Automated Analysis of Source Code Patches using Machine Learning Algorithms

Antonio Castro Lechtaler[1,2], Julio César Liporace[1], Marcelo Cipriano[1], Edith García[1], Ariel Maiorano[1], Eduardo Malvacio[1], Néstor Tapia[1]

[1] Grupo de Investigación en Criptografía y Seguridad Informática (GICSI), Instituto Universitario del Ejército (IUE), [2] Universidad Nacional de Chilecito (UNdeC), Argentina

{antonio.castrolechtaler, edithxgarcia, jcliporace,maiorano, cipriano1.618, edumalvacio, tapianestor87}@gmail.com

Abstract. An updated version of a tool for automated analysis of source code patches and branch differences is presented. The upgrade involves the use of machine learning techniques on source code, comments, and messages. It aims to help analysts, code reviewers, or auditors perform repetitive tasks continuously. The environment designed encourages collaborative work. It systematizes certain tasks pertaining to reviewing or auditing processes. Currently, the scope of the automated test is limited. Current work aims to increase the volume of source code analyzed per time unit, letting users focus on alerts automatically generated. The tool is distributed as open source software. This work also aims to provide arguments in support of the use of this type of tool. A brief overview of security problems in open source software is presented. It is argued that these problems were or may have been discovered reviewing patches and branch differences, released before the vulnerability was disclosed.

Keywords: automated, source code review, source code analysis, patch analysis, machine learning, text mining, software quality.

1 Introduction

This work presents a software tool for automated source code patches and branch differences analysis. It aims to systematize updates and source code reviews to alert on potential bugs, implying some degree of system security compromise, or vulnerabilities. The tool is distributed as an open source project and available at http://github.com/gicsi/aap. Currently, we have not found other open source tools available for the proposed functionality presented here. Other projects have been published but they largely deal with software engineering. For instance: trackable recoveries between source code and corrected bugs through patch analysis [5], or the use of patches for bug reporting. In the first case, a data processing tool was presented for Bugzilla systems, identifying the information with CVS tags.

In the case of open source analysis with cryptographic functionalities, Android case studies show that, for instance, only 17% of 269 vulnerabilities - reported in the period between January 2011 and May 2014 - were attributed to glitches in the

cryptographic libraries. The remaining 83% were attributed to flaws in the application´s use of these libraries [14].

Furthermore, Android applications were analyzed in [8]: 10.327 out of 11.748 applications (88%) reported at least one implementation error. This last work presented an automatic detection tool, but was not distributed freely [15].

Other tools are available for general use to assist in source code management. However, they do not focus on security problems or patch reviews . They operate over the entire development cycle. Among the most common available open source alternatives, [12] and [23] are worth mentioning.

In the next section, references are surveyed on open and commercial software alternatives, dealing with security issues [4, 19, and 22].

It should be noted that control systems aiming at specific revisions have their own difficulties and limitations.

A study analyzing 210 samples showed that over 40% were not reported in the evaluation of C/C++ source code with five analytical tools; while only 7% of those samples were appropriately reported by the five tools [30]. Analogous results were found in the same study, using six tools based on Java.

The Open project OWASP [22] has identified the weaknesses of automated tests. It has also pointed out their strengths: ability to perform fast and repetitive analysis over a high volume of source code, the capacity to detect high probability errors, and the specificity provided in the reports.

2 Source Code Analysis

2.1 The Cost of Software Quality

Source code analysis and inspections are a continuous best-practice, aiming to improve software quality. It should be noted that although the quantity of bugs in a software project is not the only quality indicator, it constitutes a valuable measure for control and potential enhancements to development processes. Strictly, the goal of these reviews is to reduce the cost of software quality by identifying and correcting bugs at early stages of development [25].

It has been shown empirically [9] that software quality depends on control mechanisms used as an integral part of processes. Furthermore, the rate of personal inspections (measured in quantity of source code lines per time unit) has been shown to affect the effectiveness in detection (and correction) of bugs. Data shows, for instance, that review quality declines as this rate exceeds its recommended maximum of 200 code lines per hour. The authors cite previous articles in which a rate of 125 lines per hour is considered the optimum [3]. They also note: *"it is almost one of the laws of nature about inspections, i.e., the faster an inspection, the fewer defects removed"* [25].

2.2 Analysis Systematization

When dealing with software quality and the effectiveness of source code review, analysis systematization may yield efficient quality and development control

processes when accounting for the time demanded from an analyst. Quality assurance tools are an essential resource in the improvement of software applications.

In a NIST publication, the minimum required functionality specifications in a source code analyzer for searching vulnerabilities are laid out [2]. The article suggests that it should identify security weaknesses, and report them tagging their type and location.

According to SEI/CERT, relying exclusively on policies, standards, and good practices for software development have proven inadequate [30]. It points out that these factors are not being applied consistently and that manual source code audits can be complemented and enhanced with systematized and automated tests.

2.2.1 Current Available Tests

Different organizations maintain lists from which a great range of tools for testing source code are referenced. Among them: the NIST webpage, Source Code Security Analyzers [19], and the Secure Coding Tools from CERT [4].

Finally, the OWASP project is an obliged reference, maintaining a thorough list of source code analysis tools [22].

3 Post Patch Vulnerabilities

3.1 Errors in the Categorization of Vulnerabilities

In his recent thesis [34] and revising previous work [35, 36, 33], Jason Wright - best known for his work in the cryptographic framework of the OpenBSD operating system [13] – emphasizes the importance of patch reviews. In the third section of his last publication, he introduces the concept of *hidden impact bugs*. The concept is also found in [35] and [33]. In the latter the term used is *hidden impact vulnerabilities*, referring to previous work [1]. However, they all deal with the same concept; i.e., vulnerabilities which were reported as bugs and whose real impact – their vulnerability category – was recognized after issuing the initial report which included a patch to solve what was believed to be a software flaw or defect with no security implications.

Impact delay is defined as the time elapsed since the report of the flaw – as a patch – until the time when the bug was tagged with a CVE. In 2012, the author, contributing with others at a second stage of analysis, published a review of hidden impact bugs in the source code (patches) of the Linux Kernel and MySQL database [33]. Later, in 2013, they published an extended analysis, focusing on MySQL exclusively [35].

The tests consisted of detailed reviews of a subset of the reported bugs. The source code associated to the bugs is analyzed to determine the percentage of unclassified vulnerabilities. The authors extrapolate the results to yield an estimate of the total percentage on reported flaws. From the available data, the results are summed up as follows:

- In the second stage analysis of the Linux kernel - from January 2009 to April 2011 - results show that from a total of 185 hidden impact bugs, 73 (39%) had

an impact delay of less than 2 weeks; 55 bugs (30%) had a delay of less than 4 weeks; and 29 (16%) had a delay of less than 8 weeks.

- In the MySQL study, total hidden impact bugs sum up to 29. From them, 19 (65%) had an impact delay of less than 2 weeks, also 19 had a delay of less than 4 weeks, and 16 bugs (55%) had a delay of less than 8 weeks.

3.2 Fixed Issues in NTP-devel

A reference implementation of the protocol used to synchronize clocks over the network, known as *Network Time Protcol* (NTP), is distributed as part of the Linux operating systems, among others. In December 2014, a series of security issues – some of them categorized as critical – were identified. The information went public along with patches to fix the problems. However, the bug database used by the development team showed that some of these bugs had already been fixed in the *devel* version, years before.

Additional information is available in their security webpage [16]. A brief outline of the particular details is given below to illustrate the way in which a comparison between two versions can point out critical fixes not implemented in the stable version.

Although Bugzilla´s historical data show that the information was made accessible to the public on 12/12/2014, messages on bug 2665 [17] - regarding a weak cryptographic default key - point out that the vulnerability had already been "corrected" four years before, in the NTP-devel (4.2.7), particularly "(4.2.7p11) 2010/01/28."

An additional example involves bug 2666 [18] dealing with a random number generator – cryptographically insecure with a weak initialization seed. Although it was later fixed once again before the release of version 4.2.8, messages point out that the development version n 4.2.7 p30 (of November 1st, 2011) had already corrected the problem, referring an enhancement of the seed management.

3.3 Multiple Patches for Shellshok Vulnerabilities

Recently discovered in the UNIX bash shell system, the Shellshok vulnerabilities represent an example of critical security problems which are not corrected completely or adequately in the first published patches. The importance and criticality of the vulnerability was highlighted in The Register [28], in which they recommended immediate installation of the patch to avoid wide open access to Linux and OS X operating systems. The news, along with a public notice from Red Hat [26], and the updated packages are dated September 24th, 2014. Nonetheless, later, related problems were identified and for which Red Hat did not offer updates until September 26th, 2014 [27].

3.4 Simultaneous Discovery of the Heartbleed Vulnerability

During 2014, a critical vulnerability, known as Heartbleed, was disclosed. It consisted of an exploitable bug in the OpenSSL library. According to Bruce Schneier,

half a million sites were infected, turning the problem catastrophic [29]. A Red Hat Security representative [6] considered that the coincidence of two simultaneous findings of a single problem would increase the risk of maintaining the vulnerability secret. Consequently, it can be inferred that this rationale could have encouraged OpenSSL to release updated packets hastily. The rush in the releases may be responsible for the lack of coordination in issuing the patches. Thus, this illustrates how, under certain circumstances, several organizations can publish patches in an non-coordinated manner.

3.5 Vulnerability in OpenBSD after a Misclassified Bug

In 2007, an IPv6 vulnerability was found in the OpenBSD kernel [21]. A specially crafted ICMP packet could generate a remote buffer overflow, when handled by an internal function which miscalculated the buffer´s length. According to the author, the vulnerability was identified while analyzing – and trying to reproduce – a problem already fixed by a patch rated as reliability fix instead of vulnerability or security fix [20].

4 Patch Analysis Tool

4.1 Automated Analysis of Patches, or AAP, version 0.2b

4.1.1 Idea and Rationale

Beside obvious differences in source code and binary analysis, the project is based on the criteria used by automated malware analysis tools, such as the Cuckoo Sandbox.

In [11], two problems are identified when performing this type of analysis manually: the increasing volume of malware in the wild and the time required for a thorough inspection.

The goal in this work also involves an efficacy assessment in the use of IA techniques, especially text mining and machine learning, applied to source code patches and “texts” in general comments.

The rationale relies on published work regarding the discovery of vulnerabilities in the patches distribution of open source code projects. The references in the third section of this article constitute examples of this problem.

4.1.2 Machine Learning

Although beyond the scope of this article, a brief overview is presented here on the techniques and applications of machine learning implemented in the tool presented in this article:

Natural Language Processing (NLP). Natural language is understood as the language used by human beings to communicate: English, Spanish, ... Generally, NLP involves the manipulation of a natural language through the use of computer systems. For instance, text analysis enables the detection of positive or negative sentiments in tweets – in favor or against.

Supervised Classification. Classifying comprises assigning tags of the appropriate category to particular entries or inputs. The set of tags is defined beforehand. Some classification examples are SPAM identification or news categorization (sports, finance, and others). A classifier is supervised when it is trained based on data (corpora) containing the correct tag for each entry or input.

AAP Implementation. For the generation of tagged data, bug reports, with attached patches, were downloaded from Red Hat and OpenBSD public bug database system and errata pages, respectively. The labels "vulnerability" and "bug" were defined according to the criticality established in the original reports. With this set of tagged entries or inputs, classifiers - of the naïve Bayes and decision tree types - were trained. These classifiers can then be used from a plugin to estimate the appropriate tag for each automatically analyzed patch.

4.1.3 Functionality Currently Implemented

Through its web interface, the tool maintains and organize information of software projects, reference to its repositories, branches, updates, and analysts users assigned to them.

In addition, the web interface handles the general configuration for the automatic reviews of patches and different versions. These rules are implemented in the form of plugins, programmed in Python and editable on the interface itself.

The verifications of these rules, automatically executed over patches or over branch differences of a software project update, activates alerts whenever necessary. Currently, these rules include, among others, searching for configured patterns in source code and comments, detecting source code implementing cryptographic functionality, and checking for changes in specific files or done by particular authors or committers.

Since version 0.2b, the tool contemplates the application of machine learning techniques through trained classifiers with patches and comments from public bug information repositories. Classification analysis applies a per "vulnerability" or "bug" criteria.

Configured rule inspections are performed over patches recovered from GIT repositories [10].

The periodic batch process updates the repositories and automatically performs the analysis configured for each of its branches. The outcome includes the generation of system alerts -and optionally sending notification emails with analysis results- to the assigned analysts.

4.1.4 Technologies

The tool consists of a web-based application developed in the Python language [24] and using Web Django framework [7]. It makes use of GIT versatility for updating and handling source codes [10]. Other external tools may be used optionally.

Currently, plugins to classify patches and comments are implemented using an open source library or toolkit, also developed in Python: NLTK (Natural Language

Toolkit). It includes software, data and documentation, and can be freely downloaded from http://nltk.org/.

NLTK Trainer constitutes another resource – a set of Python scripts for NLTK model training.

Particular plugins for static analysis were implemented for different languages (C/C++, Python, Perl, PHP) based on the use of open source tools like FlawFinder, RATS, Splint and CppCheck.

4.1.5 Open Source Software

The tool's source code is publicly published using a free service for open source projects management: http://github.com/gicsi/aap. It is licensed under the terms of the GPL, or GNU General Public License, from the Free Software Foundation.

5 Further Work

Functionality enhancements are under design. The expectation is to capture the interest of other developers and encourage them to get involved with the project. Among other improvements, the following features are under consideration: to achieve a higher degree of sophistication in rule and algorithm design, to extend the use of machine learning in the tool, to incorporate information retrieved from developers´ discussion forums, and to integrate the tool with other databases dealing with vulnerability reports and security advisories.

References

1. Arnold, J., Abbott, T., Daher, W., Price, G., Elhage, N., Thomas, G.A. Kaseorg Security Impact Ratings Considered Harmful. Proceedings 12th Conference on Hot Topics in Operating Systems. USENIX. May 2009. [Online – accessed 29/12/2014: http://www.inl.gov/technicalpublications/Documents/5588153.pdf].
2. Black, P., Kass, M., Koo, M., Fong, M. Source Code Security Analysis Tool Functional Specification Version 1.1. NIST Special Publication 500-268 v1.1. February 2011. [Online – accessed 29/12/2014: http://samate.nist.gov/docs/source_code_security_analysis_spec_SP500-268_v1.1.pdf].
3. Buck, F. Indicators of Quality Inspections. IBM Technical Report TR21.802, Systems Comm. December 1981.
4. CERT Division - Secure Coding Secure Coding Tools. CERT, Software Engineering Institute (SEI), Carnegie Mellon University. [Online - accessed 29/12/2014: http://www.cert.org/secure-coding/tools/index.cfm].
5. Corley, C., Etzkorn, L., Kraft, N., Lukins, S. Recovering Traceability Links between Source Code and Fixed Bugs via Patch Analysis. University of Alabama. 2008. [online - accessed 29/12/2014: http://www.cs.wm.edu/semeru/tefse2011/papers/p31-corley.pdf].
6. Cox, M. Heartbleed. Mark Cox, J. Google+. [Online - accessed 29/12/2014: https://plus.google.com/+MarkJCox/posts/TmCbp3BhJma].
7. Django Framework. Django overview. Django Software Foundation. [Online - accessed 29/12/2014: https://www.djangoproject.com/start/overview/].
8. Egele, M., Brumley, D., Fratantonio, Y., Kruegel, C. An empirical study of cryptographic misuse in android applications. CCS '13 Proceedings of the 2013 ACM SIGSAC

conference on computer & communications security. Pages 73-84. 2013. U.S.A. [Online – accessed 29/12/2014: http://www.cs.ucsb.edu/~chris/research/doc/ccs13_cryptolint.pdf].
9. Kemerer, C., Paulk, M. The Impact of Design and Code Reviews on Soft-ware Quality: An Empirical Study Based on PSP Data. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 35, NO. XX April 2009. [Online - accessed el 29/12/2014: http://www.pitt.edu/~ckemerer/PSP_Data.pdf].
10. Git SCM. About Git. Git - Software Freedom Conservancy. [Online: http://git-scm.com/about - accessed 2912/2014].
11. Guarnieri, C. One Flew Over the Cuckoo's Nest. Hack in the Box 2012. May 2012. Netherlands. [Online – accessed 29/12/2014: http://sebug.net/paper/Meeting-Documents/hitbsecconf2012ams/D1T1%20-%20Claudio%20Guarnieri%20-%20One%20Flew%20Over%20the%20Cuckoos%20Nest.pdf].
12. Gerrit Website Gerrit Code Review. Google Inc. [online: https://code.google.com/p/gerrit/ - accessed 29/12/2014].
13. Keromytis, A., Wright, J., de, T. Raadt. The Design of the Open BSD Cryptographic Framework. International Conference on Human System Interactions (HSI). June 2012. Australia. [Online - accessed 29/12/2014: http://www.thought.net/papers/ocf.pdf].
14. Lazar, D., Chen, H., Wang, X., Zeldovich, N. Why does cryptographic software fail? a case study and open problems. Proceedings of 5th Asia-Pacific Workshop on Systems Article No. 7. 2014. U.S.A. [online - accessed 29/12/2014: http://pdos.csail.mit.edu/papers/cryptobugs:apsys14.pdf].
15. Mujic, A. Reimplementation of CryptoLint tool. Blog for and by my students. December 2013. [Online - accessed 29/12/2014: http://sgros-students.blogspot.com.ar/2013/12/reimplementation-of-cryptolint-tool.html].
16. Network Time Protocol project. NTP Security Notice. NTP support website. Network Time Foundation. [Online - accessed 29/12/2014: http://support.ntp.org/bin/view/Main/WebHome].
17. Network Time Protocol project. Bug 2665 - Weak default key. NTP Bugzilla. Network Time Foundation. [Online - accessed 29/12/2014: http://bugs.ntp.org/show_bug.cgi?id=2665].
18. Network Time Protocol project. Bug 2666 - non-cryptographic random number generator with weak seed. NTP Bugzilla. Network Time Foundation. [Online - accessed 29/12/2014: http://bugs.ntp.org/show_bug.cgi?id=2666].
19. NIST Source Code Security Analyzers. SAMATE - NIST. [Online - accessed 29/12/2014: http://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html].
20. Ortega, A. OpenBSD Remote Exploit. Core Security. Julio de 2007. [Online – accessed 29/12/2014: https://www.blackhat.com/presentations/bh-usa-07/Ortega/Whitepaper/bh-usa-07-ortega-WP.pdf].
21. Ortega, A., Richarte, G. OpenBSD Remote Exploit. Core Security. April 2007. [Online - accessed 29/12/2014: https://www.blackhat.com/presentations/bh-usa-07/Ortega/Whitepaper/bh-usa-07-ortega-WP.pdf].
22. OWASP Wiki. Source Code Analysis Tools. The Open Web Application Security Project (OWASP). Ult. mod. 29/10/2014. [Online: https://www.owasp.org/index.php/Source_Code_Analysis_Tools - accessed 29/12/2014].
23. Phabricator Website. Phabricator, an open source, software engineering platform. Phacility, Inc. [online: http://phabricator.org/ - accessed 29/12/2014].
24. Python Website. About Python. Python Software Foundation. [Online: https://www.python.org/about/ - accessed 29/12/2014].
25. Radice, R. High Quality Low Cost Software Inspections. Paradoxicon Publishing. 2002.
26. Red Hat. Security. CVE Databases. CVE-2014-6271. Red Hat Customer portal. September 24th, 2014. [Online: https://access.redhat.com/security/cve/CVE-2014-6271 - accessed 29/12/2014].

27. Red Hat. Security. CVE Databases. CVE-2014-7169. Red Hat Customer portal. September 24th, 2014. [Online: https://access.redhat.com/security/cve/CVE-2014-7169 - accessed 29/12/2014].
28. The Register. Leyden, J. Patch Bash NOW: 'Shellshock' bug blasts OSX, Linux systems wide open. The Register online tech publication. September 24th, 2014. [Online: http://www.theregister.co.uk/2014/09/24/bash_shell_vuln/ - accessed 29/12/2014].
29. Schneier, B. Heartbleed. Schneier on Security, Blog. April 2014. [Online: https://www.schneier.com/blog/archives/2014/04/heartbleed.html – accessed 29/12/2014].
30. Seacord, R., Dormann, W., McCurley, J., Miller, P., Stoddard, R., D.Svoboda, Welch, J. Source Code Analysis Laboratory (SCALe). CERT, Software Engineering Institute (SEI), Carnegie Mellon University. April 2012. [Online: https://resources.sei.cmu.edu/asset_files/TechnicalNote/2012_004_001_15440.pdf - accessed 29/12/2014].
31. W. Weimer. Patches as Better Bug Reports. University of Virginia. 2006. [Online: https://www.cs.virginia.edu/~weimer/p/p181-weimer.pdf - accessed 29/12/2014].
32. Wheeler, D. Flawfinder. David Wheeler, A.'s Personal Home Page- Flawfinder Home Page. [Online: http://www.dwheeler.com/flawfinder/ - accessed 29/12/2014].
33. Wijayasekara, D., Manic, M., Wright, J., McQueen, M. Mining Bug Databases for Unidentified Software Vulnerabilities. Proceedings International Conference on Human System Interactions (HSI). June 2012, Perth, Australia. [Online: http://www.inl.gov/technicalpublications/Documents/5588153.pdf – accessed 29/12/2014].
34. Wright, J. Software Vulnerabilities: Lifespans, Metrics, and Case Study. Master of Science Thesis. University of Idaho. May 2014. [Online: http://www.thought.net/papers/thesis.pdf – accessed 29/12/2014].
35. Wright, J., Larsen, J., McQueen, M. Estimating Software Vulnerabilities: A Case Study Based on the Misclassification of Bugs in MySQL Server. Proceedings International Conference of Availability, Reliability, and Security (ARES). September 2013. pp. 72-81. Regensburg, Germany. [Online – accessed 29/12/2014: http://www.inl.gov/technicalpublications/Documents/5842499.pdf].
36. Wright, J., McQueen, M., Wellman, L. Analyses of Two End-User Software Vulnerability Exposure Metrics (Extended Version). Information Security Technical Report, 17(4), Elsevier. April 2013. pp. 44-55. [Online: http://www.thought.net/papers/INL-JOU-12-27465-preprint.pdf – accessed 29/12/2014].