

Verification and Validation of an Open Source Product

Part – I Static Analysis

Wenguang Li

P-number: 9002229699

E-mail: weili15@student.bth.se

Sai Priyatham Dongoor

P-number: 9406173337

E-mail: sado16@student.bth.se

Sai Mohan Harsha Kota

P-number: 9409131514

E-mail: sakf16@student.bth.se

Akshay Kumar Jilla

P-number: 9508109015

E-mail: akji16@student.bth.se

Panchajanya Varanasi

P-number: 9302205613

E-mail: pava15@student.bth.se

Abstract: *This paper provides a detailed description of static and dynamic code analysis of an open source community plugin for WordPress – BuddyPress. This paper is divided into two parts. Part I contains the static analysis reports of this open source plugin using an open source static analysis tool RIPS. Part II contains the dynamic analysis reports of this plugin, which is described in detail in further sections.*

Keywords: Static analysis, dynamic analysis, testing, vulnerabilities, open source software.

I INTRODUCTION

Since its appearance in the early 1990s, the World Wide Web evolved from a platform to access text and other media to a framework for running complex web applications [1]. However, web applications have been plagued with a lot of vulnerabilities and security problems. With increasing popularity of open source software, everybody is accessible to the code. The intent is to improve software by giving everyone an opportunity to modify the code. But we cannot always ensure that the modified code is secure because there may be attackers who might intentionally modify the code for carrying out malicious attacks on all the users of the

software. So we need to examine the code for security vulnerabilities. In this context, we have examined an open source plugin called BuddyPress both statically and dynamically to analyze its security.

BuddyPress is an open source plugin for WordPress that provides social networking platform for sites [2]. It has a number of features that websites can take advantage. It is mostly written using PHP and MySQL.

Part I is further divided into sub sections in which we explain our inspection goals, Static analysis tools, results. We also try to evaluate the usefulness of static code analysis.

II INSPECTION GOALS

Although a large research effort on web application security has been going for more than a decade, the security of web applications continues to be a challenging problem. An important part of this problem derives from vulnerable source code often written in languages like PHP [1]. With increasing demand for dynamically adaptive web applications more and more complex code is being written but not without security related vulnerabilities. Static code analysis tools are a solution to find vulnerabilities. Our primary inspection goal is to detect the security related vulnerabilities associated with BuddyPress plugin. The security

vulnerabilities related to PHP applications of this type can be classified into two types namely server side vulnerabilities and client side vulnerabilities. The table 1 describes server side vulnerabilities and table 2 describes client side vulnerabilities respectively.

Possible Attack	Description
Command Injection	It is an attack on the host operating system via a vulnerable application. Command injection is possible when application passes unsafe data (cookies, etc.) to a system shell.
File Disclosure	An attacker might read local files with this vulnerability.
File Manipulation	An attacker might write to arbitrary files or inject code into file with this vulnerability in the application
PHP object Injection	It is an application level vulnerability that could allow an attacker to perform different kind's malicious attacks like Code Injection, SQL Injection, and Application Denial of Service.
SQL Injection	An attacker might execute arbitrary SQL commands on the database server with this vulnerability. An attacker can inject his own SQL queries thus manipulating the database completely.
Header Injection	An attacker can exploit this issue by enticing an unsuspecting user to follow a malicious URI
XPath Injection	Similar to SQL Injection, XPath Injection attacks occur when an application uses user-supplied information to construct an XPath query for XML data.
Denial of Service	An attacker might gain access to make arbitrary HTTP requests which can be used to exhaust server resources.
Sensitive Sink	A sensitive sink is every construct/function that can cause a vulnerability when tainted data is given as a parameter.

Table 1 Possible Server side Attacks [3].

Possible Attack	Description
Cross Site Scripting (XSS)	Cross-Site Scripting are a type of injection, in which malicious scripts are injected into otherwise trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a client side script to a different end user.
HTTP Response Splitting	HTTP response splitting is a means to an end, not an end in itself. An attacker passes malicious data to a vulnerable application and the application includes data in an HTTP response header.
Session Hijacking	Session Hijacking consists of the exploitation of the web session control mechanism, which is normally managed using a session token. The attack compromises the session token by stealing a valid token to gain unauthorized access to the web server.

Table 2 Possible Client side Attacks [3].

Why security is our primary goal?

Our primary goal is to check for the above mentioned security vulnerabilities because web applications are the becoming susceptible to malicious attacks these days which result in loss of sensitive data and might also compromise the privacy of the user. Since BuddyPress is an open source plugin which is extensively used by many people around the world, it should be secure. So we have decided our primary goal is to detect security vulnerabilities.

Our secondary goal is check for size of the code and duplicated code because memory constraints continue to be a major problem and also duplicated code is a waste of resources.

III STATIC CODE ANALYSIS TOOL SELECTION

Static code analysis tools find the vulnerabilities in the code without executing the code. We have to observe and understand the working of various static code analysis tools. Open source static code analysis tools for PHP are limited. We got the tools by searching the google search engine for static analysis tools for PHP. We chose the appropriate tool which enables us to meet our inspection goals. We have shortlisted these 7 open source tools. Out of these 7 tools

we have chosen RIPS tool as given below because it satisfies both our primary goal of finding security vulnerabilities and secondary goal of checking for duplicated code and code size. The table 3 shows the evaluation table for the static analysis tools.

We have eliminated some tools like Klockwork, Fortify, Checkmarx static code analyzer and PHP Storm from initial study because they were not open source and are commercial tools.

S.no	Static Code Analysis Tool	Primary Goal (Security Vulnerabilities)	Secondary Goal (Duplicated Code and code size)	Decision Taken
1.	PHP Mess Detector	No	Yes	Rejected
2.	PHP code Sniffer	No	No	Rejected
3.	Visual Code Grepper	Yes	No	Rejected
4.	PHP Metrics	No	Yes	Rejected
5.	PHP LOC	No	Yes	Rejected
6.	DevBug	Yes	No	Rejected
7.	RIPS	Yes	Yes	Accepted

Table 3 Evaluation Table for Static code analysis Tools

A brief overview of the RIPS tool [9] we selected is given in the table below:

Name of The Tool	RIPS
Version	0.55
Supported Languages	PHP
License	General Public License
Special Features	Over 100 test cases, Detection of Backdoors
Written in	PHP

Table 4 General Information about RIPS Static code analysis tool

We have used the data from the official websites of these tools to find out the features of each tool and it helped us to compare the tools and generate this evaluation table 3. Since RIPS satisfies our inspection goals, we have chosen RIPS as a static code analysis tool for this evaluation.

In the next section we describe the procedure for running the static analysis tool. We also displayed the results with a valid interpretation and motivation. We have also drawn certain conclusions basing on the results. Screenshots of the working of the RIPS tool were also included after conclusion.

IV RUNNING THE STATIC CODE ANALYSIS TOOL

The step by step process is given below:

1. We need a server to run the RIPS tool. We have used XAMPP software to turn the system into a localhost.
2. We have then started RIPS tool.
3. After starting the RIPS tool, we need to specify the path to the source code we want to analyze. We have set it to the BuddyPress source code.
4. After selecting the path we need to select the verbosity level from five different levels.
5. We have chosen 'user tainted mode' as our verbosity level because we wanted to find the vulnerabilities during user input from the malicious user point of view. Then the tool would detect more vulnerabilities. Tainted data is nothing but the data that comes from outside the script.
6. After selecting our level, we have to choose the vulnerabilities that we wish to find out. We have run the static analysis to uncover both server side vulnerabilities and also client side vulnerabilities.
7. Finally, we can click on scan to run the static code analysis tool.
8. After scanning all the source code files, the tool gives a statistical report of the findings.
9. We then need to interpret the results to find out whether to detected vulnerability is a true positive or false positive.
10. All the authors have followed this process and analyzed the outcomes individually.
11. After that the authors have discussed about the vulnerabilities and formulated a modified report.

The individual reports and group report were included in an HTML file.

The following table 5 shows the time taken by each author to give his individual report on the detected vulnerabilities.

Author	Vulnerabilities detected	Inspection Time (in hrs.)
Author 1	81	3
Author 2	81	3
Author 3	81	2
Author 4	81	2
Author 5	81	2

Table 5 Time log

V REPORTS AND INTERPRETATION

After running the static code analysis tool, the authors have analyzed the warnings. Based on the review the authors classified the vulnerabilities into 3 categories namely true positive, false positive and uncertain.

1. True positive – The warnings that are really true.
2. False positive – The violations that were incorrectly named as warnings. These are not true.
3. Uncertain – The warnings for which the authors were not able to come to a decision. These can be either true of false.

The following chart shows the distribution of the vulnerabilities found.

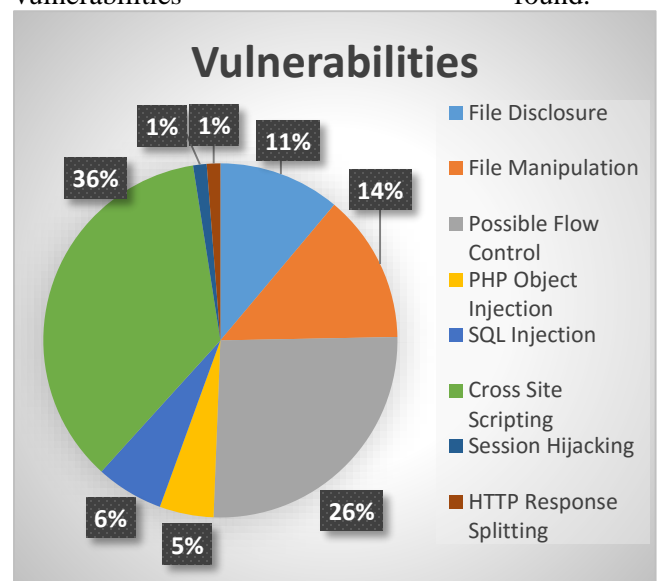


Figure 1 Distribution of Vulnerabilities

After preparing the individual assessment report, the authors have discussed within the group and prepared a group report. The final results were provided in the graph below.

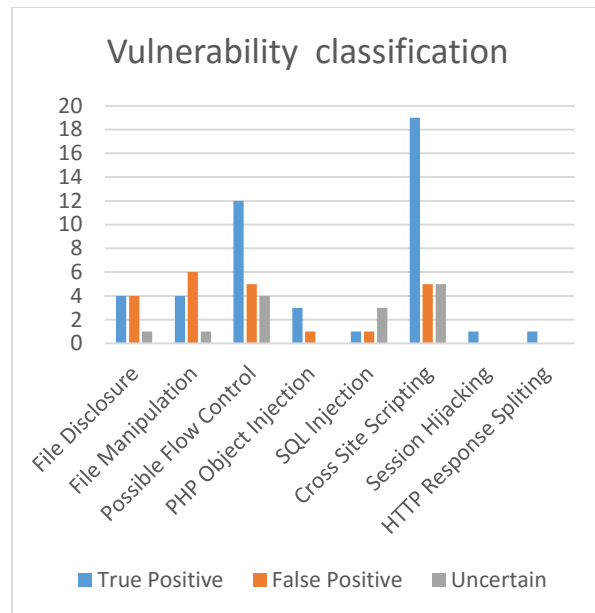


Figure 2 Vulnerability Classification after consensus

Out of 81 vulnerabilities reported, 45 of the vulnerabilities were considered as true positive, 22 vulnerabilities were considered as false positives and 14 were considered uncertain.

The following table 6 describes the overall results after consensus.

Errors Classified as	Count
True Positive	45
False Positive	22
Uncertain	14
Total	81

Table 6 Classification of Vulnerabilities

Among the vulnerabilities reported, Cross Site Scripting vulnerability was reported many times, followed by Control flow errors, File manipulation, File disclosure, SQL Injection and PHP object Injection. Session Hijacking and HTTP Response splitting were least reported by the static analysis tool. Some of the errors were not considered as they were repetitive.

VI CONCLUSIONS

Upon running the static code analysis, the following conclusions were drawn by the authors.

- A. Was the tool helpful in achieving our goals?

We have used RIPS tool among from others tools listed above because RIPS would detect the most commonly exploitable vulnerabilities in web applications. Many vulnerabilities were detected but most of them are false positives mainly because the tool RIPS was also written in PHP. And also the graphs and reports were prepared using excel. Overall the tool is good but it could have been more accurate and it could also have the option to export the reports.

- B. Degree of Agreement upon comparing individual answers.

The authors were mostly in agreement but there were some exceptions. In those exceptional cases each author states his reason for his review. The most convincing reason was considered. In some cases the authors were not sure about the vulnerability. Those warnings were named uncertain.

- C. Is Static code analysis cost effective?

There are many static code analysis tools in the market. Some of them are open source while some of them are commercial. It depends upon the tool selection, which is dependent on the inspection goals. A static analysis tool which is open source, capable of detecting all the possible vulnerabilities, within limited time, which also generates reports for interpretation is really cost effective tool to work with. Anyone who has basic knowledge about the language can easily make use of the tool effectively.

VII SCREENSHOTS

The image showing the running of a local server using XAMPP Software.

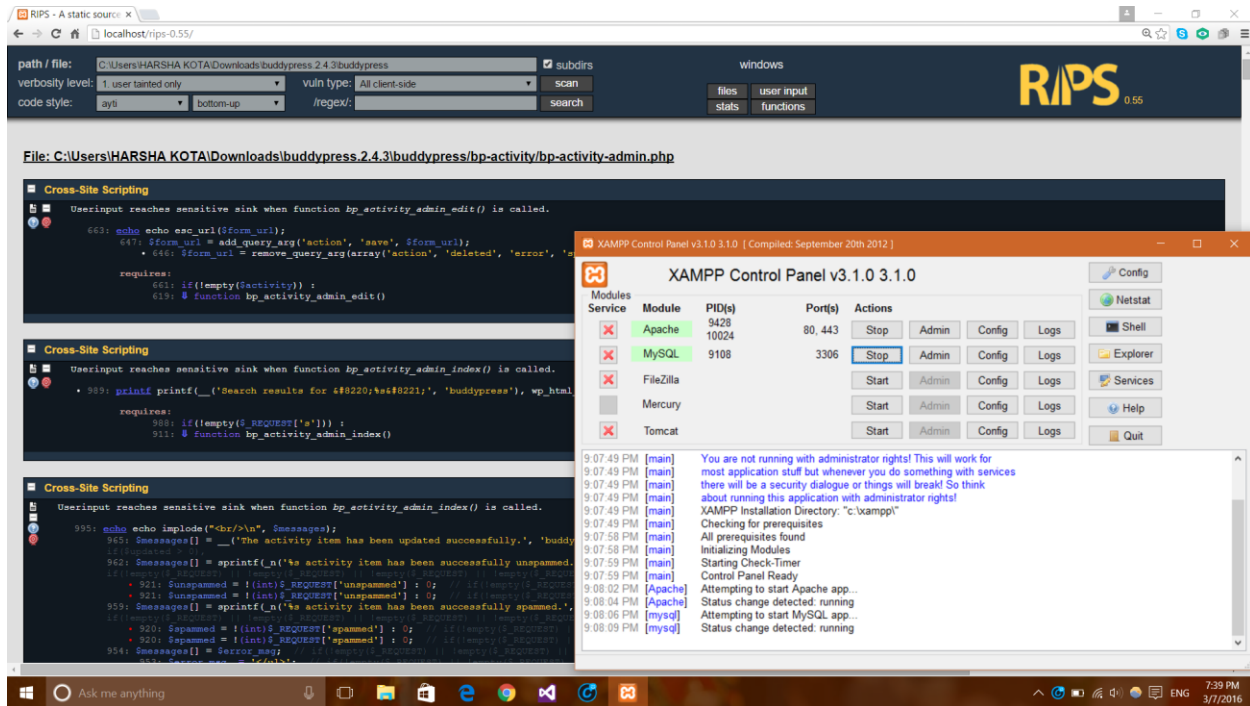


Figure 3 Running Local server to run RIPS tool

The image showing the working of RIPS tool.

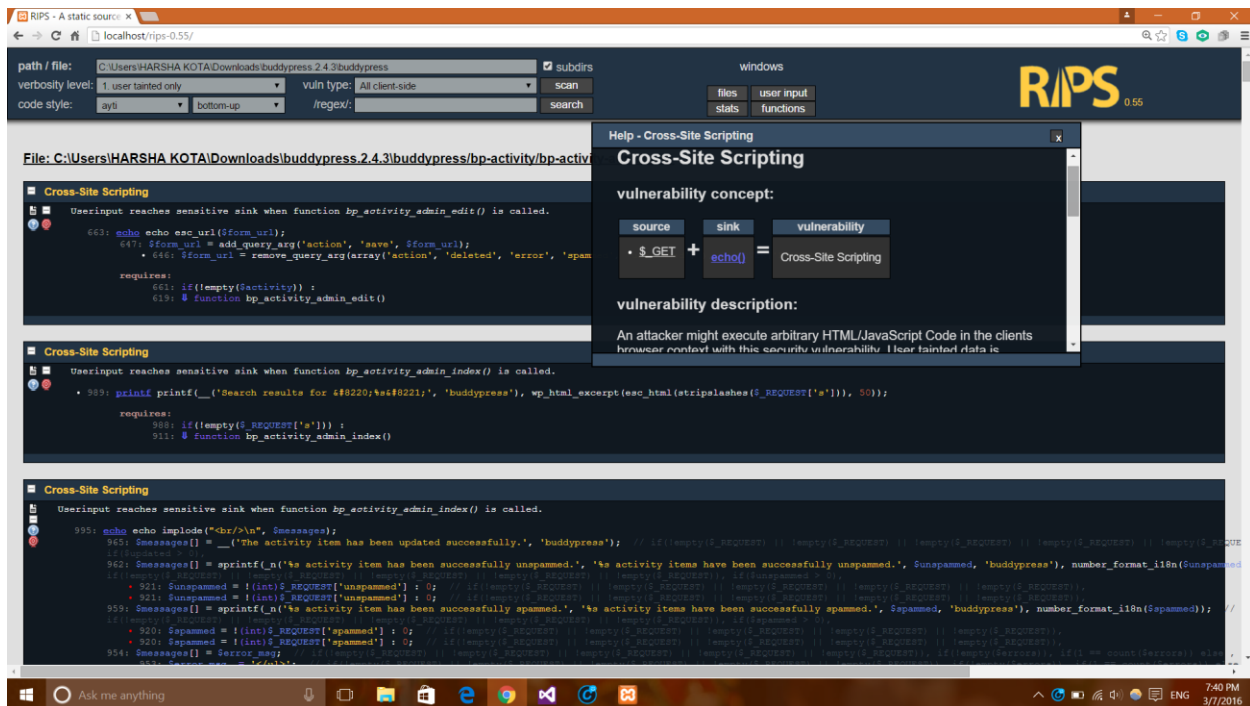


Figure 4 Working of RIPS tool

Part II - Dynamic Analysis

I INTRODUCTION

The number of reported web application vulnerabilities is increasing dramatically. The most common way of securing the web applications is searching and eliminating vulnerabilities therein [4]. Since manual code review consume lot of time, automated approaches for detecting security vulnerabilities were developed. They can be divided into two categories: black-box testing and white-box testing [4]. The second approach is based on web application analysis from the server side, with the assumption that source code of the application is available. In this case static or dynamic techniques can be applied. We have seen the application of static analysis techniques on the BuddyPress so far. This section covers the dynamic analysis of the BuddyPress plugin.

Dynamic code analysis is the process of executing the code to detect vulnerabilities during runtime. Dynamic analysis is usually used during the deployment phase, to ensure web application runtime protection [4].

II SCOPE OF THIS DYNAMIC ANALYSIS

Dynamic analysis can be used at different testing levels like unit testing, acceptance testing, usability testing, etc. A test level is a separate test effort each having its own goals and procedure [6]. Since we are working with web applications, security is our prime focus area. Also the web application we are dealing is an open source application. So the authors have limited the scope of this dynamic analysis to penetration testing and acceptance testing.

Why Penetration testing?

Penetration testing is the testing a web application to find vulnerabilities that an attacker could exploit. The main intention of penetration testing is to determine potential entry points in the application which can be used by the attacker to exploit [3]. As our primary goal is to analyze the security of the web application, so the authors

have used dynamic code analysis at penetration testing level.

Why Acceptance testing?

Acceptance testing is conducted to establish whether a system satisfies its acceptance criteria and to enable the customer to determine whether to accept the system [6]. User acceptance is one important factor for the success of web applications. Since BuddyPress has a huge community of users, the authors decided to conduct acceptance testing to see if the application satisfies its acceptance criteria.

III INSPECTION GOALS

Our inspection goals for this dynamic analysis are twofold. They are:

1. Primary goal is to analyze the security vulnerabilities in web applications. In this context we have chosen penetration testing as one of our test level.
2. Our secondary goal is to check if the system is in acceptance with its criteria. For this we have chosen acceptance testing as our other testing level.

The following sections describe the tools that we have used and the criteria for selecting the tools. We have also detailed the dynamic testing process.

IV SELECTION OF DYNAMIC CODE ANALYSIS TOOLS

On comparison with static code analysis tools, the number of dynamic code analysis tools is less. Some tools which we found were commercial. Some tools were named as hybrid tools which combine the functionality of both static analysis tools and dynamic analysis tools. Some tools have limited functionality like limited testing levels. Out the few tools we found, we have selected two tools which enable us to meet our inspection goals. The following table 7 shows the tools we have found and their evaluation.

S.no	Dynamic Code Analysis Tool	Primary Goal (Security Vulnerabilities)	Secondary Goal (User Acceptance)	Decision Taken
1.	Simple Test	No	No	Rejected
2.	Lime	No	No	Rejected
3.	Ojes	No	No	Rejected
4.	Codeception	Yes	Yes	Accepted
5.	OWASP ZAP Tool	Yes	No	Accepted

Table 7 Evaluation of static code analysis tools

V PENETRATION TESTING USING OWASP ZAP TOOL

The tools like Simple test, Lime and Ojes have limited functionality like they can perform unit testing only. Also it is often difficult to read and rewrite those tests as they are in TDD (Test Driven Development) style where the tests are difficult to interpret.

Why Codeception?

Codeception has acceptance testing, functionality testing and also unit testing capabilities. It readily integrates into a number of IDE's for PHP [7]. The tests are in an easily readable BDD (Behavior driven development) style. BDD uses a more verbose style so that it can be read almost like a sentence. Thus if we can read and understand tests fluidly, we will naturally write better and more comprehensive tests. So the authors have decided to use Codeception.

Why OWASP ZAP Tool?

OWASP ZAP is an easy to use integrated penetration testing tool for finding security related vulnerabilities in web applications [8].

It has a very nice feature called "spider" [8]. The spider is a feature that is used to automatically discover new resources (URLs) on the target application. This is useful in covering the entire web application. So the authors have decided to use OWASP ZAP tool.

These two tools satisfy enable us to meet the two fold objective and hence these tools were used to perform dynamic code analysis. The following sections provide the detailed description of the testing process used for both the tools.

The step by step process is given below:

1. Firstly, we need to download the open source dynamic tool OWASP ZAP and install it.
2. This runs on a server. We first need to use XAMPP software to run a local server.
3. After setting the server, we should specify the URL on which ZAP has to perform the penetration testing.
4. Run the BuddyPress on the local server.
5. Give the URL in the browser to the ZAP tool. Ex: <http://localhost:80/buddypress/>
6. After specifying the URL for ZAP tool, click on Attack Button.
7. The testing process begins it takes a while to complete. The process checks the target code for the security vulnerabilities mentioned in table 2 and 3 during the runtime.
8. After completion the results are grouped under 5 categories namely high, medium, low, informational and false positive.
9. The results are then interpreted.

The next section shows the results and reports of ZAP tool.

VI SCREENSHOTS

The following screenshots show the opening screen of the OWASP ZAP tool and the results of the tool.

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	2
Informational	0

Alert Detail

Medium (Medium)	Application Error Disclosure
Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	http://localhost:8080/buddypress/
Parameter	N/A
Evidence	Parent Directory
URL	http://localhost/buddypress/?C=N.O=D
Parameter	N/A
Evidence	Parent Directory
URL	http://localhost/buddypress/?C=M.O=A
Parameter	N/A
Evidence	Parent Directory
URL	http://localhost/buddypress/?C=S.O=A
Parameter	N/A
Evidence	Parent Directory
URL	http://localhost/buddypress/?C=D.O=A
Parameter	N/A
Evidence	Parent Directory
URL	http://localhost/buddypress/tp-blogs/
Parameter	N/A
Evidence	Parent Directory
URL	http://localhost/buddypress/tp-activity/
Parameter	N/A
Evidence	Parent Directory

Figure 7 The generated html report

VII RESULTS AND INTERPRETATION

The results of the ZAP tool are represented in the following table:

S.no	Alert	Count	Flagged as
1.	Application Error Disclosure	502	Medium
2.	Directory Browsing	57	Medium
3.	X- Frame Options Header Not Set	621	Medium
4.	Web Browser XSS Protection Not Enabled	621	Low
5.	X-Content Type Options header missing	621	Low

Table 8 Alerts classification of OWASP ZAP tool

Application Error Disclosure: This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page. The solution is implementing a mechanism to provide a unique error reference/identifier to the client (browser)

while logging the details on the server side and not exposing them to the user.

Directory Browsing: It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files, backup source files etc. which can be accessed to read sensitive information. Solution is to disable directory browsing.

X-Frame Options Header: X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks. ClickJacking is an UI attack of maliciously tricking a web user into clicking on something different from what the user perceives they are clicking on. Solution is that most browsers support X-frame options. Ensure it is set on all web pages.

Web Browser XSS Protection: Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server. Ensure that it is enabled in the web browser.

X-Content Type Options Header: The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'no sniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Mostly False positive since modern browsers aren't vulnerable.

VIII ACCEPTANCE TESTING USING CODECEPTION

The authors want to test a widget of BuddyPress which helps us to log in and log out easy on homepage using Codeception tool. The following table shows the setup required to perform this test.

Tool	Version	Comments
Lenovo Laptop	Y480N	
Windows 10	Professional	
XAMPP Software	5.6.15	Host a local server
WordPress	4.4.2	Framework of BuddyPress
BuddyPress	2.5.0	Target Application
Codeception Software	2.1.6	BDD testing frame work for PHP
Chrome	48.0.2564.97m	A fast Browser

Table 9 Required Setup for this test

The step by step process is given below:

1. Start the local server using XAMPP software.
2. We have downloaded the WordPress source files and we started running those files on the server. Then we can directly access BuddyPress plugin on top of it.
3. Since we want to perform acceptance testing on a widget, we have added the BuddyPress login widget to the homepage as shown in figure 8.
4. Then we can see the widget on the home screen. Figure 9 shows the home page after adding the widget.
5. In order to perform acceptance testing on the widget using Codeception, we need to develop the test scenarios. The authors need to create test scenarios.
6. The authors designed to two test scenarios one for log-in and one for log-out.
7. The following table shows the scenarios the authors created.

Test Scenario 1 for testing Log-in	
1.	Open Homepage
2.	Fill username
3.	Fill Password
4.	Click "Log in" Button
5.	See Dashboard

Table 10 Test Script 1 in BDD style

Test Scenario 2 for testing Log-out	
1.	Click "WordPress" button and back to homepage
2.	Click "log out" Button
3.	See homepage which is same as before log in

Table 11 Test Script 2 in BDD style

8. After learning the quick start guide of Codeception the authors entered the following files into the workspace of Codeception as they were required to perform the test. The figures 10, 11, 12 show the files we added.
9. After adding the test scenarios to the workspace, the authors ran the following command in the cmd.

```
<php codecept.phar run --steps --xml --html >
```

The above command is used to test the acceptance testing file we designed above displays the result step by step and also the final result in html file.

The screenshots of the working process have been shown in the next section followed by the results and conclusions.

IX SCREENSHOTS

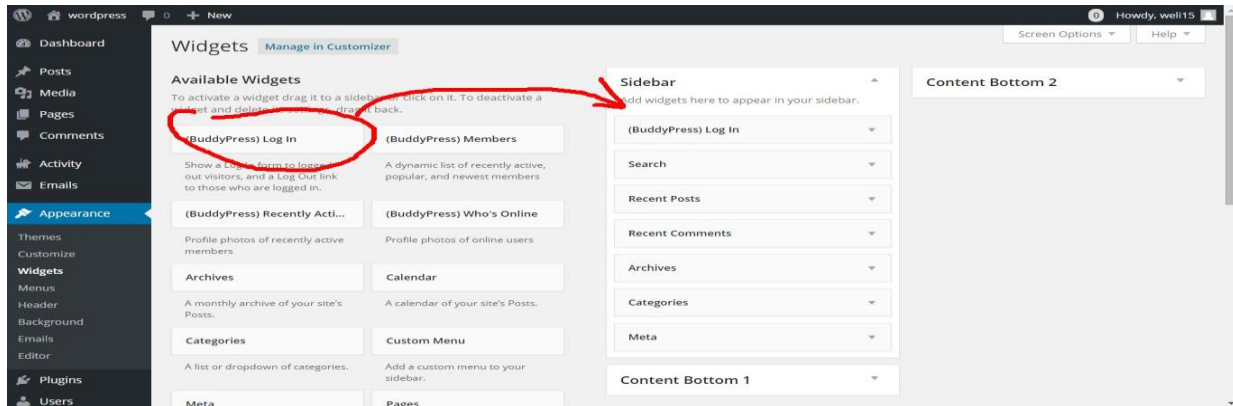


Figure 8 Homepage of WordPress running on the local server

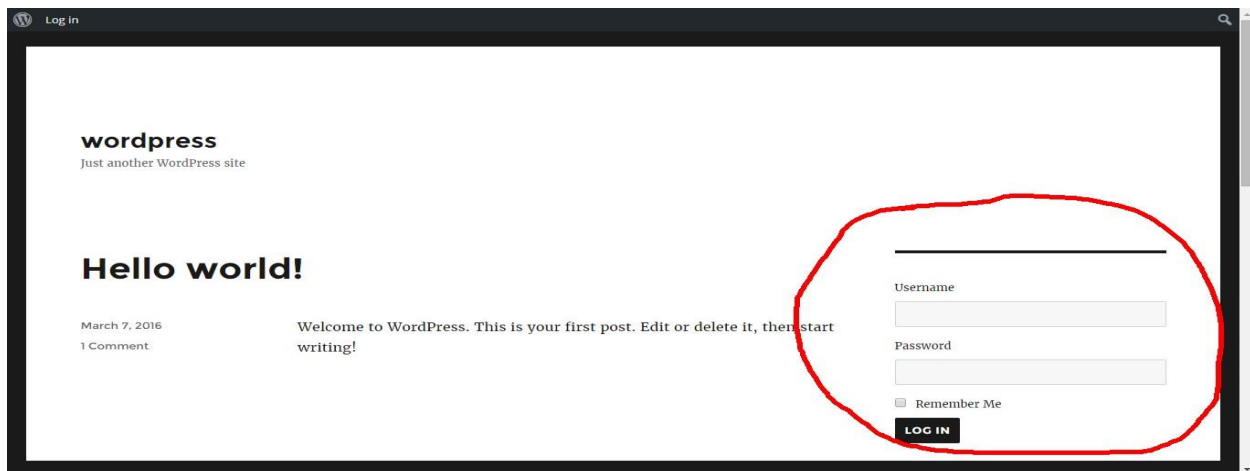


Figure 9 Homepage after adding the BuddyPress Widget

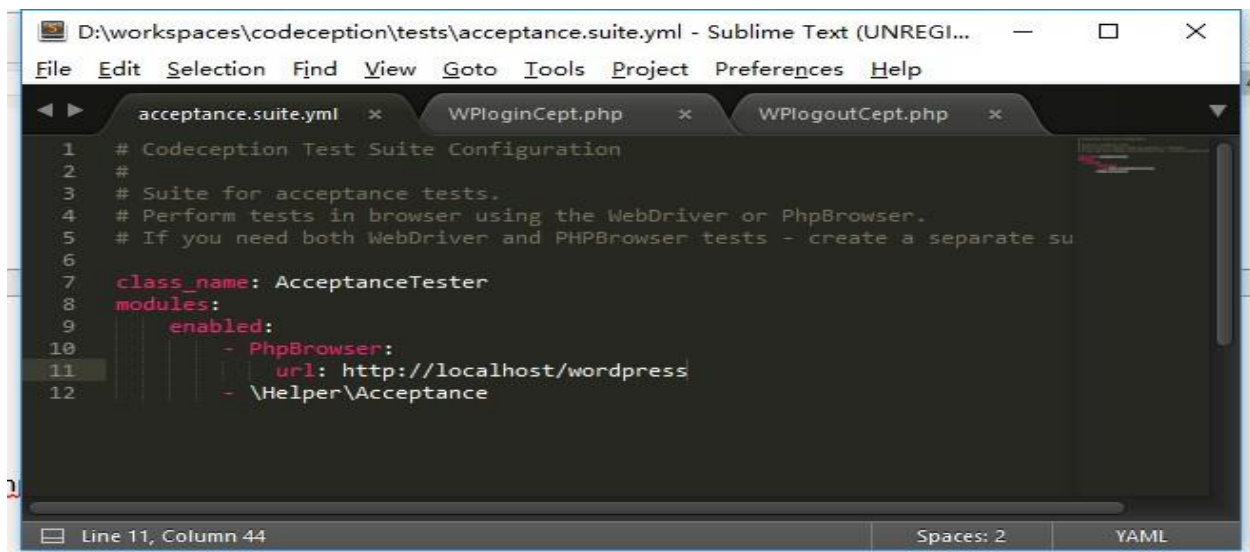
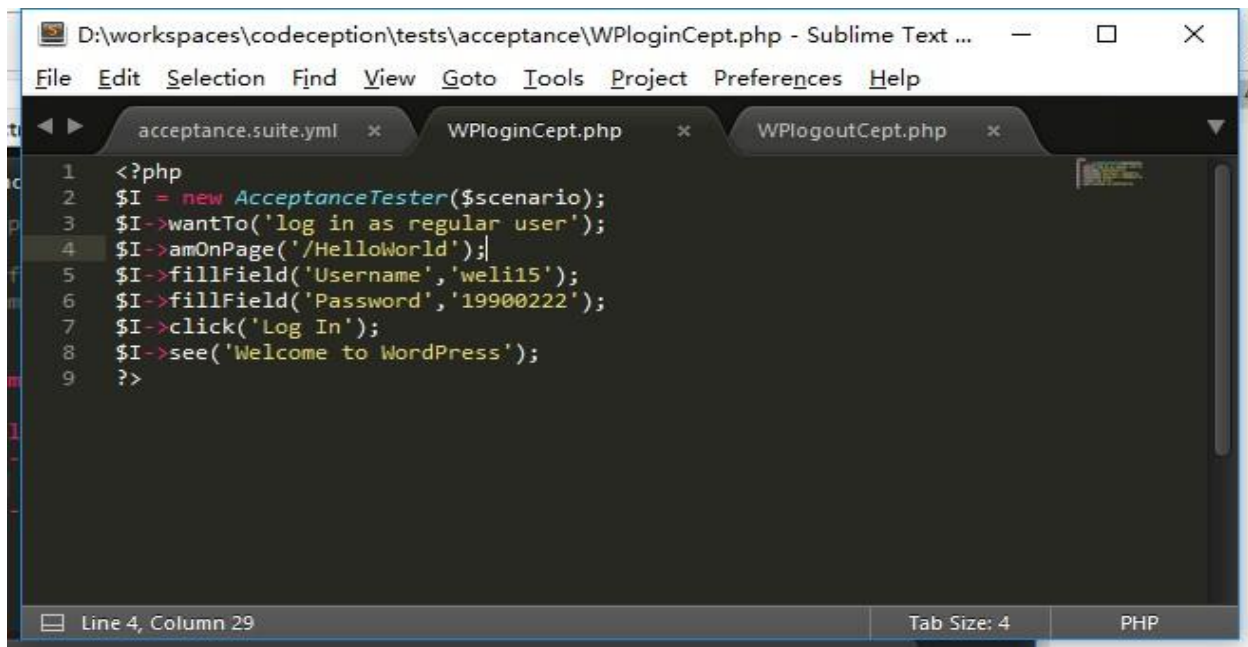


Figure 10 <acceptance.suite.yml>

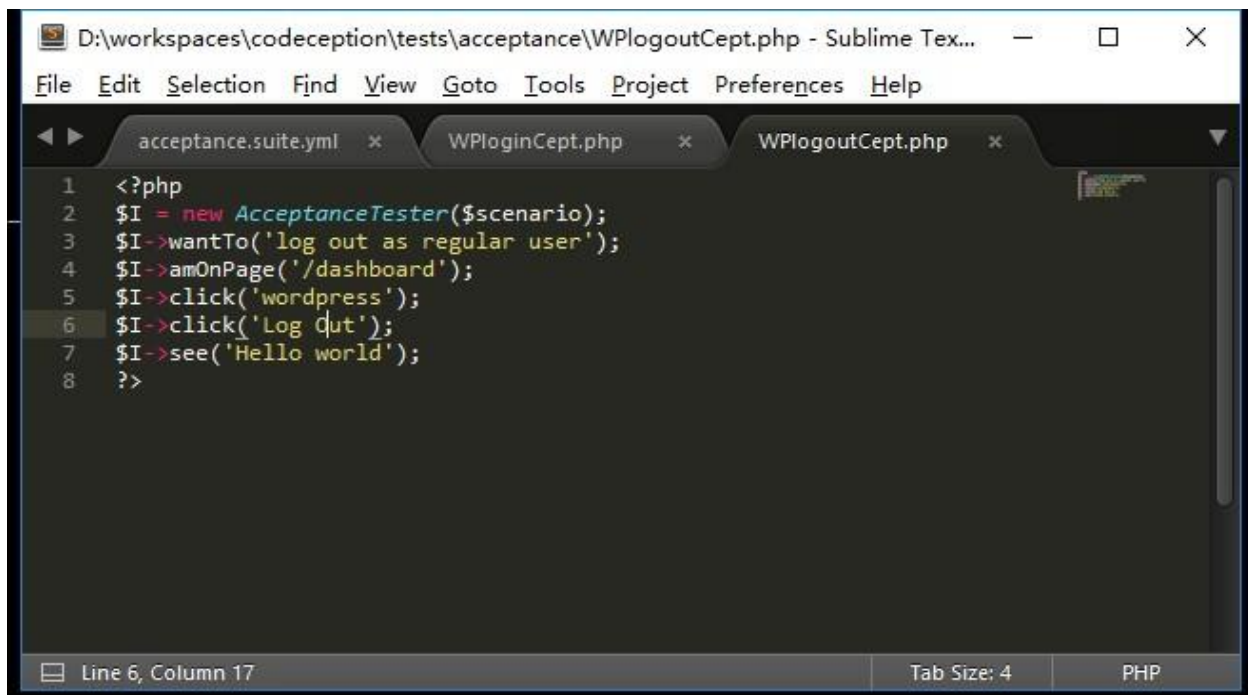


The screenshot shows a Sublime Text editor window with the title bar "D:\workspaces\codeception\tests\acceptance\WPloginCept.php - Sublime Text ...". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The tab bar shows three tabs: acceptance.suite.yml, WPloginCept.php (active), and WLogoutCept.php. The code in WPloginCept.php is as follows:

```
1 <?php
2 $I = new AcceptanceTester($scenario);
3 $I->wantTo('log in as regular user');
4 $I->amOnPage('/HelloWorld');
5 $I->fillField('Username','weli15');
6 $I->fillField('Password','19900222');
7 $I->click('Log In');
8 $I->see('Welcome to WordPress');
9 ?>
```

The status bar at the bottom indicates "Line 4, Column 29", "Tab Size: 4", and "PHP".

Figure 11 <WPloginCent.php>



The screenshot shows a Sublime Text editor window with the title bar "D:\workspaces\codeception\tests\acceptance\WLogoutCept.php - Sublime Tex...". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The tab bar shows three tabs: acceptance.suite.yml, WPloginCept.php, and WLogoutCept.php (active). The code in WLogoutCent.php is as follows:

```
1 <?php
2 $I = new AcceptanceTester($scenario);
3 $I->wantTo('log out as regular user');
4 $I->amOnPage('/dashboard');
5 $I->click('wordpress');
6 $I->click('Log Out');
7 $I->see('Hello world');
8 ?>
```

The status bar at the bottom indicates "Line 6, Column 17", "Tab Size: 4", and "PHP".

Figure 12 <WLogoutCent.php>

```
D:\workspaces\codeception>php codecept.phar run acceptance --steps
Codeception PHP Testing Framework v2.1.6
Powered by PHPUnit 4.8.22 by Sebastian Bergmann and contributors.
```

```
Acceptance Tests (2) -----
Log in as regular user (WPloginCept)
```

```
Scenario:
```

```
* I am on page "/HelloWorld"
* I fill field "Username","weli15"
* I fill field "Password","19900222"
* I click "Log In"
* I see "Welcome to WordPress"
PASSED
```

```
Log out as regular user (WPlogoutCept)
```

```
Scenario:
```

```
* I am on page "/dashboard"
* I click "wordpress"
* I click "Log Out"
FAIL
```

Figure 13 Testing Result

```
Time: 5.31 seconds, Memory: 9.75Mb
```

```
There was 1 failure:
```

```
1) Failed to log out as regular user in WPlogoutCept (tests\acceptance\WPlogoutCept.php)
```

```
Step I click "Log Out"
```

```
Fail Link or Button by name or CSS or XPath element with 'Log Out' was not found.
```

```
Scenario Steps:
```

```
3. $I->click("Log Out")
2. $I->click("wordpress")
1. $I->amOnPage("/dashboard")
```

```
FAILURES!
```

```
Tests: 2, Assertions: 1, Failures: 1.
```

Figure 14 Failure reason for logout scenario

Acceptance Tests

○	[+] Log in as regular user (WPloginCept) 1.5s
✓ 1	I am on page "/HelloWorld"
	I fill field "Username","weli15"
X 1	I fill field "Password","19900222"
	I click "Log In"
	I see "Welcome to WordPress"
S 0	[+] Log out as regular user (WPlogoutCept) 3.4s
I 0	I am on page "/dashboard"
	I click "wordpress"
	I click "Log Out"
	Link or Button by name or CSS or XPath element with 'Log Out' was not found.

Summary

Successful scenarios:	1
Failed scenarios:	1
Skipped scenarios:	0
Incomplete scenarios:	0

Figure 15 Final result html file

X RESULTS AND INTERPRETATIONS

From the results, we can see that log in scenario is passed but log out scenario is failed. We have tried different kinds of format to implement log out function such as the format in the user guide of Codeception but none of them was useful. Some of them showed the same result and others didn't compile successfully. As shown in the figure 14 there was a problem with XPath element in the code.

```
<?php
// By specifying locator type
$I->click(['link' => 'Login']);
$I->click(['class' => 'btn']);
?>
```

Figure 16 parameter format 1

```
<?php
$I->click('Log in');
// CSS selector applied
$I->click('#login a');
// XPath
$I->click('//a[@id=login]');
// Using context as second argument
$I->click('Login', '.nav');
?>
```

Figure 17 parameter format 2

XI COMPARISON OF RESULTS OF BOTH THE TOOLS

As the objective of the authors is twofold, we have selected two different tools to meet both the primary and secondary inspection goals. OWASP ZAP tool provided a good insight into the security vulnerabilities present in the application whereas Codeception enabled us to perform acceptance testing on one of the functional widget of BuddyPress using manually written test cases. Since BuddyPress is used by many people, acceptance testing enables us to understand the system reactions to various user actions. Both of these results are from different test levels each having its own importance.

XII CONCLUSIONS

After performing both static and dynamic analysis, the authors come to the following conclusions:

Both static and dynamic code analysis aim at detecting the vulnerabilities in the code at different stages. Static at the developing stage while dynamic at the deployment phase. Ultimately the goal of both these analysis approaches is to deliver applications with zero or minimum vulnerabilities making the applications more productive for the user.

REFERENCES

- [1]. I. Medeiros, N. Neves and M. Correia, "Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining," in *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 54-69, March 2016.
- [2]. "BuddyPress - open source plugin." [Online]. Available: <https://wordpress.org/plugins/buddypress/>. [Accessed: 29-feb-2016]
- [3]. "OWASP – Open Source Web Application Security Project." [Online]. Available: https://www.owasp.org/index.php/Main_Page. [Accessed: 01-mar-2016]
- [4]. Petukhov, Andrey, and Dmitry Kozlov. "Detecting security vulnerabilities in web applications using dynamic analysis with penetration testing." *Computing Systems Lab, Department of Computer Science, Moscow State University*, 2008.
- [5]. Curphey, M., Wiesman, A., Van der Stock, A., Stirbei, R.: "A Guide to Building Secure Web Applications and Web Services". OWASP 2005.
- [6]. IEEE Standard for Software and System Test Documentation - Redline," in *IEEE Std 829-2008 (Revision of IEEE Std 829-1998) - Redline* , vol., no., pp.1-161, July 18 2008

[7]. “Codeception – BDD style PHP testing.”
[Online]. Available: <http://codeception.com/>.
[Accessed: 02-mar-2016]

[8]. “OWASP ZAP Tool – Penetration testing tool.” [Online]. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. [Accessed: 02-mar-2016]

[9]. “RIPS – PHP static code analyzer.”
[Online]. Available: <http://rips-scanner.sourceforge.net/>. [Accessed: 01-mar-2016]

Appendix: A

Contribution statement table

Name	Time spent on the work (in hours)	Percentage of contribution to overall work result
Wenguang Li	16	20%
Akshay Kumar Jilla	16	20%
Sai Priyatham Dongoor	16	20%
Panchajanya Varanasi	16	20%
Sai Mohan Harsha Kota	16	20%