



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Leadership Essentials for Managers (Cybersecurity Leadership 512)"
at <http://www.giac.org/registration/gslc>

Changing the DevOps Culture One Security Scan at a Time

GIAC (GSLC) Gold Certification and ISE 5501

Author: Jon-Michael Lacek, jon-michael.lacek@student.sans.edu

Advisor: *David Fletcher*

Accepted: 8/22/2019

Abstract

Information Security has always been considered a roadblock when it comes to project management and execution. This mentality is even further solidified when discussing Information Security from a DevOps perspective. A fundamental principle of a DevOps lifecycle is a development and operations approach to delivering a product that supports automation and continuous delivery. When an Information Technology (IT) Security team has to manually obtain the application code and scan it for vulnerabilities each time a DevOps team wants to perform a release, the goals of DevOps can be significantly impacted. This frequently leads to IT Security teams and their tools being left out of the release management lifecycle. The research presented in this paper will demonstrate that available pipeline plugins do not introduce significant delays into the release process and are able to identify all of the vulnerabilities detected by traditional application scanning tools. The art of DevOps is driving organizations to produce and release code at speeds faster than ever before, which means that IT Security teams need to figure out a way to insert themselves into this practice.

1. Introduction

The DevOps process is steadily gaining popularity throughout organizations; however, IT security continues to remain absent within the process. There have been several studies, such as the Ponemon Institute survey and Gartner that indicate the security tools available are too complex to integrate into a DevOps release pipeline or that they cannot perform an adequate security assessment compared to the stand-alone appliances.

In organizations where application security assessments are being conducted, they are traditionally assessed towards the end of the project plan as one of the final steps before the application is scheduled to be released. The IT security team will obtain the source code to perform static code analysis, followed by a dynamic assessment which is typically managed in a certification environment where the application has been built and deployed. The results of these scans are then compiled and presented to the application developers. At this point, the development team and security team collaborate to understand the vulnerabilities presented in the document, determine how to correct them, and finally, estimate how quickly they can perform remediation. Project managers and business stakeholders have a decision to make: delay the application release in order to allow developers time to remediate or place compensating controls around the discovered vulnerabilities, or accept the risk that the known vulnerabilities present to the organization.

When organizations involve the IT security team early in the software development lifecycle, the overall risk to the organization is significantly reduced. One solution for early integration is to configure scanning tools to be utilized in the DevOps pipeline build and release process. Not only will security teams be involved from the beginning, but this integration will now produce a continual feedback loop for developers each time they check their code back into their code repository. Since this option is available in the most common release pipeline toolsets such as Jenkins, Azure DevOps, or AWS, why has the practice of securing an application not become a standard exercise across all organizations with a DevOps culture?

Author Name, email@address jon-michael.lacek@student.sans.edu

The research presented in this paper will explore the work effort needed to integrate some of the existing application scanning extensions available in the most frequently used DevOps pipeline release products. It will also evaluate the quality of scanning that they provide compared to the tools used in a traditional source code and dynamic analysis engine.

2. Literature Review

The digital transformation is well underway across all business verticals, and the culture of DevOps is at the heart of the movement. High performing organizations, such as Google, Amazon, Facebook, Etsy, and Netflix, are routinely and reliably deploying code into production hundreds, or even thousands, of times per day, using the Continuous Integration Continuous Deployment (CI/CD) methodology (Kim, Debois, Willis, Humble, & Allspaw, 2017). How do organizations continue to create new applications or integrate feature enhancements to existing applications and deliver them at such a rapid pace all while ensuring the deployed code does not contain any vulnerabilities? A continual feedback loop is the cornerstone of the Agile development process. Therefore, when executed correctly, developers will continually receive feedback on the vulnerabilities present in their code. This new feedback loop will slowly change the security mindset of the developers, consequently making secure coding a fundamental skill within the organization, and thus creating a culture commonly referred to as DevSecOps.

While executives strive for and expect the continual growth of application deployments and feature updates, they, for the most part, are aware of the security concerns with such a rapid deployment schedule. Of the C-suite respondents surveyed in the State of DevOps report, 64 percent believe security teams are involved in technology design and deployment versus 39 percent of respondents whose primary role is that of an individual contributor at the team level (“State of DevOps Report”, p. 6). To address this gap and offer solutions, it is suggested that “The best way to get everyone on the same page is through reinforcing the DevOps pillars of automation and measurement. Automated systems enable better reporting of metrics that can be shared across the

Author Name, email@address jon-michael.lacek@student.sans.edu

business” (“State of DevOps Report”, p. 6). To start gathering metrics related to the overall security of the deployed applications, organizations need to automate the scanning process. The only way to gather comparable, consistent metrics is to integrate the application scanning tools into the CI/CD pipeline.

More than ever, effective management of technology is critical for business competitiveness. For decades, technology leaders have struggled to balance agility, reliability, and security (Kim, Debois, Willis, Humble, & Allspaw, 2017). With 87 percent of people surveyed in the Ponemon survey believing that digital transformation is essential to business, and 63 percent stating that IT security is essential to supporting innovation with minimal impact on the goals of digital transformation, why are security tools missing in the CI/CD pipeline (“Ponemon Survey”, p. 14)?

One of the challenges presented in the Ponemon survey was that a barrier to achieving a secure digital transformation process was due to the complexity of the business processes, which 56% of respondents supported. Additionally, 44% reported that there is a lack of adequate security technology solutions available to successfully inject IT security into the continual release life-cycle (“Ponemon Survey”, p. 7). The good news for the IT security community is that effectively delivering DevSecOps has been one of the fastest-growing areas of interest of Gartner clients. These concerns illustrate the need for a culture shift within the IT security field. Integration, however, cannot continue to start at the end of the Software Development Life Cycle (SDLC), as it has traditionally. If IT security teams want to partner with businesses, they must not continue to be a roadblock. Information security must adapt to development processes and tools, not the other way around (MacDonald, 2017).

This study aims to demonstrate that the complexity of integrating the necessary tools in a CI/CD pipeline is no greater than the time or expertise needed to provide a security assessment of a web application when performed independent of the deployment of that same application. Moreover, “when security is integrated into the DevOps culture, high performing teams spend 50 percent less time remediating security issues than low performers. This is because they build security into the SDLC in contrast to retrofitting security at the end” (“State of DevOps Report 2018”, p. 72).

Author Name, email@address jon-michael.lacek@student.sans.edu

By enabling the automation of static and dynamic analysis tools in a release pipeline, development teams automatically receive the vulnerability information present within their code each time the application is checked into its repository or sent through the build and release pipelines. The time savings realized when utilizing this process is two-fold. First, developers will be able to deploy applications or feature releases quicker as their secure coding skills increase. Security analysts can now spend their time on dynamic assessment, ethical hacking or red teaming instead of using their time continually configuring tools to scan applications. Perhaps the greatest benefit isn't the time savings itself, but rather that application scanning is taking place daily, at minimum, rather than only during the times IT security teams are made aware of application or feature releases and are given the appropriate amount of time to assess those deployments.

3. Research Method

3.1. Lab Design

This research is being conducted using the Microsoft suite of tools available in the Azure cloud. An Ubuntu server has been created to host an application. WebGoat, “a deliberately insecure web application” (OWASP.org), will be used as the application to evaluate the Azure DevOps pipeline extensions capabilities. The testing will include both the static code analysis along with the dynamic application scanning aspects that IT security analysts typically take when evaluating the overall security of a web application that is ready for consumption by the customer.

Static code analysis will be conducted during the build process, while the dynamic application scan will take place during the release pipeline. The configuration of the build pipeline references the WebGoat code repository which has been downloaded and stored in the Azure Git repositories for analysis. When the application is configured to be deployed, a Docker container will be utilized to run WebGoat on a virtual Ubuntu server in the Microsoft Azure Portal. The OWASP Zed Attack Proxy (ZAP) has been installed on a second Ubuntu server. The ZAP tool will be configured to point to the deployed WebGoat URLs for application analysis.

Author Name, email@address jon-michael.lacek@student.sans.edu

3.2. Testing Methodology

3.2.1. Establishing a build and release baseline

Establishing a baseline requires measuring the time it takes to build and release the WebGoat application without tying in any security extensions. WebGoat will be compiled using the Maven extension, “a tool that can now be used for building and managing any Java-based application” (“Apache Maven Project”, 2019). Once built, security extensions such as SonarQube and ZAP will be integrated into the pipeline configurations to automatically evaluate the application from a static code analysis standpoint, as well as a dynamic analysis standpoint while it is in the process of deployment.

The first phase in the pipeline release process involves the build of the application. The WebGoat version 8.0 application files have been downloaded and stored in a Microsoft Azure Git repository. The build pipeline is configured to reference this code repository and compile the application. See Appendix 1 for the overview of the configuration. Capturing the time to build WebGoat without any static code analysis taking place establishes the baseline. The Microsoft Azure DevOps pipeline interface provides timings for each step in the build process, which takes approximately 3 minutes and 29 seconds to compile, as shown in Figure 1 below.

✓ Prepare job · succeeded	<1s
✓ Initialize job · succeeded	<1s
✓ Checkout · succeeded	9s
✓ Maven pom.xml · succeeded	3m 16s
✓ Post-job: Checkout · succeeded	<1s
✓ Finalize Job · succeeded	<1s

Figure 1. Baseline build pipeline results.

The application is then deployed using the Azure DevOps release pipeline configuration to a virtual Ubuntu 18.04 server in the Microsoft Azure cloud. See Appendix 2 to view the release pipeline configuration for this stage. This configuration references a docker-compose.yml file that is used to deploy the WebGoat application in a

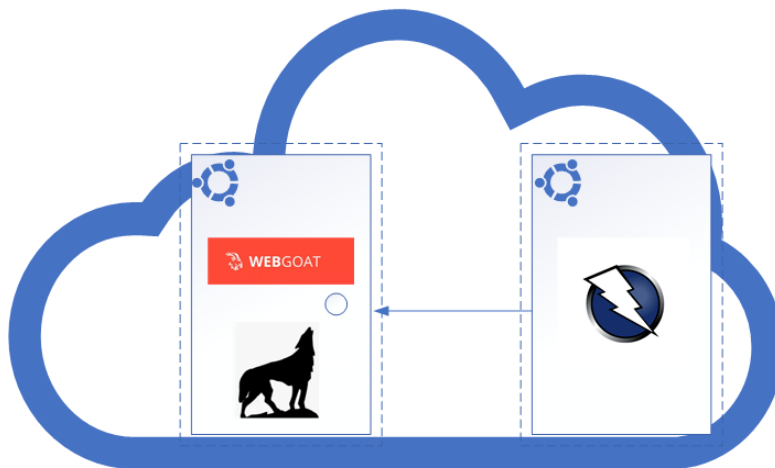
Docker container hosted on the virtual Ubuntu server. When initiated, this process takes approximately 28 seconds, as shown in the figure below.

✓ Initialize job · succeeded	2s
✓ Download Artifacts · succeeded	12s
✓ Copy Files to: /home/researchaccount/ · succeeded	<1s
✓ Command Line Script · succeeded	12s
✓ Finalize Job · succeeded	<1s

Figure 2. Baseline release pipeline results.

3.2.2. Establishing a ZAP scan baseline

At this point, the WebGoat application is running in a container on a VM. The installation of the OWASP Zed Attack Proxy takes place on a second virtual Ubuntu 18.04 server running in the Microsoft Azure portal. The configuration of the proxy is then conducted to reference the WebGoat application previously installed and referenced in Section 3.2.1. Figure 3 depicts the two virtual Ubuntu servers hosted within the Azure cloud with one server hosting WebGoat and the other hosting ZAP. This process is what is typically involved in a traditional means of dynamic application scanning where IT security analysts are provided with the web applications URL for configuration and analysis within the application scanning tool.



Author Name, email@address jon-michael.lacek@student.sans.edu

Figure 3. Virtual servers with hosted applications in Azure cloud

The WebGoat application is composed of two applications that interact with each other, http://host_ip_or_dns-name:8080/WebGoat/ and http://host_ip_or_dns-name:9090/WebWolf/. When the ZAP tool is configured to point to those URLs and perform a spider of the sites, followed by an active scan, the following results are obtained.

Time to spider the WebGoat application	20 seconds
Time to scan the WebGoat application	1 minute, 56 seconds
Time to spider the WebWolf application	21 seconds
Time to scan the WebWolf application	2 minutes, 5 seconds

Table 1. Baseline application scan results

The following chart documents the results of the active application security scan of both applications in the Docker container:

Vulnerabilities	Critical	High	Medium	Informational	Total Number of URLs
WebGoat	2	1	3	0	11
WebWolf	2	0	0	0	5

Table 2. Baseline vulnerability scan results

3.2.3. Automatic Build Integration

With the baseline timing to build the WebGoat application established, it is time to gather statistics indicating how much additional time static code analysis tests take when integrating with a build pipeline. The original build pipeline configuration referenced in Section 3.2.1 has been modified to incorporate SonarQube extensions: “SonarQube is an open source product for continuous inspection of code quality” (Dockerhub.com). Since WebGoat is a Java-based application, SonarQube was chosen as the Azure pipeline extension to utilize. See Appendix 3 to view the pipeline configuration

Author Name, email@address jon-michael.lacek@student.sans.edu

used in this analysis. To find additional static code analysis applications, visit www.owasp.org/index.php/Static_Code_Analysis. The list presented on the OWASP site provides the available coding languages that are supported by each tool.

When this build process starts, the *Prepare analysis on SonarQube*, *Bash Script*, and *Publish Quality Gate Result* steps are the additional steps configured in the pipeline. The *Maven pom.xml* step includes a checkbox that could integrate directly with SonarQube. However, to capture the additional time introduced to the build process for this research, the SonarQube integration commands have been configured using the *Bash Script* step.

✓ Prepare job · succeeded	<1s
✓ Initialize job · succeeded	2s
✓ Checkout · succeeded	9s
✓ Prepare analysis on SonarQube · succeeded	<1s
✓ Maven pom.xml · succeeded	3m 9s
✓ Bash Script · succeeded	38s
✓ Publish Quality Gate Result · succeeded	36s
✓ Post-job: Checkout · succeeded	<1s
✓ Finalize Job · succeeded	<1s

Figure 4. Build pipeline results with SonarQube integration.

As Figure 4 demonstrates, integration with SonarQube only takes an additional 75 seconds in this build pipeline. The steps required to prepare SonarQube and reconfigure the build pipeline to publish the testing results to the SonarQube portal cannot be captured systematically. The steps needed for initial integration include deploying a SonarQube server, or virtual instance of SonarQube, configuring a new project within SonarQube, and obtaining the necessary code snippet that is needed for configuration into the build pipeline for integration. Deploying and configuring a SonarQube server is a one-time setup. Configuration of each project pipeline that organizations are looking to integrate occurs once per project. After connecting the SonarQube project within each

application pipeline, developers now have the benefit of continually receiving feedback about their code immediately after the completion of the build process.

Now that the SonarQube extension has been configured to integrate with the build pipeline, the build is initiated and then completes which subsequently publishes the results of the static code analysis to the SonarQube project page. Developers now have an extremely easy-to-use tool that describes what the problem is, why it is a problem, and the location of that problem within the code.

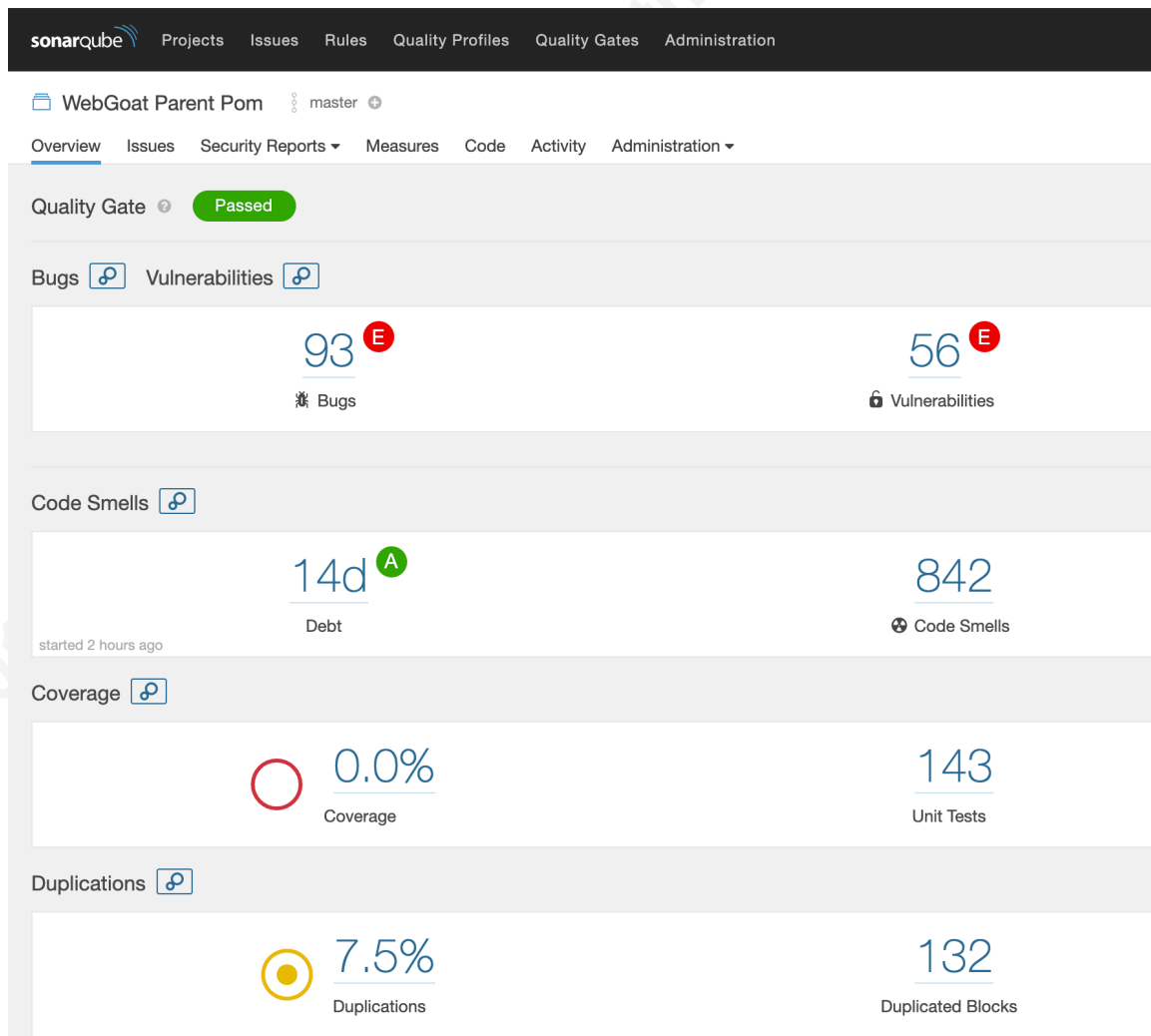


Figure 5. Results of WebGoat static code analysis

As depicted in Figure 5, developers have a summarized view of issues within their application. The SonarQube extension evaluates the application and documents any piece of code that breaks a coding rule within its analysis. The evaluation of the application's

Author Name, email@address jon-michael.lacek@student.sans.edu

code is broken down into one of three categories. These categories are bugs, vulnerabilities, and code smells (SonarQube Resources). A bug is a problem within the code that will break the application, while a vulnerability is a point in the code that is open to attack. An algorithm is then used to generate the report and assign an overall grade to the application. While the scores will be interpreted differently across organizations, it is important to note that the developer now has a tool that provides immediate feedback each time the code is built. Tools like SonarQube provide resources to understand why that particular coding practice creates a vulnerability or bug and examples on how to correct the problems detected within the code. By repetitively reviewing problems within their code, over time, developers will change their poor coding habits.

In addition to the benefits realized from an application security perspective, most of the pipeline extensions provide valuable reports that contribute to enhancing the quality of the application in development. The SonarQube extension contains a section referred to as code smells, which highlights segments of the code that are confusing and difficult to maintain. The technical debt feature estimates the time it would take to correct the code smells. Code coverage test results is another extremely beneficial practice that should be exercised in a DevOps pipeline process as organizations mature. The more testing that takes place against specific variables, functions, or subroutines, the lower the chance of running into a previously undetected bug. The final measurement provided in Figure 5 is duplications. This metric captures the amount of repetitive code used throughout the code base, which plays a critical role in identifying where poor coding practices have sprawled throughout the application.

3.2.4. Automatic Release Integration

After the completion of the build process, a release pipeline can be constructed to deploy those artifacts. The WebGoat application will be deployed using a container to demonstrate the various options available in common pipeline toolsets, not the artifacts previously created in Section 3.2.3. Figure 6 shows the screenshot of the pipeline configuration with the addition of the ZAP baseline stage that deploys and utilizes the OWASP ZAP Docker image for application scanning.

Author Name, email@address jon-michael.lacek@student.sans.edu

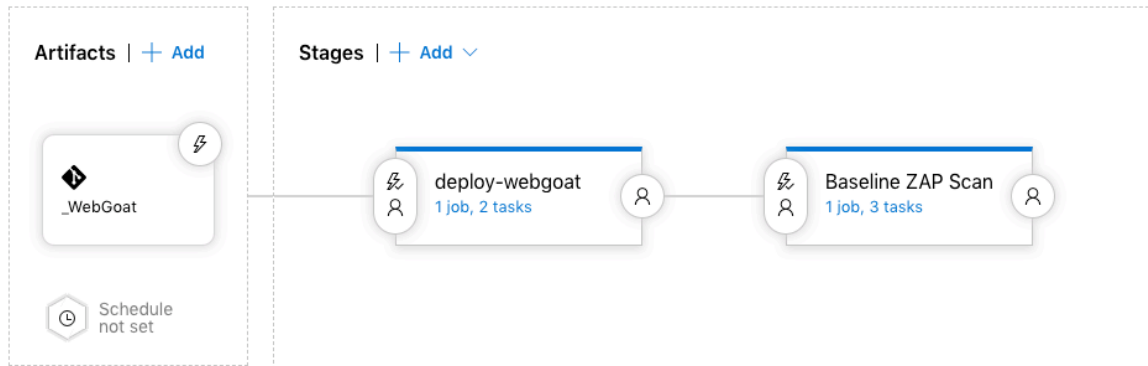


Figure 6. Azure release pipeline with application scanning tool integration

After the application has been successfully deployed in the first stage of the release, the initiation of the baseline scan takes place within the second stage, named *Baseline ZAP Scan*. This scan will first spider the site, followed by an active scan. As shown in Figure 7 below, the log messages within the console contain the results of the scan in the form of warnings. See Appendix 4 for the commands used to initiate the Docker scan within the release pipeline.

```

PASS: Insecure HTTP Method [90028]
PASS: Loosely Scoped Cookie [90033]
WARN-NEW: Cookie Without SameSite Attribute [10054] x 2
  http://10.0.0.7:9090/WebWolf/
  http://10.0.0.7:9090/login
WARN-NEW: Absence of Anti-CSRF Tokens [10202] x 3
  http://10.0.0.7:9090/login
  http://10.0.0.7:9090/login?logout
  http://10.0.0.7:9090/login?error=true
WARN-NEW: Anti CSRF Tokens Scanner [20012] x 4
  http://10.0.0.7:9090/login?error=true
  http://10.0.0.7:9090/login
  http://10.0.0.7:9090/login?logout
  http://10.0.0.7:9090/webjars/jquery/3.2.1/jquery.min.js
FAIL-NEW: 0 FAIL-INPROG: 0 WARN-NEW: 3 WARN-INPROG: 0 INFO: 0 IGNORE: 0 PASS: 68
Full ZAP Scan - Ending
  
```

Figure 7. Output from the ZAP active scan initiated from the release pipeline.

The commands used in the scan steps above combined the spidering of the URLs along with the scanning of the site. The addition of this stage adds a total of 3 minutes, 35 seconds to the release pipeline as shown in Figure 8.

✓ Initialize job · succeeded	1s
✓ Download Artifacts · succeeded	15s
✓ Download latest ZAP Docker Release · succeeded	<1s
✓ WebGoat Scan · succeeded	1m 49s
✓ WebWolf Scan · succeeded	1m 28s
✓ Finalize Job · succeeded	<1s

Figure 8. Results of each phase of the release pipeline.

The following two charts show the time taken to scan each application along with the number of vulnerabilities discovered for each URL. By default, the vulnerability results of the integrated scan do not provide the severity levels. Instead, the console logs include a warning message with the name of the vulnerability discovered along with the URL where it exists. Security analysts can modify a configuration file that allows for the configuration of warning types to better align with their organizational goals. This research will utilize the default configuration to measure the number of vulnerabilities discovered for each of the two applications.

Time to spider and scan the WebGoat application	1 minute, 49 seconds
Time to spider and scan the WebWolf application	1 minute, 28 seconds

Table 3. Application spider and scan timings.

	Number of Vulnerabilities Discovered	Number of URLs
WebGoat	4	10
WebWolf	3	9

Table 4. Application vulnerability count in automated pipeline integration.

4. Findings and Discussion

The results of the testing conducted in this research confirmed the hypothesis that the tools available in CI/CD pipelines have the same capabilities that stand-alone products have. Additionally, the time needed to integrate these tools is not significantly shorter or longer than the manual scanning methodology that traditional security teams utilize.

The success of integrating application scanning extensions within a CI/CD pipeline hinges on a well-planned process. The research conducted here demonstrated easy-to-accomplish steps to determine the work effort needed for organizations to make the leap and start integrating security tools into their application development pipelines. Once the integration of security tools into the DevOps pipelines conclude, security analysts have the ability to continue to enhance the scanning capabilities and the actions taken as a result of those scans. This configuration brings security into the CI/CD pipeline process by allowing them to continue to build upon the basic scanning tests conducted in this research.

Much like the developer of the applications utilizing a CI/CD pipeline, the security analysts responsible for integrating these extensions will continually be able to introduce enhancements in the form of advanced application checks. Accomplishing these enhancements starts by thinking about how a security analyst typically scans an application from the manual scan standpoint. When manually scanning, application security analysts use results displayed in the GUI from different portions of the scan to pivot to a more in-depth analysis of the code. For example, when starting an application scan, a security analyst may typically spider the application. Using the results of the spider, the analyst may then choose to attack each of the URLs detected. The next step may be to use the discovered forms and begin to fuzz the fields available. These steps would then correlate to the continued evolution of the security scanning within the CI/CD pipelines.

4.1. Baseline Comparison Timing

The captured timings in this research focus on the difference in the time it takes for a build and release pipeline to complete both before and after implementing application scanning extensions. The results of this testing introduced an additional 38 seconds to scan the application code and 36 seconds to publish those results to the SonarQube extension, for a total of 1 minute, 14 seconds.

There isn't a consistent and repeatable method of establishing data that depicts the time needed to configure the initial connection from an application scanning tool to the application being scanned using the traditional means of dynamic code analysis. The steps typically involved in that process include a security analyst obtaining the URL the web application resides, opening their application scanner of choice, creating a project, configuring the new project with the corresponding settings such as URL and finally, launching the scan. Within a CI/CD pipeline, the analyst would obtain the web application URL, open the CI/CD pipeline tool, add the extension that integrates the application scanner of choice with the application to be scanned and add the command line syntax to the extension.

Although there is no significant difference in time between the traditional application scanning configuration versus the initial CI/CD configuration, a notable amount of time will be saved from that point on. Since the execution of the application scanning extension will be performed each time the application is deployed, there are no future time requirements involved. On the contrary, notification of each release must include IT security, giving them time to analyze the new code if organizations continue to use the traditional method of analysis. Historically, IT security teams are left out of future releases to existing applications, which can present a significant risk to the organization if vulnerabilities are introduced to the code.

4.2. Baseline Vulnerability Comparison Counts

The testing captured in this research compared the use of the ZAP tool used in a traditional means of dynamic code analysis compared to the integration within a CI/CD pipeline. The testing captured the time taken to spider the application URLs, perform an

Author Name, email@address jon-michael.lacek@student.sans.edu

active scan on each URL, and finally, the number of vulnerabilities discovered within the applications. The following charts compare the baseline times to the integration of the security scanning tools to the release pipeline, while the second chart summarizes the number of vulnerabilities discovered using the stand-alone client versus the command line integration conducted within the release pipeline:

	Baseline timing	ZAP integration timing
Time to spider and scan the WebGoat application	2 minutes, 16 seconds	1 minute, 49 seconds
Time to spider and scan the WebWolf application	2 minutes, 26 seconds	1 minute, 28 seconds

Table 5. Application spider and scan comparison timings.

	Number of Vulnerabilities		Number of Vulnerable URLs	
	Stand-alone	Pipeline	Stand-alone	Pipeline
WebGoat	6	4	11	10
WebWolf	2	3	5	9

Table 6. Application vulnerability scan comparison timings.

Although the numbers are not identical, this testing confirms similar functionality exists between the client version of the ZAP tool on a stand-alone server, versus the dynamic integration using a temporary container in the CI/CD pipeline process. This step might be the most significant change to the interaction between the development and security teams. When security analysts utilize the client version of ZAP, the application scanning methodology and tests might not follow the same process each time due to a number of different variables. For example, one inconsistency could be the interpretation of the returned results by the security analyst performing and reviewing the scan. An important point to note is that organizations would need to adopt a mentality that the integrated application scanners in the CI/CD pipeline would be evaluating the most

common, well-known vulnerabilities and common coding mistakes. Understanding this would allow security analysts to dedicate more time to the in-depth functionality testing of the application while knowing that the baseline scan remains consistent during each scan.

4.3. Enhanced Vulnerability Scans

As organizations continue to mature in their DevOps initiatives, the capabilities within their pipelines continue to grow. Maturity comes in the form of unit testing that is built out to ensure functionality within each aspect of the code or in the logic of the gates between each stage in a pipeline. Incorporating this mentality from a security perspective requires a strong partnership with the development teams whose goal continues to be to deploy application and feature updates regularly. A good starting point would be the integration of the analysis tools into the pipelines without impacting the build and release functionality. This step would be to gain visibility to the vulnerabilities and poor coding practices that exist in the application. When development teams have had a chance to review the output of the scans and consult with security analysts to understand and correct the vulnerabilities, a decision should be made to implement a gate that would fail the progression to the next stage in a pipeline should a vulnerability be detected.

An additional benefit of integration within a CI/CD pipeline is for applications that are not released on a regular or frequent cadence. Security analysts could work with the developers to set a scheduled release, which would trigger the application scan extensions. This re-occurring schedule would not impact the functionality of the application since the source code has not changed. Many of the dynamic code analysis tools, including ZAP, are continually developing the tests that are executed within their scans to look for the up-to-date vulnerabilities discovered in the wild. For example, after the identification of the Heartbleed vulnerability, the ZAP community configured tests into the active scanner functionality to test each application it scans for its presence. If a pipeline is configured using security extensions and is set to release on a regular schedule, even if there have not been any changes to the code, notification would be sent to the developers if their application is vulnerable to the newly discovered vulnerability.

Author Name, email@address jon-michael.lacek@student.sans.edu

Another benefit of integrating security tools in the CI/CD pipeline is how the notification of vulnerabilities within an application can be configured. Many of the extensions have built-in functionality that integrates back into the tools that Agile teams use to manage their work. For example, in the Microsoft Azure DevOps toolsets, security analysts have the option of automatically creating work orders or bug items for each detected vulnerability and placing them in the developer's list of backlog tasks. Using this capability will help drive the adoption of integrating security into the DevOps culture. Developers will be far more likely to address the vulnerabilities within their code if they don't need to take manual steps to review and understand the discovered vulnerabilities. The extensions available provide the details about the vulnerability, which include resources on how to correct them.

5. Recommendations and Implications

Organizations responsible for delivering a secure product must integrate application scanning extensions into their DevOps pipelines. Like the DevOps process, this culture change is a slow continual process. Integrating static and dynamic code analysis tools can and should be implemented over time as the culture shifts within the organization.

5.1. Recommendations for Practice

DevOps pipelines have a wide variety of control mechanisms within them that control the progression of an application through the pipeline. The concept of gates, which are configured to control whether or not the next stage of the pipeline can start, should be utilized as a maturity mechanism. As organizations begin introducing the security tools into their release pipelines, they could decide to configure the extensions passively. Using this methodology, organizations can automate the analysis of the source code and the scanning of the application, all while still allowing the release of the product to the customers. Reports are generated as a result of the security extension integration, allowing development teams time to review the vulnerability reports. Developers and security analysts can then prioritize them accordingly within their backlog of work.

Author Name, email@address jon-michael.lacek@student.sans.edu

As teams mature and become comfortable with the use of these application scanning extensions, the release pipelines should be reconfigured to fail when an application release does not meet a predetermined set of criteria, or risk acceptance, as a result of the scan. For example, if you are a financial institution deploying an application responsible for sending and receiving funds electronically, you may decide that the results of the application scans should contain no known vulnerabilities of any severity. If there is a discovery of any vulnerability during the build and release process, the deployment should fail. However, if you are a marketing firm, you may be willing to accept the risk that informational, low, or medium severity vulnerabilities present in your application, while only failing the release upon the discovery of a critical or high vulnerability.

The development of security-related extensions within the most common DevOps pipeline tools continues to evolve. Automatic integration into an Agile process is a must if the DevOps culture will include security. Many of these extensions can be configured to automatically integrate with the Agile methodologies such as work orders, user stories, and bug fixes that get mapped out on a Kanban board. An example of this is the advanced integration of the OWASP ZAP extension. Pipeline administrators can configure the pipeline to automatically create individual work orders or bugs for each vulnerability detected within the developer's backlog as part of their code development. This automatic process becomes invaluable in terms of time savings while helping to provide input to the continual feedback loop that ensures the release of application features continue in a timely and secure manner.

5.2. Implications for Future Research

As security scanning tools begin making their way into the software development life-cycle, they will start to gain the attention of developers that may have previously discounted their importance. As teams mature and the use of these tools shift left in the SDLC, developers will begin to understand the best practices of secure coding through on-the-job training. This research was unable to capture the time saved for building an application that incorporates application scanning from the start, rather than at the end. Measurements could be taken to indicate the amount of time needed to correct a

Author Name, email@address jon-michael.lacek@student.sans.edu

vulnerability discovered in code and then multiply that out across the application for how many times the utilization of that coding style was employed. If the development of methods or functions occurs in an insecure manner, the developer would be made aware of that poor coding practice after checking their code into the code repository for the first time, rather than at the end of the applications development where that coding practice may have sprawled throughout numerous other methods or functions.

6. Conclusion

Changing the culture in any organization involves the commitment of multiple teams and doesn't happen overnight. With the tools available today and the desire to continually release applications or feature updates to existing applications at an increasingly rapid pace, the IT security community must help keep those applications and the data they process safe and secure. By leveraging the available CI/CD pipeline extensions, organizations can start integrating a security mindset in the DevOps process and put secure coding at the forefront in the minds of the developers. Achieving this must and can be accomplished without forcing developers to abandon their continuous integration continuous deployment tools they are accustomed to today.

References

- 2018 State of DevOps Report. (n.d.). Retrieved from <https://puppet.com/resources/whitepaper/state-of-devops-report>
- Azure Resources. (2019). Retrieved from <https://docs.microsoft.com/en-us/azure/devops/?view=azure-devops>
- Apache Maven Project. (2019). What is Maven?. Retrieved from <https://maven.apache.org/what-is-maven.html>
- Docker Hub. (n.d.). Retrieved from <https://hub.docker.com/>
- Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. (2017). *The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations*. Portland, OR: IT Revolution Press, LLC.
- MacDonald, N., Head, I. (2017) *10 Things to Get Right for Successful DevSecOps*. Retrieved from <https://www.gartner.com/document/3811369>
- OWASP WebGoat Project. (2019). Retrieved from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
- OWASP Zap Attack Proxy Project. (2019). Retrieved from https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
- Ponemon Institute – Bridging the Digital Transformation Divide: Leaders Must Balance Risk & Growth (2018) Retrieved from <https://www.ibm.com/downloads/cas/ON8MVMXW>
- SonarQube Resources. (2019). Retrieved from <https://sonarqube.org/features/integration/>
- Veracode State of Software Security Report: Focus on Industry Verticals. (2018). Retrieved from <https://info.veracode.com/analyst-report-devsecops-global-skill-survey.html>

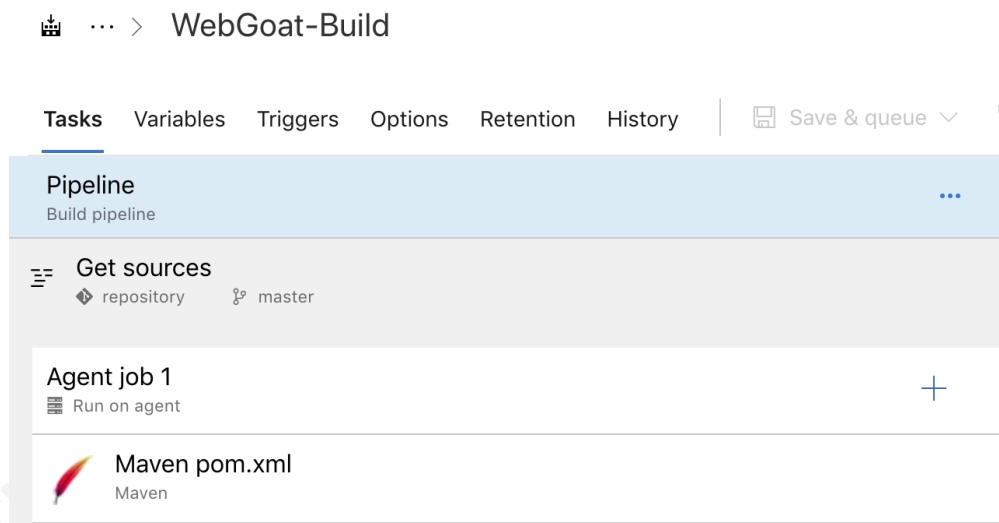
Verizon. (2019). Data Breach Investigations Report. Retrieved from Verizon website:

<https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf>

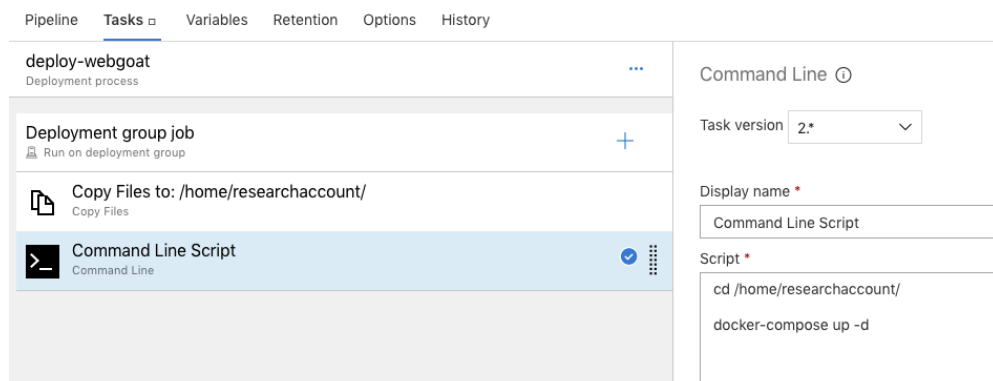
Author Name, email@address jon-michael.lacek@student.sans.edu

Appendix

1. Build pipeline policy configuration. The *Get Sources* step is configured to reference the WebGoat application files stored in a Git repository. The *agent* step is the integration needed to tell the pipeline where to build the code in the Microsoft Azure cloud. This phase connects to an Ubuntu virtual machine in the Microsoft Azure Portal. The final stage, *Maven pom.xml*, is the step that compiles the Java-based WebGoat application.

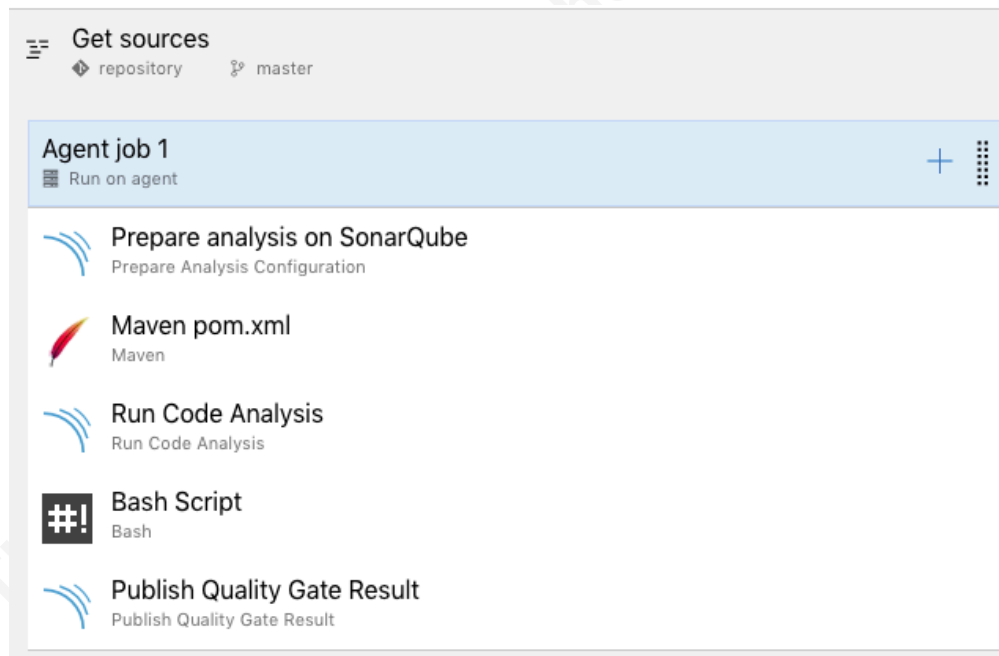


2. Release pipeline policy configuration. The *Deployment group job* step is configured to point to the previously constructed Ubuntu server. The *Copy Files to:* step copies the artifacts needed to start the application on the Ubuntu server, and the final phase *Command Line Script* initiates the command to start the application, which is *docker-compose up -d*.



Author Name, email@address jon-michael.lacek@student.sans.edu

3. SonarQube pipeline extension integration. The *Prepare analysis on SonarQube* step performs a quick check against the configured SonarQube server that is configured within this stage. The *Maven pom.xml* step compiles the application code followed by the *Run Code Analysis* step that performs the SonarQube static code analysis. The *Bash Script* is used to update the SonarQube project with the results of the scan, and the *Publish Quality Gate Result* is used to provide the Boolean condition back to the pipeline indicating whether or not the code quality met the acceptable threshold.



4. Commands used for ZAP scan via a Docker instance. The following three commands correspond to the three command line steps in the diagram below:

```
docker pull owasp/zap2docker-weekly
```


```
docker run -t owasp/zap2docker-weekly zap-full-scan.py -t
http://10.0.0.7:8080/WebGoat/
```


```
docker run -t owasp/zap2docker-weekly zap-full-scan.py -t
http://10.0.0.7:9090/WebWolf/
```


Pipeline **Tasks** ▾ Variables Retention Options History

Baseline ZAP Scan ...
Deployment process

Deployment group job +
Run on deployment group

 **Download latest ZAP Docker Release**
Command Line

 **WebGoat Scan**
Command Line

 **WebWolf Scan**
Command Line