Quality Of WordPress Plug-Ins: An Overview of Security and User Ratings

Teemu Koskinen, Petri Ihantola, and Ville Karavirta
Aalto University, School of Science
Department of Computer Science and Engineering
teemu@teemukoskinen.com, petri.ihantola@aalto.fi, and ville.karavirta@aalto.fi

***Abstract*—We have applied static analysis to find out how vulnerable the plugins available at the official WordPress plugin directory are to well known security exploits. We have compared the amount of potential vulnerabilities and vulnerability density to the user ratings, to determine if user ratings can be used for finding secure plugins. We conclude that the quality of the plugins varies and there is no clear correlation between the ratings of plugins and the number of vulnerabilities detected in them. Indeed, an additional manual review exposed a simple but severe SQL injection vulnerability in a plugin, which has both good user ratings and a high download count. We recommend plugins to be individually inspected for typical vulnerabilities before using them in any WordPress powered site.**

I. Introduction

WordPress is a highly popular online content management platform used to power many blogs, home pages, and social communities (e.g. Mashable, Plazaa, etc). There are over 73 million WordPress sites in the world and 300+ million people view 2.5+ billion pages each month[1]. The vast amount of plugins is one of the reasons for the popularity of WordPress. At the time of writing, there were over 19,073 plugins in the official WordPress plugin directory[2] downloaded in total of 286,416,770 times. The plugins are often developed by individual developers and provided to others for free. There are plugins for social communities (e.g. BuddyPress), picture galleries, payment systems, etc.

While some of the plugins are more security critical than others, a vulnerability in any plugin may expose the whole system to attackers. Therefore, all plugins go through a manual review before being accepted in the official plugin directory[3]. This, however, is not able to prevent all the problems. At the time of writing, there were 377 WordPress related vulnerabilities reported in the National Vulnerability Database[4], 122 (32%) of which were related to a plugin or extension [7].

To help find high quality plugins, WordPress allows the user community to rate the plugins and displays ratings and download counts for each plugin. Our research question is related to the usefulness of these ratings: *"Do plugin ratings predict the amount of implementation related vulnerabilities in WordPress plugins?"*

To detect the potential vulnerabilities, we applied an open source static analysis tool called RIPS [2] and compared our findings with user ratings. Static analysis can lead to false positive findings. Therefore, in this report, when speaking about the amount of vulnerabilities we actually refer to the potential vulnerabilities detected in static analysis. We restricted our analysis to the backends of the plugins, which, like WordPress itself, are written in PHP.

This paper is structured as follows. In Chapter II, we overview some previous research. Chapter III presents our research methods, and results are discussed in Chapter IV. We conclude our work in Chapter V.

II. Background and Related Research

A. User Reviews and Ratings

User reviews are a popular way for users to share their experiences with on-line products and services. Typical forms of review include *star ratings*, where the users can grade a product on a scale from one to, lets say, five stars, and *verbal feedback*, where the users write short comments.

The impact of online ratings and reviews to user behavior has been studied extensively. Chevalier and Mayzlin [1], for example, have described how improved reviews lead to better sales. They also state that the verbal feedback has a bigger impact than summary statistics (i.e. star ratings). More recently, Mudambi and Schuff [6] also concluded that the helpfulness depends on the depth of the review, and varies between product type. In the star rating context, the users typically only see an average rating and total number of reviews. The depth of such information is clearly shallow, which makes it less helpful[5].

B. Types of Vulnerabilities

According to the Open Web Application Security Project (OWASP), the top 5 PHP related vulnerabilities are[6]: remote code execution, cross-site scripting (XSS), SQL injection (SQLI), PHP configuration and file system attacks. All but the misconfigured PHP environment are implementation vulnerabilities possible to detect in static analysis. The list is rather consistent with the more recent, language agnostic, OWASP list of most critical web application security risks.[7]

[1] http://en.wordpress.com/stats/
[2] http://wordpress.org/extend/
[3] http://wordpress.org/extend/plugins/about/
[4] http://nvd.nist.gov/
[5] TornadoGuard, http://xkcd.com/937/
[6] https://www.owasp.org/index.php/PHP_Top_5
[7] http://code.google.com/p/owasptop10/

The top three items in this list are injection, XSS, and broken authentication and session management.

Based on [8], we argue that certain language design choices make it relatively easy for less experienced programmers to cause implementation vulnerabilities into PHP web applications. Web applications created by experienced PHP programmers, however, are equally secure as web applications written by users of other programming languages [4]. Unfortunately, not all plugin developers are experienced PHP programmers.

C. Automated Detection of Vulnerabilities

The size and complexity of web applications make manual code reviews challenging. Thus, automated approaches to detect security vulnerabilities are widely applied. These can be divided between *black box* (i.e. testing) and *white box* (e.g. static analysis) approaches.

Black box security scanners access web services through the HTTP interface the same way users do. These tools are thus independent of the backend technologies. Doupé et al. [3] compared eleven security scanners. They concluded that, providing meaningful/valid test data is challenging, and, therefore, black box scanners struggle to crawl the applications deep enough to identify various vulnerabilities. The rate of false negatives (i.e. a tool not being able to find a vulnerability) are typically between 60 and 90% [3, 4].

White box approaches are based on analyzing the source code. *Static program analysis* can be seen as an extreme among white box approaches. It analyses software without executing (or compiling) the code. Static analysis can answer, for example, if it is possible that a variable used to construct an SQL query originates from unsafe user input. Whereas false negatives are the problem in fully automatic black box security testing, the problem with static analysis is false positives. A safe program sequence can be marked as vulnerable because static analysis can not guarantee the safety. Fonseca et al. [5] have evaluated three widely used commercial security scanners for PHP applications. Their method included injecting typical programming errors in two web-based applications, one of them being WordPress, and then using each of the scanners to determine if injected SQLI and XSS vulnerabilities were detected. The three scanners had false positive rates varying from 20% to 77%.

The two approaches complement each other, but, at least in our context, black box testing would have required configuring the tools to focus on UI behavior that is relevant for each plugin. This is why we decide to apply static analysis that can be done effectively without configuring parameters as we see that uniform configurations make the results more comparable between plugins.

There are commercial (e.g. Fortify Source Code Analyzer[8]) and open source (e.g. Pixy[9], RATS[10], RIPS[11]) static analysis tools to reason about security of web applications written in PHP. In this study, we chose to apply RIPS because it is open source and it supports detecting more vulnerability types than any other free tool we are aware of.

III. Methods

We downloaded a randomized sample of 322 plugins listed in the WordPress plugin directory. The plugins were downloaded directly from version control repository[12]. We continued by running static security analysis for each of the plugins. For this, we used a slightly modified[13] version of RIPS on verbosity level 2. Verbosity level one creates smallest amount of warnings whereas most warning are created on level 5. Finally, for each plugin, we calculated the *vulnerability density* (PV/KLOC[14]), that is the number of vulnerabilities divided by the number of actual lines of code (i.e. comment and empty lines removed). Ratings and the number of downloads were also retrieved from the plugin directory.

IV. Results

A. Vulnerabilities and Vulnerability Density

In our analysis of the 322 plugins, we discovered 860 vulnerabilities from 127 plugins. Vulnerabilities were distributed unevenly across the plugins as illustrated in Figure 1. Most of the vulnerabilities were related to XSS (see Table I).

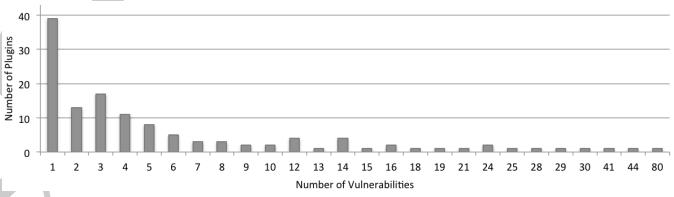

Fig. 1. Distribution of vulnerabilities per plugin

TABLE I
VULNERABILITIES BY TYPE, 322 PLUGINS

Vulnerability type	Total	In how many plugins
Cross Site Scripting (XSS)	615 (72%)	107
File affect (FA)	86 (10%)	37
File read (FR)	38 (4%)	20
Unserialize (POP)	32 (4%)	15
Other	29 (3%)	14
Code execution (Code)	17 (2%)	6
Header injection	12 (1%)	7
File include (FI)	12 (1%)	9
SQL Injection (SQLI)	11 (1%)	3
Control flow (Con)	8 (1%)	7

The vulnerability densities varied among the plugins from 0 to an alarming 200 PV/KLOC. The highest amount of vulnerabilities, 80, was found in a plugin having 446 lines of PHP code (i.e. vulnerability density of 179 PV/KLOC). The number of code lines correlates mildly with the number of vulnerabilities (Spearman's $\rho = 0.53, p < 0.01$) and vulnerability density (Spearman's $\rho = 0.37, p < 0.01$).

[8]https://www.fortify.com/products/hpfssc/source-code-analyzer.html
[9]http://pixybox.seclab.tuwien.ac.at/pixy/
[10]https://www.fortify.com/ssa-elements/threat-intelligence/rats.html
[11]http://rips-scanner.sourceforge.net/
[12]http://plugins.svn.wordpress.org/
[13]We added batch processing and serialization of results to a textual format.
[14]Potential Vulnerabilities divided by Kilo Lines of Code

B. The Number of Ratings and Downloads

How many times a plugin is graded contributes to the trustworthiness of the star rating. Our sample of plugins were downloaded a total of 3,792,711 times and rated 2,783 times. Thus, only 0.07% of downloads lead to writing a review. Interestingly, five most popular plugins contribute to more than 200,000 downloads which is 35.5% from all downloads in our sample. There is a strong linear correlation between the number of downloads and the number of ratings (Pearson's $r = 0.89, p < 0.01$). Distribution of the number of ratings per plugin is skewed towards the lower end as illustrated in Figure 2.

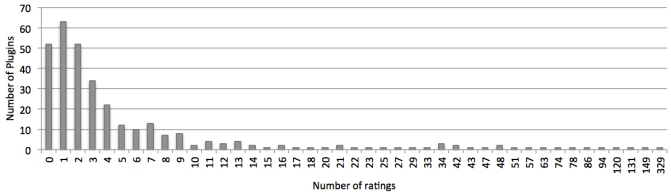

Fig. 2. Distribution of the number of ratings per plugin.

C. Ratings and Vulnerabilities

Ratings and total amount of vulnerabilities as well as ratings and vulnerability densities are illustrated in Figure 3. The figure shows three versions of both with all plugins, plugins with at least five ratings, and plugins with at least ten ratings. The smallest grade a plugin can get is 20 (i.e. one star). Thus no stars (i.e. rating = 0) implies that no-one has graded the plugin. After removing such plugins, we found a weak negative correlation between the ratings and the number of vulnerabilities (Spearman's $\rho = -0.23, p < 0.01$) and between the ratings and vulnerability density (Spearman's $\rho = -0.20, p < 0.01$). When focusing to plugins rated at least 5 or 10 times, we found correlation coefficients very similar but no longer statistically significant $p > 0.05$. This could be due to the smaller sample size. From the 270 rated plugins, 99 were rated at least 5 times and 49 at least 10 times (see Figure 2).

D. Manual Review

After the static analysis we decided to make a closer inspection by manually reviewing the plugin with most vulnerabilities. With 4,113 downloads, this was the 92nd most downloaded plugin among the sample with a good average rating of 77.6 out of 100. Five of the detected potential vulnerabilities were SQLI vulnerabilities. We confirmed all of those to be real so that with a well designed input data, a malicious user could alter the database[15] or even take over the site. These vulnerabilities could be fixed by validating the type of the input and by replacing special characters with escaped ones.

[15]http://xkcd.com/327/

V. Discussion

Although the security of the open source web platforms themselves has been steadily increasing [9], 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. The overall security of the plugins was promising, as over half of the plugins passed the static analysis with no vulnerabilities detected. However, we discovered that the quality was not consistent across different plugins. The brief manual review of just one plugin revealed that vulnerable plugins can easily be found and that they may be exploited easily. Inconsistency of security among individual plugins was observed in terms of both vulnerability counts and densities in the plugins.

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.

When studying if plugins with a high rating contain less vulnerabilities, we found only weak evidence supporting that. It is therefore questionable if the two values presented at the plugin catalog, rating and download count, help users choose high quality plugins. At least in terms of security.

The vulnerability densities are a measure to compare the results with previous work and other applications. In our research, the plugins contained a total of 179,393 lines of PHP code with the overall vulnerability density 4.79 PV/KLOC. This is consistent with similar web applications written in PHP [9]. Although the earlier research, where vulnerability densities ranged between 3.30 and 8.88PV/KLOC, was conducted with a different analysis tool, the overall security of the WordPress plugins appears to be on par with the typical quality of open source PHP applications.

A. Concerns about Validity

The two main concerns related to the validity of the results are that our data (i.e. plugins and ratings) are not representative and that our approach to detect vulnerabilities suffers from both false negative and false positive results. Related to the first concern, some of the plugins are also available through other distribution channels and therefore may have additional reviews elsewhere. In addition, rating a plugin requires logging in to the site, which may also alter the demographics of the raters. On the other hand, this reduces fabricated ratings. Related to the second concern, we tried different verbosity levels and doing quick manual overviews of the reported vulnerabilities, we decided to apply level 2. Earlier, Dahse [2] have applied the same setting with good success rate. However, to decrease the number of false positives, a better alternative might be to use several vulnerability detection tools in parallel.

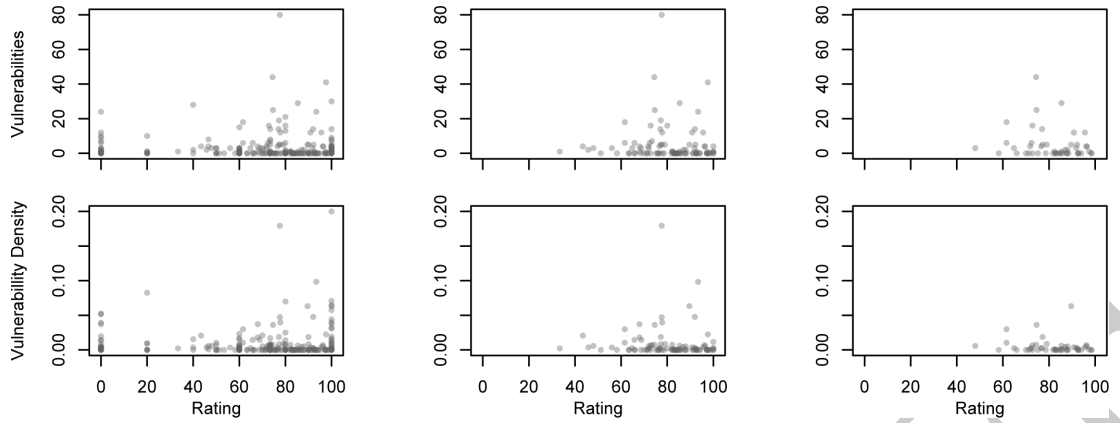

Fig. 3. Ratings and the number of vulnerabilities (top) and vulnerability densities (bottom) of all plugins (left), plugins rated at least 5 times (middle) and plugins rated at least 10 times (right).

VI. Conclusion

To answer our research question *"Do plugin ratings predict the amount of implementation related vulnerabilities in WordPress plugins?"*, we have analyzed 322 randomly selected WordPress plugins. Wordpress plugin site allows users to grade plugins and we have compared these ratings of the plugins with the vulnerabilities identified through static analysis. We found only a weak non-linear correlation between ratings and number of vulnerabilities.

Based on our findings, we are confident that there are real risks involved when using third-party plugins on a WordPress site. Many plugins appeared not to be vulnerable, but as the user ratings and download counts do not assist in finding secure plugins, proper inspection should be done by static analysis or manual review before using any plugin on a WordPress site. The cost of software development and fast schedules in the industry make installing plugins an attractive solution, but we hope our findings encourage developers to take the time to inspect the code before using it.

A. Future Work

The obvious future improvement, which we will implement, is to run the same tests with a larger sample of plugins. We also plan to manually inspect a higher number of potential vulnerabilities to better understand how many of those are real. Furthermore, it could be beneficial to research if other qualities of the plugins have more impact on their popularity. Properties such as usability, usefulness or even the plugin description itself may have more impact on user satisfaction than security. It could also be useful to combine the results from this study with the results made by Mudambi and Schuff [6], and find out if the extremeness of user ratings has an impact on their usefulness. This would, however, require access to not just the average rating of plugins, but also the distribution of individual ratings in the five star scale, which are not publicly available. Finally, more research on how often people downloading the plugins use the ratings to support their decision making is also needed. Is it possible that the minority of people who rate have an impact on the majority of people downloading the plugins?

References

[1] Chevalier, J. A., and Mayzlin, D. The effect of word of mouth on sales: Online book reviews. *Journal of Marketing Research 43*, 3 (2006), 345–354.

[2] Dahse, J. RIPS – a static source code analyser for vulnerabilities in PHP scripts. http://www.php-security.org/downloads/rips.pdf, May 2010.

[3] Doupé, A., Cova, M., and Vigna, G. Why Johnny can't pentest: an analysis of black-box web vulnerability scanners. In *Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment* (Berlin, Heidelberg, 2010), DIMVA'10, Springer-Verlag, pp. 111–131.

[4] Finifter, M., and Wagner, D. Exploring the relationship betweenweb application development tools and security. In *Proceedings of the 2nd USENIX conference on Web application development* (Berkeley, CA, USA, 2011), WebApps'11, USENIX Association, pp. 99–111.

[5] Fonseca, J., Vieira, M., and Madeira, H. Testing and comparing web vulnerability scanning tools for sql injection and xss attacks. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing* (Washington, DC, USA, 2007), PRDC '07, IEEE Computer Society, pp. 365–372.

[6] Mudambi, S. M., and Schuff, D. What makes a helpful online review? a study of customer reviews on amazon.com. *MIS Quarterly 34*, 1 (2010), 185–200.

[7] NIST Computer Security Division. National Vulnerability Database. http://web.nvd.nist.gov/, 2012.

[8] Shiflett, C. *PHP Security*. O'Reilly Open Source Convention, Portland, Oregon, USA, 2004.

[9] Walden, J., Doyle, M., Welch, G. A., and Whelan, M. Security of open source web applications. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement* (Washington, DC, USA, 2009), ESEM '09, IEEE Computer Society, pp. 545–553.