BuddyPress Testing

Part I: Static Analysis

Li Xian, Fan Ling Zhi, Struchkov Innokentiy, Kothawar Sravika, Avutu Neeraj Reddy
Blekinge Institute of Technology
Karlskrona, Sweden
1451194024@qq.com, 591029368@qq.com, i.t.struchkov@mail.ru, srko16@student.bth.se, neav16@student.bth.se

Abstract—**This part introduces a static code analysis of BuddyPress using RIPS as the tool. Generally, it describes the inspection goals and corresponding motivations, comparisons between alternative tools and RIPS, and then it's the performance evaluation of RIPS and our findings.**

Keywords—static analysis; BuddyPress; RIPS; vulnerabilities

I. Introduction

BuddyPress is a powerful community plugin for WordPress that takes your site beyond the blog. It includes all of the features you've come to expect from any online community, like user profiles, groups, activity streams, notifications, and more[1]. We'd like to do the static code analysis of buddyPress. Static Code Analysis is usually performed as part of a Code Review and is carried out at the Implementation phase of a Security Development Lifecycle (SDL). Static Code Analysis commonly refers to the running of Static Code Analysis tools that attempt to highlight possible vulnerabilities within 'static' (non-running) source code by using techniques such as Taint Analysis and Data Flow Analysis[2]. So we chose RIPS as our tool to perform the static code analysis. RIPS is a tool written in PHP to find vulnerabilities in PHP applications using static code analysis. By tokenizing and parsing all source code files, RIPS is able to transform PHP source code into a program model and to detect sensitive sinks (potentially vulnerable functions) that can be tainted by user input (influenced by a malicious user) during the program flow[3]. This paper is organized as follows: section II describes how our inspection goals were produced. Section III describes why we chose RIPS instead of other alternative tools. Section IV describes the performance of RIPS, warnings classified by individually and corresponding reasons. Section V is the conclusion about Part I.

II. Inspection Goals

In order to decide which static code analysis tool for testing BuddyPress, we read some literature, from[4], we found the following potential goals: (1) Maintain quality. (2) Reach zero uninspected detects. (3) Reach zero outstanding defects. From[5], we found the following potential goals: (4) Type checking. (5) Style checking. (6) Security review. Besides, when we looked into the details of RIPS, we came up with the goals of checking vulnerability types and checking user input/file/function list. And all of the goals above formed the potential set of our inspection goals.

After our discussion, we reached an agreement on our inspection goals, which are maintaining quality, security review, checking vulnerability types and checking user input/file/function list. We chose quality for our first goal because of that we were required to assure good quality of BuddyPress for further extending and modernizing. And quality is a more suitable criteria for assessing software products, we can detect the defects of BuddyPress and maintain its quality. For the goal of security review, since BuddyPress includes many users' privacy information, such as friend connections, internal messaging, extended profiles and site tracking, so we want to find some security defects and protect users' privacy from being attacking. We chose the third goal of checking vulnerability types because we want to investigate as many defects as possible, and classify the security defects into different types for giving better solutions. And there are many kinds of vulnerabilities, such as cross-site scripting, possible flow control, SQL injection, file manipulation, etc. For the forth goal of checking user input/file/function list, since BuddyPress was written by PHP language, but none of our group members were familiar with it, so we want to get the list of user input lists, file lists and function lists to get a rough understanding of this product. Other very good reason to start using static analysis tools for maintaining the quality of the project is that they can detect most vulnerable and weak parts of the code of the entire project. Based on the collected data, some static analyzers can automatically build descriptive tables, graphs, visual connections, practical lists and show critical weak points of the project. Some advanced analyzers can even extend their reports with context-based links and hints all over the page which greatly improves visual and logic understanding of reports. Tools that utilizes demonstrative metrics and context-based hints are a necessity for the large-scaled project with tens, hundreds and thousands of source files. BuddyPress has more than 700 files of source code, so in this case, it will be very useful to apply a static analyzer that can provide good visual metrics and graphs. Besides, it will help us to make comparisons with the dynamic analysis part.

III. ALTERNATIVE STATIC ANALYSIS TOOLS

In the static code analysis part, we chose RIPS as our tool based on our inspection goals. And we provide four alternative tools which are PHP mess detector, PHP copy/paste detector, PHP analyzer and PHP storm to make comparisons and explain the reasons why we finally chose RIPS.

(1) PHP Mess Detector

PHP Mess Detector is a tool which can check source code written in PHP for possible problems. PHPMD can detect possible bugs, sub-optimal code, unused parameters and so on. Also, PHPMD can check code style and complexity and suggests advice which can optimize the project's code. This tool is recommended to use with PHP Storm IDE, in that case PHPMD will take all advantages of IDE. Otherwise, PHPMD runs as a locally configurable script. The user can create own rule sets in XML format and combine multiple rule sets for code analysis process. There are three possible ways to get analysis reports: XML, plain text and HTML.

As we can see PHPMD is a nice extension to basic static analyzers of IDE. It can suggest very useful recommendations regarding project's source code quality and readability, which IDE can't provide. And it's freeware status can be considered as a convenience by newly established companies with humble economic conditions.

But the advantages of PHPMD don't fit the given scenario in our opinion. Scenario: "You have taken over the project for the system specified above and your product manager asked you to first conduct a static analysis for the product to assure good quality prior to further extending and modernizing it." Our case' s project is the Buddypress. A very popular open-source plugin for Wordpress that adds full social capabilities for that content management system. Even newly installed copies of Wordpress offers to an administrator to install this plugin right away after initial set up. Because its popularity and open-source origins, Buddypress has very polished code style already and it follows Wordpress's code guidelines very distinctly. Therefore the use of PHPMD can be quite inefficient in given scenario. PHPMD is also not a very flexible tool for quick discovering and locating of code flaws. It only provides a single static report file containing the simple list of warnings.

We should concentrate our attention on security problems of BuddyPress because of social nature of the plugin. PHPMD doesn't provide functionality necessary to us in basic set and also requires some knowledge and time to setting up. But PHPMD can be used in future to further extending and modernizing of the code-base of project.

(2) PHP Copy/Paste Detector

PHPCPD is a PEAR tool. Duplication of code is generally a bad programming style, while it might be appealing and faster to duplicate certain fragments, it often comes at the cost of increased maintaining efforts like harder to change code. Duplication occurs when copy/paste is done. it also occurs when two developers write similar code without knowing it. For avoiding it, PHP copy/paste detector generally scans a project for duplicate code. It detects the repeated code in PHP file or project. PHP Copy/Paste Detector operates on the token stream which is generated by PHP's token_Get_all function and uses string hashing techniques to find duplicates. This makes it good at finding large literal code duplication.

With PHPCPD, we cannot meet our requirements, i.e. list the warnings or the vulnerabilities in the software. With this we can only remove the duplication in the code, but cannot meet our goals. We can use it later in order to check the duplication of code. So, we are using RIPS in order to meet up with the goals set by us, i.e. doesn't meet the functionality set needed by us.

(3) PHP Analyzer

PHP Analyzer performs the same analyzes that a compiler would like, for example, type inference or other flow analyzes, ensuring that every line of code and every potential execution path are tested. It uses multiple techniques to ensure deep, accurate analysis including Interprocedural Dataflow Analysis; Reverse Abstract Interpretation; Design Pattern Intelligence; Framework Intelligence. Table 1 below shows the difference between with PHP analyzer and RIPS.

Table 1 Difference between with PHP analyzer and RIPS

PHP analyzer	RIPS
Focus on the methods and classes	Detect backdoor
Detect simple vulnerability	Detect a lot of vulnerabilities
Slow, take a long time	Fast
lack of grasping traits	Can grasp traits
command	Web check, localhost

From Table.1, we found RIPS was better than PHP analyzer, it can detect more information and faster than PHP analyzer, also including grasping traits.

Also, we found that RIPS can check more vulnerabilities than PHP Analyzer. And it suits for our goal, It has a very comprehensive inspection according to many paths. We can get more reliable information which includes the source and reasons, so we decide to choose RIPS.

(4) PHP Storm

PHP storm is a platform where PHP,HTML and java script can be written, it provides the on-the-fly error prevention, best auto-correction, code analysis, code refactoring, zero configuration, debugging and an extended HTML,CSS and Java Script editor, i.e. Iit has all the functionalities of web storm which full fledged support for PHP and databases.

PHP storm supports multiple operating systems. It can be quick started and easy on memory. It is full featured with development IDE. It supports code completion, documentation and lint. It can synchronize files from local to server and vice-versa. It can be extended with plug-ins. It can support ZEN-coding.

We have several disadvantages with PHP Storm. Here are some of them. It requires Oracle Java for it to run properly.

This tool is complex for us to understand, i.e. for developers. It is too commercial to use.

With the above information regarding PHP storm, we are having a brief idea regarding it. With that it can be said that PHP Storm doesn't fit the given synopsis. Our project is to test BuddyPress statically and dynamically, which is a very popular plug-in for word press. It has very polished code style already and follows guidelines of WordPress. As PHP storm is efficient to use, PHP Storm is also best in use, when we are developing a software. We cannot discuss flaws in the code using storm or locate the vulnerabilities efficiently. As we are amateurs, it is very hard for us to understand PHP Storm, within the given time, for making the best use of it. PHP storm can be used in the future, for the reading of the software, improvement of the project. So, RIPS is better to use comparatively, which can ably detect sensitive sinks that can be tainted by user input during the program. We can search for vulnerabilities in different parts of the BuddyPress efficiently.

IV. Findings and individual comparisons

1. Findings

Through scanning BuddyPress using RIPS which is a static code analysis tool for PHP security. We scanned all the files of BuddyPress, RIPS detected 79 vulnerabilities in total. BuddyPress includes 13 folders and some independent files out of folders.

In the folder of bp-activity and its sub-directories, RIPS detected 3 cross-site scripting vulnerabilities. In the folder of bp-blogs and its sub-directories, RIPS detected 4 cross-site scripting vulnerabilities. In the folder of bp-core and its sub-directories, RIPS detected 3 file disclosure vulnerabilities, 3 file manipulation vulnerabilities, 1 cross-site scripting vulnerability and 1 possible flow control vulnerability. In the folder of bp-forums and its sub-directories, RIPS detected 1 file manipulation vulnerability, 1 SQL injection vulnerability, 16 cross-site scripting, 1 HTTP response splitting vulnerability, 1 session fixation vulnerability, 10 possible flow control vulnerabilities and 1 PHP object injection vulnerability. In the folder of bp-groups and its sub-directories, RIPS detected 11 cross-site scripting vulnerabilities. In the folder of bp-members and its sub-directories, RIPS detected 1 file manipulation vulnerability and 11 cross-site scripting vulnerabilities. In the folder of bp-messages and its sub-directories, RIPS detected 1 cross-site scripting vulnerability. In the folder of bp-templates and its sub-directories, RIPS detected 2 cross-site scripting. In the folder of bp-themes and its sub-directories, RIPS detected 5 cross-site scripting vulnerabilities. In the folder of bp-xprofile and its sub-directories, RIPS detected 2 possible flow control vulnerabilities.

2. Individual Comparisons

When we got the results, we need to classify these warnings into three types which are true positive, false positive and undecided. When we individually decided on the warnings, we took many factors into consideration, some of us checked the source code and got the deep understanding of calling mechanism of functions, some of us checked user inputs of the warnings and found the dependencies among the lists, and some of us checked the guideline of how PHP and BuddyPress work, and learned many sample code. After we finished listing the warnings individually, most of us made a similar classification of those warnings. And we classified most of the warnings into false positive, some true positive warnings, and some undecided warnings. Even though, when we discussed our own results, some disagreements occurred, there are many reasons for that, one reason is that someone thought that some specific types of faults easier to judge than others, such as the warnings of echo and printf, echo is used for outputting all parameters, printf is used for producing output according to format. The second reason is that the members of group have different PHP language backgrounds, some of us are good at understanding the source code, and some of us may don’t understand the source code at all, so we may classify the same warning into different types. The third reason is that we individually may has some biases on classifying the warnings, some of us would just look at the description of one specific warning and then marking it as true positive. And some of us would investigate the details of the same warning, such as the functions calling and referencing code and then marking it as false positive.

During the discussion of the warnings, we read the warning descriptions and searched the details of every warning, how it works, how does it be used in BuddyPress, and what the relationships between this warning and the similar warnings, also, we checked the user input lists, file lists and function lists of each warning. Finally, we reached the consensus of all the 79 warnings detected by RIPS, we marked 7 warnings as true positive, 4 warnings as undecided and 68 warnings as false positive.

V. Discussion and conclusion

1. Discussion

RIPS, as a static code analysis tool, was very helpful in achieving our goals. For maintaining quality and security review, since in static analysis part, we were required to assure good quality prior to further extending, and quality includes many aspects, such as security, testability, modifiability, etc. Although the security of the open source web platforms themselves has been steadily increasing, plugins can introduce new security threats to websites. Because installing a plugin does not necessarily require technical skills nor reviewing the plugin codebase, security of the plugins in the first place is extremely important[6]. Since BuddyPress is a plugin of WordPress, and RIPS is a good tool for detecting vulnerability, so RIPS was helpful in maintaining the quality of BuddyPress. For the goal of checking vulnerability types, RIPS is a tool written in PHP to find vulnerabilities in PHP applications using static code analysis[3], and it can detect many kinds of vulnerabilities. Taking the concept of taint-style vulnerabilities in PHP as an example which showed by figure 1[7]:

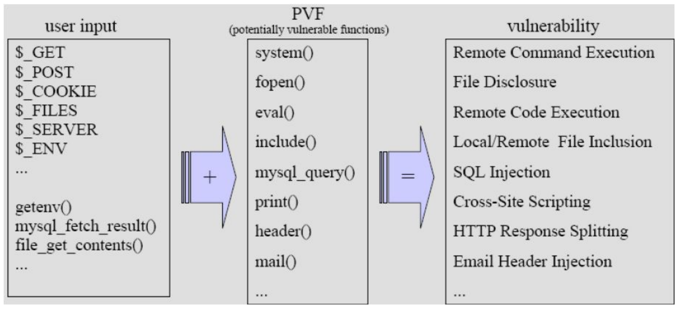
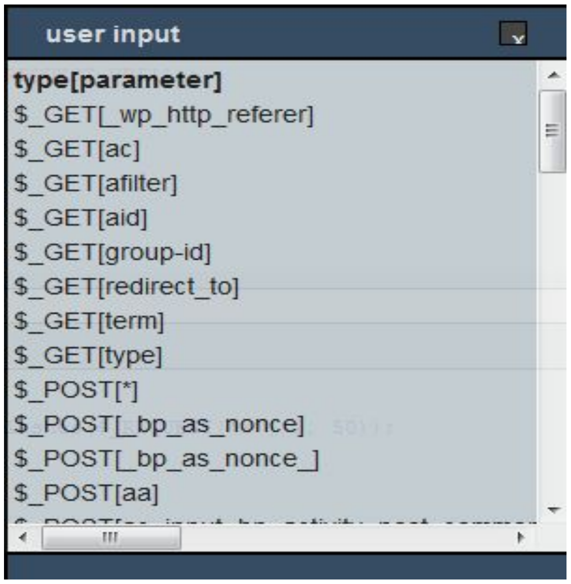


Figure 1: Concept of taint-style vulnerabilities in PHP

All the vulnerability types above can be detected by RIPS, in reality, RIPS has helped us detected 79 vulnerabilities of BuddyPress totally, in which it includes 53 cross-site scripting vulnerabilities, 13 possible flow control vulnerabilities, etc. Therefore, RIPS was helpful in achieving our goal of checking vulnerabilities. For the goal of checking user input/file list/function list, since most of our group members didn't familiar with PHP language, so we want to know about the user inputs, file list and functions list of one specific warning to get some basic information, and after we got that , it would be easier for us to make the decision whether the warning was true positive, false positive or undecided. RIPS can provide all these lists, figure 2 below shows the user input list of the warning "echo":


user input

type[parameter]
$_GET[_wp_http_referer]
$_GET[ac]
$_GET[afilter]
$_GET[aid]
$_GET[group-id]
$_GET[redirect_to]
$_GET[term]
$_GET[type]
$_POST[*]
$_POST[_bp_as_nonce]
$_POST[_bp_as_nonce_]
$_POST[aa]

Figure 2: User input list of echo

Figure 3 below shows the file list of echo:

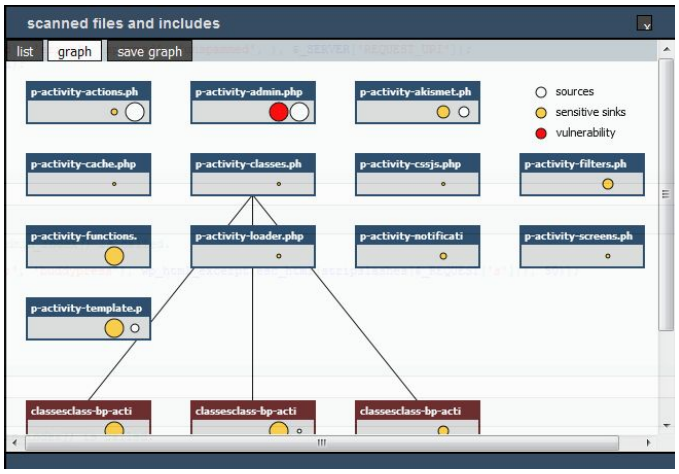
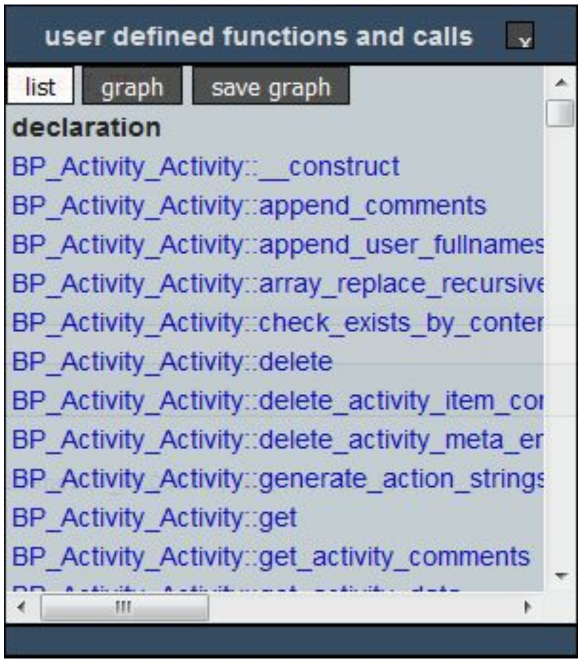


Figure 3: File list of echo

Figure 4 below shows the function list of echo:


user defined functions and calls

list graph save graph

declaration
BP_Activity_Activity::__construct
BP_Activity_Activity::append_comments
BP_Activity_Activity::append_user_fullnames
BP_Activity_Activity::array_replace_recursive
BP_Activity_Activity::check_exists_by_conter
BP_Activity_Activity::delete
BP_Activity_Activity::delete_activity_item_coi
BP_Activity_Activity::delete_activity_meta_er
BP_Activity_Activity::generate_action_strings
BP_Activity_Activity::get
BP_Activity_Activity::get_activity_comments

Figure 4: Function list of echo

From the figures above, we could know more information about the warning of echo. In this way, RIPS was helpful in achieving our goal of checking user input/file/function lists.

However, from a practical point of view, RIPS didn't perform very well in accurately identifying the faults. Static source code analysis tools often generate a lot of false positives and warnings. This may occur due to incorrect semantic or flow analysis or circumstances that are needed for a correct identification of a vulnerability but which could not be evaluated by the tool correctly. Therefore it is important to present the results in such a way that the user can easily decide between a correct vulnerability detection or a false positive himself[7]. Static analysis tools cannot solve all of the security problems, mainly because these tools look for a fixed set of patterns, or rules, in the code. The output of static analysis tools still requires human evaluation. There's no way for a tool to know exactly which problems are more or less important for the programmer automatically, so there is no way to avoid studying the output and making a judgment call about which issues should be fixed and which ones represent an acceptable level of risk[8]. In terms of our findings, RIPS detected 79 vulnerabilities of BuddyPress totally, and we concluded that 68 vulnerabilities were false positive, so it could not detect more real faults exist in the project.

When we list the warnings which detected by RIPS individually, we also logged the time we spent on listing every warning. Static analysis proved out to be very fast and cost efficient in analyzing the quality of the large codebase, even though manual review would be required in order to verify the issues and to fix them. The vulnerabilities that were detected in the analysis can not automatically be considered as defects, but they are implemented in a way which makes them, under certain conditions, vulnerable to attacks[6]. Excepting for the scanning time, our first reviewer spent about 3 hours to list the warnings and make assessment of them, the second reviewer spent about 2 hours on that, the third and fifth reviewer spent about 2.5 hours on that, and the fourth reviewer spent about 4 hours on that. Besides, we spent around 4 hours to reach an agreement of those warnings. However, we spent less time in the static analysis part compared with the dynamic analysis part since finding suitable testing tools for dynamic analysis took us much time.

2. Conclusion

In conclusion, security is one of the most important issues software developers have to have in mind when creating it. This fact led to developing various ways of securing the software. One of those ways is Static Code Analysis[5]. Static analysis allows one to examine all the different execution paths through a program at once and make assessments that apply to every possible permutation of inputs. During the analysis step configured security knowledge can be used to detect vulnerabilities[7]. RIPS has helped us in achieving all the goals relevant to vulnerabilities detecting, RIPS found 79 vulnerabilities, and we marked 68 of them as false positive, 7 of them were true false positive, and the rest were undecided. Static analysis can decrease the amount of testing and debugging necessary for the software to be deemed ready[8]. And static code analysis has proved to be the fast and cost-effective way of defects detecting in projects. However, static code analysis would produce many false positive warnings, and may give developers and testers a false sense of security. So we'd like to do a dynamic analysis of BuddyPress in Part II.

PART II: DYNAMIC ANALYSIS

Abstract—**This part introduces a dynamic code analysis of BuddyPress using PHP vulnerability Hunter and OWASP ZAP as the tools. We described our inspection goals and motivations first, then we provided several alternative tools with their properties and depicted the reasons why we chose the two tools. Finally, we compared the outputs produced by the tools, evaluated their performance and made our conclusion.**

Keywords—dynamic analysis; BuddyPress; PHP Vulnerability Hunter; OWASP ZAP; Outputs

VI. INTRODUCTION

Dynamic analysis is the analysis of the properties of a running program. In contrast to static analysis, which examines a program's text to derive properties that hold for all executions, dynamic analysis derives properties that hold for one or more executions by examination of the running program. Dynamic analysis can detect violations of properties as well as provide useful information to programmers about the behavior of their programs[9]. Dynamic code analysis includes vulnerability testing and penetration testing. In this part, firstly, we chose PHP vulnerability Hunter for vulnerability testing of BuddyPress, OWAP ZAP for penetration testing, and then we compared the outputs of the tools and explained the differences. This part is organized as follows: section VII describes our dynamic inspection goals and their importance and rationales. Section VIII describes the tools we selected for BuddyPress's dynamic analysis and the testing techniques they implemented , also, described alternative tools and their properties. Section IX describes the two tools' performance and comparisons of different outputs produced by these two tools and corresponding reasons. Section X describes the discussion and our conclusion. Section XI is the contribution table which shows our individual contributions during the whole BuddyPress's testing process.

VII. INSPECTION GOALS

Since we got many false positive warnings from static analysis part, in dynamic part we want to detect more real faults of BuddyPress. In this process, we found some goals from literature which gave us some hints in specifying our inspection goals. From[9], we got the potential goals of precision of information and dependence on program inputs. From[10], we realized the goal of different approaches to dynamic analysis would result in different quality, also, we got the idea that malware would lead security issues to researchers. And we were asked to compare the outputs produce by the tools which inspired us to make it as one of our potential goals. After we discussed and prioritized those goals, we formed our inspection goals: (1) Check security vulnerabilities. (2) Get precise information about the warnings. (3) Examine the testability which is a criteria of quality. (4) Check the dependencies on program outputs. (5) Check the difference of the outputs produced by different testing tools.

For the first goal, because We have to provide a good privacy of users' personal data because of social nature of BuddyPress. From the perspective of analysis management the use of dynamic analysis has a great advantage. In the context of the dynamic analysis, the search for security issues is the search for issues in the most used and executed parts of code. It means that we will likely locate and resolve most conspicuous for hypothetical attackers problems of our program as well. In that way, we will be able to quickly prioritize details of our “ To Do ” list which will speed up security review. For the second goal, since we got many false positive warnings from static analysis part, so we want to get some more accurate information about those warnings so as to find more real faults. For the third goal, testability is the tendency for software to find its defects or faults during the testing phase, this can be a most important issue for verification and the quality assurance of the software. Dynamic testing is done for assessing the ease of the software program by giving the input and examining the outputs. In this part, we chose two tools to test BuddyPress, we expect to test BuddyPress's testability through the execution time and warnings finding by the tools. For the fourth goal since we will get many warnings from the tools, we want to find the dependencies between the warnings which will be good for improving the product's quality. For the fifth goal, we'd like to investigate the differences between the outputs provided by the tools, from which we would get some useful information in explaining these differences. That's all the reasons for our inspection goals.

VIII. SELECTED TOOLS AND ALTERNATIVE TOOLS

We chose two tools for the dynamic analysis of BuddyPress, they are PHP Vulnerability Hunter and OWASP ZAP, the first tool was used for vulnerability testing, and the other was used for penetration testing. Vulnerability testing looks for known vulnerabilities in your systems and reports potential exposures[11]. Penetration testing is the most fre-

quently and commonly applied of all software security best practices, in part because it's an attractive late life-cycle activity. Once an application is finished, its owners subject it to penetration testing as part of the final acceptance regimen[12].

1. Selected Testing Tools for Dynamic Analysis

(1) PHP Vulnerability Hunter

PHP Vulnerability Hunter is an whitebox fuzz testing tool capable of detected several classes of vulnerabilities in PHP web applications. PHP vulnerability hunter uses a combination of static and dynamic analysis to automatically map the target application. Because it works by instrumenting application, PHP Vulnerability Hunter can detect inputs that are not referenced in the forms of the rendered page. PHP Vulnerability Hunter is aware of many different types of vulnerabilities found in PHP applications, from the most common such as cross-site scripting and local file inclusion to the lesser known, such as user controlled function invocation and class instantiation. PHP Vulnerability Hunter can get measurements of how much code was executed during a scan, broken down by scan plugin and page[13]. PHP Vulnerability Hunter mainly implements three techniques which are automated input mapping, several scan modes and code coverage. They were implemented through the scanning phases. The PHP Vulnerability Hunter scans through three phases: Initialization, Scan, and Uninitialization Phase. During the initialization phase, code is annotated and static analysis is performed on code to detect inputs. After code is annotated and static analysis is performed, the scanning starts to detect bugs inside the code. This is where dynamic analysis comes into play. Dynamic Analysis is performed to discover new inputs or bugs within the code. Once the scan is complete, all of the files are restored from backups made from initialization phase and results are posted[14].

(2) OWASP ZAP

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing as well as experienced security professionals[15]. ZAP implements four techniques[16]:

A. **Proxy browser based regression tests** through ZAP in order to explore the application in a realistic way.

In order to explore an application manually you should configure your browser to proxy via ZAP. If you are using a recent version of Firefox then the easiest way to do this is via the 'Plug-n-Hack' button on the Quick Start tab. You can also manually configure all modern browser to use ZAP as a proxy.

ZAP is an intercepting proxy, which means you can change requests and responses on the fly. All data that passes through ZAP can be changed, including HTTP, HTTPS, WebSockets and postMessage traffic.

B. **Use the spiders** to discover content not covered by regression tests.

ZAP can only attack the pages of an application that it is aware of, which means that you must explore your application in some way. The Quick Start tab uses the ‘traditional’ spider which discovers links by examining the HTML in the application responses.

C. **Run the active scanner** to attack the application.

ZAP passively scans all of the requests and responses that it discovers via the spiders or that are proxied through it from your browser. Passive scanning does not change the responses in any way and is therefore always safe to use. Scanned is performed in a background thread to ensure that it does not slow down the exploration of an application. Passive scanning is good for finding a limited number of potential vulnerabilities, such as missing security related HTTP headers. It can be an effective way to get a sense of the state of security in a given web application.

Active scanning attempts to find potential vulnerabilities by using known attacks against the selected targets. As active scanning is an attack on those targets it is completely under user control and should only be used against applications that you have permission to test.

D. **Read the alerts** found and report any new vulnerabilities.

After you executed the specific software, you'll get different types of warnings, on the right interface, you can see the detail information about the specific warning, also, you can edit the warning's risk level based on your own experience and save it.

2. Alternative tools related to our goals

According to our inspection goals, we found some alternative tools, each of them might meet one or two of our goals, for example, taint could find some security vulnerabilities. But all of the alternative tools could not generate test cases automatically which was the reason why we didn't choose them. Since we were not familiar with PHP language, if we write test cases manually to test our project, we may make some mistakes and get incredible results. The following content is the descriptions of four alternative tools and their properties.

(1) PHPUnit

PHPUnit is a unit testing framework, it devides the code into many independent parts and tests each parts isolated, also provides a strict, written contract that the piece of code must satisfy.

PHPUnit can find problems early in the development cycle. It includes bugs and and flaws or missing parts of the specification for the unit. The process of writing a thorough set of tests forces the author to think through inputs, outputs, and error conditions, and thus more crisply define the unit's desired behavior. Allowing the programmer to refactor code, and make sure the module still works correctly. The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Reducing uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

However, PHPUnit has some disadvantages which hindered us to choose it as our selected tools. Sometimes it will test multiple functions and it would require having multiple @covers annotations. Another risk is if you change the method or function name and don't update the @covers annotation, then your code coverage won't cover the function or method you actually tested. Besides, PHPUnit will take too much time to writing up tests, some of the IDEs out there will auto-generate a set of basic tests, but writing good complete tests for your code takes some time.

(2) Taint

Taint is an extension to PHP which is used for detecting XSS codes (tainted string), SQL injection vulnerabilities, shell inject, etc and can be considered as dynamic analysis tool designed for security review. The analyzing process carried out only when web application's pages are requested. The extension works transparently on the server side and consumes very small CPU time. When Taint locates vulnerabilities it aborts execution of tainted code and instead outputs a warnings (user can configure details of the output). This can lead to breakage of web-application's layout making information collection difficult for the tester. The extension don't comes with PHP standard package. Users have to download source code of version compatible with his or her PHP server, compile it by himself/herself, and properly configure the server.

When Taint's capabilities are fits to our first and second goal (security review, precise information about warnings) its usability is questionable and that makes us to consider this tool not suitable to our case.

(3) Selenium

Selenium is a set of different software tools each with a different approach to supporting test automation. The entire suite of tools results in a rich set of testing functions specifically geared to the needs of testing of web applications of all types. These operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior. One of Selenium's key features is the support for executing one's tests on multiple browser platforms.

Selenium IDE (Integrated Development Environment) is a prototyping tool for building test scripts. It is a Firefox plugin and provides an easy-to-use interface for developing automated tests. Selenium IDE has a recording feature, which records user actions as they are performed and then exports them as a reusable script in one of many programming languages that can be later executed.

Selenium-Grid allows the Selenium RC solution to scale for large test suites and for test suites that must be run in multiple environments. Selenium Grid allows you to run your tests in parallel, that is, different tests can be run at the same time on different remote machines.

(4) Atoum

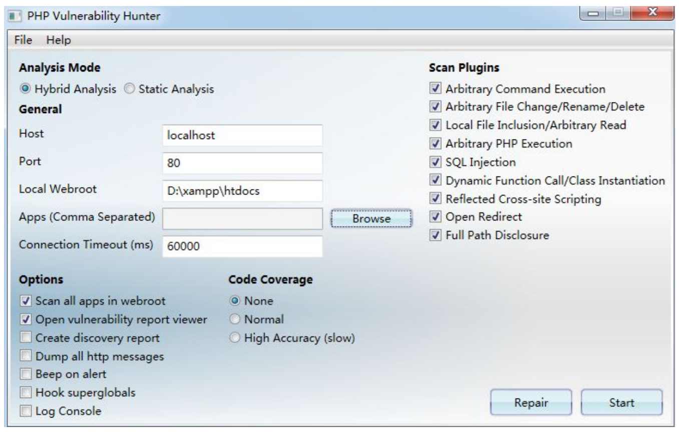
Atoum is a simple, modern and intuitive unit testing which is similar to PHPunit and has its own specialities: Support for PHP name-spaces; Each test can be run directly. Tests can be run quickly and Atoum performs rigid checking. It is simple and easy to learn; Its syntax is similar to English language. Even though it changes constantly, due to its backward compatibility it is preferred. Virtual streams to mock file system. Adapters to fake PHP functions. Full-featured mock engine. Tests isolation and parallelization to get better performances and avoid side-effects on tests. It support for xUnit and Clover reports.

Atoum's capabilities meets our 1st, 2nd and 3rd goals but testing with this tool creates some problems which leads to a race condition, so this tool is not suitable for our case.

IX. OUTPUTS COMPARISON AND PERFORMANCE EVALUATION

1. Outputs produced by the tools

When using PHP vulnerability Hunter as the tool to scan BuddyPress, we got the following results, Figure 5 shows the settings we set, and Figure 6 show the outputs produced by PHP Vulnerability Hunter.



Analysis Mode
[✓ Hybrid Analysis] Static Analysis

General

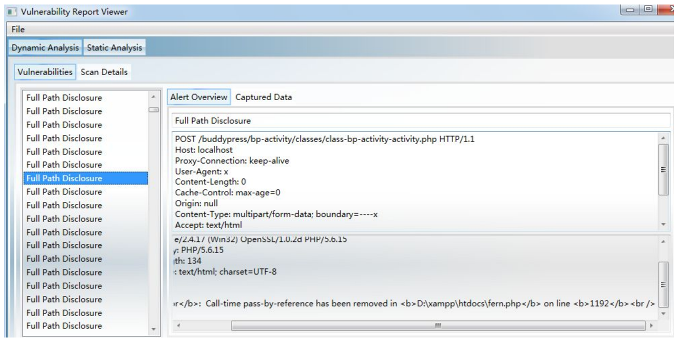
Host	localhost
Port	80
Local Webroot	D:\xampp\htdocs
Apps (Comma Separated)	
Connection Timeout (ms)	60000

Options
- ✓ Scan all apps in webroot
- ✓ Open vulnerability report viewer
- ☐ Create discovery report
- ☐ Dump all http messages
- ☐ Beep on alert
- ☐ Hook superglobals
- ☐ Log Console

Code Coverage
[✓ None] Normal High Accuracy (slow)

Scan Plugins
- ✓ Arbitrary Command Execution
- ✓ Arbitrary File Change/Rename/Delete
- ✓ Local File Inclusion/Arbitrary Read
- ✓ Arbitrary PHP Execution
- ✓ SQL Injection
- ✓ Dynamic Function Call/Class Instantiation
- ✓ Reflected Cross-site Scripting
- ✓ Open Redirect
- ✓ Full Path Disclosure

Figure 5 Set Settings for PHPVH



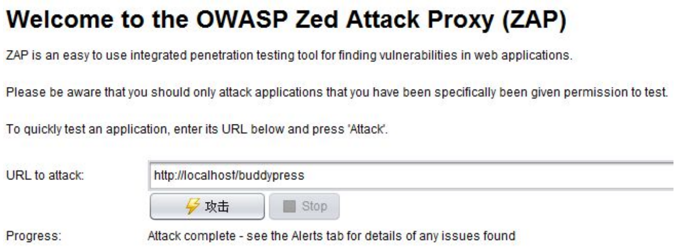
```
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
Full Path Disclosure
```

```
Full Path Disclosure

POST /buddypress/bp-activity/classes/class-bp-activity-activity.php HTTP/1.1
Host: localhost
Proxy-Connection: keep-alive
User-Agent: x
Content-Length: 0
Cache-Control: max-age=0
Origin: null
Content-Type: multipart/form-data; boundary=----x
Accept: text/html

e/2.4.17 (Win32) OpenSSL/1.0.2d PHP/5.6.15
y: PHP/5.6.15
th: 134
: text/html; charset=UTF-8

r</b>:  Call-time pass-by-reference has been removed in <b>D:\xampp\htdocs\fern.php</b> on line <b>1192</b><br />
```

Figure 6 Outputs of PHPVH

The outputs produce by PHPVH shows that it detected 318 warnings, even though we want to detect all types of vulnerabilities, but it only detected the type of full path disclosure.

When using ZAP as the tool to attack BuddyPress, we gor the following results. Figure 7 shows the attack path we chose, Figure 8 shows the outputs produced by ZAP.


Welcome to the OWASP Zed Attack Proxy (ZAP)

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

To quickly test an application, enter its URL below and press 'Attack'.

URL to attack: http://localhost/buddypress

Progress: Attack complete - see the Alerts tab for details of any issues found

Figure 7 Attack Path

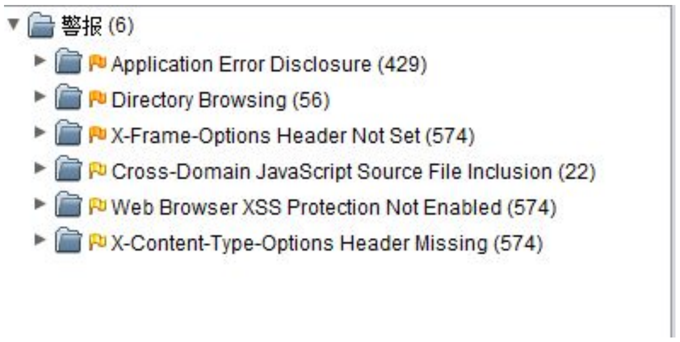

```
警报 (6)
  Application Error Disclosure (429)
  Directory Browsing (56)
  X-Frame-Options Header Not Set (574)
  Cross-Domain JavaScript Source File Inclusion (22)
  Web Browser XSS Protection Not Enabled (574)
  X-Content-Type-Options Header Missing (574)
```

Figure 8 Outputs Produced by ZAP

From the figure above, we know that ZAP detected 2229 warnings which including 429 application error disclosure warnings, 56 directory browsing warnings, 574 x-frame-options header not set warnings, 22 cross-domain javascript source file inclusion warnings, 574 web browser xss protection not enabled warnings and 574 x-content-type-options header missing warnings.

2. Outputs Comparison

When we looked into the details of the outputs produce by these two tools. We found the following differences which is showed in table 2.

Table 2 Differences Comparison

Categories	PHPVH	ZAP
Vulnerability Types	One type-full path disclosure	Many types
Number of Warnings	318	2229
Solutions to Warnings	No	Yes
Execution Time	15mins	18mins
Warning description	General	Detail

If we compare the outputs of the two testing tools, we can find that PHPVH only detected full path disclosure warnings, Full Path Disclosure (FPD) vulnerabilities enable the attacker to see the path to the webroot/file. The risks regarding FPD may produce various outcomes. For example, if the webroot is getting leaked, attackers may abuse the knowledge and use it in combination with file inclusion vulnerabilites to steal configuration files regarding the web application or the rest of the operating system[17]. However, ZAP detected many types of warnings, such as application error disclosure, directory browsing, etc. This because PHPVH's mapping mechanism, there's no other inputs exist in BuddyPress which related to other warning types except for full path disclosure[13]. ZAP implements the techniques of active and passive scanning and changing requests and responses and it can detect more types of warning[16].

For the second difference, PHPVH detected 318 warnings, ZAP detected 2229 warnings. Because the different detecting mechanism of the two testing tools which result in different results. PHPVH aims at scanning the vulnerabilities by mapping inputs, ZAP aims at attacking the project via many techniques to find more potential faults.

For the third difference, ZAP provides the solutions to warnings, but PHPVH didn't. I think this may due to the internal working mechanisms are different, even ZAP provides the solutions, but we still can edit the warning's risk level, and some of the warnings are false positive, and the solutions may meaningless.

For the forth difference, PHPVH took 15 minutes to execute BuddyPress, ZAP took 18 minutes to do that. This because they used different techniques to calculate the execution time. There are many techniques for measuring execution time, such as Resolution, Accuracy, Granularity and difficulty[18].

For the last difference, the description of warnings in PHPVH are general, but ZAP describes more detail, you can look the details of the specific warning and it will help you to make a decision whether it's true positive or false positive. This is because of the technique of proxy ZAP used which allows it can capture more accurate information about the warnings[16].

3. Performance Evaluation

Firstly, PHPVH and ZAP are both used to detect vulnerabilities of the chosen software and they are focus on security issues, and that's what they do from our first goal of checking security vulnerabilities. Secondly, ZAP can provide more precise information of the detected information, and PHPVH can find more full path disclosure warnings which we didn't find in static analysis part, in this way, these two tools are very suitable for our second goal of getting precise warning information. Thirdly, since we used two tools to test BuddyPress, we want to test their testability based on the execution time and warning findings. And PHPVH and ZAP are two popular tools which can automatically generate test cases for dynamic analysis, so the results could be more creditable if we choose them as our tools. Fourthly, the warnings produced by PHPVH are dependent which means that warnings' order is in accordance with the folders' order of BuddyPress, we can easily make the decisions of the warnings if they are real faults. For our fifth goal, we want to check the differences of the outputs of the two tools, since PHPVH provides 318 outputs of warnings, ZAP provides 2229 outputs of warnings, and we can compare them and make our conclusion.

X. DISCUSSION AND CONCLUSION

In this dynamic analysis part, we used PHPVH and ZAP to test BuddyPress and get corresponding outputs. Through comparing the outputs, we find that ZAP can find more potential faults than PHPVH, and ZAP could provide more precise information about the warnings. Some of the warnings are same which means that one fault can be detected by PHPVH and also can be detected by ZAP. Besides, we found that these two tools' stability are not good, because if we run BuddyPress on PHPVH or ZAP twice, the number of warnings they produced may be different. And the data in table 2 was an average calculation of running three times. After our group members' discussion, we drew the following conclusion.

1. Advantages of dynamic analysis[19]:

(1) It identifies vulnerabilities in a runtime environment.

(2) Automated tools provide flexibility on what to scan for.

(3) It allows for analysis of applications in which you do not have access to the actual code.

(4) It identifies vulnerabilities that might have been false negatives in the static code analysis.

(5) It permits you to validate static code analysis findings.

(6)It can be conducted against any application.

2. Limitations of dynamic analysis[19]:

(1) Automated tools provide a false sense of security that everything is being addressed.

(2) Automated tools produce false positives and false negatives.

(3)Automated tools are only as good as the rules they are using to scan with.

(4) It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

3. Conclusion

Both static analysis and dynamic analysis have advantages and disadvantages. When we perform an analysis, it depends on the specific condition. If we have no access to source code or do not understand software builds, or we want to do penetration testing, we can conduct the dynamic analysis[20]. However, if we want to do the job right, we can combine static analysis and dynamic analysis, and try to use more tools, and finally choose the most suitable and effective tools for testing our project.

XI. CONTRIBUTION TABLE DESCRIPTION

In this section, we will make a short description of what we individually had done during the static analysis and dynamic analysis of BuddyPress. Table 3 below shows the the time we spent and the percentage we contributed to the whole work.

Table 3 Individual Contribution

Name	Time spent on the work(in hours)	Percentage of contribution to overall work result
Li Xian	40	31.7%
Fan Ling Zhi	21	16.7%
Struchkov Innokentiy	22.5	17.9%
Kothawar Sravika,	21	16.7%
Avutu Neeraj Reddy	21.5	17%

In both parts, we five members would come up inspection goals and choose tools to test our project and choose alternative tools to describe. Li Xian took the responsibilities of planning group meetings, facilitating working process, searching for literature, trying different testing tools and accomplishing the whole documentation. And the rest of our group members, each of them took one inspection goal and alternative tool to describe and make comparison, they also searched many different testing tools. Besides, all of us attended the group meetings and discussed our findings and results. We finished the whole work because of all group member's excellent contribution, thanks for all our group members!

REFERENCES

[1] https://buddypress.org/about/

[2] https://www.owasp.org/index.php/Static_Code_Analysis

[3] http://rips-scanner.sourceforge.net/

[4] http://www.coverity.com/library/pdf/effective-management-of-static-analysis-vulnerabilities-and-defects.pdf

[5] Papcun J. Integrating Static Code Analysis and Defect Tracking[D]. Masarykova univerzita, Fakulta informatiky, 2014.

[6] Koskinen T, Ihantola P, Karavirta V. Quality of WordPress plug-ins: an overview of security and user ratings[C]//Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom). IEEE, 2012: 834-837.

[7] Dahse J. RIPS-A static source code analyser for vulnerabilities in PHP scripts[J]. Retrieved: February, 2010, 28: 2012.

[8] Gomes I, Morgado P, Gomes T, et al. An overview on the static code analysis approach in software development[J]. Faculdade de Engenharia da Universidade do Porto, Portugal, 2009.

[9] Ball T. The concept of dynamic analysis[C]//Software Engineering－ESEC/FSE’99. Springer Berlin Heidelberg, 1999: 216-234.

[10] Willems C, Holz T, Freiling F. Toward automated dynamic malware analysis using cwsandbox[J]. IEEE Security & Privacy, 2007 (2): 32-39.

[11] Triware Networld Systems. http://www.tns.com/PenTestvsVScan.asp

[12] Arkin B, Stender S, McGraw G. Software penetration testing[J]. IEEE Security & Privacy, 2005 (1): 84-87.

[13] CodePlex, Project Hosting for Open Source Software. https://phpvulnhunter.codeplex.com/

[14] Klos, Galiana, Marlowe, Givens. Hypertext preprocessor (php) vulnerabilities, risk and countermeasures. Florida State University iSchool, 2013.

[15] OWASP Zed Attack Proxy Project. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[16] OWASP ZAP 2.4, Getting Started GuideOpen Web Application Security Project.

[17] Full Path Disclosure. https://www.owasp.org/index.php/Full_Path_Disclosure

[18] Measuring Execution Time and Real-time Performance: Part 1. http://www.drdobbs.com/embedded-systems/measuring-execution-time-and-real-time-p/193502123

[19] Static vs. dynamic code analysis: advantages and disadvantages, 2009. https://gcn.com/articles/2009/02/09/static-vs-dynamic-code-analysis.aspx

[20] DenimGroup. StaticAnalysisDynamicAnalysisAndHowToUseThemTogether, 2008. http://www.denimgroup.com/media/pdfs/DenimGroup_StaticAnalysisDynamicAnalysisAndHowToUseThemTogether_Content.pdf