International Journal of Information and Computation Technology.
ISSN 0974-2239 Volume 3, Number 5 (2013), pp. 391-398
© International Research Publications House
http://www. irphouse.com /ijict.htm

Securing PHP Based Web Application Using Vulnerability Injection

[#1]Jamang Jayeshbha Bhalabha, [#2]Amit Doegar and [#3] Poonam Saini

[#1,2]National Institute of Technical Teachers Training and Research, Chandigarh.
[#3]PEC University of Technology, Chandigarh.

Abstract

In the past few years, Vulnerability injection technique has been given relatively lesser attention by the research community. The probable reason might be positioned in its objective which is apparently contrary to the purpose of making applications more secure. However, the same technique can be used in various research applications such as the automatic creation of vulnerable code for the purpose of educational application, systems requiring in-depth testing and evaluation of both vulnerability scanners and security teams. In the contemporary literature, a handful of work exists for securing PHP based web applications using vulnerability injection. Most of the protocols exploit the vulnerability detection rather than vulnerability injection which is a major drawback for the design of real- time and secure applications. Moreover, the existing protocols employ single type of vulnerability which limits its practical usage. Also, the dynamic nature of PHP makes the design of secure PHP based applications more challenging. In the light of above said limitations, we propose to design a prototype which targets PHP based web applications. Further, the prototype is injected with taint- style vulnerabilities i.e., Sql injection, cross-site scripting etc. The thesis would exploit vulnerabilities present in the latest versions of well known PHP applications. This, in turn, would provide a better insight into the most common type of vulnerabilities.

Index Terms: Vulnerabilities, Injection, PHP, Static analysis.

1. Introduction

Web applications in particular are being used today as front ends to many security critical systems (e.g., home banking and e-commerce), but due to their high exposure, they are particularly susceptible to being heavily attacked. This means that they require special care to make them secure and resilient against these threats.In a world growing every day more dependent on computer systems, we are slowly becoming aware of the need for security in applications to prevent attacks that could result in the loss of material, money or even human lives. Security teams and vulnerability scanners are some of the approaches used to eliminate the flaws that weaken the applications, the vulnerabilities. However, without a systematic way of evaluating them, it is difficult to choose the right security tool for the job, or find out what knowledge the security team is lacking. Evaluating vulnerability scanners might be done by comparing the results from each other, to see which tool finds more vulnerabilities with less false positives. This however only gives a result relative to the other vulnerability scanners, meaning that if all vulnerability scanners are bad the this method will not produce meaningful results. Evaluating a security team is even harder since most companies do not have the resources required to employ more than one security team, and even if they have, they probably would not want to have them doing the same work just for the sake of comparing the results.The best way to evaluate both security teams and vulnerability scanners is to have them work over an application whose vulnerabilities are already known, in order to be able to check not only which types of vulnerabilities they find best but also which ones they failed to find at all. While this can be done by using some older version of an application, with some known vulnerabilities, finding a version that has just the right vulnerabilities for the evaluation might be very difficult. Therefore, a method is necessary to give to the evaluator the control over which vulnerabilities are inserted in the test application, to ensure that the evaluations are effective and efficient. A vulnerability injection tool can also be used to estimate the number of vulnerabilities that are left to correct in an application after it has already been tested [1]

2. Literature Survey

There is not much work on vulnerability injection in our current day. The reasons for this might have to do with the fact that it looks counterproductive to the end goal of making software more secure and also that it is as hard (if not harder) as the opposite problem of detecting vulnerabilities. This might explain why only one team, Fonseca has focused on this exact problem .Fonseca builds upon this concept in two different works: in [5] they present the concept and delve deeper into it in [6]. They implement the concept by building a vulnerability injection tool for PHP that injects SQL Injection vulnerabilities. Fault injection has been used since the 1970s [7] to test the dependability of fault tolerant systems. By injecting faults into specific components of a system it was possible to verify whether the system could tolerate faults in those components, and it was also possible to learn the impact of the eventual failures. The first approach for fault injection was to inject faults directly in the hardware. These were used mostly to test the tolerance of the hardware to occasional failures and to evaluate the dependability of systems where high time resolution is required. Static Analysis is a form of analysis of computer software where the code is analyzed without being executed. It can be used to gather a lot of information about the code, from defects or bugs, unreachable code and unused variables, whether it adheres to good programming practices, software metrics, and can even be used to formally prove whether the application has some given properties. While it can also be applied on binary code, it is most commonly applied to source code. In fact, compilers use it to do their job and thus are a good model to verify what and how is static analysis done [8]. Compilers are usually composed of three main components: the front end where source code is parsed, its syntax and semantics is validated and is transformed into an intermediate representation ready for further analysis; the middle end where the code is subject to static analysis in order to be optimized and the back end where the code will be finally translated into the output language. Static analysis tools typically have a similar work flow to the front and middle end except for the optimization part which is where tools diverge according to their purpose, and thus these two components are worth a deeper look at. While the purpose of vulnerability detection is the complete opposite of what this work tries to achieve, tools that try to detect vulnerabilities statically have many characteristics in common with a potential vulnerability injection tool, mainly related to the static analysis that it needs to perform and to the vulnerability modeling that is required both to detect vulnerabilities or to detect potential injection locations. It is thus interesting to analyze tools that perform vulnerability detection on PHP.RIPS [13] is written in PHP and thus requires setting up a local web server in order to use it. Once that step is done it can be controlled completely using a practical web interface that allows scanning files for vulnerabilities while customizing the verbosity level, the vulnerabilities to analyze, and even the code style in which results are presented. Pixy is written in Java and is a command line application that can detect Cross-site Scripting and SQL Injection vulnerabilities using taint analysis, which makes it both less user-friendly and less complete than RIPS. It is run by specifying one file where vulnerabilities will be searched, which is then presented in a summary in the terminal. Alternatively, a DOT file can be produced, which can be visualized using the dot application from Graphviz [14] that represents the taint path that causes the vulnerability.

3. Proposed Prototype

The modifications made to Rips were the ones marked in red in Figure.1 The addition of a injectVulns() method in the DepClients and of a Vulnerability Injector module were required in order to transform it into a Vulnerability Injection Tool. The rest of the modifications were made in order to allow for the Vulnerability Injection Tool to be able to inject and detect some more vulnerability.

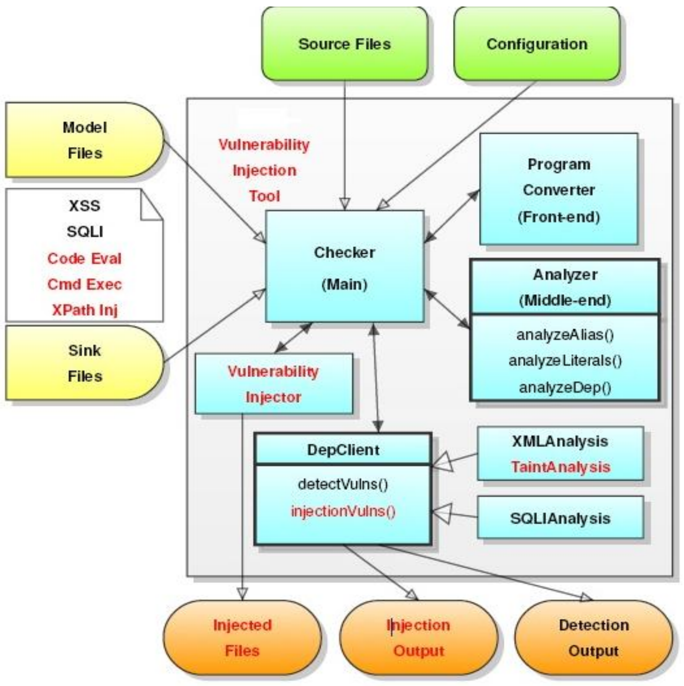


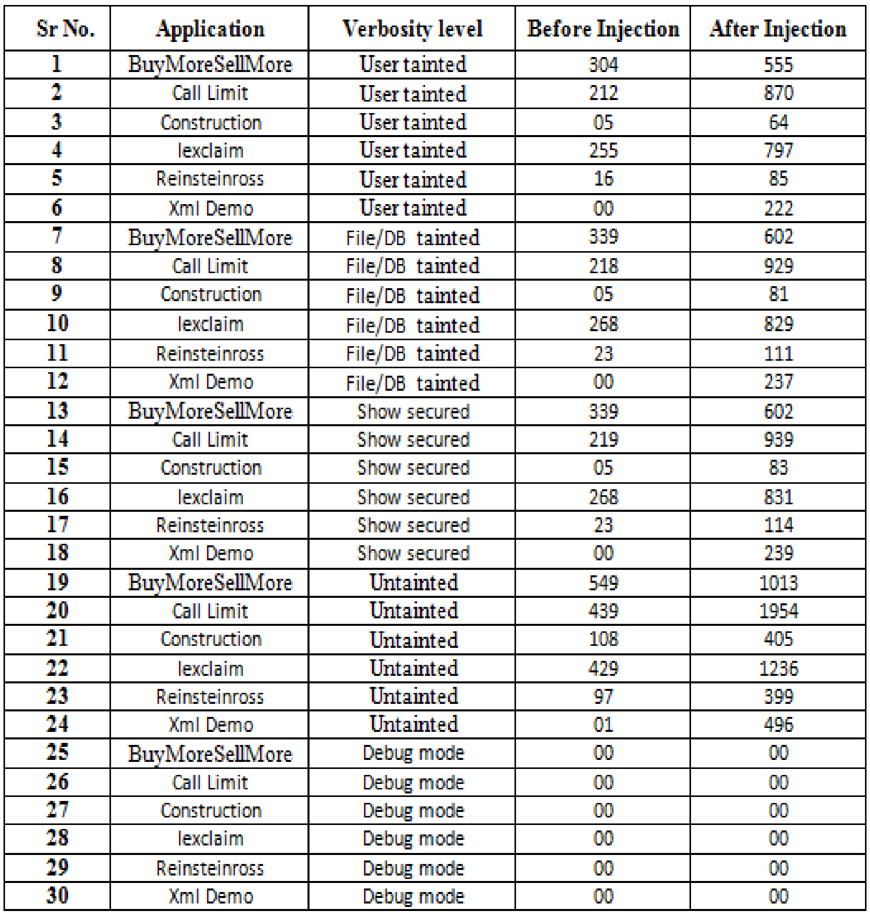
Fig. 1: Rips Plus Modification for Injection Tool.

4. Evalution

The prototype will be evaluated with some web applications in order to measure its efficiency. The four applications are a guestbook script called Talkback [17], a web chat server named Voc [18], a document management system called yaDMS [19], and RIPS [20], the vulnerability scanner that contributed to the model and sink files of the prototype. The selection of these applications was based in three requirements that had to be satisfied. The first was that the prototype had to be able to parse the application files, which was not always true because the original Pixy could not parse PHP5 applications, and thus neither could the prototype. The second is the equivalent for RIPS, i.e., RIPS had to be able to parse the application files too. This was required because RIPS was to be used to cross-check the results of the injections. The third requirement was that the applications were easy to run and test, which was required to verify whether the injections affected the apparent behavior of the application or not.

Experimental Results

Table 1: Output Data.



Sr No.	Application	Verbosity level	Before Injection	After Injection
1	BuyMoreSellMore	User tainted	304	555
2	Call Limit	User tainted	212	870
3	Construction	User tainted	05	64
4	lexclaim	User tainted	255	797
5	Reinsteinross	User tainted	16	85
6	Xml Demo	User tainted	00	222
7	BuyMoreSellMore	File/DB tainted	339	602
8	Call Limit	File/DB tainted	218	929
9	Construction	File/DB tainted	05	81
10	lexclaim	File/DB tainted	268	829
11	Reinsteinross	File/DB tainted	23	111
12	Xml Demo	File/DB tainted	00	237
13	BuyMoreSellMore	Show secured	339	602
14	Call Limit	Show secured	219	939
15	Construction	Show secured	05	83
16	lexclaim	Show secured	268	831
17	Reinsteinross	Show secured	23	114
18	Xml Demo	Show secured	00	239
19	BuyMoreSellMore	Untainted	549	1013
20	Call Limit	Untainted	439	1954
21	Construction	Untainted	108	405
22	lexclaim	Untainted	429	1236
23	Reinsteinross	Untainted	97	399
24	Xml Demo	Untainted	01	496
25	BuyMoreSellMore	Debug mode	00	00
26	Call Limit	Debug mode	00	00
27	Construction	Debug mode	00	00
28	lexclaim	Debug mode	00	00
29	Reinsteinross	Debug mode	00	00
30	Xml Demo	Debug mode	00	00

5. Conclusion

The work focus on the detection of various vulnerabilities while injecting the same through a prototype design. Vulnerability injection tool is capable of inserting realistic and attackable vulnerabilities into php based web applications. Further, the prototype is injected with taint-style vulnerabilities *i.e.*, Sql injection, cross-site scripting etc. The vulnerabilities present in the latest versions of well known php applications would be studied and analyzed. A handful of work exists for securing php based web applications using vulnerability injection. Most of the protocols exploit the vulnerability detection rather than vulnerability injection which is a major drawback for the design of real- time and secure applications. Moreover, the existing protocols employ single type of vulnerability which limits its practical usage. Also, the dynamic nature of PHP makes the design of secure php based applications more challenging. In the thesis, we proposed to design a prototype which targets PHP based web applications. This technique can be used in various research applications such as the automatic creation of vulnerable code for the purpose of educational application, systems requiring in-depth testing and evaluation of both vulnerability scanners and security teams.

References

[1] McConnell S., (1997), "Gauging software readiness with defect tracking", IEEE Journal of Software Engineering, pp. 136–135.

[2] Web Goat. URL: https://www.owasp.org/index.php/ ategory:OWASP_WebGoat_Project

[3] Biggar P. and Gregg D.(2009) "Static Analysis of Dynamic Scripting Languages", Informatics, pp. 56-60.

[4] Usage of server-side programming languages for websites. Feb. 2011. URL: http://w3techs.com/ technologies/overview/programming_language/all.

[5] Fonseca J., Vieira M. and Madeira H. (2008), "Training Security Assurance Teams using Vulnerability Injection", 14th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 297–304.

[6] Fonseca J., Vieira M. and Madeira H.(2009), "Vulnerability & attack injection for web applications", IEEE/IFIP International Conference, pp. 93–102.

[7] Carreira J.V., Costa D. and Silva J.G.(1999),"Fault injection spot-checks computer system dependability", pp. 50 –55.

[8] http://en.wikipedia.org/wiki/Compiling

[9] Chomsky N.(1956), "Three models for the description of language", pp. 113–124.

[10] Etienne Kneuss.(2010), "Static Analysis for the PHP Language", Informatics, pp. 122-126.

[11] Jovanovic N., Kruegel C. and Kirda E.(2006), "Precise alias analysis for static detection of web application vulnerabilities",pp. 27–36. ISBN: 1595933743.

[12] Xie Y. and Aiken A.(2006), "Static detection of security vulnerabilities in scripting languages", 15th USENIX Security Symposium. 2006, pp. 179–192.

[13] Dahse J.(2010), "RIPS - A static source code analyzer for vulnerabilities in PHP scripts".

[14] Graphviz(2001), "Graph Visualization Software". URL: http://www.graphviz.org/. Computing, vol. 61 (6), pp. 810-837.

[15] Alfred V. Aho, Ravi Sethi and Jeffrey D.,(1986), "Ullman. Compilers: principles, techniques, and tools". Boston, MA, USA: Addison-Wesley Longman Publishing Co., ISBN: 0-201-10088-6.

[16] Kirda E.,(2007), "A static analysis tool for detecting web application vulnerabilities", IEEE Symposium, pp. 6–263.

[17] http://www.calllimit.com/
[18] http://reinsteinross.com/.
[19] http://en.wikipedia.org/wiki/Compiling
[20] Chomsky N.(1956), "Three models for the description of language", pp. 113–124.
[21] Etienne Kneuss.(2010), "Static Analysis for the PHP Language", Informatics, pp. 122-126.
[22] Jovanovic N., Kruegel C. and Kirda E.(2006), "Precise alias analysis for static detection of web application vulnerabilities",pp. 27–36. ISBN: 1595933743.