CSEC 793 Capstone in Computing Security
Project Report

MS CAPSTONE LITERATURE REVIEW

February 21, 2018

Ryan Whittier
Department of Computing Security
College of Computing and Information Sciences
Rochester Institute of Technology
rjw4910@g.rit.edu

1 Introduction

The project I propose is the development of a security focused gitlab continuous integration (CI) pipeline. I wish to make a pipeline that focuses on security in an attempt to improve application security. This pipeline will automatically run a series of tests against a repository of configuration files and code. These tests will enforce correct configurations and security practices within committed changes.

CI is a technique that is used to lead to better development practices, but also to speed up development releases. Those benefits can also be tailored to improve security in development. Many penetration tests or bug reports revolve around the same information repeated for different cases. These reports often have a common solution. If we discuss web applications, one common vulnerability is cross site scripting (XSS). XSS vulnerabilities have the short term recommendation of encoding all user input to avoid users input being interpreted as part of the webpage. The long term recommendation is to set up a full Content Security Policy (CSP), limiting where scripts and HTML tags can be loaded from. Companies generally choose the short term route of fixing each individual case, but this will cost them time and money in the long term, eventually they decide to implement a long term solution. In this example companies will often implement encoding, but as they expand the web application, they find the same bug. The company eventually switches to creating a CSP and configuring it correctly so new expansions can not be affected by a XSS bug.

Everyday applications are getting larger and are handling enormous amounts of data. Many of these applications are services that companies provide, such as Facebook and Netflix. All of these applications need to provide security for both their internal company data and the end user's data. The problem is that as these applications grow, their logic gets more complex and mistakes will be made, potentially introducing vulnerabilities into the codebase without the developers knowing it. Sometimes these mistakes are simply forgotten flags on cookies and other times they can be complex flaws in the business logic of the application. These mistakes can result in damage ranging from defacement of a company site, to a complete breach of a company's internal network.

A bug does not need to be exploited by a malicious actor to cause financial loss for the company. Many companies offer bug bounty programs, in which a researcher can discover and report a bug for a reward. The rewards vary by company, but often times the researcher will receive a substantial monetary reward, depending on the severity of the bug discovered. This means that each time the company releases updates to their applications, researchers will be scavenging for more vulnerabilities, which can costly for the company. It may also cost the company's security team a significant amount of time, as they need to look into and verify each report that is submitted. Often times, researchers can find very common vulnerabilities in the application, that usually result from developers attempting to push out new features at such a rapid pace.

CI can be used to catch simple mistakes automatically, by ensuring that the common bugs are found before the changes are deployed into production. By finding these bugs

before they're introduced into production, the number of submissions from bug bounty programs will decrease dramatically. This means that the bug reports that are submitted by researchers will be of higher quality and provide more value to the company.

2 Literature Review

- How, when the problem was raised by whom, what event, etc.
- Any attempts have been proposed to attack the problems
- Pros and Cons of each existing solutions
- How does your proposed solution fit within here, i.e., your solution is improving an existing solution, just another solution, or revolutionizing the way of thinking?

Make sure to cite references. Here is how to cite the great textbook written by Knuth [6]in the latex. Here is the way how to cite a journal article written by the father of computer science Alan Turing [8].

2.1 Introduction

Continuous Integration (CI) has grown in popularity, being used in both enterprise and open source projects. CI provides an increase in productivity to programming projects by creating an environment where building, testing, and often deployment is done automatically. When developers do not need to do these tasks manually they can spend more time programming and finding bugs that trigger a failed build or failed test. Most research into CI looks at increasing productivity of developers and the enforcement of development standards. The potential use of CI for security has been mostly overlooked with most research looking at exploiting and protecting a CI system or a CI pipeline and not at the benefits of a pipeline focusing on application security.

A separate pipeline for security could provide a number of benefits for a security baseline. It could force certain configurations such as Content Security Policy in web applications, or lack of use of depreciated crypto algorithms such as DES or TripleDES[9]. A pipeline that looks only at big win configurations such as these would serve a similar function as Automated Source Code Analysis Tools (ASCAT) do when looking at code meeting project coding guidelines[10]. The pipeline could also include ASCATs that search for security bugs in a warning stage to avoid wasted time with failed builds from false positive. Another potential feature is an inclusion of fuzzers, which were recently used against memory forensic tools to look into anti-forensics techniques through crashing the tools[2].

2.2 Test case generation

Test case generation is a discipline that focuses around automatically generate unit tests for Software Engineering projects. The discipline sometimes focuses on generating tests from a base, such as the source code or a config file, and generating a set of tests from the base or it focuses on taking existing tests and exanpding them to cover more features.

One paper from the 2014 Automated Software Engineering conference looked at generating test cases for web from an existing selenium test suite[3]. This paper goes through the method that is used to generate the new test cases. The method starts by taking a selenium test suite and going though it keeping track of states, http element interactions, and test case assertions. It uses the base states and element interactions to create a State-Flow Graph, which is a graph of the states with the interations that that move from one state to the next and the assertions that are involved to a single state. It then uses a web crawler on each state to discover alternate paths and new states. It then makes use of assertions on the base states to generate new tests for the crawler discovered states. An example given in the paper is that an application that allows you to create notes is tested with selenium by logging in, clicking the create note element, and then verifiying that an id element had specific text. The test suite was then expanded by crawling to find the edit note interaction and the delete note interaction. These were then clicked and tested using the assertion, verifying that the id was present and changing the text to match the new interactions result text.

2.3 Effective use of CI

Continuous integration has been a fantastic addition to Software Engineering practices by removing repetitive administrative tasks. The less steps a software engineer has to go through when producing new code, the more time they can spend on adding features, squashing bugs, or otherwise improving the project code base. There is plenty of research done into the use of CI including many case studies, papers and presentations looking at how to setup CI and Continuous Deployment (CD), and papers looking at specific configurations of CI.

CI can be integrated with all sorts of tools to increase productivity. The paper How Open Source Projects use Static Code Analysis Tools in Continuous Integration Pipelines by Fiorella Zampett looks at how Static Code Analysis tools in a CI pipeline effects projects[10]. The paper found that the most utilized feature of ASCATs was to ensure a project's code was consistent by ensuring the developers coding guidelines were met. The paper also found that ASCATs caused broken builds to be fixed quickly in an average of 8 hours and one build. There are a number of recommendations that this paper provides which outline how to set up ASCATs in a CI pipeline, what to think about when doing so, and what to expect to maintain the ASCAT. Adding a source code analysis tool will help keep a project consistent while helping point out bugs that could be missed by developer made unit tests.

Two papers look at CI and how it affects projects in general. Continuous Integration and Quality Assurance: A Case Study of Two Open Source Projects by Jesper Holck and Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects by Michael Hilton both look at open source projects and how CI affects them[4, 5]. The papers found that CI often replaces the practice of having developers make formal design documents. The developers instead just pick from a list of tasks and works on completing them and

merging them into the code base. The papers also found that most developers like CI and plan to use it again in future projects. For projects that did not use CI it was found that the reason was usually just that the developers were not familiar with how to set up and use CI. Both papers found CI to be successful in increasing productivity, causing projects to release twice as often, accept pull requests quicker, and have developers less worried about breaking the project.

2.4 Security in CI pipelines

Security conferences often look into how to break systems as a way of pointing out the flaws of a configuration. CI servers have also had their share of security professionals testing for exploits and looking at the consequences of compromise. CI has also had a little bit of research into how to secure a CI pipeline.

A presentation at DEFCON named Exploiting Continuous Integration (CI) and Automated Build Systems talked about the consequences of a CI enabled projects[7]. The presentation found that exploiting a repo that holds a CI integration ends with a huge amount of access. If the repo links into an internal CI server, then the attacker ends up with internal network access. If the repo links to a CI server with multiple CI instances or also runs the CD, then the attacker will get more source code and access to the deployment machines because the CI holds a way to connect to the deployment servers. Otherwise the attacker ends up with environment variables which often hold extremely sensitive information.

A paper by Len Bass called Securing a Deployment Pipeline looks at how to secure a CI pipeline to limit the damage in case of exploitation[1]. The paper details a way to break a pipeline down into trusted and untrustworthy parts, segmenting operations until an untrustworthy segment cannot be broken down any more. The CI pipeline then holds parts that are guaranteed to be trustworthy and run as expected, minus specific cases outlined in the paper, and parts that may provide untrustworthy output. By limiting the scope of untrustworthy parts, the rest of the pipeline can run as expected and it is possible to see where the most risk lies. Then the owners of the pipeline can work to limit access to the untrustworthy portions and look into solutions to make those portions trustworthy.

2.5 Use of CI for Security

There is little research into the use of CI for security.[1] One presentation by Mozilla looks at the use of CI to tackle easy fixes in response to their bug bounty program. The presentation looks at using CI to ensure that the production environment contains configurations that mirror best practices for common web application security bugs. Some examples include HSTS is enabled, CSP for XSS bugs, various X-OPTIONS headers, Cookies have secure, Cross origin sharing, and Subresource integrity checks. The presentation recommends figuring out a security baseline for a projects CI pipeline, drive testing from the CI pipeline,

and empower the team to fix the issues. Another recommendation is not to break on deployment into production as that could break the production site if configured poorly. The end result of mozilla's CI setup was a large drop in bug reports that the CI tests aimed to fix.

3 Work and method

The work into CI so far has shown that CI is useful tool for developers to ease the creation of new features. CI has spread to open source projects and is deployed in most organizations that have at least one large code project. The security side of how a CI is dangerous if exploited and how to secure a pipeline against attacks has had little research. The most interesting thing that I think is lacking is the look into how CI can be used to improve the security the project that it is integrated on.

There are two ways I feel CI could help improve the security of a project. One is in targeting common bugs that are already fixed. Some examples are ensuring that CSP is enabled, HSTS is enabled, parameterized queries are used, binary protections are enabled in compilation scripts, and unsafe functions are not used. Another use could be in including fuzzing in a pipeline. I have not seen any research into automating fuzzing into testing an application. Depending on the project it could be a very useful tool to find bugs.

I am implementing a security focused pipeline which will recommend a number of the common bugs to test for. The pipeline will also include some ASCATs, looking at the output from them in a warning state. The output can still be followed up on to decide whether or not it is a false positive.

References

[1] Len Bass. “Securing a Deployment Pipeline”. In: (2015).

[2] Andrew Case. “Gaslight: A comprehensive fuzzing architecture for memory forensics frameworks”. In: (2017).

[3] Amin Milani Fard. “Leveraging Existing Tests in Automated Test Generation for Web Applications”. In: (2014).

[4] Michael Hilton. “Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects TODO”. In: ().

[5] Jesper Holck. “Continuous Integration and Quality Assurance: a case study of two open source projects”. In: (2017).

[6] Donald E Knuth. *Art of Computer Programming, Volume 4, Fascicle 4, The: Generating All Trees–History of Combinatorial Generation*. Addison-Wesley Professional, 2006.

[7] spaceb0x. *Exploiting Continous Integration (CI) and Automated Build Systems, DEFCon 25*. URL: `https://media.defcon.org/DEF\%20CON\%2025/DEF\%20CON\%2025\%20presentations/DEFCON-25-spaceB0x-Exploiting-Continuous-Integration-UPDATED.pdf`.

[8] Alan M Turing. “Computing machinery and intelligence”. In: *Mind* 59.236 (1950), pp. 433–460.

[9] Julien Vehent. *Test Driven Security in Coninouos Integration, Enigma*. 2017.

[10] Fiorella Zampetti. “How Open Source Projects use Static Code Analysis Tools in Continuous Integration Pipelines”. In: (2017).