Steady - Secure Software Development Report

Nimesh Ryan Silva
Florida Institute of Technology
20 PUBLICATIONS **0** CITATIONS

CYB 5660 Final Report

Nimesh R. Silva

Florida Institute of Technology

Author Note

Dr. Bulumulle

Introduction

Over the last few years, the adoption of open-source software (OSS) in the software industry has accelerated, and many commercial products now include a variety of OSS libraries. Any of these OSS libraries' vulnerabilities can have a significant impact on the security of the commercial product that bundles them. OWASP recognized the importance of this issue by included "A9-Using Components with Known Flaws" among the Top-10 security vulnerabilities in 2013. The public exposure of vulnerabilities such as Heartbleed and ShellShock in 2014 helped to increase awareness of the problem even more.

OSS libraries with known vulnerabilities are discovered to be utilized for some time after a corrected version has been provided, despite the deceptive simplicity of the available fixes (the most evident being: update to a more recent, patched version). At development time, updating to a more recent, non-vulnerable version of a library is a simple approach. When a vulnerable OSS library is part of a system that has already been deployed and made available to its users, however, the problem might be much more difficult to solve. Any change (including updates) to major enterprise systems that support business-critical functions may result in system downtime and comes with a cost.

This study proposes a pragmatic approach to decision-making that helps to simplify the process. This is accomplished by automatically creating actual proof supporting the argument for immediate patching (where possible). The application uses (part of) a library for which a security fix has been released in response to a vulnerability. The technique integrates seamlessly into the standard development cycle without demanding additional effort from developers and is independent of programming languages and vulnerability kinds.

Background

To determine whether a certain vulnerability in an OSS library is important for a certain application, look at the accompanying security patch, which is the set of changes made to the library's source code in response to the vulnerability. After that, the strategy is founded on the following pragmatic assumption:

- There is a considerable possibility of the vulnerability being exploited whenever a program that includes a library (known to be susceptible) runs a fragment of the library that would be updated in a security patch.

If the traces were gathered before the introduction of a security patch and all existing library versions are affected by the vulnerability, the library versions can be ignored. In other circumstances (for example, if traces are older than a patch and the corresponding constructs exist in both vulnerable and patched library versions), the version of the library producing the trace must be identified and compared to the versions affected by the vulnerability.

System Design and implementation

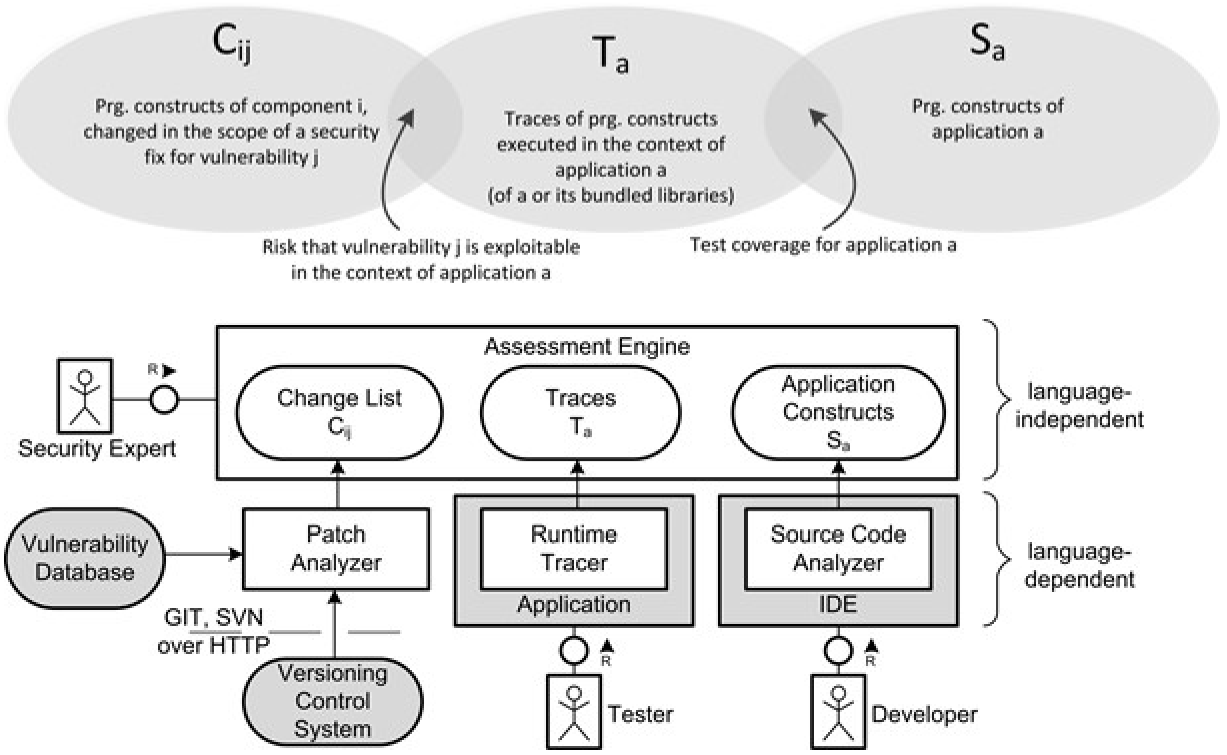
The source code can be founded here https://github.com/nrs011/steady. It is forked from eclipse/steady.

When a new vulnerability for an open-source library is published, the Patch Analyzer is triggered. It communicates with the appropriate Version Control System (VCS), detects all programming constructs that have been modified to address the vulnerability, and uploads their signatures to the central Assessment Engine. The vulnerable and patched revisions of all relevant source code files of the library are compared to create this change list. The relevant commit revisions can be found in the vulnerability database, searched in the commit log, or manually defined.

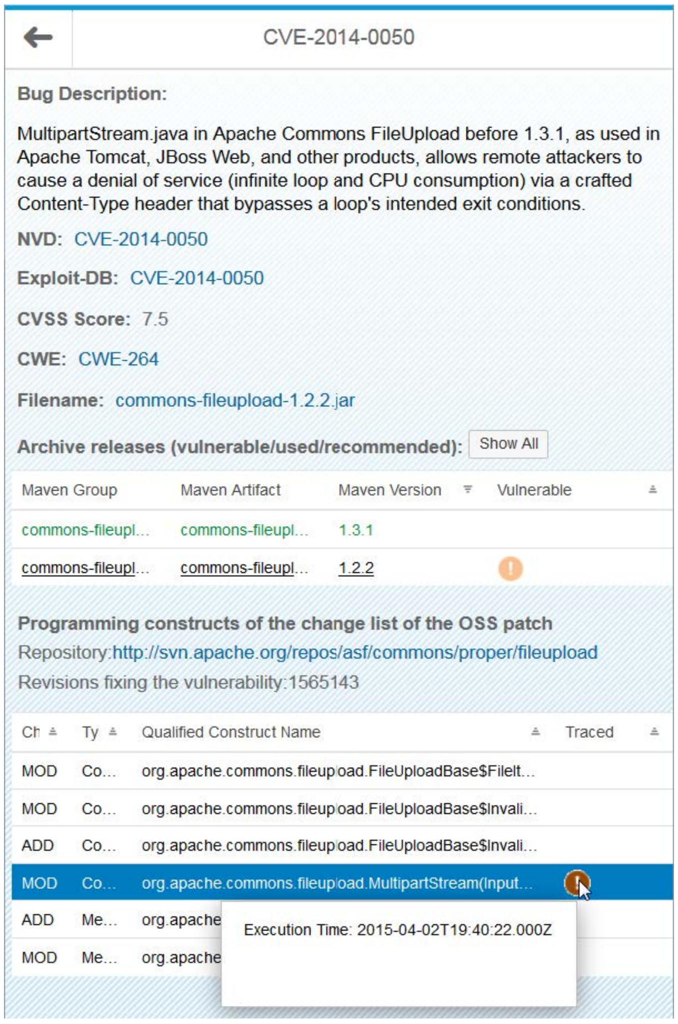
The Runtime Tracer gathers and uploads execution traces of programming constructs to the central engine. This is accomplished by introducing instrumentation code into all of the application's programming constructs as well as all of the included libraries. Instrumentation can be done in two ways: dynamically during runtime or statically before the application is deployed. All programming structures utilized at runtime, including libraries included at runtime and portions of the runtime environment, can be traced using the former.

The main disadvantage of the Runtime Tracer is its influence on application startup, especially when a large number of components must be loaded before the application can be made available to users, as in the case of application containers. Static instrumentation does not affect the startup time of the program and can be used in situations where the runtime environment cannot be set up for dynamic instrumentation, such as PaaS systems. It cannot, however, ensure that all programming constructs used at runtime are covered.

The Source Code Analyzer reads the application's source code, detects all of its programming constructs, and uploads their signatures to the central engine along with an application identifier.


C_{ij}
Prg. constructs of component i, changed in the scope of a security fix for vulnerability j
T_a
Traces of prg. constructs executed in the context of application a (of a or its bundled libraries)
S_a
Prg. constructs of application a
Risk that vulnerability j is exploitable in the context of application a
Test coverage for application a
Assessment Engine
Change List C_{ij}
Traces T_a
Application Constructs S_a
Security Expert
R
language-independent
Vulnerability Database
Patch Analyzer
Runtime Tracer
Application
Source Code Analyzer
IDE
language-dependent
GIT, SVN over HTTP
Versioning Control System
Tester
Developer

The Assessment Engine is realized utilizing an SAP Hana database where the change-list, trace-list, and programming constructs are stored and manipulated. The results are accessible via a web frontend.



CVE-2014-0050

Bug Description:

MultipartStream.java in Apache Commons FileUpload before 1.3.1, as used in Apache Tomcat, JBoss Web, and other products, allows remote attackers to cause a denial of service (infinite loop and CPU consumption) via a crafted Content-Type header that bypasses a loop's intended exit conditions.

NVD: CVE-2014-0050

Exploit-DB: CVE-2014-0050

CVSS Score: 7.5

CWE: CWE-264

Filename: commons-fileupload-1.2.2.jar

Archive releases (vulnerable/used/recommended): Show All

Maven Group	Maven Artifact	Maven Version	Vulnerable
commons-fileupl…	commons-fileupl…	1.3.1	
commons-fileupl…	commons-fileupl…	1.2.2	!

Programming constructs of the change list of the OSS patch
Repository:http://svn.apache.org/repos/asf/commons/proper/fileupload
Revisions fixing the vulnerability:1565143

Ch	Ty	Qualified Construct Name	Traced
MOD	Co…	org.apache.commons.fileupload.FileUploadBase$FileIt…	
MOD	Co…	org.apache.commons.fileupload.FileUploadBase$Invali…	
ADD	Co…	org.apache.commons.fileupload.FileUploadBase$Invali…	
MOD	Co…	org.apache.commons.fileupload.MultipartStream(Input…	!
ADD	Me…	org.apache	
MOD	Me…	org.apache	

Execution Time: 2015-04-02T19:40:22.000Z

Discussion

This methodology has natural applicability in continuous build and integration systems when considered as part of the whole software development lifecycle. When integrated into such systems, this tool may gather traces regularly and provide a timely warning when one or more of the libraries in use are discovered to be vulnerable.

At the time of writing, work on adapting this prototype to run as part of a Jenkins build is underway. It is planned to complete this implementation and assess it in significant development projects in the future (e.g., with over a hundred libraries).

Conclusion

This research took a practical approach to answer a fundamental question: Does a vulnerability in bundled open-source libraries influence an application? This method aids in determining whether a vulnerability requires immediate repair. It can be smoothly incorporated into industry-scale build and integration systems and is generic in terms of programming languages and types of vulnerabilities.

The conceptual approach, as well as a physical implementation as a tool, were described in this work, with the capabilities shown using an exemplary case.

References

OWASP. (2017). OWASP Top Ten. Retrieved June 5, 2021, from Owasp.org website: https://owasp.org/www-project-top-ten/

Williams, J. (2012). *THE UNFORTUNATE REALITY OF INSECURE LIBRARIES*. Retrieved from website: https://owasp.org/www-pdf-archive/ASDC12-The_Unfortunate_Reality_of_Insecure_Libraries.pdf