirect nesting of HTML contexts and sharing of dataflow paths in the application. Specifically, inconsistent sanitization on 1,207 paths were found (representing 4.7% of the total paths) and observed an additional 285 instances of inconsistent multiple sanitization. Based on the findings, Saxena et al., suggested the use of automatic sanitizer placement such that, in an application, automatic selection of the suitable sanity checks would be applied on a dataflow path from a possibly untrusted data source to an HTML output sink. Saxena et al., provided knowledge on how to place sanitizers for effective countermeasures of injection flaws however, knowledge on vulnerabilities associated with web application frameworks was not provided so users with low security abilities such as SMEs and individuals have no knowledge on the kind of vulnerabilities that may exist in these templates.

According to Weinberger et al., Cross-site scripting (XSS) attacks has been a notorious threat to existing and emerging web applications. Even Larger web service organizations such as Google Analytics, Facebook and Twitter have not been spared with XSS attacks. In their contribution, Weinberger et al., conducted a systematic analysis of XSS sanitization in WAFs[4]. The study thoroughly studied into the security of the XSS sanitization concepts frameworks and develop a model of the web browser to determine the challenges of XSS sanitization. The result was used to systematically evaluate the XSS abstractions in some web application framework including CodeIgniter, ASP.NET Request Validation, XSS terminate Rails plugin, Django, GWT SafeHtml. Ctemplate and CLEAESILVER[4]. The study disclosed that the frameworks were not addressing critical parts of the XSS problem. There was huge difference between the abstractions provided by frameworks and the requirements of applications.

Some of the WAF disclosed the security mechanisms in their systems. WordPress claim that the default html tags permitted on its framework are healthy choice to let people use html in their comments and posts, without compromising the security or safety of the user's data or server [21]. Joomla!

Developed JInput class for input filtering at its default setting[22]. Moodle has exhibited its security issues of which XSS partially protected but default password was a serious threat[23]. Obviously, these frameworks claim to have some kind of security protection, but the objective investigation would reveal the actual security the states of WAFs.

3 Materials and Methods

A literature survey was conducted in IEEE-Xplore, Google Scholar, Science Direct/Elsevier and Springer, to determine related studies of security measures in considering the security challenges associated with the usage of WAFs. In Addition, some investigation methods were used to assess the vulnerabilities on commonly used Web Application Frameworks (WAFs) and templates. Traditional system development life circle (SDLC) was used to develop web application purported for the security implementation framework of WAFs for SMEs. The investigation techniques were again used to evaluate the security measures built into the developed web application. Generally, Open Web Application Security Project (OWASP) standard of testing and securing against the vulnerabilities and other standards were adopted [24].

The Web Application Frameworks (WAFs) outlined in Table 1, were used as victims of attack in the study and were tested for different vulnerabilities. A web application (WebPOS) was also developed and used as a tool to attack the WAFs and was a victim of attack to test for the security resilience after the implementation of the security measures against the assessed vulnerabilities. Details of these materials are shown in Table 1.

Table 1: Materials and their attributes used

Resource/Tool	**Version**	**Hosting**	**Usage Type**
WordPress	4.5.3	hosted with xampp	Victim
Joomla!	3.5.1	Hosted with Xampp	Victim
Moodle	3.1.1+	Hosted with Xampp	Victim
MVC C# .Net	4.0.30319	Hosted with Internet Information Services (IIS)	Victim
WebPOS	Designed with php version 5.6	Hosted on Azure web services	Attacker/Victim
Tryit editor	3.0	W3Schools	Attacker
MySQL	Ver 15.1 Distrib 10.1.28-MariaDB, for Win32 (AMD64)	Locally hosted with Xampp and remotely with Azure web service	Victim/Attacker
html	5.0	N/A	Attacker

With reference to Table 1, the Resource or Tool column indicates the names of the resource or tool used in the study. The version column indicates the version of the material used while the hosting column indicates the server and where the service or material was hosted. The type of usage column indicates how the material was used. A victim indicates that the material or tool was used as a target while an attacker means that the WAFs was used as an attacker site to attack the victims.

Default settings of some web application frameworks(WAFs) were tested for vulnerabilities. The vulnerabilities tested were Cross Site Request Forgery (CSRF), Cross Site Scripting (XSS), Reverse Shell or File upload (RSA), SQL injection (SQLi), Session Hijacking (SHJ), Exposed Accessed Credential (EAC) and Brute Force Attack (BFA). Guideline for selecting these vulnerabilities were taken from OWASP top ten vulnerabilities[25]. A common approach of testing for each of these vulnerabilities was adopted for the WAFs such as WordPress, Joomla, Moodle and .Net framework (C#) as shown in Table 1. A scoring scheme was developed as shown in Table 2 and it consists of the WAF column which corresponds to the web template system, to be tested, such as WordPress. The Vulnerability ID (VUL. ID) uniquely identify the vulnerability to be tested. The vulnerability column corresponds to the vulnerabilities tested.

Table 2: Scoring scheme of vulnerabilities

WAF	VUL. ID	VULNERABILITY	SECURITY LEVEL
WordPress	XSS	cross-site scripting (XSS)	
	BFA	brute force attack	
	SQLi	SQL Injection	
	RSA	File upload or Reverse Shell Attack	
	SHJ	Session hijacking	
	XSRF	Cross-site Request forgery	
	EAC	Exposed Access Credential	

Further, the security level column indicates the test score of the security level of the web application framework in respect to the corresponding vulnerability. The vulnerability levels were rated as follows[26];

i. A security level of 1 means that the system was completely anonymous to vulnerability test and test really proved that vulnerability was not present in the template. This gives no clue to the hacker.
ii. A security level of 2 means that there was an incomplete protection against the vulnerability. The test indicated that the vulnerability was protected in the template, but the system was not completely anonymous. Example, in a vulnerability test, the protection mechanism resulted in giving feedback to user or attacker on the level of protection or revealing system information which turn to be educative to the hacker.
iii. A security level of 3 means that the vulnerability was present.

The vulnerabilities were tested manually by using a blend of Black-box and Gray-box methods[27, 28]. Black-box analysis estimates an application’s security level from the point of view of an external attacker, with no insider knowledge of the application. Gray-box testing was done such that it involves an attacker being considered a user with some system privileges, such as an employee[25, 27-29]. Some vulnerability scoring systems which are commonly used include Common Vulnerability Scoring System (CVSS) [30], Vulnerability Rating and Scoring System (VRSS)[31] and Microsoft security update severity rating [32] but their target users are highly skillful and IT security expert. They include vendors who might be bias in their scoring and IT security experts may be expensive for hiring affordability of SMEs. In view of this, a simpler but effective tool (SS) was developed for scoring the vulnerabilities in this study.

3.1 Methods used in testing for vulnerabilities

Cross-Site Request Forgery (CSRF) is a kind of attack which forces an end user to execute unwanted actions on a web application in which the user successfully authenticated [33]. The basic steps to determine if a target website was vulnerable is to check if that target web page was able to be loaded into an iframe[34]. This was done by determining if that target or victim web page could be loaded into an iframe. This was done by creating a web page with an iframe on WebPOS[34] as follows:

```
<html><head> <title>Clickjack test page</title></head><body><p>Website is vulnerable to
clickjacking!</p><iframe src="http://localhost:85/frameworks/wordpress/wp-
admin/post.php?post=1&action=edit" width="500" height="500"></iframe></body></html>
```

The expected result is that if the message in the paragraph tag within the body tag (Website is vulnerable to clickjacking!) at the top of the page was seen and the target web page was successfully loaded into the frame, then that target site was vulnerable and has no type of protection against Clickjacking attacks[34]. XSS attacks occur when a hacker uses a web application to send bad code, mostly in the form of a browser side script, to a target end user. XSS can occur anywhere a web application receives input from a user without the output or the input being validated or encoded[35]. In this studies, malicious codes were injected into the database of the website using one of the forms of the website. The victim requests a page from that website which included the malicious string from the database. The success of attack of the inject of the malicious inject was then accessed [36]. File upload or Reverse shell attack testing procedure was basically determining if the file size and or type had some limitations during file upload[37]. SQL injection occurs when a hacker sends untrusted codes into an interpreter. The initial step to determine if a website was vulnerable to SQL injection (SQLi) was to inject a single quote at the end of the website url of a page that showed vulnerable[38]. If the page remains anonymous or gives error messages without revealing system information, then the site was not vulnerable to SQL injection. SQL injection can also be done through other user inputs[2, 38]. Conversely, if the page returns error revealing the system information, it is an indication that the website was vulnerable to SQL Injection[2, 38].

Exposed Access Credential occurs when a file containing user credential is able to be executed by the server leading to the exposure of the user credential[25, 39]. A basic test of this weakness was done by running the file containing the user credentials. If the credentials were displayed in plain text, it indicated that the web applicant had no protection against EAC vulnerability[25, 39]. Brute Force Attack consists of an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response[25]. It involves number of tries, with the predicted values. To test for the brute force attack on a login screen, the attacker attempts logging in with several password with a particular user name. If the system disables the account of that user after some number of attempts for instance, it means there was brute force protection mechanism in place [40, 41]. Session Hijacking occurs when an attacker steals victims authenticated session identification and use that to hijack the victim's authenticated system without the knowledge of the victim[42]. This was tested by finding out basically if there was basic session expiration implemented on the website. This was done by logging on to the various pages and being inactive on them for an average of 60 minutes. If resources of those pages could not be accessed and the sessions expire, it means some session protection mechanism was in place [42].

3.2 Security implementation methods on webPOS

To effectively secure web applications against the studied vulnerabilities, a web application(webPOS) was developed. The purpose was to use the system to evaluate a security implementation framework on how the web application could be shielded from the studied attacks. The vulnerabilities which were prevented in the implementation are SQLi, XSS, CSRF, BFA, SHJ, RSA and EAC. OWASP and other standards were adopted[25, 43]. With SQLi, one of the ways used to protect WebPOS against SQLi is by using prepared statement[16]. Further to this, the function, mysqli_real_escape_string(), was used to filter all user inputs. This function escaped special characters in user inputs, by using the current charset of the connection [27, 44]. CSRF was protected by using generated session token. On the user input form a random token was generated into a token session by using the function openssl_random_pseudo_bytes(16)[45]. The generated session token was stored in a hidden input field. During processing, the token submitted by “POST” method was compared with the token stored in the session[45]. If there was a variance, an error would be generated and an alert of Invalid AntiCSRF displayed. For XSS all the user inputs were passed through some filters such as stripslashes() and htmlspecialchars(). These functions convert all special characters into html equivalents of all scripts injected as client data[2]. In the implementation against BFA, the system keeps a log of failed login attempt each time a wrong password was entered against a correct user name[40, 41]. If the password entered was incorrect against that username for 3 times, that user account was disabled. A feedback message was sent to the user in notification about the account being disabled. The user was directed to contact the system administrator. The key point was for the account to be disabled if a wrong password was entered against a correct username for a continues number of times. For SHJ, IP binding and session timeout methods were used to secure the code against session high jacking[42]. At login and during session creation, the remote IP address of the client and time logged were stored in respective sessions. On accessing a page, the stored IP address was compared with the current client accessing the page. If the IPs differed, that client was logged out. Also, if there was an inactivity for more than 30 minutes, in attempt to access the secured page, the session expires and the user was logged out[42]. A secure code was developed by limiting the file type and size[25] against RSA while EAC security measure was implemented by using the file type .php for storing the user credentials[39].

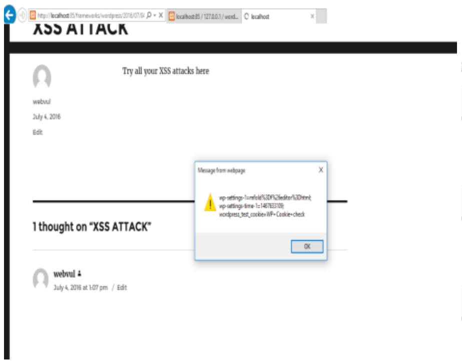
In evaluating usability, SUS scale, which is a Likert scale which results in a single number representing the total measure of the entire usability of a system under study, was used [46]. This was adopted in this study to measure the usability of the custom designed web application (WEBPOS). The score for individual items are not meaningful on their own in SUS study[46]. The SUS score calculation was done by, first summing the score contributions from each question. Each question’s score contribution ranges from 0 to 4. For questions 1, 3, 5, 7 and 9 the score contribution calculation is the scale position minus 1. For question 2, 4, 6, 8 and 10, the contribution is 5 minus the scale position. The sum of the scores were then multiplied by 2.5 to obtain the overall value of SU. The average and standard deviation were then computed to determine the variation[46].

4 Results

The results of the various frameworks (WordPress, Joomla, Moodle and MVC.Net Framework) which were tested is presented. The security test and implementation results of the custom designed web point of Sales System (WEBPOS) and SUS tests were also presented in this section.

4.1 Security Tests and Results on WAFs

In testing for XSS, Iinitially, the script,” <script>Alert (document.cookie);</script>“ was injected through a comment input box of a WordPress template and the script was able to be executed as shown in Figure 2. The attack was further performed to steal the session cookie onto the attacker web server. This was done by injecting the “<script>'http://webposproject.azurewebsites. net/ addEmployeeForm.php ?id='+ document.cookie</script>”, in the comment box of. This was able to steal the user session information to the hacker’s web server ”http://webpos-project.azurewebsites.net/addEmployeeForm.php?id=wp-settings 1=mfold%3Df%2 6editor %3Dhtml; wp-settings-time-1=1467633109; wordpress_test_cookie=WP+Cookie+check” which is indicated in Figure 2. This led to a score of a vulnerability level of 3 as indicated in Table 3. Each of the outlined vulnerabilities (Vul. ID) in Table 2 was assessed subsequently in each of the WAFs in Table 2. The vulnerability scores were then recorded in Table 3 in accordance with the scoring which was specified in section3.



XSS ATTACK

Try all your XSS attacks here

webvul
July 4, 2016
Edit

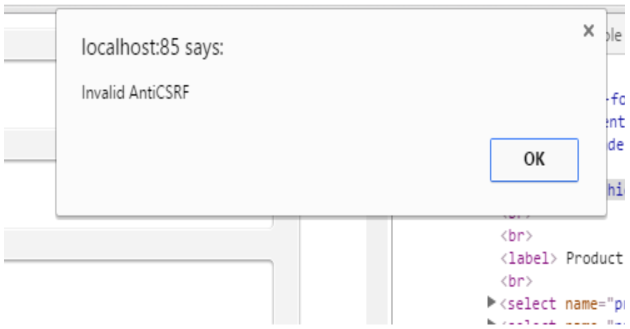
Message from webpage

wp-settings-1=mfold%3Df%26editor%3Dhtml; wp-settings-time-1=1467633109; wordpress_test_cookie=WP+Cookie+check

1 thought on "XSS ATTACK"

webvul
July 4, 2016 at 1:07 pm / Edit

Figure 2: Injection of script on comment box on WordPress Site.



localhost:85 says:

Invalid AntiCSRF

```
<br>
<label> Product
<br>
<select name="p
```

Figure 3: Resilience against CSRF

Table 3: Tabulated Vulnerability Scores of WAFs

	.Net	Joomla!	Moodle	WebPOS	WordPress	All
BFA	3	3	3	1	3	13
EAC	2	1	1	1	1	6
RSA	3	1	1	1	1	7
SHJ	3	1	1	1	3	9
SQLi	2	1	1	1	1	6
XSRF	2	3	2	1	2	10
XSS	2	1	1	1	3	8
All	17	11	10	7	14	59

With reference to Table 3, WebPOS registered the lowest vulnerability cumulative sum of 7 while .Net framework registered the highest cumulative value of 17.

The WebPOS development included pages for system settings, creating and adding products, updating a sales transaction page, user login and a report center. The system also has interfaces for user accounts creation and login. In the WebPOS application, the major focus was to implement security framework against the vulnerabilities investigated in this study. The security implementation framework against the various vulnerabilities are outlined below:

- The code below provided protection against XSS.

  $partname=stripslashes($partname);
  $partname=mysqli_real_escape_string($db,$partname);$partname= tmlspecialchars($partname);
  $partname = trim($partname);
  $partnumber = stripslashes($partnumber);

- Security implementation against brute force attack:

  In this code, the user is disabled if the wrong password was submitted to the database against a correct user name for more than three times.
  if($attempt>3){ $stmt=$sqlcon->prepare(**"UPDATE WebPOS_user SET userStatus=0 WHERE username=? "**);

$stmt->bind_param(**'s'**,$username);
$stmt->execute(); $error= **"Your user account is disabled! Contact Admin"**.;}

- Ssecurity Implementation Against SQLi

Real escape string function and prepared statement were used to protect against SQLi. The real_escape_string function was used to escape malicious input characters while the prepared statement was used to bind parameters or place holders for receiving user input data. Once there is bonding, the input data is placed into an already constructed SQL statement and sent to the database to be processed[47] as shown in the code below;
$discount=mysqli_real_escape_string($db,$discount);
$discount = htmlspecialchars($discount);
$discount = trim($discount);
$msg=**""**;
//prepare statement
if ($stmt = $sqlcon->prepare(**"SELECT discountID FROM discount where percentDiscountRate LIKE ?"**)) {
$stmt->bind_param(**'i'**, $discount);

- Security implementation against File upload or Reverse Shell Attack

As shown in the code below, the file size and type were restrict to prevent RSA.
if((strtolower($uploaded_ext) == **"jpg"** || strtolower($uploaded_ext) == **"jpeg"** || strtolower($uploaded_ext) == **"png"**) &&($uploaded_size < 100000) && getimagesize($uploaded_tmp)) {
else $msg = **"Your image was not uploaded. You can only upload JPEG ,PNG images and Max. file size of 100KB"**}

- Security implementation against Session hijacking

With reference to the code below, IP binding and session time out were implemented to protect this weakness as shown below;$ip=$_SESSION[**"ip"**]; $timeout=$_SESSION [**"timeout"**]; **if** (!($ip==$_SERVER[**'REMOTE_ADDR'**])){ header(**"location: logout.php"**); *// Redirecting To Other Page*} **if**($_SESSION [**"timeout"**]+1800 < time()){ *//session timed out* header(**"location: logout.php"**); *// Redirecting To Other Page* }**else**{ *//reset session time* _SESSION[**'timeout'**]=time();}

- Security implementation against Cross-site Request forgery

The function openssl_random_pseudo_bytes(16) and session time out were used to implement this security measure.

- *Security implementation against Exposed Access Credential*

A file type of .php was given to the access credentials page labeled connect and this was executed but details of the credentials were not exposed.

4.2 Security Test Results on WebPOS

After the security implementations in section 3.2, the custom developed web application (WebPOS) was tested against the outlined vulnerabilities. This was done to determine if indeed the security measures were efficient and effective enough to shield the system against the vulnerabilities.

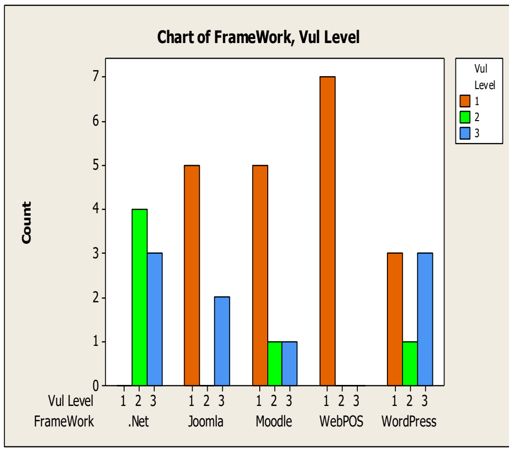


Figure 4: Chart of Frameworks, Vul. Level

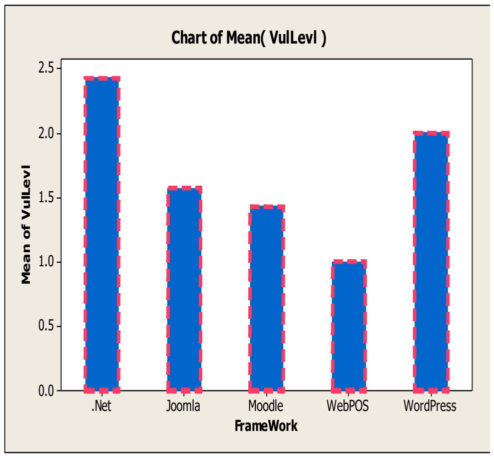


Figure 5: Chart of Mean of vul. score

All the tests for the outlined vulnerabilities were done in line with their respective methods defined under subsection 3.1 and the results recorded in Table 3. For instance, for security test of XSS on WebPOS, a script was injected in a product description field but the output of the test resulted in a disruption of the script as it was encoded as shown under “Product Description:” in figure 6.

From Figure 4, WebPOS registered the highest number of low vulnerability score of 1 followed by Joomla! and Moodle. From the chart of mean, Microsoft .Net framework had the highest mean score while WebPOS had the lowest as shown in Figure 5. In the usability study, the SUS test of the average of the ten users, the overall value representing the SU was 72.5 +/-22.1.

5 Discussion

The main objective of this research was to ascertain if some vulnerabilities exist in WAFs which could be driving the lead in web-based systems attack. The intention was to provide the vulnerabilities level in WAFs to users to serve as a guide in their choice of usage. The approach was guided by OWASP and other standards. The purpose of the security implementation framework in the WebPOS was to demonstrate how the vulnerabilities could be shielded in implemented systems. The WebPOS also served as an attack tool for testing the vulnerabilities in the WAFs. From the summary of the statistical results in Table 3, all the WAFs had some vulnerabilities at varying level of protection. Security implementation against the various vulnerabilities were implemented and a test result on the implemented system (WebPOS) resulted in the lowest vulnerability score. The implemented system also passed an SUS test with a score of 72.5 +/-22.1.

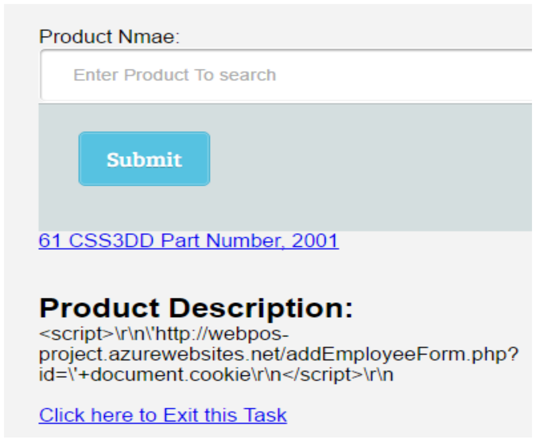
Among the four WAFs (.Net, Moodle, Joomla and Wordpress) studied, C# MVC .Net Framework (.Net) had the highest cumulative vulnerability score count of 17. The .Net framework did not obtain a lower score of 1 for any of the vulnerabilities tested. This implies that there was no complete protection against any of the vulnerabilities tested. Aside brute force attack, reverse shell attack and session hijacking vulnerabilities, the C# MVC .Net framework had a midway (level 2) security protection against EAC, SQLi, CSRF and XSS vulnerabilities as shown in Table 3. The high vulnerability score of the .Net framwork was registered due to the verboseness of the error messages in the implementation of the security measures which could turn to be educative to the attacker[4, 16]. For instance, with reference to the Table 3, all the frameworks studied had complete protection for EAC but the .Net framework had incomplete protection due to the educative error messages which could aid secondary attack[4, 16]. OWASP has noted that, hackers are most interested in error messages since they might learn from some information leaked through error messages that could lead to further attacks[48]. Acunetix also disclosed that, error messages can contain sensitive information such as the location of the file that produced the unhandled exception[49]. Furthermore, an attacker can use the information returned by error messages to deduce the technologies used in Web applications, determine if an attempted attack was successful and to gather hints to explore for future attacks [50]. In this regard, sensitive information ranging from web application version to physical file locations were disclosed in the error messages which lead to each of those vulnerabilities scoring vulnerability levels of 2. So, the .Net framework results was the lest secure framework largely because of its unhealthy error messages.

Conversely, Moodle had the lowest cumulative vulnerability score count of 10 among the four WAFs studied as indicated in Table 3 and Figure 4. The test result have complete protection for EAC, RSA, SHJ,SQLi and XSS. For instance, in a file upload attack or RSA, an incorrect file type was uploaded onto Moodle site, but the file was not accepted to be uploaded and an error message “Filetype cannot be accepted”, was returned to the attacker. In this case, by reading the error message, the attacker would have no clue of resource location, version of application or any information that could enhance the attack. This led to a low vulnerability score of 1 as shown in Table 3.

EAC and SQLi vulnerabilities had security measures implemented against them in all the frameworks. EAC is a serious vulnerability since access to the credential can enable hackers to take total control of the entire system. Potential problems arise when this file is somewhere within document root[39]. Usually, web servers render the content of a file in plain text if they are unable to work out the file type or the extension[39]. Such files containing user access credentials are displayed in plain text and this exposes the user credentials. The serious nature of issues of EAC could have been the reason why all the frameworks hd a security protection mechanism in their default settings against the EAC vulnerability.

SQLi is one of the OWASP top ten application vulnerabilities[51]. The high threat of this vulnerability is the reason why all the frameworks had protection mechanisms implemented in their default settings against SQLi. SQLi attack is done by injecting malicious codes onto user input space or appending it onto the url of the resource[2, 50, 52]. These malicious codes take advantage of vulnerabilities in SQL code structure. SQL injection of these codes have the potential of causing harm such as data deletion, updating, inserting, unauthorized access and many more[2, 50, 52]. SQL case insensitivity has become a dream for hackers since it offers flexibility to craft codes for attacks[2, 50, 52] but a nightmare for security defenses as it is challenging to implement blacklisting defense. Precedence of evaluation of logical operators is another possible point of attack through SQL injection[4, 16]. In an operation containing NOT, AND, OR, the NOT operator is evaluated first followed by AND operator and the OR operator. In the logical operation, the OR operator is always evaluated last and so the attacker creates a malicious code on existing dynamic SQL code to retrieve data from the system. A code of OR 1=1 injected onto the login screen could have bypassed the login credentials and compel the query to respond positively since 1=1 is always True[4, 16, 47].

In the WebPOS, the product page which had most of the security implementations basically enables a user to enter product type, product name, product classification or category and other related attributes. The product is then submitted to the database and can be accessed for sale if required through the sale screen. One of the reasons for using PHP in the WebPOS implementation, was because the majority of the WAFs tested were developed with PHP so the framework could be representative enough. With regards to the defensive programming, prepared statement is known to be used for shielding against injection attacks[26, 47]. In prepared statements, SQL statements are predefined, so the user input is independent on the SQL statement parsing [4, 16, 26, 47]. Therefore, the user data is treated as data alone and does not affect the SQL statement (SQLi) or code structure. As a result, the SQL injections, are not processed by the database in a malicious manner [4, 16, 26, 47]. Additionally, quote escape functions were used to prevent SQLi[26]. Such functions which are already in built into systems such as real_escape_string in PHP for MySql, were used for mitigation. Developers can create their own quote escaping functions to protect in systems such as MicroSoftSQL Server and Oracle[26] systems which do not have such inbuilt functions.


Product Nmae:

Enter Product To search

Submit

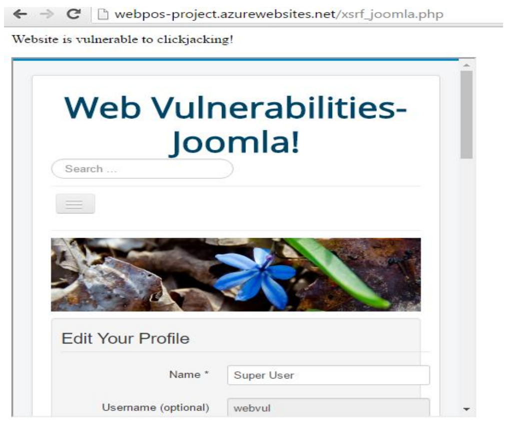
61 CSS3DD Part Number, 2001

Product Description:

<script>\r\n\'http://webpos-project.azurewebsites.net/addEmployeeForm.php?id=\'+document.cookie\r\n</script>\r\n

Click here to Exit this Task

Figure 6: XSS test Result on WebPOSS


webpos-project.azurewebsites.net/xsrf_joomla.php

Website is vulnerable to clickjacking!

Web Vulnerabilities-Joomla!

Search ...

Edit Your Profile

Name * Super User

Username (optional) webvul

Figure 7 Test for CSRF in Joomla!

IP binding and session timeout methods were used to secure the code against session high jacking. At login and during session creation, the remote IP address of the client and time logged-in were stored in respective sessions[42]. On accessing the page frmAddproducts.php, the stored IP address was compared with the current client's IP accessing the page. If the ip differs, that client was logged out[42, 53]. Also, if there is an inactivity for more than 30 minutes, in attempt to access the secured page, the session would expire and the user would be logged out[42, 53]. CSRF was protected by using generated session token[33]. On the user input form a random token was being generated into a token session by using the function openssl_random_pseudo_bytes(16)[54]. During processing, the token submitted by POST method was compared with the token stored in the session. If there was any variance, an alert message was generated and an alert of Invalid AntiCSRF displayed as shown in Figure 3.

In protecting against EAC, a file type of .php was used to store the system access credentials. Once a file type was provided at run time, the system does not display the access credentials in plain text[50]. Basic file type and size restriction were used to protect against RSA[37]. Usually, the major aim of RSA is to upload some scripts or codes to the system and also finding a way to get the code executed[37]. If a shell script is uploaded, the attacker can navigate through the url and browse the server[37]. The result of file upload attack include complete system takeover, defacement, overloaded file system or database and forwarding attacks to back-end systems[37]. File size and type restriction method prevents upload of scripts and large file size or type that could have advert effect on the system [37].

The WebPOS was also evaluated for the vulnerabilities using the same vulnerability test method as outlined in Table 2. This provided a common denominator for comparison of the security test results since a common method was used to test both the WAFs and the WebPOS. The results indicated that the WebPOS had the lowest vulnerability score for all the vulnerabilities tested and this resulted in a cumulative vulnerability score of 7. For instance, in testing for CSRF, Joomla! Web page was able to be loaded onto the iframe and also displayed the message "website is vulnerable to clickjacking" as shown in Figure 7. This attracted a vulnerability score of 3 since the test indicated no protection for the CSRF vulnerability. However, the webPOS page detected the malice and displayed the message "Invalid antiCSRF" as shown in figure 3. This attracted a vulnerability score of 1.

Furthermore, the SUS result, was 72.5 +/-22.1 meaning that the, custom-built WebPOS, was at least usable at SU of 50.3 up to 94.6. Generally, custom built systems are considered less usable in comparison with WAFs. For instance, a study of The Wayne State University (WSU) College of Engineering (COE) Content Management System was deemed to be unusable in comparison with Joomla WAF used to design other similar systems. A change was proposed to use Joomla! instead of the custom-build. Because Joomla offered better usability and maintain ease of administration. Joomla also consists of an array of templates in an organized fashion that facilitates easy navigation, an extension system to offer new features. It also consists of a user management system to administer user privileges and other several usability features [55]. It was noted that WAFs bring in open templates and plugins enabling the look and feel of the site and features to be altered, added, or removed easily. A lot of issues were associated with the custom-built CMS. Scalability, unavailability of programmers, insufficient documentation of custom-built systems, inconsistent navigations, hardcoded database leading to difficulties in testing for new features are some of the issues which were associated with the custom-built system[55]. Conversely, open-source CMS provided complete documentation of the source code and allowed multiple developers to make improvements and bug fixes. In open source, the WAF'S original source code was available to view, download, compile, and change as the user wants for free. An open-source CMS uses a plugin architecture which allows additional features to be created as and when they are needed[55]. An ideal WAF is powerful enough to handle hundreds of users with thousands of webpages while being easy enough for users to navigate, edit, and administer. WAFs also employs Lightweight Directory Access Protocol (LDAP) for authentication of servers[55]. User access control are included and able to restrict access to specific pages[55]. This suggests that there are more benefits in terms of usability in open source CMS but not in terms of security as shown in this study.

The time spent in downloading and installing a WAF was about just an hour. Basically, MySQL database was downloaded and installed followed by WAFs. However, the development of WebPOS as at the stage it had reached (from login to adding product in the system) took about four weeks consisting of 20 working days. 8 hours were spent on each of these days leading to 160 hours. This implies that in comparison between configuration of a WAF verses developing a custom application costs an extra 159 hours. Therefore, there is a huge cost savings in developing application using WAF as compared to custom built system development from scratch.

What is also quite essential in this study is the adoption of a simple but effective web vulnerability scoring scheme (SS). SS classifies or rates web vulnerabilities based on the depth of penetration of an attack or malicious injects. A complete penetration of an attack in this study had a score value of 3, incomplete attracted a score of 2 and none- penetration or complete resistance to the attack led to a score value of 1. Some vulnerability scoring systems which are commonly used include Common Vulnerability Scoring System (CVSS) [30], Vulnerability Rating and Scoring System (VRSS)[31] and Microsoft security update severity rating [32] but their target users are security experts and highly skillful IT personnel [3][4]. They include vendors who might be bias in their scoring and IT security experts may be expensive for hiring affordability of SMEs. These commonly used vulnerability scoring systems target user groups are Software Application Vendors, Vulnerability Scanning and Management, Security (Risk) Management. The user groups excludes SMEs you are not security experts[3][4]. In view of this, a simpler but effective tool (SS) was developed for scoring the vulnerabilities in this study. This would afford low skilled IT security practitioners, who may be associated with SMEs, to be able to use the SS for vulnerability related analysis.

The study was tested on default settings of the WAFs since novice might lack the technical capabilities to configure the security settings on them prior to their usage. So even if there was a security provision for any of the vulnerabilities which was not implemented in the default settings, the security test would miss it. Further, an SUS study of WAFs would have given an absolute comparison and evaluation in terms of usability. In fact, the methodology of the analysis of usability of WebPOS representing custom-built system differs from that of the WAFs. In the WebPOS, the qualitative data of the user perception of the system was quantified by using the SUS method which represented a single measure. But the qualitative data of the user experience studied in the WAFs [55] was not quantified. Despites this, the comparison still holds because certain shortfalls of the custom-built systems are unarguable. For instance, in a custom-designed system (WebPOS), if the programmer was not available and documentation of the application was not sufficient, it implies that scalability and maintainability of the code would become a challenge. Whereas in the case of WAFs, multiple developers and documentation exist, leading to efficient scalability and maintainability. WAFs are used by numerous of users and so, the feedback of the users were used to improve upon the systems which resulted in high usability[6].

Conclusion

Due to the high rate of usage of web application frameworks and the high rate of increased in data breaches through attacks on web applications, it becomes necessary to investigate into the most commonly used WAFs for the presence of the most common web vulnerabilities. The broad objective was to determine if web vulnerabilities were present in the default WAFs which were driving the lead in frequency of attack. WordPress, Moodle, Joomla and MVC C# .Net Framework, were investigated. The vulnerabilities which were studied are SQLi, XSS, File upload, Cross-site Request Forgery CSRF and Brute Force Attack. In the result, Moodle showed less vulnerabilities and proved to be more secure among the four WAFs studied aside other factors. Essentially, there were some vulnerabilities associated with the web templates studied. These vulnerabilities had the potentials to be used to gain ingress into systems. A Point of Sales system (WebPOS) was then developed and used for implementing a security framework against the tested vulnerabilities. With these findings, SMEs and other users would know the vulnerabilities associated with the various templates. The vulnerability levels which were discovered in the WAFs would provide knowleged for users in their choice of WAFs. The implementation framework for mitigation would also guide users to implement the appropriate sanitization for related vulnerabilities if they desire to use a WAFs which has these vulnerabilities. Such related research results would reduce the numerous vulnerabilities and attacks of systems through web applications.

Finally, it would be a great addition to the body of knowledge and an interesting piece to expand the study to determine the cost benefit analysis of installing and securing a WAF verses building and securing custom-built system. Evaluation in terms of performance between custom built systems and WAF could also be conducted in the future since the scope of this project did not cover that. Further to this, there still exists other commonly used web application systems such as Drupal and more recent vulnerabilities such as Server-Side Template Injection (SSTI). These common frameworks and recent vulnerabilities would be highly considered in further works.

6 Reference

[1] Semantec, Internet Security Threat Report (ISTR) 2019 | Symantec. 2019.

[2] Rahul, J. and S. Pankaj. A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection. in Proceedings of the 2012 International Conference on Communication Systems and Network Technologies. 2012. IEEE Computer Society.

[3] Warkentin, M. and R. Willison, Behavioral and policy issues in information systems security: the insider threat | SpringerLink. 2009.

[4] Weinberger, J., P. Saxena, D. Akhawe, et al., A systematic analysis of XSS sanitization in web application frameworks, in Proceedings of the 16th European conference on Research in computer security. 2011, Springer-Verlag: Leuven, Belgium. p. 150-171.

[5] Bitnami, Packaged Applications for Any Platform. 2016.

[6] Bitnami. Bitnami for XAMPP Application Modules. 2018; Available from: https://bitnami.com/stack/xampp?utm_source=bitnami&utm_medium=installer&utm_campaign=XAMPP%2BInstaller.

[7] Bioko B., Content Management Bible (Hungry Minds. 2003. 26(11): p. 35-70.

[8] Allwebco. Free templates included with hosting. 2016; Available from: http://www.allwebco-templates.com/support/S_free_templates.htm.

[9] Heckman, J.M. and E.J. Glantz, Web content management: a collaborative approach. . Information Processing & Management, 2003. 39(4): p. 667-668.

[10] Haug, A., The implementation of enterprise content management systems in SMEs. Journal of Enterprise Information Management, 2012. 25(4): p. 349-372.

[11] Yii. The Definitive Guide to Yii 1.1. Security 2019 [cited 2019 18.08.2019]; Available from: https://www.yiiframework.com/doc/guide/1.1/en/topics.security.

[12] OECD. OECD SME and Entrepreneurship Outlook 2019 | READ online. 2019; Available from: https://read.oecd-ilibrary.org/industry-and-services/oecd-sme-and-entrepreneurship-outlook-2019_34907e9c-en.

[13] Zec, M. and M. Kajtazi. Examining how IT Professionals in SMEs Take Decisions About Implementing Cyber Security Strategy. in Proceedings of the 9th European Conference on IS Management and Evaluation 2015. Academic Conferences Limited.

[14] Utomo, H. and M. Dodgson, Contributing Factors to the Diffusion of IT within Small and Medium-Sized Firms in Indonesia. Journal of Global Information Technology Management, 2001. 4(2): p. 22-37.

[15] Gäre, K. and U. Melin, SMEs need formative infrastructure for business transformation. Journal of Enterprise Information Management, 2011. 24(6): p. 520-533.

[16] De Meo, F., M. Rocchetto, and L. Viganò, Formal Analysis of Vulnerabilities of Web Applications Based on SQL Injection (Extended Version). 2016.

[17] von Oheimb, D. and S. Mödersheim. ASLan++ — A Formal Security Specification Language for Distributed Systems. 2012. Berlin, Heidelberg: Springer Berlin Heidelberg.

[18] Armando, A., W. Arsac, T. Avanesov, et al. The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures. 2012. Berlin, Heidelberg: Springer Berlin Heidelberg.

[19] OWASP. OWASP Top 10 Application Security Risks - 2017. 2019 27-03-2018 [cited 2019 17-08-2019]; Available from: https://www.owasp.org/index.php/Top_10-2017_Top_10.

[20] Saxena, P., D. Molnar, and B. Livshits, SCRIPTGARD: Preventing Script Injection Attacks in Legacy Web Applications with Automatic Sanitization, in ACM Conference on Computer and Communications Security. 2010: Chicago, Chicago, IL, USA.

[21] WordPress. WordPress is Secure. 2018 2018-03-28 [cited 2019 01.09.2019]; Available from: https://wordpress.org/about/security/.

[22] Joomla! Secure coding guidelines - Joomla! Documentation. 2019 [cited 2019 01.09.2019]; Available from: https://docs.joomla.org/Secure_coding_guidelines.

[23] Moodle. Security overview report - MoodleDocs. 2016 29 June 2016]; Available from: https://docs.moodle.org/31/en/Security_overview_report.

[24] OWASP. PHP Security Cheat Sheet - OWASP. 2018; Available from: https://www.owasp.org/index.php/PHP_Security_Cheat_Sheet.

[25] OWSAP, Category:OWASP Top Ten Project - OWASP. 2013.

[26] OWASP, OWASP Backend Security Project PHP Security Programming - OWASP. 2018.

[27] Petukhov, A. and D. Kozlov, Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing 2008.

[28] Auronen, L., Tool-Based Approach to Assessing Web Application Security. 2002.

[29] Verison, 2016 Data Breach Investigations Report | Verizon Enterprise Solutions. 2016.

[30] Mell;, P. and K. Scarfone;, Improving the Common Vulnerability Scoring System. IET Information Security, 2007. 1(3): p. 119 – 127.

[31] Liu, Q. and Y. Zhang, VRSS: A new system for rating and scoring vulnerabilities. Computer Communications, 2011. 34(3): p. 264-273.

[32] Microsoft. Security Update Severity Rating System. 2019; Available from: https://www.microsoft.com/en-us/msrc/security-update-severity-rating-system.

[33] OWASP. Cross-Site Request Forgery (CSRF) - OWASP. 2018 [cited 2016 01 August]; Available from: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[34] OWSAP, Cross-Site Request Forgery (CSRF) - OWASP. 2016.

[35] OWSAP, Cross-site Scripting (XSS) - OWASP. 2016.

[36] Jakob Kallin, I.L.V. Excess XSS: A comprehensive tutorial on cross-site scripting. 2016; Available from: https://excess-xss.com/.

[37] OWASP. Protect FileUpload Against Malicious File - OWASP. 2018; Available from: https://www.owasp.org/index.php/Protect_FileUpload_Against_Malicious_File.

[38] Vanan, M.a.S., M.V.,, Information Gathering. 2016.

[39] Shiflett, C., Essential PHP security. 2016.

[40] OWASP. Brute force attack - OWASP. 2016; Available from: https://www.owasp.org/index.php/Brute_force_attack.

[41] OWASP. Blocking Brute Force Attacks - OWASP. 2018; Available from: https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks.

[42] Burgers, W., R. Verdult, and M. van Eekelen. Prevent Session Hijacking by Binding the Session to the Cryptographic Network Credentials. 2013. Berlin, Heidelberg: Springer Berlin Heidelberg.

[43] Pietraszek, T. and C.V. Berghe, Defending against Injection Attacks through Context-Sensitive String Evaluation. 2005.

[44] PHP Group. License Information and Time Related Extensions. 2001 [cited 2016 15 August]; Available from: https://secure.php.net/license/.

[45] Rochkind, M., Security, Forms, and Error Handling, in Expert PHP and MySQL: Application Design and Development. 2013, Apress: Berkeley, CA. p. 191-247.

[46] Holt, O.B.P., Usability & Ux Evaluation: Methods & Techniques. 2016.

[47] Petukhov, A. and D. Kozlov, Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing. 2008.

[48] OWASP, Error Handling, Auditing and Logging - OWASP. 2018.

[49] Acunetix. Application error message - Vulnerabilities - Acunetix. 2018; Available from: https://www.acunetix.com/vulnerabilities/web/application-error-message.

[50] EC-Council, Ethical Hacking and Countermeasures: Web Applications and Data Servers. 2009.

[51] OWASP. Category:OWASP Top Ten Project. 2018.

[52] informationtreasure, Hacking Websites Using SQL Injection Manually. 2014.

[53] OWSAP, Session hijacking attack - OWASP. 2014.

[54] OWASP, Time and Randomness Management Library - OWASP. 2013.

[55] Martens, B. and A. George, Wayne State University College of Engineering Content Management System and Joomla! CMS. 2009.