William Burger
Johns Hopkins Engineering for Professionals
Java Security 695.711.81, Fall 2018
Prepared for: Ebrima N Ceesay

Problem Description

One way that people try to secure their code is by making their programs open source. An advantage of open source projects is that other people can contribute and help fix bugs or algorithms. Open source is directly in contrast to the idea that keeping code and algorithms confidential can keep them secure. However, just because projects are open source doesn't necessarily mean they are immune to poorly written code or using vulnerable modules. The purpose of this paper is to gain a grasp of how vulnerable open source tools are by looking at prior research and creating a program that uses the Github API to see if a script can identify specifically java programs that have dependencies on modules known to be vulnerable.

Motivation

The motivation behind this paper and project is to make the community aware of just how vulnerable open source tools can be. This won't necessarily mean that the tools are bad in comparison to paid for proprietary tools, because those tools are also susceptible to poorly written code and lack of patching by developers. The vulnerabilities that will be checked will be ones that are public knowledge, meaning that people who developed the code are either unaware or unwilling to patch their vulnerable code. This is also interesting because Github itself monitors for known vulnerabilities on OWASP and MITRE websites, so it should be quick and easy to make the updates to code. The question becomes whether or not people are still spending resources to keep projects up to date, and whether these projects remain vulnerable on Github for any extended period of time.

Preliminary Literature Review

There is prior art in the field of analyzing open source tools for vulnerabilities. Github has integrations and even has done work finding vulnerabilities itself and alerting its users. While this may be the case, it doesn't look like Github does this specifically for Java. Second, from the perspective of the attacker, it doesn't matter that 500,000 vulnerabilities were corrected if there are many more programs that still have vulnerabilities on Github. Overall there is a consensus that because there are a lot of eyes on open source tools that vulnerabilities are detected and repaired more quickly than in the commercial software space. The counter argument to this is that vulnerabilities are also directly visible to attackers who can write scripts to go on Github and find vulnerabilities.

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=java

> Currently on the MITRE Common Vulnerabilities and Exposures website there are 1828 CVEs effecting Java in some way. This was found using a "Java" keyword search. This will be useful to the paper because vulnerabilities will need to be selected for searching through

Payne, Christian. "On the security of open source software." *Information systems journal* 12.1 (2002): 61-78.

> Its important to understand the generic pros and cons presented in this paper about the open source software community.

Hissam, Scott A., Daniel Plakosh, and C. Weinstock. "Trust and vulnerability in open source software." *IEE Proceedings-Software* 149.1 (2002): 47-51.

This paper provides the point of view of a cybercriminal – that open source has its benefit for an attacker because they can see the source code. The downside to the cybercriminal is that bugs frequently get fixed quickly. It would be interesting to get a metric on approximately how quickly bugs are fixed.

https://help.github.com/articles/about-security-alerts-for-vulnerable-dependencies/

Github itself provides an alerting feature for its users for Ruby, NPM and Python packages. The question is – are users of tools like this, or others that provide similar functionality, able to convince people to actually update the code or does the code remain insecure for enough period of time for someone to find it if they have an automated script that crawls the github pages?

https://blog.github.com/2018-03-21-security-alerting-a-first-look-at-community-responses/

This is a blog providing some interesting metrics on the GitHub vulnerability finding tool. Github published that they found over 4 million vulnerabilities in open source software using their tool, and then also mentioned that 450000 of these vulnerabilities were patched shortly after their vulnerability finding tool was patched. Is the tool doing enough? Are these vulnerabilities easy to find if you just have a crawler, making it easy for an attacker to choose a low-hanging fruit that no one has bothered to patch?

Proposed Approach

The proposed approach is to first find papers that mention how many vulnerabilities are expected to be found in open source software versus commercial software. The goal here is to get a sense for what other people are claiming they have found in terms of number of number of vulnerabilities in open source software and if there is a general consensus that open source software is less vulnerable than commercial software. If similar results are found that have assessed specifically java dependency vulnerabilities in open source tools, then the purpose of this project will be to reproduce those results. Once preliminary research is done on prior art, then some CVEs will be selected in Java packages from the MITRE or OWASP pages that will be used to assess whether or not any github projects contain those vulnerabilities. Then, a crawler will be written in Java that assesses open source github projects. The overall approach will be to make requests to the Github API searching for projects with the keyword “Java”, and then parsing these projects dependencies for matches to the chosen CVEs.

A count will be kept for the CVE that was hit and how frequent a project contains a vulnerability that was chosen. Once these metrics are collected it will be compared to the prior art statistics for expected number of vulnerabilities in open source software and in commercial software.

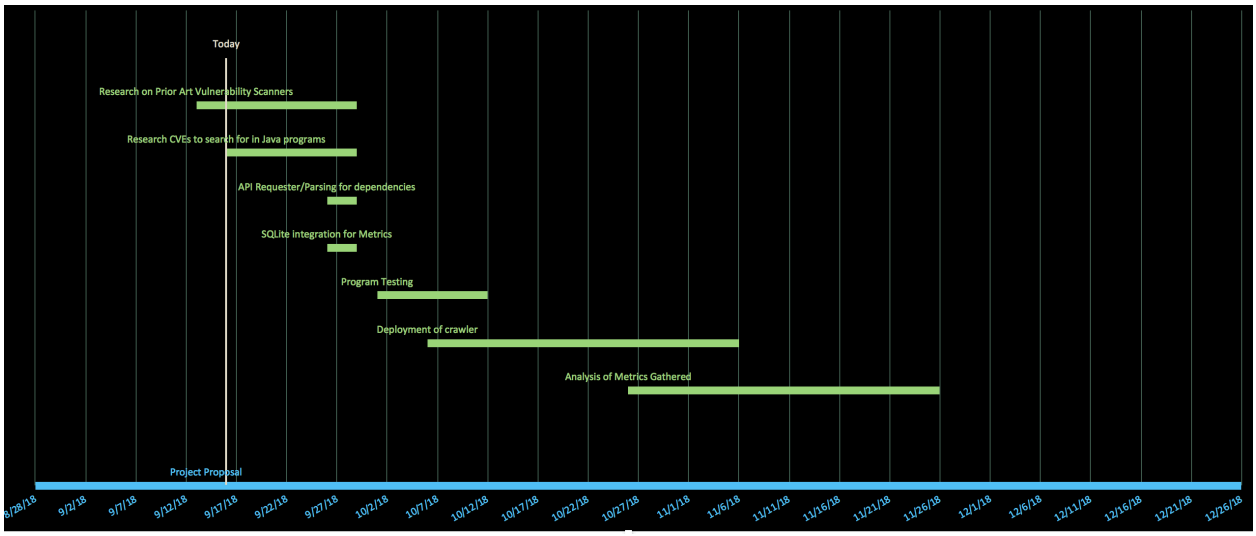
Another interesting metric to check will be date of release of the CVE to see if more recent CVEs are more likely to patched, or if in general some software is just more vulnerable than others. This should influence how the test CVEs are chosen – some should be recently uncovered vulnerabilities while others should be older.

Proposed Method for Evaluation
Success criteria will lie on how useful the metrics are that are collected:

- Do the metrics reflect prior research on the number of CVEs in open source software?
- Did the program manage to detect any vulnerabilities at all?
- Were appropriate candidate vulnerabilities selected to be easily detected in Java programs?
- Why would the program fails to detect any vulnerabilities?
    - Are open source programs secured?
    - Did it take too long to scan for the vulnerabilities, so not enough programs could be assessed to see if they had the vulnerability that was looked for?

Finally, the research and results should be well presented and documented in a final paper.

Time Plan:
Below is a gantt chart created for the project. The initial phase is research and planning for the project, while the middle phase is building and deploying the crawling program while the end of the class will be dedicated to the final paper.



Milestones

No.	Position	date	Milestone
1		9/14/18	Project Proposal
2		10/12/18	Project Milestone 1
3		11/9/18	Project Milestone 2
4		12/7/18	Project Final Paper

Tasks

No.	Start Date	End Date	Task
1	9/13/18	9/28/18	Research on Prior Art Vulnerability Scanners
2	9/16/18	10/6/18	Research CVEs to search for in Java programs
3	9/26/18	10/18/18	API Requester/Parsing for dependencies
4	9/26/18	10/3/18	SQLite integration for Metrics
5	10/1/18	10/11/18	Program Testing
6	10/6/18	11/5/18	Deployment of crawler
7	10/26/18	11/25/18	Analysis of Metrics Gathered