A New Framework of Security Vulnerabilities Detection in PHP Web Application

Jingling Zhao, Rulin Gong
School of Computer, Beijing University of Posts & Telecommunications
National Engineering Laboratory for Mobile Network Security
Beijing, China
gongrulin1990@163.com

***Abstract*—Nowadays, Web applications provide us most of the internet services, but also give birth to more and more new types of internet applications. While, according to the developers' programming techniques and safety awareness, there are many kinds of web application security flaws and vulnerabilities hiding in the program. So it is very important to improve their reliability and security. Usually people use code review based on static or dynamic analysis to detect security vulnerabilities, but each method has shortcomings that can't overcome easily which can result in a big number of false positives and omission. To address this issue, this paper proposed a new framework of detecting security vulnerabilities of PHP web application. In this framework, we combine dynamic and static analysis to make full use of the advantages of the two, greatly improve the efficiency of detection. An implementation based on this framework has also been completed and it will also be presented in the paper.**

Keywords—code audit, static analysis, dynamic analysis, PHP code review

I. INTRODUCTION

Nowadays, web applications support more and more our daily activities. Unfortunately web applications are suffering from more and more attacking. According to the report of OWASP [2] in the last decade, the traditional security vulnerabilities such as SQLI (SQL injection), XSS (cross-site scripting) and so on are still the main thread to the web application.

Web applications are mostly developed by scripting language. As of January 2013, PHP was installed on more than 240 million websites and 2.1 million web servers. The CVE is a dictionary of vulnerabilities maintained by the MITRE Corp. On the whole database, about 29% of vulnerabilities (CVE) are related to PHP.

Actually there are two classical methods in scripting languages security vulnerabilities detection, static analysis and dynamic analysis. Static analysis is a very import technique which is widely used in code audit. It is used to detect errors, refactor code and follow the compilation process of the code. However, for dynamic scripting languages like PHP, there are too many features that dynamic analysis can't handle easily, for example, run-time code generation, run-time source inclusion, weak typing, dynamic typing, run-time aliasing and array creation. It is very difficult to deal with these newly added features. While , the other one, dynamic analysis is usually used in black box testing, it uses a huge number of attack vectors to simulate the behaviour of the attackers, testing if there is abnormal result, from which we can judge whether the program is reliable or not. But this method is highly blind, can result in very low efficiency.

In this paper, we combine dynamic analysis and static analysis, we firstly implement the static analysis based on HHVM (HipHop Virtual Machine) [6], from which we get the static analysis result and special path, variables, parameters and others, we then put them into the dynamic analysis to construct a dynamic test set, which will be used in real dynamic detection. In this way, we effectively make use of their both advantages, overcoming shortcomings.

The rest of the paper is organized as follows, we first introduce some PHP security vulnerabilities (section 2), then present our implement of static analysis (section 3), Section 4 describes the core of the framework *–URL Reconstructed and Fuzz based dynamic analysis*, Section 5 describes the experiment of our approach, Section 6 discusses related work and section 7 concludes.

II. PHP SECURITY VULNERABILITY

The new framework of Security Vulnerabilities Detection in PHP Web Application we propose aims at detecting nearly all kinds of PHP security vulnerabilities. While, it is a huge project for us to complete, so in this paper, we just take SQLI and XSS for examples. Actually both them are still the most devastating and common in the web applications. But, here we introduce three kinds of security vulnerabilities which are the kinds of vulnerabilities our framework aims to detect in the first step.

A. SQL Injection

SQL Injection is one of the many web attack mechanisms used by hackers to steal data from web databases. It is perhaps one of the most common application layer attack techniques used today. Nearly every web application needs to interact with a database, in which we will use SQL statements to get, delete, create or update web application data. Using the SQL Injection, attackers try to insert malicious SQL statements into the normal SQL statements, then the malicious statements work in the process of execution. Attackers can perform any database operations, even the system operation by modifying the inserted malicious. Figure 1 is an example of SQLI attack, attackers can insert malicious statements into variable $sql by

978-1-4799-8873-0/11 $31.00 © 20115 IEEE
DOI 10.1109/IMIS.2015.42

assigning malicious SQL statements to variable $_GET[‘username’] or $_GET[‘password’].

```
1 <?php
2  $user=$_GET[‘username’];
3  $pass=$_GET[‘password’];
4  $sql=”SELECT * FROM users WHERE username
   =’$user’ and password=’$pass’”;
5  $result=mysql_query($sql);
6 ?>
```

Figure 1: An example of SQL injection

If attackers assign $_GET[‘username’] by value of “’or 1=1#”, then value of variable $sql changes to ”SELECT * FROM users WHERE username=’’ or 1=1#’ and password=’xxyy’”,we can find semantic was changed by attackers, it will allow the attackers pass the identity authentication.

B. *Cross-Site Scripting*

Cross-Site Scripting (XSS) attacks are a type of injection, in which attackers can inject malicious client-side script into Web pages viewed by other users. The malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site, so that the attackers can steal all these from the client browser. Figure 2 is an example of XSS attack, attackers can easily do XSS attack by assigning $_GET[‘username’] with a piece of malicious script, usually in JS(JavaScript).

```
1 <?php
2   $name=$_GET[‘username’];
3   echo $name;
4 ?>
```

Figure 2: An example of Cross-Site Scripting

If attackers assign $_GET[‘username’] by value of “<script src=”http://evil/steal.js”></script>”, and in the steal.js which deployed in the attacker’s server, there is a piece of code like what shows in Figure 3.

```
1 alert(document.cookie)// In real attack
   scenario, the code here is much more
              aggressive.
```

Figure 3: An example of Cross-Site Scripting

Then attacker will send the constructed URL to victim, the latter click the URL, then the cookie of victim will be pop up. In real scenario, cookies will be passed to the attackers. Attackers will get the privileges belongs to normal users.

Actually XSS can be divided into three categories, 1) Stored, 2) Reflected, and 3) DOM based XSS. It differs in the way attackers use and the features it exhibit.

C. *Any File Inclusion*

File inclusion vulnerability is also a type of vulnerability most often found on websites. It allows attacker to include a file in the website or in the remote server. This can lead to something as minimal as outputting the contents of the file or more serious events such as:

- Code execution on the web server
- Code execution on the client-side such as JavaScript which can lead to other attacks such as cross site scripting (XSS)
- Denial of service (DoS)
- Data theft/manipulation

Figure 4 is an example of Any File Inclusion attack, attackers can include any file by assigning $_GET[‘filename’] with the existing filename.

```
1  <?php
2  $file=$_GET[‘filename’];
3  require_once($file);
4 ?>
```

Figure 4: An example of Any File Inclusion

In this case, attackers can assign $_GET[‘filename’] with “../conf/database.conf”.Then this piece of script will include the database configuration file of the web application, which is very dangerous.

File Inclusion can be divided into 2 categories, 1) LFI (local file inclusion), 2) RFI (remote file inclusion). LFI means attacker can include any file in the local web server like the example before, RFI means attacker can include a file which lives in remote server, in other words, attacker can execution any script.

III. HHVM Based Static Analysis

Static analysis is very necessary, because by static analysis we can go deep into the core of code to make clear how the code works and track the compilation process from which we can get more information about our research. During the static analysis, the PHP source code should be firstly parsed to abstract syntax tree (AST). AST is the middle form of the source code and also an effective and simple PHP source code data structure, then we can form control flow and data flow out of the data structure. With control flow and data flow, it will be easier for us to analyze the transfer and spread of the specific data.

Actually, there are already some kinds of parser which we can use in our research, such as Zend Engine, PhpParser, Phc, HHVM and so on. Zend Engine is developed by PHP official and is also the part of the PHP parser by default. But Zend Engine translates PHP source code into opcode which is an intermediate form in the execution of PHP like Java bytecode, so it can`t be used by us for static analysis. PhpParser, Phc, HHVM and some other parsers all can parse PHP source code into AST, and some of them even have been already used in static analysis in some very good projects. For example, PhpParser is used in the famous PHP source code static analysis project –Pixy, which is very instructive to our project. PhpParser is also used in Stranger, Phc had been used in some similar projects, which ware unfortunately stopped now.

A. *HHVM*

We choose HHVM as the front-end to translate PHP source code into AST. HHVM is an open source virtual machine designed by Facebook, it is used to compile PHP or

Hack script to speed up service delivery. Actually there are three stages in the born of HipHop Virtual Machine--HPHPC=>HPHPI=>HHVM.

- HPHPC is static compilation, that is, to convert PHP into c + +.
- HPHPI is a transitional product, similar to PHP zend virtual machine, the performance is not as good as the zend virtual machine, but can be run to check the effect .
- HHVM is on the basis of HPHPI, application of the JIT technology, performance is close to the HPHPC, According to Facebook, HHVM have gained significant improvement in speed, PHP interpreter 60% faster than currently in use, and use 90% less memory

Figure 5 is an example of AST which is given out by HHVM, the PHP source code is what shows in the Figure 2.

```
Dumping control flow: Function php$test2_php
7fd4981cfb70 (1)
  InDegree: 0
  OutDegree: 0
  -> 0x7fd49cd34ac0 StatementList(6) 2 test2.php:1@1
   -> 0x7fd49cd65e80 ExpStatement(22) 1
       test2.php:2@24
      -> 0x7fd49d3dd700 AssignmentExpression(1) 2
        (AssignEffect)(LocalAltered) test2.php:2@23
     -> 0x7fd49d3dd500 SimpleVariable(2) [name] 0
(LValue|NoLValueWrapper|AssignmentLHS|DeepAssignmentL
HS)(LocalAltered)(NoRefInfo)(NoObjInfo)test2.php:2@5
       -> 0x7fd49cd39940 ArrayElementExpression(5) 2
(AssignmentRHS)(AccessorEffect)(LocalAltered)test2.ph
p:2@23
       -> 0x7fd49d3dd600 SimpleVariable(2) [_GET] 0
     (NoLValueWrapper|AccessContext) (LocalAltered)
     (NoRefInfo) (NoObjInfo) test2.php:2@11
      -> 0x7fd49cd39800 ScalarExpression(8)
      ['username'] 0 (LocalAltered) test2.php:2@22
       -> 0x7fd49cd65f00 EchoStatement(20) 1
       test2.php:3@11
   -> 0x7fd49d3dd900 ExpressionList(0) 1
   (LocalAltered) test2.php:3@10
   -> 0x7fd49d3dd800 SimpleVariable(2) [name] 0
   (NoLValueWrapper) (LocalAltered)
   (NoRefInfo)(NoObjInfo) test2.php:3@10
   7fd4981cfbe0 (0)
 InDegree: 0
 OutDegree: 0
```

Figure 5: An example of HHVM AST

B. Data flow analysis

Data flow analysis is the core of static analysis, the purpose of the static analysis is to track data transmission and communication.

Data flow analysis can be divided into 3 steps:

- Parse PHP source code into AST
- Build CFG(Control Flow Graph) from AST
- Travel the whole CFG, detect vulnerabilities, record variables and parameters

HHVM has finished the first step for us, because there is a parser in HHVM which can parser PHP source code to AST very well.

Then, for the second step, we should construct CFG from the AST. The AST is composed of a number of blocks, each block represents several program statements whose executions can be combined into one in order. The CFG is composed of a number of nodes, each node corresponds to a block and edges which are used to represent jumping in the control flow. Then we travel all the AST, translate blocks to nodes and connect them with edges (We use the forward data-flow analysis) to keep the right order and logic.

The third step, we need to travel the whole CFG to follow and analyze possible vulnerabilities. In our project, for example, we first mark some specific variable as pollute such as $_GET or $_POST, because these variables can be assigned by any value including SQL Inject malicious statements or Cross-Site script. Then we track the transmission and communication of the pollute variables, record them in the data structure.

C. Data Storage structure and extraction

As a simple static analysis, section 3.2 is the whole content, but in our research that was not enough, what we need is all the variables and parameters. In our research, we design a kind of data structure to store all these. Figure 6 is the basic data structure.

<table>
<tr><td>Index</td><td>value</td><td>…</td><td>index</td><td>value</td></tr>
<tr><td></td><td>Value node</td><td></td><td></td><td>Value node</td></tr>
<tr><td colspan="2">Index node</td><td colspan="3">Index node</td></tr>
<tr><td colspan="5">Storage node</td></tr>
</table>

Figure 6: The data structure of Static Analysis

In this data structure, we design a storage node for each variable like $username, $_GET, $_SERVER and so on. Each storage node can be divided into three categories, Symbol Table, Array and Object. In every storage node, there are some index nodes which can also be divided into three categories, NoRef, RefToIndex and RefToStorage. This will be necessary and useful when we come to alias analysis. In each index node, there is only one value node that is the value of the variable. In each value node, there are three marks Tianted, UnTainted and UnDef which can help to mark if the variable is polluted.

For example, $_GET[`username`]=`xiaowang`, _GET uses a storage node, username uses an index node in the storage node, `xiaowang` uses a value node in the index node.

In the static analysis, we realized the function of basic analysis of the pollution, so that we can give out the determined analysis results. On the other hand, we recoded all the variables for the next step – Dynamic Analysis.

IV. URL Reconstruct and Fuzz Based Dynamic Analysis

Dynamic analysis is always used in black box testing. In our search, we use fuzz testing for dynamic analysis. Even though dynamic analysis has a big disadvantage which is blindness, it may reduce the efficiency of analysis, but it is also necessary. The reason is dynamic analysis has a very low rate of false positives, which has long been criticized in static analysis. On the other hand, dynamic analysis is much more trusted because its analysis process is very similar to attacker`s behavior, and it is easy for us to perform special or temporary test by modifying the testing set.

In our research, we make dynamic analysis into 3 steps.

- Crawl all the URLs
- Break up the URL parameter and value
- Restructuring the URL parameter and value, doing fuzz test

In the section 4.1-4.2, we will introduce the previous two, the last and also the most important one we will introduce in the section 4.3.

A. Crawler

We developed a crawler in python. Python is script language which is very good at writing crawler. It has a lot of development packages which we can use in our research to reduce the complexity of the development. In the real project, we use multithreaded programming to improve the efficiency of crawl, and we put the main function in to a class, it will be convenient for second development or modifying.

Also, python provides us HTML data parsing packages, we choose to use a package named `HTMLParser`. With this package, we rewrite the HTML tags analytic function to find the URL fragments we need, and construct real URLs.

B. URL Treatment

At the early stage of our research, we read the dozens of popular CMS (Content Management System) code written in PHP including Discuz, Metinfo, PhpYun, Phpmywind and so on. We notice that nearly all these mature CMS adopt MVC architecture. MVC means M(module) V(view) C(control). It divides PHP web application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. The model directly manages the data, logic and rules of the application. A view can be any output representation of information, such as a chart or a diagram; multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the controller, accepts input and converts it to commands for the model or view.

The URLs of PHP web application using MVC architecture have obvious characteristics. Figure 7 is an example, the second URL is from the web application developed by MVC architecture.

```
1.http://www.test.com/user/info/show/index
.php?stu_id=2013110736
2.http://www.test.com/index.php?role=user&
module=info&op=show&stu_id=2013110736
```

Figure 7: An example of normal URL and URL of CMS in MVC architecture

We can easily find variables in the URL are different. Variables like `role`, `module` and `op` work in the Control layer, it directs URL to the corresponding code which is common written in class or function to call it. While these variables can`t be fuzzed easily, because it is not involved the specific code, just works like a guider, fuzzing for them makes no sense. But for variables like `stu_id`, it works.

In this step, we just break the URL into some parameters and values and divide them into 2 kinds, one for free fuzz parameters like `stu_id`, these parameters can by freely fuzzed, one for limit fuzz parameters like `role`, `module` and `op`, which can only be valued by given values,.

C. Combination

From the section 4.2, we get 2 kinds of parameters also with their values. From the section 3, we get the all the variables inside the code, these variables are in fact can`t be obtained by dynamic analysis. Here, we are going to combine them together.

First we merge all the variables and parameters, from which we choose all the limit fuzz parameters and their values, just like what show in the Figure 8.

```
Example URL:
http://www.test.com/index.php?role=user&modu
le=info&op=show&stu_id=2013110736
   role: user, teacher, admin, anyone
   module: info, news, class, mail,
           favourite, comment, detail
   op: show, add, edit, del
   api: normal, mobile
```

Figure 8: An example of merged and classified parameters

Then we freely combine all this limit fuzz parameters to construct new URL links which can direct it to every function of the source code.

Figure 9 is an example of the intermediate results.

```
Constructed URLs:
http://www.test.com/index.php?role=user&modu
le=info&op=show&api=normal
http://www.test.com/index.php?role=user&modu
le=news&op=show&api=web
http://www.test.com/index.php?role=admin&mod
ule=info&op=show&api=normal
http://www.test.com/index.php?role=anyone&mo
dule=favourite&op=edit&api=web
……
```

Figure 9: An example of transformation and combination to URL

In this example, we can get as most as 4*7*4*2 = 224 URLs, but that was not the end. Then we add the free fuzz parameters to the URL like `stu_id` ,`id` ,`q_id` and so on.

Figure 10 is an example of the final constructed results.

```
Constructed URLs:
http://www.test.com/index.php?role=user&mod
ule=info&op=show&api=normal&stu_id=1&id=2&q
_id=3
http://www.test.com/index.php?role=user&mod
ule=news&op=show&api=web&stu_id=1&id=2&q_id
=3
http://www.test.com/index.php?role=admin&mo
dule=info&op=show&api=normal&stu_id=1&id=2&
q_id=3
http://www.test.com/index.php?role=anyone&m
odule=favourite&op=edit&api=web&stu_id=1&id
=2&q_id=3
……
```

Figure 10: An example of the URLs after transformation

At last, we get the full URL links like what is showed in Figure10, we then do fuzz testing to free fuzz parameters of each link with malicious code set which is consisted of a lot of scripts who can cause SQL injection, XSS, or something like that. From the fuzz result, we will be able to get valuable dynamic analysis results.

Then we get double result, static one from HHVM based static analysis and dynamic from fuzz testing. The front one can give us a very good analysis result as just a normal static analysis. As we know, it will give out some false positives even though we have made a lot of optimization and some determined result. The determined result can be listed as one part of the final result. With variables and values it provided, the dynamic part breaks up and recombine the URL links to optimize the efficiency of fuzz, give out the dynamic analysis result.

From the combination, we can clearly find the key is the efficiency of dynamic analysis. Static analysis provides dynamic one a lot of variables and values inside the code, which can help later one`s fuzz testing cover more code than before. At the same time, it can also provide the later one more parameters which can`t be find, and that gives dynamic analysis a wider view about the code, of course can give out a better result.

V. Experimental Evaluation

In order to evaluate our approach, we use the micro-benchmark described in [12] for our last paper [11] in which we realize our static analysis. The micro-benchmark consists of 110 small hand-written PHP programs comprising 55 test cases, each one focusing on a single language feature, vulnerability type or sanitization routine. Each test case consists of a BAD program that includes a vulnerability, and an OK program which patches the vulnerability. However, tools may pass the BAD test or OK test with lucky guess, so we give discrimination results, a tool passes the discrimination test for a single case if and only if it passes both the BAD and OK tests for that case. Table 1 is the experimental result, it clearly shows that some indicators slightly ascending.

subject	*BAD tests*			*OK tests*			*Discriminates*		
	pass	*total*	*perc*	*pass*	*total*	*perc*	*Pass*	*total*	*perc*
All	24	30	80%	30	30	100%	24	30	80%
Aliasing	4	4	100%	4	4	100%	4	4	100%
Arrays	2	2	100%	2	2	100%	2	2	100%
Constants	2	2	100%	2	2	100%	2	2	100%
Functions	4	5	80%	5	5	100%	4	5	80%
Dynamic Inclusion	3	3	100%	3	3	100%	3	3	100%
Object	6	8	75%	8	8	100%	6	8	75%
Strings	3	3	100%	3	3	100%	3	3	100%
Variable variables	0	3	0%	3	3	100%	0	3	0%

(a)Language support

Subject	*BAD tests*			*OK tests*			*Discriminates*		
	pass	*total*	*perc*	*pass*	*total*	*perc*	*pass*	*total*	*perc*
All	11	18	61%	17	18	94%	8	18	44%
Argument injection	0	1	0%	1	1	100%	0	1	0%
Command injection	0	2	0%	2	2	100%	0	2	0%
Code injection	0	2	0%	2	2	100%	0	2	0%
SQL injection	6	6	100%	5	6	83%	5	6	83%
Server-side include	2	2	100%	2	2	0%	0	2	0%
XPath injection	0	2	0%	2	2	100%	0	2	0%
Cross-site scripting	3	3	100%	3	3	100%	3	3	100%

(b)Injection vulnerability detection

Subject	*BAD tests*			*OK tests*			*Discriminates*		
	pass	*total*	*perc*	*pass*	*total*	*perc*	*pass*	*total*	*perc*
All	7	7	100%	4	7	67%	4	7	67%
Regular expressions	2	2	0%	2	2	100%	2	2	100%
SQL injection	1	1	100%	1	1	100%	1	1	100%
Strings	2	2	100%	0	2	0%	0	2	0%
Cross-site scripting	2	2	100%	1	2	50%	1	2	50%

(c)Sanitization support

Table 1: Our approach micro-benchmark test results

In this paper, we focus on the combination of static analysis and dynamic analysis, but micro-benchmark test can just evaluate static analysis. Actually, the coverage of the dynamic analysis is also an important part, because our research is to design a framework which contains dynamic and static ones, and for dynamic analysis what we do is to enlarge the code coverage to achieve the accuracy, so we evaluate our framework with some mature CMS with MVC architecture adopted. We use TipaskV2.5, YxcmsV1.2.6, EspcmsV6.2 and some other CMS, the result of code coverage is nearly 92% much more than 76% when we just use crawler to get the URL links before, with bigger coverage, our framework can get a better result theoretically.

VI. Related Work

Before writing paper, we have studied some work similar to ours`, Here are some approaches that can deal with detection of web application security vulnerabilities especially for PHP in static analysis .Some of them even give the thinking about combining static and dynamic analysis, but the methods differ from ours`.

Huang et al. [1] were the first to consider the static analysis of PHP language, they used a lattice-based analysis algorithm derived from type systems and type state, and compared it to a technique based on bounded model checking in their follow-up paper, while their works' result were not proved by researchers and their tools were limited in PHP version, only for 4.

Pixy [3] presents taint analysis of PHP programs which provides information about the flow of tainted data using dependence graph. It greatly helped our research in static analysis even though we choose HHVM for our static analysis engine. The latest version support PHP5, which we still consider is the best open source static analysis tool.

Balzarotti et al. [4] combine static and dynamic analysis techniques to verify PHP programs. Their approach is based on Pixy and has the same limitations. They use dynamic phase to examine all those program paths from input sources to sensitive sinks that the static analysis has identified as suspicious. But clearly they don`t consider the feature of URL links of MVC pattern .

Yu et al. [5] developed an automata-based string analysis, they use Extended Static Single Assignment to simplify taint flow analysis which is very creatively and give us another choice.

Ariel et al. proposed a kind of dynamic analysis and realized it, its advantage is similar to ours`, but the recording of the variable spreading is weaker than static analysis, but we

adopt the thinking about dynamic analysis and will try to realize it in our later version in some way.

Biggar et al. [8] performs context-sensitive, flow-sensitive, inter-procedural static analysis for PHP. They model most features of PHP, their analysis is the most comprehensive in static analysis.

Rimsa et al. [10] they use Extended Static Single Assignment(ESSA) to simplify taint flow analysis which is very creatively and give us another method and choice.

Hauzar et al. [9] try to model the data model using the associative arrays, it helps us a lot when we model data model in static analysis, at last we don`t adopt arrays because of time and energy.

Cuibj et al. [11] who are in our research group have developed the first version about the static analysis, it is based on the HHVM and can give out a good analysis result. In this paper we optimize the static analysis and from that we find the new thinking of combination and realize it.

VII. CONCLUSIONS

In the area of PHP source code audit, no matter dynamic analysis or static analysis, each one has great defect that can`t be solved perfectly right now. While until now, what related works do is just to repair the defect of the two, but not propose a new method or thinking. We think that was not enough.

In this paper, we design HHVM based static analysis tools, realize the traditional static analysis and add various PHP features support. What's more, we creatively combine the static analysis and dynamic analysis, static analysis gives a big hand to dynamic one, makes up the shortcoming of blindness, greatly improve the efficiency.

We evaluate our approach by using micro-benchmark test cases like what we do in [11]. Also we used it to find bugs in some old-version CMS, analyzing the result about detection rate and coverage. We are going to use it find bugs in real web application when we finish all the work in development. Our future work will focus on adding more kinds of vulnerabilities detection and doing more research in dynamic analysis.

Even though we propose a framework of detecting vulnerabilities in PHP source code, this framework and thinking can also be used on other languages for web development, actually we are realizing it on JSP web application.

VIII. ACKNOWLEDGEMENT

This work was supported by National Natural Science Foundation of China (No. 61170268, 61100047, and 61272493)

REFERENCES

[1] Huang, Y. W., Yu, F., Hang, C., Tsai, C. H., Lee, D. T., & Kuo, S. Y. (2004, May). Securing web application code by static analysis and runtime protection. In Proceedings of the 13th international conference on World Wide Web (pp. 40-52). ACM.

[2] OWASP. Top ten project. http://www.owasp.org/, 2013.

[3] Jovanovic, N., Kruegel, C., & Kirda, E. (2006, May). Pixy: A static analysis tool for detecting web application vulnerabilities. In Security and Privacy, 2006 IEEE Symposium on (pp. 6-pp). IEEE.

[4] Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G. (2008, May). Saner: Composing static and dynamic analysis to validate sanitization in web applications. In Security and Privacy, 2008. SP 2008. IEEE Symposium on (pp. 387-401). IEEE.

[5] Yu, F., Alkhalaf, M., & Bultan, T. (2010). Stranger: An automata-based string analysis tool for PHP. In Tools and Algorithms for the Construction and Analysis of Systems (pp. 154-157). Springer Berlin Heidelberg.

[6] HHVM. A virtual machine designed for executing programs written in Hack and PHP. https://github.com/facebook/hhvm

[7] Ariel Futoransky, Eze quie l Gutesman and Ariel Waissbein. A dynamic technique for enhancing the security andn privacy of web applications. In:Black Hat USA Briefings(August 2007)

[8] Biggar, P., & Gregg, D. (2009). Static analysis of dynamic scripting languages. Draft: Monday 17th August.

[9] Hauzar, D., Kofroň, J., & Baštecký, P. (2014). Data-flow Analysis of Programs with Associative Arrays. arXiv preprint arXiv:1405.1116.

[10] Rimsa, A., d'Amorim, M., & Pereira, F. M. Q. (2011, January). Tainted flow analysis on e-SSA-form programs. In Compiler Construction (pp. 124-143). Springer Berlin Heidelberg.

[11] Baojiangcui,Zhingpenghuo,Weifuzhou,Rulingong(2014) Static Detection of PHP Web Application Security Vulnerabilities. ICCSA2014(Paper ID:SA3280)

[12] de Poel, N. L., Brokken, F. B., & de Lavalette, G. R. R. (2010). Automated security review of php web applications with static code analysis. Master's thesis, 5.