

Abdollah Shajadi

**AUTOMATING SECURITY TESTS FOR WEB APPLICATIONS IN  
CONTINUOUS INTEGRATION AND DEPLOYMENT ENVIRON-  
MENT**

**AUTOMATING SECURITY TESTS FOR WEB APPLICATIONS IN  
CONTINUOUS INTEGRATION AND DEPLOYMENT ENVIRON-  
MENT**

Abdollah Shajadi  
Bachelor's Thesis  
Autumn 2018  
Information Technology  
Oulu University of Applied Sciences

## ABSTRACT

Oulu University of Applied Sciences  
Degree programme, Information Technology

---

Author: Abdollah Shajadi

Title of the bachelor's thesis: Automating Security Tests for Web Applications in Continuous Integration and Deployment Environment

Supervisor: Kari Laitinen

Term and year of completion: Autumn 2018, Number of pages: 61 + 3 appendices

---

This thesis was a research project commissioned by Liana Technologies for creating tools and processes to implement automated security tests for web applications.

Discovering and testing available tools and concepts to achieve the aims of this project was the starting point. Burp Suite Pro, the Python programming language and Gitlab CI/CD were the main technologies that helped with the progress.

The result of this research was a Python script called Skinner that automated security testing with Burp Suite Pro in the Gitlab CI pipeline. The procedure of implementing this technology and reaching the best practice of DevSecOps is the main ingredient of the developed solution in this thesis.

---

Keywords: Web Application, Security, Automation, DevSecOps, DevOps, Python

# CONTENTS

ABSTRACT	3
TABLE OF CONTENTS	4
VOCABULARY	5
1 INTRODUCTION	6
1.1 History of Agile in software development and production	7
1.2 Agile information security	9
1.3 Challenges of DevSecOps	9
2 SECURITY CONCEPTS	11
2.1 Shifting security to the left	12
2.2 DevOps Infrastructures	15
2.3 DevSecOps integrations	17
2.4 Log management, Metrics dashboard, monitoring system and alerting	21
2.5 Cultural modification	24
3 SECURITY TESTING TOOLS	28
3.1 Dynamic Application Security Testing (DAST)	28
3.2 Burp Suite Pro	30
4 DEVELOPMENT PROCESS OF SKINNER	32
4.1 Challenges of developing a DAST solution	32
4.2 Life cycle of Skinner	39
4.3 Reporting and Feedback Loop	47
5 FUTURE DEVELOPMENT POSSIBILITIES	51
5.1 Joining several tools to work together	51
5.2 Implementation of AI and Machine Learning	53
5.3 Proactive defence and repair	54
6 CONCLUSION	56
REFERENCES	58
APPENDICES	62

## VOCABULARY

<b>TERMS AND ABBREVIATIONS</b>	<b>EXPLANATION</b>
CI	Continuous Integration
CD	Continuous Deployment
DevSecOps	Development, Security, Operations
SDLC	System Development Life Cycle
OS	Operating System
DB	Database
SAST	Static Application Security Testing
DAST	Dynamic Application Security Testing
IAST	Interactive Application Security Testing
WAF	Web Application Firewall

# 1 INTRODUCTION

In today's practice of software development and production, speed plays an important role. Programmers create and use different methodologies in their development cycle to become more and more agile and it helps them to produce better quality code in less time.

The idea of being agile does not stop with programmers, system administrators also think about new methods of automating their tasks and handling their operations, in the Agile way.

A number of tools and methods is developed to get implemented in infrastructures for sysadmins to make the whole process of development and production faster, easier and more precise.

Nowadays, the rise of security concerns is witnessed in different aspects of digital life and even more in the software development and production.

Testing security problems of software has always been a process of manual scanning which requires its own checklists and long timeframe because shipment of the software products without security checks may cause an irreparable damage to the software, the company and its users. This makes the security testing procedures crucial before each release.

The need to introduce an agile process to security scans becomes more evident when agile development and deployment procedures are already in place in other departments.

The current way of security testing is the bottleneck in the product shipment lifecycle and it is only one of the reasons why it is necessary to automate part of the security scans in the software production pipeline.

During the writing of this thesis and while attempting to gather information on the topic, an acute shortage of relevant literature was discovered, nonetheless, all available resources were used to their extent and were supplemented with the experience gained from the full-time job at Liana Technologies as a security engineer in the R&D team.

## **1.1 History of Agile in software development and production**

The word “agile” is defined as “able to move quickly and easily” by Oxford dictionary (1) and it has been in use by software developers as Agile (with capital A) since 2001.

Seventeen software developers met to work on better development methods: Jeff Sutherland, Ken Schwaber, Jim Highsmith and Bob Martin can be named among the people, who published the “Manifesto for Agile Software Development.” (2)

However, project management methods called Iterative and incremental development methods can be traced back to 1957. (3)

The Agile software development manifesto has four main values:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

While there is an importance in the items on the right, the items on the left are valued more (2). According to these values, with right tools it is possible to achieve agility in the software development process. By its natural means, it is a culture and practice that can be implemented in an everyday work environment.

These four values seem simple, but the agile software development targets a complex product development with dynamic and non-linear characteristics that can be done with different methodologies and practices using a range of tools.

For instance, Scrum is one of the most famous Agile frameworks for managing work and tasks in the software development process.

At the 2008 Agile Toronto conference, the “DevOps” term introduced by Andrew Shafer and Patrick Debois in their talk on "Agile Infrastructure". The DevOps term has been steadily promoted from 2009 (4).

The word DevOps is constructed with “development” and “operations”, The main goal of the DevOps is to automate and monitor all steps of software production from integration, testing and releasing to deployment and infrastructure management.

Agile methods focus more extensively on development and DevOps techniques focus on deployment. The frequency of development increases in general. Therefore, it is possible to test and ship the code as fast as possible. Introducing Agile to an infrastructure is a big step towards reaching the level of high-speed Agile developments that are already in place. Thus, the code can be shipped as fast with the help of a DevOps toolchain.

The DevOps toolchain can be broken down into 7 categories as follows:

1. Development and reviewing of Code.
2. Continuous integration tools.
3. Continuous testing tools.
4. Staging and packaging tools to store and manage artifacts for releases.
5. Tools for automation of releases of the software.
6. Tools for configuration management of infrastructure.
7. Tools for monitoring the process and performance.

Many tools and companies can be found within the industry that provide open source and premium services or products for each of these categories. Building a seamless solution with all the above-mentioned tools is the main task for the DevOps engineers with the collaboration of programmers and the management team.



## **1.2 Agile information security**

The idea of being Agile with security has always been the case. However, having both the infrastructure and the culture ready for it, is the point that makes it hard to implement and achieve. Scanning and performing a security test are one of the most important parts of the software development that need to be addressed based on the needs of today's environment.

The classic way of conducting a security test of software is time-consuming, therefore it is against the Agile culture and it slows down the deployment process. Thus, designing a set of tools and processes to integrate a security test into the Agile development and infrastructure is the primary concern.

With this mindset, three parts are integrated: Development, Security and Operations and consequently the term "DevSecOps" is born. It is a practice that aims at integrating security into every aspect of an application life cycle from design to development, testing, production and ongoing operations. Thus, in new DevSecOps world, automation is the main focus.

## **1.3 Challenges of DevSecOps**

If one searches the Internet for the concept of "Security in DevOps", many different names and arrangements of DevSecOps appear. However, all different terminology defines one concept, and this shows that there is no specific standard available for DevSecOps even though there is an existing standard for the secure development life cycle called ISO/IEC 27034-1 (5). Because each environment is different and each company operates in its own way and set of tools, it is hard to agree with a standard. For this thesis the term "DevSecOps" is being used to define the process of implementing a security test in the DevOps infrastructure and procedures.

One of the biggest challenges of DevSecOps is keeping up with the speed of DevOps for that implementation of tools and practices are needed considering the following points:

- Implementing security as code.
- Tools for automating security scans.
- Tools for monitoring.
- Integrating the results and processes with development, operations and security teams.

All the above-mentioned practices help to implement security but not necessarily make it faster. The goal is finding low hanging fruits of bugs being produced in the early stages of development inside the continuous integration system. Therefore, security scan tools need to be configured to not to scan deeply and thoroughly. Also introducing pre-commit hooks to statically check the code can make the Security part of the DevSecOps process faster.

Maintaining a proper teamwork can become a challenge when we introduce different teams from multiple departments to each other. With adopting the security tests into DevOps, new teams that are not used to working together will collaborate more even though the final goal is having an automated process. But it needs collaboration between the security team, developers and the operation team to build up the culture throughout the company and complete the process.

The aftermath of the automated DevSecOps process is a huge amount of data. Managing and processing those logs and data and piping them through security engineers and developers to handle tasks based on them can be a challenge, too.

## 2 SECURITY CONCEPTS

The environment in practice already has a running DevOps infrastructure. A working Continuous Integration and Continuous Delivery (CI/CD) environment is prerequisite. In practice, Developers commit and push code to the version control system (Git) and then the CI/CD pipeline will be triggered.

Before introducing DevSecOps, companies were going through a time-consuming process of testing the application for security holes before releasing it to production. The process can be something like the following list:

1. Developer pushes the feature
2. A release version is being made
3. Software is ready to deploy to production
4. Security team holds the deployment for tests
5. Security report is submitted to the developer
6. Patch is being made, and a new release version is created
7. After the security team's approval, the new release deploys to production

A big gap between the development and deployment can be seen, security tests are completely separated from CI/CD and it causes the whole production process to halt and wait for the security team. Therefore, before trying to implement the security automation and DevSecOps idea, DevOps infrastructure should be dissected to find out which tools are being used and how they interact with each other and developers.



FIGURE 1. Far left and far right of security testing.

To integrate DevSecOps seamlessly in a company it is important to have a proper understanding of the tools programmers and engineers are using and how they use them. It will give us a better perspective of the general culture of the working environment.

## **2.1 Shifting security to the left**

DevSecOps is about shifting security to the left, as far as possible, running tests against the developed code as soon as they pushed the new code makes security everyone's problem. Having security tests starting on the far left of the development process helps to divide the attack surface between different sections of security tests and each sections of security problem will be taken care of separately.

This way security team changes from a team that tells people what to do and what not to do to a team that works hand in hand with the development team.

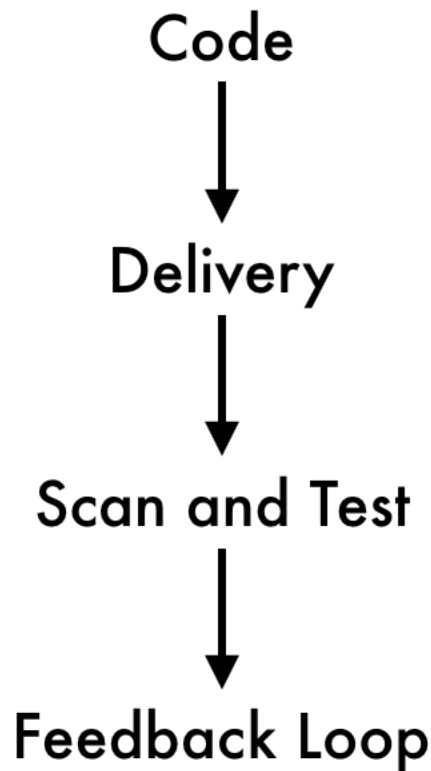


FIGURE 2. Top down process of development.

As shown in the Figure 1 on the far left, there is the static analysis, a pre-commit hook which runs when a developer tries to commit and push the code to the pipeline. It scans for different kinds of defects which can be found with the static analysis.

On the far right, there is penetration testing. It can be performed by a third-party penetration tester every year or every six months. It is a very expensive process but with DevSecOps in place, most of the application can be tested during the development. Penetration testing projects only apply to the production-ready and shipped code. It can be narrowed down to vague features which cannot be tested on pre-commit hooks and during the pipeline.

Most of the changes occur in the pipeline and during the development process. For instance, Implementation of tools to gather metrics and run automated static

and dynamic security tests against testing and staging environment on the pipeline or running a continuous bug bounty project helps to maintain a secure application throughout the year and not only when it is time to doing the penetration test.

The figure 2 shows the process of development in the pipeline which will be discussed in depth in the next chapter. After the code is pushed to the source code management, The CI/CD pipeline starts running and result of it will be a deployed staging version of the application. Static and dynamic scans can be performed in this segment. The results of those scans will be generated then the feedback loop triggers and the whole process starts again for the next pipeline run.

The feedback loop can be a new issue tracker ticket or a message on the internal chat or an alert on the monitoring system or even a meeting with the team. DevSecOps will adopt successfully when everyone starts engaging with the security tests. To reach the successful stage of integration, different kinds of tools should be work in the SDLC (System Development Life Cycle). These tools are being implemented and used by developers, operation engineers, security engineers, management and team leaders.

Tools can be categorized by environment, virtualizations and containers, Dashboard, log and metrics management, Alerting, Testing, source management, continuous integration and deployment system.

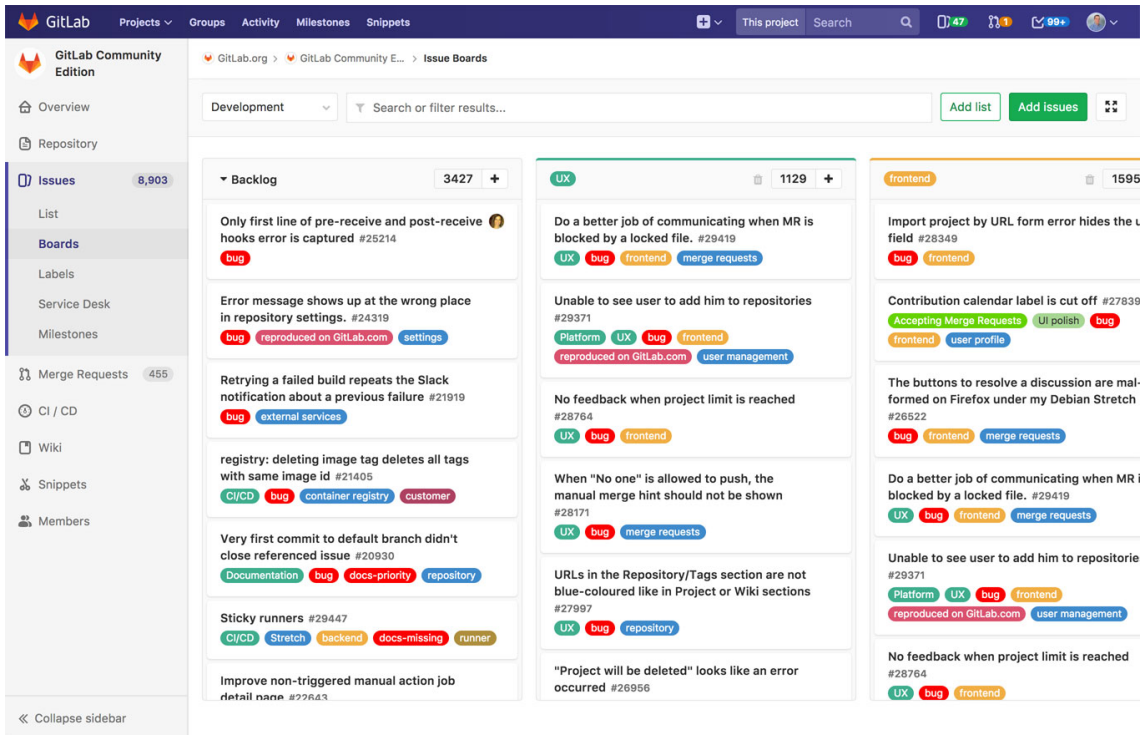


FIGURE 3. Issue Tracker's Scrum board view of Gitlab (6).

## 2.2 DevOps Infrastructures

Before adding a new module to a running system having an in-depth understanding of it is imperative, the goal is to add the security test tools to the already in place and running DevOps infrastructures and make them automated.

First and foremost, the implementation of the source code version control which goes hand in hand with the CI/CD solution has to be done. The enterprise edition of Gitlab is being used, it has its own CI/CD pipeline and Git version control with a nicely done web UI that supports Issue tracker as shown in the figure 3, it helps developers to browse through the project easily. (6)

Gitlab plays a key role in the DevSecOps implementation, it is a powerful hub for all of our tools to connect and work together. It centralizes the code, documentation and operation so managers, developers and security engineers can collaborate on tasks as productive as possible.

```
~ > platform

Usage: platform COMMAND

A tool for running projects through container environment programmable way

Options:
  -D, --debug           Enable debug mode
  -a, --data-path string Location of resource files (default "/Users/abdo/.platform")
  -d, --domain string   Domain address (default "localhost")
  -n, --network string  Network name (default "acme")

Commands:
  attach      Attach to a running container through shell
  create      Start shared platform services
  destroy     Shutdown shared platform services
  remove     Initialize single project or environment with collection of projects
  rm         Remove stopped project or environment
  run        Initialize single project or environment with collection of projects
  ssh        Wrapper to ssh* commandds
  ssl        Wrapper to openssl command
  start      Start suspended project or environment
  stop       Stop suspended project or environment

Run 'platform COMMAND --help' for more information on a command.
~ > _
```

FIGURE 4. Perl6-platform command line tool.

The next most important tool to implement is at the far left of the development process which is the developer's local environment. Perl6-Platform as shown in the figure 4, it is being used in the environment in practice, a tool developed at Liana Technologies that works with Docker containers and based on defined YAML configuration it creates, configures and runs complicated multi-container application conveniently and fast. (7)

The Platform is a tool for managing single projects or tightly coupled projects via the container environment. It is a command line tool for macOS and Unix based OS (at the writing of this document) that connects with the Docker API. With the help of its YAML based configuration file, the complicated application environment can automatically install, configure and start.

The Platform very fast became one of the "must have" tools in the development cycle. It helps developers to configure their needed tool automatically at the far



left of SDLC. Another famous tool for that purpose is Vagrant (8) from Hashicorp that does the same thing as the Perl6-platform and supports Linux, macOS and Windows.

One of the most important parts of being a DevOps engineer is to configure servers for each stage of deployment which is a very sensitive and time-consuming task. The importance of using an automation tool for that is significant. Puppet, an automated administrative engine for the Linux, Unix, and Windows is being used for DevOps automation. It performs administrative tasks (such as adding users, installing packages, and updating server configurations) based on a centralized specification (9).

Terraform is another tool from Hashicorp for building, changing, and versioning infrastructure safely and efficiently. Terraform hand in hand with other Hashicorp tool stack can manage existing and popular service providers as well as custom in-house solutions (10). It helps to define configuration as code even more throughout the whole system and it enables to automate DevOps tasks even more deeply, it gives security engineers the power to automate security test in infrastructure, too.

With the usage of these automation tools deploying new fully configured and fully automated servers is possible. The metrics agents and tools are pre-installed, and they can be used in AWS as well as private cloud and servers.

Now that the base of the DevOps infrastructure is being discussed, it is time to go through tools that concerns a log management and alerting and dashboards.

### **2.3 DevSecOps integrations**

It has been discussed about the challenges on DevSecOps in the section 1.3, cultural changes are the most important part of the DevSecOps integration. How they proceed with security problems in the team and company-wide, as well as the role of the security team and developers when they engage with each other.

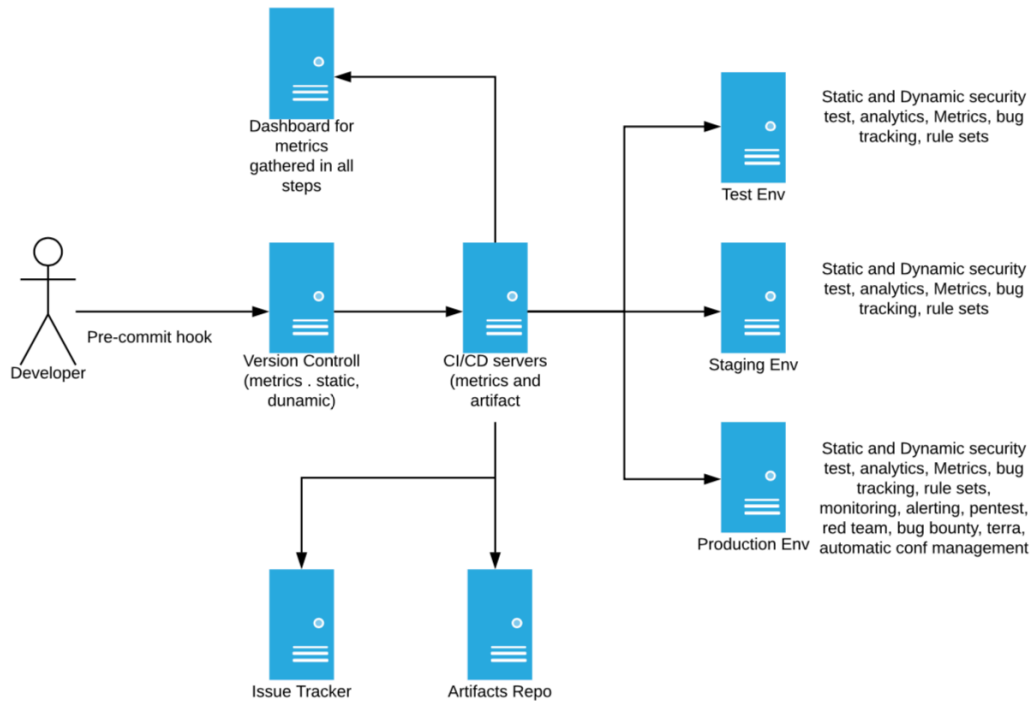


FIGURE 5. DevSecOps integration to DevOps infrastructure. (24)

This will be analyzed, and a solution will be found for these kinds of cultural issues in the current chapter, section 2.5.

The next important part of DevSecOps integration is designing a practical cycle with a security procedure inside each section, the way that if a manager slices out each section of the development cycle from far left (developers) to far right (shipped code), they will be able to see that the security test are running and the metrics are generating and being sent to monitoring dashboard.

Having a fully organized and automated process like that needs hours of planning and thinking and it can be drastically different in each environment and company.

Figure 5 is an example of a minimum recommended development process with security integrated (DevSecOps cycle). As shown in the diagram there is the version control server and CI/CD servers which both are managed by Gitlab. There is an Artifact repository which is a database for all the artifacts generated via

testing tools in the CI/CD pipeline. These artifacts can be e.g. log files, screenshots, binary files and DB dumps.

The Issue tracker that is managed by Gitlab also functions as a feedback loop for security findings between developers and security team. All the security issues that needs to be fixed by developers are created as a new ticket in the project's issue tracker, developers can assign the tickets to themselves and start working on a fix right away.

The far right of the development process, there are three kinds of servers for deployments. The latest version of application lives there for different purposes, testing environment, staging environment and production environment. A set of logic tests and functional tests and also security tests will run against each of these environments during those stages of development.

Based on the defined DevSecOps procedure, automating security starts from before the developer pushed their code to the version control and it continues to fetch metrics and artifact on all the stages in between and in the end from the production deployment, the issues are reported back to the developers with the issue tracker.

The feedback loop gets completed with metrics and dashboard. Developers, the security team and the management can see what is going on in the system. The next section, log management and how the dashboard works and what tools are being used will be discussed in detail.

Security tests throughout the development cycle from the very beginning until after shipment of the code and deployment can be in different forms. Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST) and Interactive Application Security Testing (IAST) are the ones that are implemented in the procedure. As it can be seen from the figure 5, at least one or more of these methods can be used at each step of development.

The first step on the far left is when the developer writes the code and wants to commit and push it to Gitlab. Static code analyzer (SAST) is there as a pre-commit hook that alarms the developer about issues with the code even before it leaves its machine.

On the next steps the version control and CI/CD pipeline are presented, metrics and artifacts are gathered and sent to artifact DB and dashboard at all of these steps. When CI/CD pipeline is running it runs SAST and DAST based on the instructions against one of the environments, mostly Testing and Staging environments.

On the far right of the development, interactive application security testing (IAST) program that runs against the staging/production machines as a penetration testing project or an on-going bug bounty program. Finally, to complete the cycle automatically or manually (based on the process) a new issue on the issue tracker is created to produce the feedback loop in place.

Gathering metrics at every step of the DevSecOps integrated system is what has been focused on. To process and understand all of those metrics a set of tools need to be installed which is the topic of the next section, it helps to understand what exactly happens in this complicated and fast paced procedure.

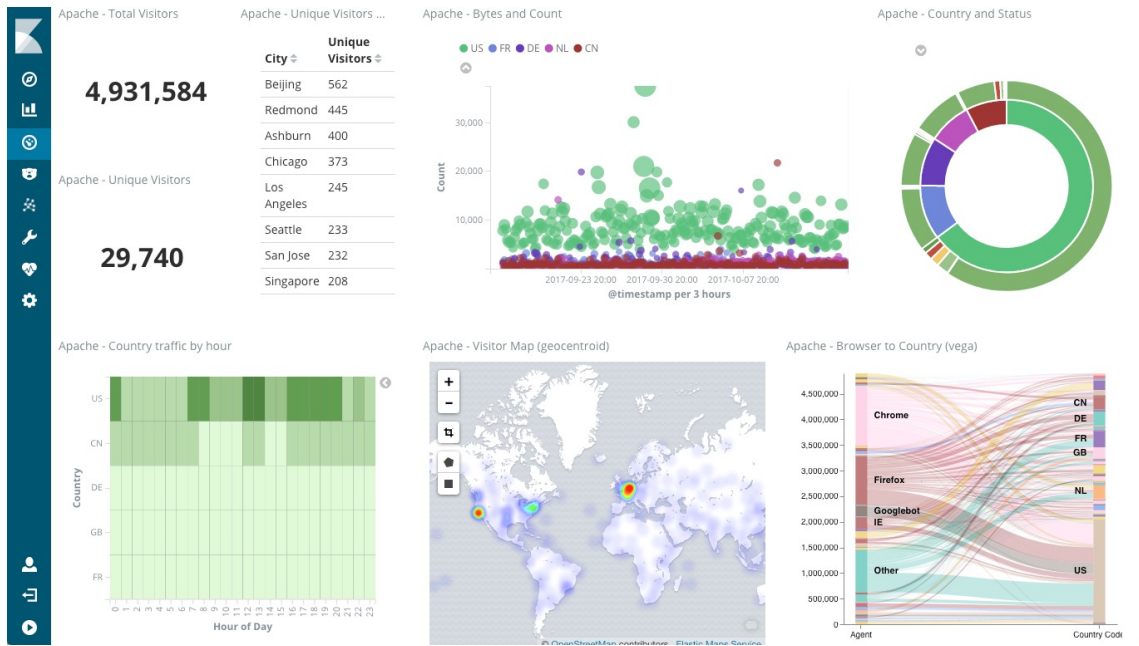


FIGURE 6. An example of Kibana dashboard with graphs and charts (11).

## 2.4 Log management, Metrics dashboard, monitoring system and alerting

Considering the discussed DevSecOps solution as a whole, a machine, which works as one body with the duty of making and shipping software, has many different, complicated internal organs. It should contain and present a working security tests and respected metrics at each part of it. The system generates an enormous amount of log files and metrics on each run of the DevSecOps pipeline as described in section 2.3.

These metrics need to be gathered, processed, filtered, indexed and then presented in a nice and understandable manner on a web-based dashboard so that security, development and management teams can see and use this data fast and accurate, according to their need and decide based on them.

The solution which has been used for this process is the ELK stack which is a stack of software made from Elasticsearch, Logstash and Kibana. In this section, the use case of ELK stack and how to implement it in the environment will be

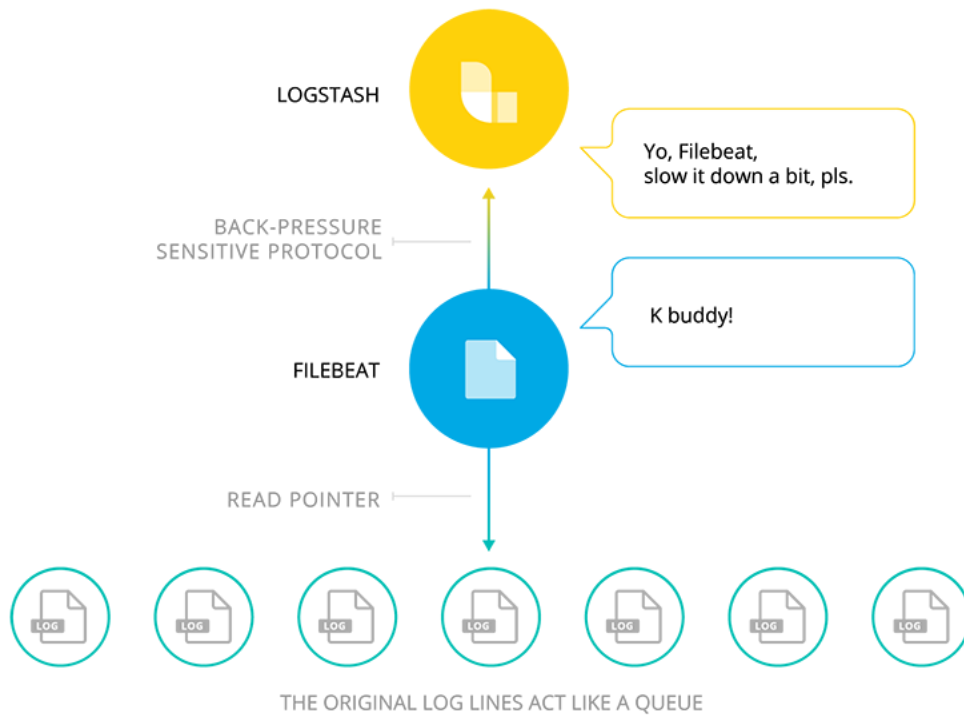


FIGURE 7. Filebeat workflow (11).

discussed. The ELK stack or as they call it “The Elastic Stack” is built on an open source foundation. It helps to reliably and securely fetch logs from any source, in any format and search, analyze, and visualize them in real time (11).

Kibana gives an analytical shape to the data and is the extensible user interface for configuring and managing all aspects of the Elastic Stack. It has a nicely built web GUI that helps to build and define charts and graphs also to browse logs which are being indexed by Elasticsearch.

Elasticsearch is a distributed, JSON-based search and analytics engine designed for horizontal scalability, maximum reliability, and ease of management. It indexes all the data which being filtered and processed by Logstash and made visible by Kibana.

Logstash is a dynamic data collection pipeline with an extensible plugin ecosystem and a strong Elasticsearch synergy. With Logstash it is possible to gather the

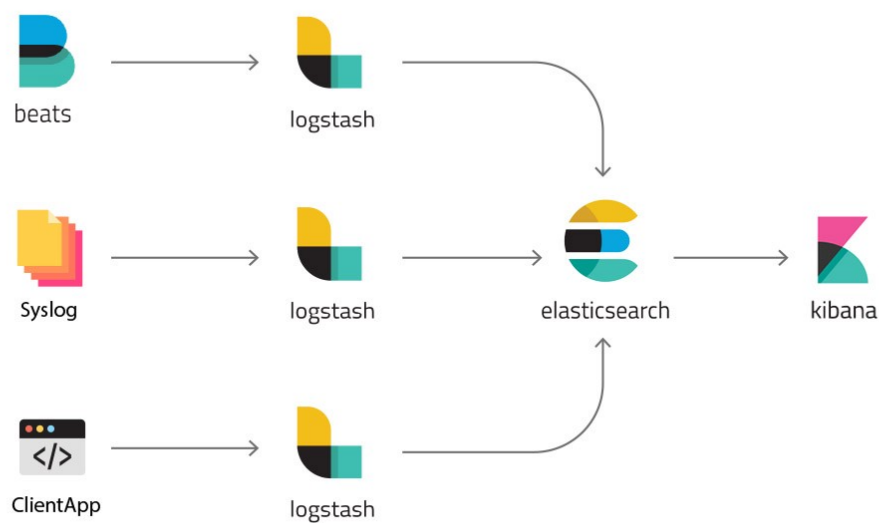


FIGURE 8. Elastic stack workflow. (11)

metrics and logs from a different part of the system and application via multiple methods, e.g. Beats, then filter and format the logs and send it to Elasticsearch

for indexing. Logstash support numerous amounts of filter plugin to aggregate and format any kind of logs. The most famous plugin called Grok that parse arbitrary log texts and structure it with the help of its internal patterns and widely used regular expressions.

Beats is a platform of lightweight shippers that sends data from edge machines to Logstash and Elasticsearch, the Beats are Logstash's agents. There are several different Beats with different specialties e.g. Metricbeat, Auditbeat, Packetbeat, Heartbeat and Filebeat.

The Filebeat is being used in the environment in practice because the application generates different sets of log files in different formats all in one folder and the best beat for shipping and keeping track of all those log files is Filebeat.

As you can see in figure 7 on page 22, Filebeat keeps track of log files and new lines of logs in each of them and ships them to Logstash. Because of differences

on the process speed between Logstash and Filebeat, one of them can become slower or faster for that reason Filebeat supports “Back-pressure sensitive protocol” to take care of caching and the speed of pushing the logs to Logstash and not losing any line in the process.

After finishing the configuration of the ELK stack, the journey of a logs starts from the application writing the event on the designated log file that Filebeat points at, as shown in figure 8 on page 23. It is possible to submit logs to Logstash via tools other than Beats but in this case Filebeat is the proper choice. It can pick up the logs and send them to Logstash then it filters and process them, the result will be processed and filtered logs in JSON.

Logstash submits the processed logs to Elasticsearch which indexes for search. It is possible to search through indexed logs in Kibana, but it has a lot more to offer than just presenting raw logs on the browser, as shown in figure 6 on page 21, it is possible to make useful charts and graphs from the processed and searchable data and put them together on a dashboard.

One of the most important parts of DevSecOps is making sense of all the technologies which works together in the process, a good dashboard can enable everyone to actually use all the generated data and act based on it toward a better workflow.

## **2.5 Cultural modification**

As mentioned in earlier chapters, the employees working habits or better say “culture”, throughout the development cycle and even the daily normal human interaction plays an important role in security. Therefore, cultural modification, integration of new habits and how everyone deal with security issues is the most important part of the DevSecOps.

First step is shifting security to the left as far as possible which was the topic of section 2.1. It means programmers are the first line of security testing, making programmers feel responsible about security as a part of their daily routine and



not a set of rules that are enforced by security team is a big step toward a better DevSecOps integration.

On the other hand, the security team needs to understand how developers work and what they need, they need to stop generating bulky outdated reports full of false positives and commanding everyone about security parameters.

The DevSecOps idea is about being Agile, a considerable amount of cultural modification is toward security team itself. Developers and operation teams already know how to work in an Agile way, it is security team that needs to change their habits and start learning more about development, operation and production.

Working in a fast-paced environment together with operation and development teams will teach security team about how they should proceed with their tasks. It makes them a friendly entity that is there to help developers to secure the software in a productive manner with their own tools, not dictate what they should do and what they should not.

The main task as a security team during implementation of DevSecOps is to make "Security" everyone's problem, when they reach to the point that every person dealing with software feel that they are first hand responsible for the security of what they are doing they reached the DevSecOps nirvana, for that it is imperative to give security more transparency.

Before the DevSecOps security teams was scan the software and hold the production and give the other teams reports of the security problems weeks after development of that part of the code, then the development team tries to fix the issues and get the security approval, after that the code can get ready to ship.

With this approach, none of the teams has a big picture of security performance of the application on each step of it teams are blind to each other's progress and it makes the procedure slow and prone to error.

When security becomes everyone's problem and companies implement it the way that it works on each iteration of development with the speed of the operation team the security team process the scan data and show it in a user-friendly dashboard as described in the section 2.4, then every department can see the value of what they are doing and they can increase their knowledge base and prioritize their task list based on it.

With security become transparent to teams, it enables them to actually care about security all the time, to the point that everyone become a member of the security team and the people in security team become freer to take care of more important issues which cannot be achieved with automatic scanning.

The DevSecOps and automated security scanning does not eliminate the need of security teams, penetration testers and bug bounty projects, it just narrows down their job to the more important parts of the application.

Automated security tools can cover a good amount of security problems that populates most of the security reports. The DevSecOps integration finds and reports all of those issues on the time of development so the application is more secure right after first push to the version control software.

There is always some problem that an automated test cannot find, it needs the human interaction (IAST). The DevSecOps makes security teams free to focus on them and become more efficient on their job, it also makes the third-party penetration testing projects become less expensive.

All the security concepts needed to know for a successful DevSecOps implementation is covered in this chapter, the pre-requisites that is needed from DevOps team for CI/CD and issue tracking to Log Management and dashboards. How The DevSecOps can be implemented in the environment, explained the behavior of culture and how it should change toward a better DevSecOps experience.

Next chapter is dedicated to the candidate DAST tools and will describe the tool which has been used to develop the DAST part of the DevSecOps implementation.

## 3 SECURITY TESTING TOOLS

During a full run of the pipeline with the DevSecOps proposal in figure 5 on the page 18, it scans the application in different ways, static testing (SAST), dynamic testing (DAST) and interactive testing (IAST). Sometimes it does all of them at one stage of the pipeline. These different ways of testing have been defined in the previous chapter and discussed about how they work, the most problematic and of them to implement is the DAST.

In this chapter, it is about the tools that are available to security teams to use as an automatic dynamic testing in pipeline. One of them will be picked to develop the solution based on it in the next chapter.

### 3.1 Dynamic Application Security Testing (DAST)

Dynamic is about black box testing of the whole or parts of the application with scanners and to perform that the application needs to be up and running.

The dynamic scan tools do not have access to the source code and only can browse and scan applications the way a user experiences it, either the backend or frontend, admin pages or public access pages, there is a set of open source and premium tools with different capabilities ready to use.

All the tools in talk for dynamic testing of the web applications either crawl the whole application or accept URLs to scan, then sends the HTTP based attacks to the application and logs the results, they can perform known types of web attacks e.g. SQL injection and XSS.

There is a range of different tools to choose, from full-fledged web application scanners with all the numerous features to small specialized tools like fuzzers. From the big list of scanners, three of them will be discussed. Burp Suite Pro (12), OWAST ZAP (13) and W3af (14). All of these products provide dynamic security testing and each of them has their own pros and cons.

W3af is an open source command-line based application written in Python, it is modular and supports its own scripting system to execute tasks and automate them.

There is graphical user interface available for this tool but the fact that it natively runs on Linux terminal makes it very easy to use for implementing it as an automated tool in the CI/CD pipeline. it makes this tool a good choice for the goals.

An example of W3af audit script is in Appendix 1 (15), the audit script can run with the following command:

```
$ ./w3af_console --s MyScript.w3af
```

Next tool in the list is OWASP ZAP (Zed Attack Proxy), one of the most famous, opensource web application testing by OWASP maintained by hundreds of volunteers written in Java with an easy to use GUI.

The pros of OWASP ZAP is that it has plugins for famous CI/CD systems like Jenkins that makes is very easy to implement Automated scanning with it, in the current case which is Gitlab CI/CD it can use its API which provides natively by the application.

The last application in the list is Burp Suite Pro by Portswigger. Burp Suite Pro is the de facto industrial security testing tool for web applications with a big list of plugins and features, Portswigger provides a free version but considering the number of features that paid version enables the Burp Suite Pro will be chosen one.

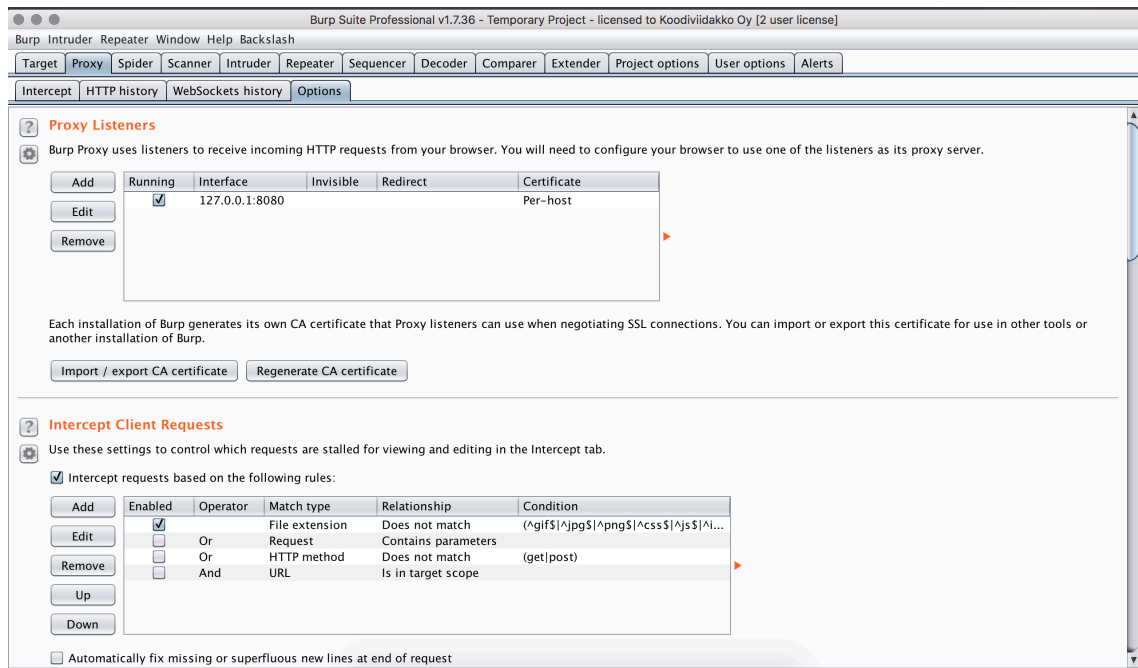


FIGURE 9. Options page of Proxy feature of Burp Suite Pro.

### 3.2 Burp Suite Pro

Burp Suite is a graphical security test tool that works as a proxy and intercepts all the traffic from user to the web application and tests them. It is written in Java and supports two versions, free edition and professional edition. Because the free edition does not support most of the useful features, the professional edition will be used. It supports functionalities like the proxy server, scanner and intruder. The advanced options are the spider, repeater, decoder, comparer, extender and a sequencer.

What will be used mostly is the scanner for the submitted URL during the pipeline runs via API, whole process will be discussed in the next chapter.

To use Burp Suite Pro on your local machine, the browser's proxy setting needs to be changed according to the parameters indicating at "Proxy" section in the "Options" menu of the application, as shown in figure 9.

It is possible to browse the pages under test. The requests and responses between the user and web server are showing in the “Proxy” menu and the “Intercept” tab which gives the testers the ability to directly manipulate, accept or deny them.

Burp Suite Pro logs all the target’s information for instance the content of requests and responses and the found issues about them in the “Target” page, it is possible to start other features of Burp Suite Pro against those targets e.g. Scanner, Intruder.

In this chapter, some of the testing tools which can be used for DAST solution has been discussed and Burp Suite Pro got chosen then its features and how it works on its basic usage scenario shortly explained.

## **4 DEVELOPMENT PROCESS OF SKINNER**

The Skinner is a proof of concept script written in Python to show how a DAST solution can work in an Agile manner. It works in Gitlab CI/CD pipeline, in other word, it runs directly from inside Gitlab CI configuration file (gitlab-ci.yml) that holds all the instructions and what is going to happen in the pipeline.

Burp Suite Pro scans need to be start automatically inside Gitlab CI/CD pipeline against the latest running version of the web application which is the version that recently pushed either to Master branch or other branches. Because of the resource limitation, these kinds of scans can become time-consuming, it is better to limit them to the master branch.

In this chapter, first in section 4.1, all the challenges during the development of the Skinner and the DAST solution and the development process will be discussed.

In section 4.2 the Skinner will be tested and how it works and how to use it in production will be discussed. Section 4.3, Agile feedback loop which comes with its monitoring and reporting systems is the topic.

### **4.1 Challenges of developing a DAST solution**

Developing a solution that needs to work automatically inside and within other solutions naturally makes it imperative to provide an API, preferably a REST API. Current version of Burp Suite Pro does not support any kind of REST API. It can be run on headless mode which makes it possible to develop an API plugin and run it in terminal, the first challenge is to run a version of Burp Suite Pro in terminal that supports REST API.



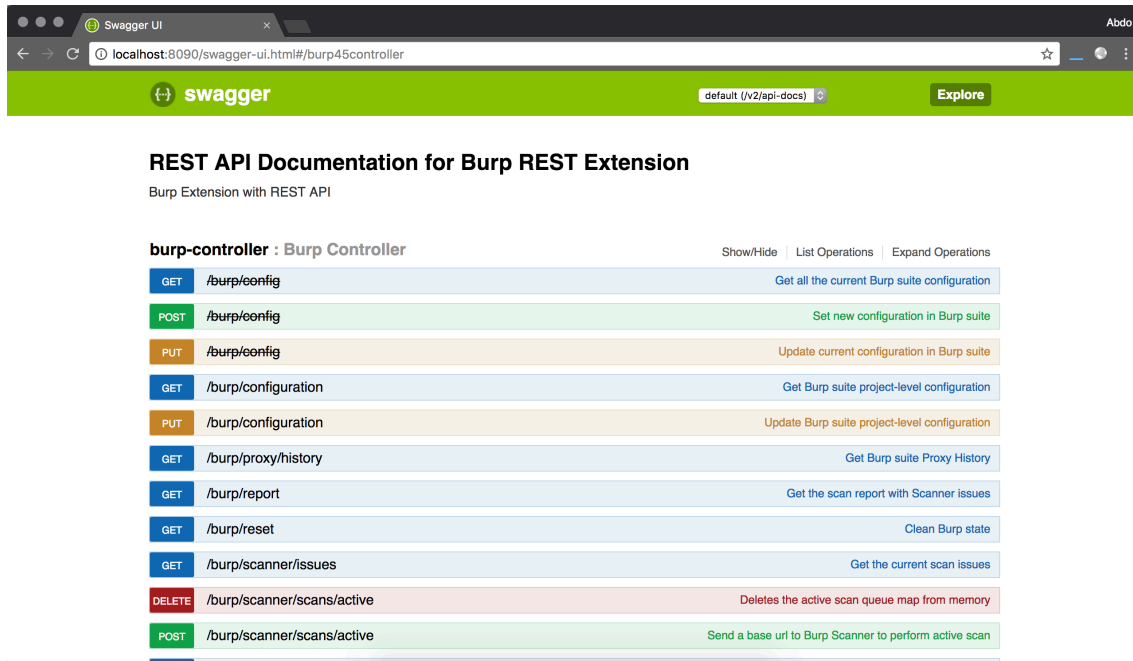


FIGURE 10. Burp Suite Pro API Swagger documentation page.

Open source third-party REST API add-ons are available, after testing several of them for instance an extension called Carbonator (16) and an API add-on developed by VMware (17), VMware solution has been chosen to be incorporated to the solution because of simplicity and fewer resource overhead.

Burp Rest API by VMware is a tool written in Java that needs to be compiled, it takes the binary file of Burp Suite Pro and after compilation generates a new Burp Suite Pro binary that supports API, the API documentation that built with Swagger (18), Burp Suite Pro API starts with the following command:

```
$ java -jar burp-rest-api-1.0.0.jar
```

As presented in figure 10, it is possible to browse API documents with the following link:

<http://localhost:8090/swagger-ui.html#/>

It is worth mentioning that at the time of writing this thesis Portswigger does not have any plan to implement internal API for Burp Suite Pro or include support for

CI/CD integration. Instead, they are working on a separate product that automates security scans on CI/CD pipelines (19). Based on a discussion on their support portal at the moment (August 2018) there is no release date specified for that product yet.

The second challenge comes to light when the Burp Suite API is running and tries to scan the application, this is a DAST solution and Burp Suite Pro scans the application as it experienced by the user (Black Box), for that reason the application needs to be deployed first.

```
43  deploy:
44    stage: deploy
45    before_script:
46      - 'which ssh-agent || ( apt-get update -y && apt-get install openssh-client -y )'
47      - eval "$(ssh-agent -s)"
48      - mkdir -p ~/.ssh
49      - echo "$DEPLOY_KEY" | tr -d '\r' | ssh-add - > /dev/null
50      - '[[ -f /.dockerenv ]] && echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config'
51    script:
52      - ssh abdollah@deploy.intra "cd /home/abdollah/environment/auth/auth && git checkout . && git checkout master && git pull"
53    only:
54      - master
55
```

FIGURE 11. Deploy configuration of Gitlab CI.

Continuous Delivery (CD) part of the pipeline is to deploy the application, there are special features ready to use in Gitlab CI/CD, with adding the configuration presented in figure 11 to the “gitlab-ci.yml” file, it is possible to deploy the last changes during pipeline. More details about the configuration will be discussed in the next section. The application is running, configured and accessible via the following URL:

<http://deploy.intra>

Current state is the “Deploy” stage of the pipeline, in the minimum pipeline configuration, unit test stage comes before deploy stage and security test stage comes after, to connect to deployment server, server's SSH key need to be added to Gitlab Variables as \$DEPLOY\_KEY so pipeline runner can connect and login with it.

This happens under before\_script, after that the connection to the deployment server is initiated then it executes the checkout to master branch and pulls latest

changes under script directive on line 51. Deploy stage is happening only if the pipeline is running for master on lines 53 and 54.

After applying the changes, the deployment server is ready and the continuous deployment (CD) is configured, now accessing to the latest pushed changes while the pipeline is running is possible. It is possible for the Burp Suite Pro API to scan the application, this leads to the next challenge that is browsing the application.

Burp Suite Pro is working as a proxy that indexes the URLs that were browsed then it queues them for the scan, it needs to perform tasks in the application like logging in, filling the forms and changing to pages.

The DAST solution needs to be fully automated, it needs a headless browser that accepts browsing instructions. There are several products for this purpose, the most famous of them is Selenium.

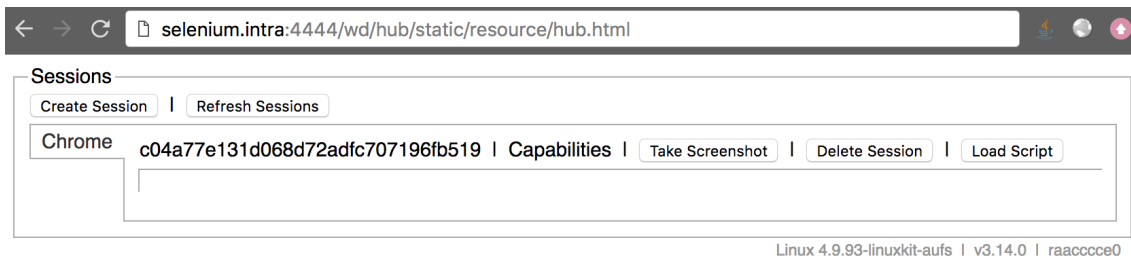


FIGURE 12. Selenium Web GUI.

To define Selenium in one sentence: “Selenium automates browsers. That's it!”, as its website asserts (20). The Selenium server can accept different types of drivers as a browser engine, the Chrome driver is being used in this case.

The application is written in PHP, for writing the Selenium instruction file in PHP Facebook/WebDriver (21) is needed which by the time of writing this thesis is the best Selenium WebDriver for PHP.

To install Selenium, Docker container has been used, because of its usability and speed. A new server on the same network as the Burp Suite Pro API installation and Deployment server initiated for Selenium called selenium.intra. It is possible to install Chrome version of Selenium with the following command:

```
$ docker run -d -p 4444:4444 selenium/standalone-chrome
```

This command exposes port 4444 of the container to the port 4444 so it is possible to access the Web GUI of the Selenium as presented in figure 12 via the following URL:

<http://selenium.intra:4444/wd/hub/static/resource/hub.html>

In the real-world situation while the DAST solution is in production this URL will never be used, everything happens automatically but during development, it is helpful to use it for debugging purposes.

In the next section, the configuration and usage of Selenium and other tools together will be explained in detail, now that the main DAST toolchain is ready it is possible to test a security scan.

```
@cwe-89
Scenario: The application should not contain SQL injection vulnerabilities
  And the SQL-Injection policy is enabled
  And the attack strength is set to High
  And the alert threshold is set to Low
  When the scanner is run
  And the following false positives are removed
    |url           |parameter           |cweId   |wascId   |
  And the XML report is written to the file build/zap/sql_injection.xml
  Then no Medium or higher risk vulnerabilities should be present
```

FIGURE 13. BDD-Security test case (22).

The questions that arises is that how long it will take and what will be scan and what will happen if this running scan holds up the pipeline for a long time and makes the whole process very slow, this is the fourth and the most important challenge.

Time and resource play a vital role in every system, the optimization of the processes should be done very well, so it does not undermine the Agility of the designed system. The focus should be on different parts of the system that makes it bottleneck of the process. This can break down into three questions.

1. How security test is written?
2. What scans are done?
3. When the scans are done?

By answering these questions, it is possible to optimize and minimize the usage of time and resource by the processes. There are two different approaches to answer the first question, first way to do it is BDD (Behavior Driven Development), there are similar solution with this approach like BDD-Security by Continuum-security (22) and Mittn by F-secure (23). To work with these applications the programmer needs to write security tests in BDD as shown in figure 13, like they write unit tests, they need to think of different possibilities and write tests for it which the result will be pass or fail.

This approach narrows down what the scanner will be going to scan to the parts of the application which recently changed and the type of vulnerabilities, it has a dramatic impact to the scanning time and usage of the resources.

The downside of this strategy is that programmers have to write test cases for every part of the application which needs to be tested, it makes them to think of different attack vectors. Next approach is that what the developed python script (Skinner) does, running scanner only for newly developed pages.

This way with Burp Suite Pro spider being disabled it only put browsed URLs that are being instructed by developer to the scanning list and the security scan engine starts to test all the ways to intrude the application.

```

5 variables:
6   POSTGRES_HOST: postgres
7   POSTGRES_DB:
8   POSTGRES_USER: runner
9   POSTGRES_PASSWORD:
10  LDAP_ORGANISATION: Koodiviidakko Oy
11  LDAP_DOMAIN: viidakko.fi
12  LDAP_ADMIN_PASSWORD:
13
14 stages:
15   - unittest
16   - deploy
17   - securitytest
18
19 image: debian:jessie
20
21 test:app:
22   stage: unittest
23   before_script:
24     - apt-get update
25     - apt-get -y install curl wget ldap-utils php5-curl php5-ldap php5 php5-common python3-pip
26     - echo "deb http://packages.viidakko.fi/ jessie main" > /etc/apt/sources.list.d/liana.list
27     - wget --quiet -O - http:// | apt-key add -
28     - apt-get update
29     - apt-get -y install liana php5-pgsql postgresql-client openssl liblianaauth-installer-php
30     - pip3 install requests mattermostdriver selenium
31     - perl -pi -e 's/short_open_tag = Off/short_open_tag = On/g' /etc/php5/cli/php.ini
32     - wget http://deploy.intra:8000/proxyplugin.zip
33   script:
34     - export PGPASSWORD=$POSTGRES_PASSWORD
35     - psql -h "postgres" -U "$POSTGRES_USER" -d "$POSTGRES_DB" -c "SELECT 'OK' AS status;"
36     - openssl genrsa -out Data/server-key.key 4096
37     - openssl rsa -in Data/server-key.key -pubout > Data/server-key.crt
38     - wget http://deploy.intra:8000/skinner-0.0.1.tar.gz
39     - tar xvf skinner-0.0.1.tar.gz
40     - python3 ./skinner-0.0.1/skinner/main.py -b http://burp.intra -id $CI_PROJECT_ID
41     - conforms -fu
42
43 deploy:
44   stage: deploy
45   before_script:
46     - 'which ssh-agent || ( apt-get update -y && apt-get install openssh-client -y )'
47     - eval $(ssh-agent -s)
48     - mkdir -p ~/.ssh
49     - echo "$DEPLOY_KEY" | tr -d '\r' | ssh-add - > /dev/null
50     - '[[ -f ~/.dockerenv ]] && echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config'
51   script:
52     - ssh abdollah@deploy.intra "cd /home/abdollah/environment/auth/auth && git checkout . && git checkout master && git pull"
53   only:
54     - master
55
56 security:
57   stage: securitytest
58   before_script:
59     - apt-get update
60     - apt-get -y install curl wget python3-pip
61     - pip3 install requests mattermostdriver selenium
62   script:
63     - wget http://deploy.intra:8000/skinner-0.0.1.tar.gz
64     - tar xvf skinner-0.0.1.tar.gz
65     - cd skinner-0.0.1
66     - python3 ./skinner/main.py -b http://burp.intra -u http://deploy.intra -s selenium.intra -id $CI_PROJECT_ID
67   only:
68     - master

```

FIGURE 14. gitlab-ci.yml example.

It will make scan time longer but in the other hand what programmers have to do is much easier and that improves the integration of DevSecOps. This solution is user-friendly, and developers can start using it faster. Also those pages are the target pages of the application being scanned, this answers the second question (What scans are done?).

Programmers instruct in Selenium config file only the parts of the application they want to get queued for scan. This way it reduces time on writing test case and also scanning time. The detail of how to write a Selenium instruction file is in section 4.2.

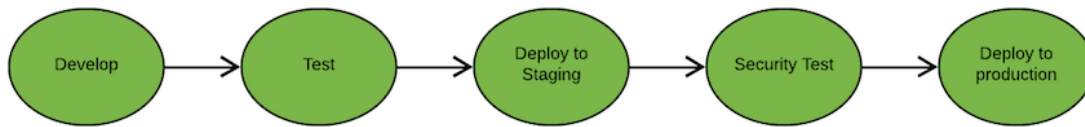


FIGURE 15. Development to Production timeline.

The final question to answer is: When the scans are done? the scan triggers and starts during the pipeline to answer this question first the question of “what happens in pipeline?” need to be answered, that is the topic of next section.

In the minimum configuration possible, there is three stages, unittest, deploy and securitytest. The position of security test stage is crucial for the process, so security test needs to be the last thing that happens in pipeline right before production release.

## 4.2 Life cycle of Skinner

In section 4.1, the challenges and solutions of DAST has been discussed and the introduction of toolchain that will enable security team to reach the automated dynamic security tests like Burp Suite Pro API as the security scanner that runs on the following URL:

<http://burp.intra>

Next is the deployment server that fetches latest changes during pipeline runs which is accessible via the following URL:

<http://deploy.intra>

Last but not least, Selenium server with Chrome driver which automates browser tasks that is accessible with the following URL:

<http://selenium.intra>





```

1  <?php
2  namespace Facebook\WebDriver;
3  use Facebook\WebDriver\Remote\DesiredCapabilities;
4  use Facebook\WebDriver\Remote\RemoteWebDriver;
5  use Facebook\WebDriver\Remote\RemoteKeyboard;
6  use Facebook\WebDriver\Remote\WebDriverCapabilityType;
7  use Facebook\WebDriver\Chrome\ChromeOptions;
8  require_once('vendor/autoload.php');
9
10 $host = 'http://selenium.intra:4444/wd/hub';
11
12 $pluginForProxyLogin = 'proxyplugin.zip';
13
14 $options = new ChromeOptions();
15 $options->addExtensions([$pluginForProxyLogin]);
16 $capabilities = DesiredCapabilities::chrome();
17 $capabilities->setCapability(ChromeOptions::CAPABILITY, $options);
18 $driver = RemoteWebDriver::create($host, $capabilities, 5000);
19
20 //Login to auth following page will redirect to auth page
21 $driver->get('http://deploy.intra');
22 $targetUsername = '***@***.com';
23 $targetPassword = '***';
24 sleep(5);
25 $element = $driver->findElement(WebDriverBy::name('login'));
26 $driver->getKeyboard()->sendKeys($targetUsername);
27 $driver->getKeyboard()->pressKey(WebDriverKeys::ENTER);
28 sleep(5);
29 $element = $driver->findElement(WebDriverBy::name('password'));
30 $driver->getKeyboard()->sendKeys($targetPassword);
31 $driver->getKeyboard()->pressKey(WebDriverKeys::ENTER);
32 sleep(5);
33
34
35 // Browse pages that need to be scanned
36 $driver->get('http://deploy.intra/installations');
37 sleep(5);
38 $driver->get('http://deploy.intra/accounts/create-email-list');
39 sleep(5);
40 $driver->get('http://deploy.intra/projects/add');
41
42 // Destroy Selenium session at the end
43 $driver->quit();
44

```

FIGURE 17. Selenium instruction PHP file.

Figure 15 on page 39, illustrates the big picture of stages of the pipeline, after developer pushed the changes to Gitlab, CI/CD pipeline triggers and starts running, the full example of “gitlab-ci.yml”, it is available on figure 14 on page 38.

First stage that will run is unittest which executes all the functional tests written by the developer after passing of this stage developers can continue their work on the next commits, they will be notified about security issues found by the DAST solution later via chatOps or issue tracker.

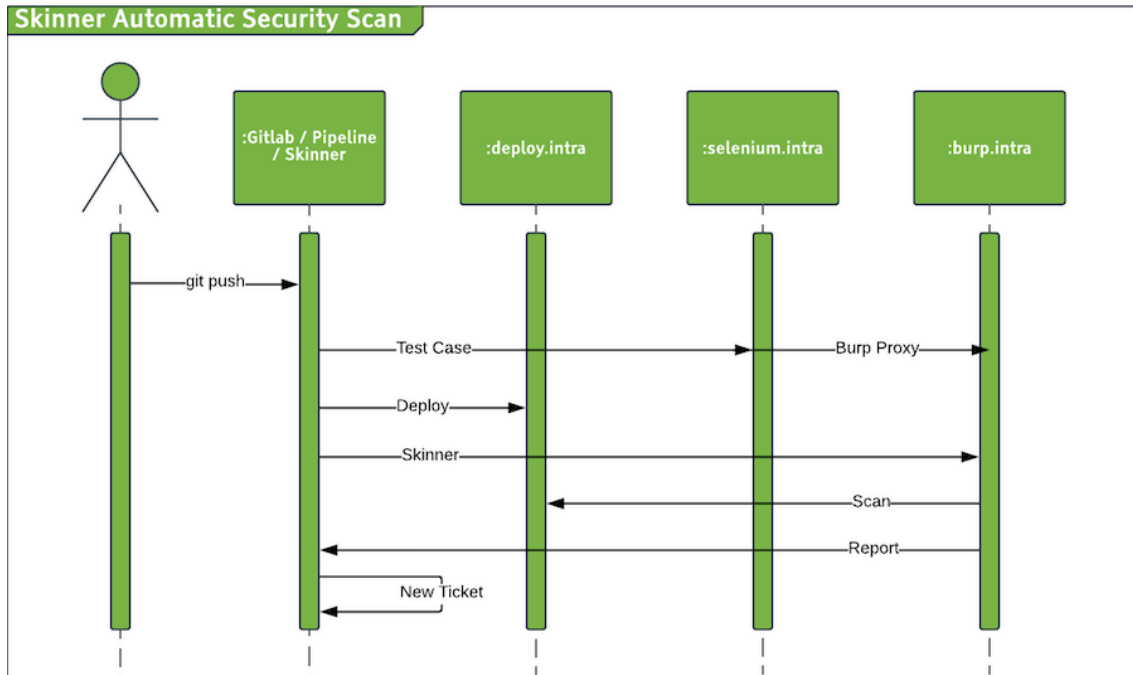
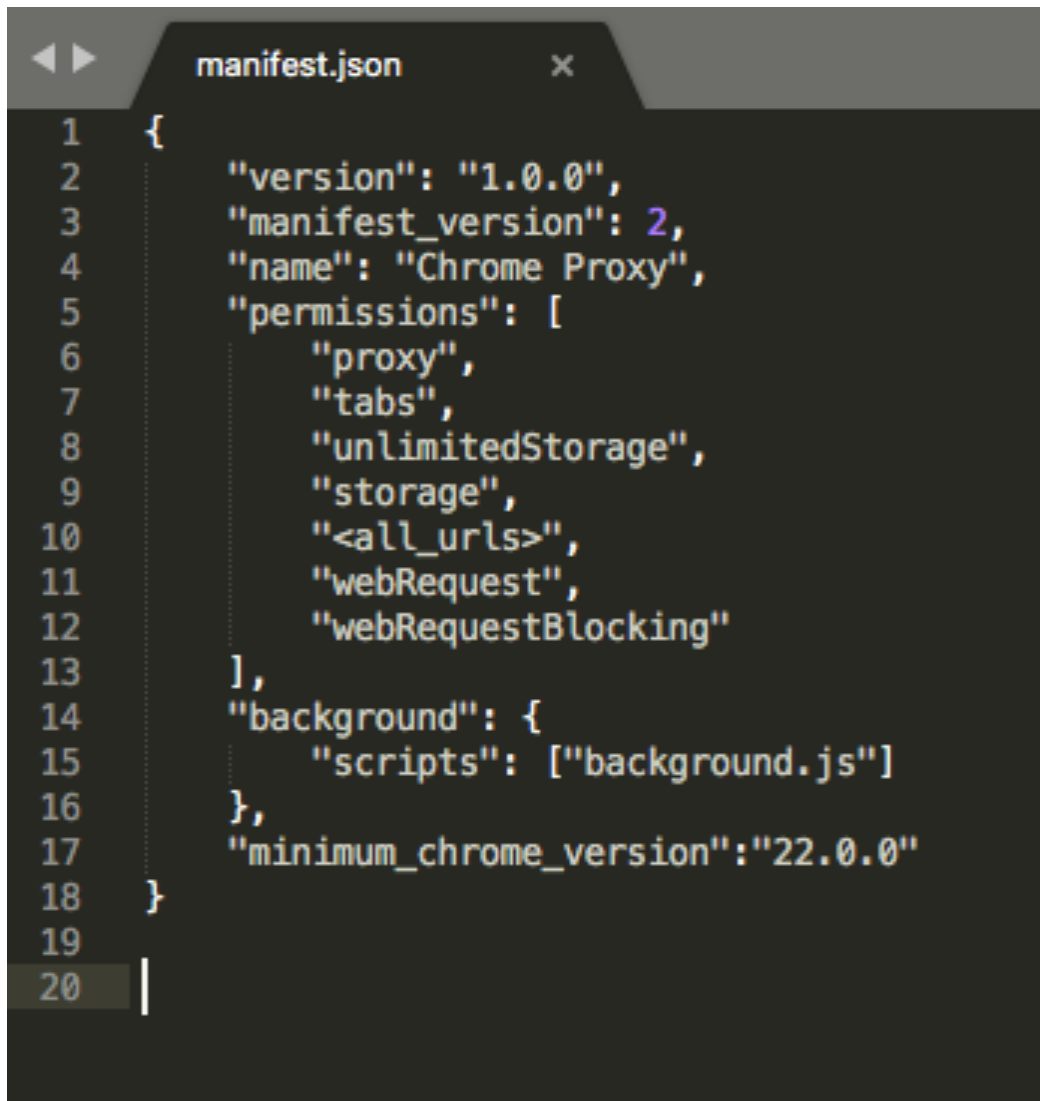


FIGURE 18. The life cycle of Skinner during a pipeline run.

Next is the deployment stage which makes the latest change of the application available on the deployment server after that pipeline moves to the securitytest stage that runs the Skinner. Finally, if the new version is ready to deploy to production it will go for the procedure.

As it can be seen from figure 14 on page 38, securitytest stage is the last stage of the pipeline which starts on line 56, like other stages there is before\_script, script, and limitation of branch that the stage executes for which is the master branch.

On before\_script it installs needed libraries for the running Skinner that consists of curl, wget, Python, and with Python package manager (pip3), it then installs Requests package and MatterMost driver (for ChatOps) that will be explained in section 4.3, Also Selenium library for python which enable Skinner to be able to connect to Selenium.



```
1  {
2      "version": "1.0.0",
3      "manifest_version": 2,
4      "name": "Chrome Proxy",
5      "permissions": [
6          "proxy",
7          "tabs",
8          "unlimitedStorage",
9          "storage",
10         "<all_urls>",
11         "webRequest",
12         "webRequestBlocking"
13     ],
14     "background": {
15         "scripts": ["background.js"]
16     },
17     "minimum_chrome_version": "22.0.0"
18 }
19
20
```

FIGURE 19. The Manifest.json file, Chrome proxy plugin.

Under the script directive the Skinner is being prepared to run on lines 63 to 65, it runs with the following command:

```
$ python3 ./skinner/main.py -b http://burp.intra -u http://deploy.intra -sU selenium.intra -id $CI_PROJECT_ID
```

The options in the command are as follow:

1. -b: Defines the Burp Suite Pro API server address.
2. -u: Defines the deployment server.
3. -sU: Defines the Selenium server.
4. -id: Defines CI project id that being created by Gitlab CI.

```

1
2
3 var config = {
4     mode: "fixed_servers",
5     rules: {
6         singleProxy: {
7             scheme: "http",
8             host: "burp.intra",
9             port: parseInt(8080)
10        },
11        bypassList: ["foobar.com"]
12    }
13 };
14
15 chrome.proxy.settings.set({value: config, scope: "regular"}, function() {});
16
17 function callbackFn(details) {
18     return {
19         authCredentials: {
20             username: "",
21             password: ""
22         }
23     };
24 }
25
26 chrome.webRequest.onAuthRequired.addListener(
27     callbackFn,
28     {urls: ["<all_urls>"]},
29     ['blocking']
30 );
31

```

FIGURE 20. The proxy.js file, Chrome proxy plugin.

Gitlab CI configuration file is ready, next file to write is the Selenium instruction in PHP which named security\_test.t.php. as shown in figure 17 on page 41.

It first defines the Facebook/WebDriver namespace and import its functions. Then comes the Selenium server host to connect with at line 10. Before start browsing the application, it needs to configure the headless automated Selenium Chrome proxy to get behind Burp Suite Pro API and send traffic through it. A custom made Chrome plugin will be used, it is the proxyplugin.zip, on line 12 that contains two files, proxy.js and manifest.json as a Chrome plugin requires, it can be found in the content of these file in figure 19 and figure 20.

From lines 14 to 18 the proxy plugin is being enabled and the Chrome driver session initiated, now it can perform the browsing tasks, the web application in this case, is restricted and needs to login to access internal pages so the first task is to log in and go through authentication procedure.

From lines 21 to 32, pre-configured username and password is defined and with special Facebook WebDriver selectors it can be accessed to login and password elements and with `sendKeys` and `pressKey` it performs the login task.

By now the login process is queued in Burp Suite Pro to scan, now it needs to set the URLs of the application to scan as follows:

```
$driver->get('http://deploy.intra/installations');
```

From lines 36 to 40, it describes 3 URLs that can be changed on different cases as the programmer desires, Selenium will browse these pages and it gets queued to Burp Suite Pro scan list that will be scanned right after. On the last line, 43, it closes the session which helps the Selenium server to stay light and clean for the next pipeline run.

The major goal on DevSecOps and the DAST solution is being Agile in security. In a practical language with the programmer's point of view it means, the programmer develops code, commits and pushes changes to Gitlab and it runs the CI/CD pipeline, after it finishes, the developer receives notification and new tickets according to the found security issues. This goal will be reached with the help of Skinner script.

Figure 18 on page 42, represents everything that happens after the programmer pushes code to Gitlab. Right after Gitlab receives new commits it runs the pipeline according to the definition on the Gitlab CI YAML file. First thing the CI runner does is executing the `unittest` stage that is a set of test cases including the Selenium security test that proxies the URLs to Burp Suite Pro and adds them to the scanning queue.



Burp Suite Pro starts the scan of the queued URLs against the updated application on the deployment server and in the end, it sends back report data via REST API to Skinner. During this time Skinner keeps track of the scan and at the end of the scan after it received the report as shown on figure 21 it prints all the security issues on Gitlab CI runner's console.

In the end, Skinner creates new tickets with full details on Gitlab Issue tracker via Gitlab's official REST API and sends a full PDF version of the report to Matter-Most channel. Skinner is a part of DevSecOps that plays the role of DAST. It has been designed in a way that it takes a minimum amount of energy and time from developers and tries to be as efficient and fast as possible during its process in pipeline.

Skinner is an open source tool that I developed at Liana Technologies which can be accessed via the following link on GitHub:

<https://github.com/LianaTech/skinner>

### **4.3 Reporting and Feedback Loop**

The developed solution is sending a full report of the scan from Burp Suite Pro API to the running Skinner in Gitlab CI pipeline, next it needs to get the most out of it, processing these data and metrics and reports are one of the most important parts of project.

The problem with old fashion penetration testing reports is that security team gives a PDF or a printed version of hundreds of pages of a technical report about security issues of the project. Going through that report for developers is a frustrating process and it ends up with having a tremendous number of false positives and unrelated issues.

**Cleartext submission of password**

URL: [http://\[redacted\]](#)  
 Severity: High  
 Confidence: Certain

**Issue Background**

Some applications transmit passwords over unencrypted connections, making them vulnerable to interception. To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.

Vulnerabilities that result in the disclosure of users' passwords can result in compromises that are extremely difficult to investigate due to obscured audit trails. Even if the application itself only handles non-sensitive information, exposing passwords puts users who have re-used their password elsewhere at risk.

**Issue Detail**

The page contains a form with the following action URL, which is submitted over clear-text HTTP:

- [http://\[redacted\]](#)

The form contains the following password field:

- password

**Remediation**

Todo: Add todo

Assignee: No assignee - assign yourself

Milestone: None

Time tracking: No estimate or time spent

Due date: No due date

Labels: security-test

Weight: None

Confidentiality: Not confidential

Lock issue: Unlocked

FIGURE 22. New ticket in Gitlab issue tracker by Skinner.

There is a “big data” problem, on each run of the pipeline it sends a big number of issues from Burp Suite Pro to Skinner to process and most of these issues are false positive and repetitive.

To handle this, a false positive database comes handy, to save all the found issues and tag them as a legit issue or false positive, every time Skinner receives an issue it compares it to false positive database and then generates report and creates new ticket based on it.

After a proper filtering and categorization an issue list is being made, it is time to generate metrics and send them to ELK stack for monitoring and further decisions. In figure 22, a screenshot of created ticket in Gitlab Issue Tracker via Skinner can be seen and the sent message in the chat system (MatterMost) in figure 23 is presented.



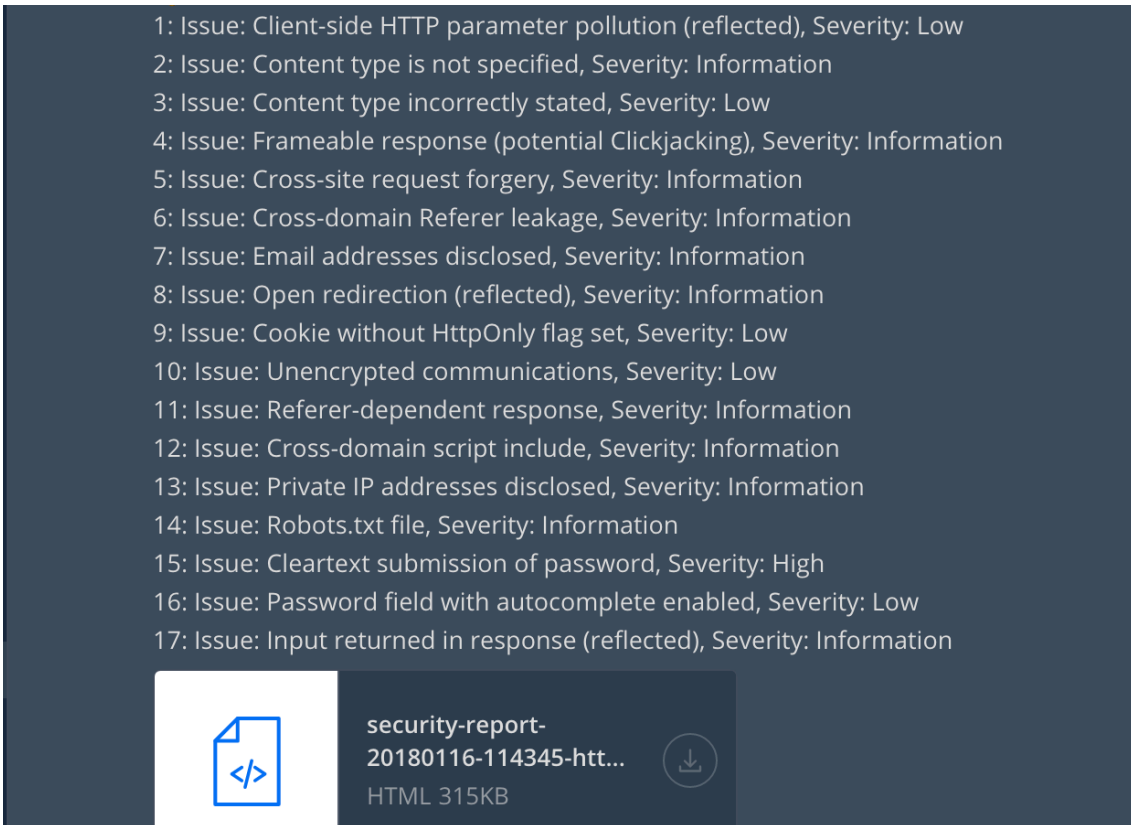


FIGURE 23. New message containing list of issues in Mattermost by Skinner.

ChatOps is a part of DevSecOps that makes connecting and messaging between people and system to people efficient. feedback loop is one of the most important concepts of DevSecOps that is going to complete the process. Everything that has been done in the pipeline and with the help of cultural modification everything will show off and start making sense in feedback loop process. It is where people connect, present and meet to see results, solve problems and decide about next steps of the development.

The tools that helps to have a productive feedback loop process like Gitlab Issue Tracker, an automatically created new ticket for critical problems via Skinner or the MatterMost Channel comes handy when the goal is sending the list of found issues.

With all these tools powered by analytical processing of ELK stack everything that needs to make security, management and development team connected together in a productive way is in use.

In this chapter, how a DAST process can work in pipeline, its relationship with Skinner script and how it is implemented in the Gitlab CI pipeline has been discussed. The most important challenges that was on the way to achieve a proper DAST solution explained and also, the way it is possible to use generated data as a bi-product of everything happened in pipeline, mostly Burp Suite Pro report.

This chapter presented what has been done for DAST part of the DevSecOps implementation in Liana Technologies infrastructure, of course, it can be developed and thought through in many different ways. Skinner is a proof of concept of a dynamic security test inside DevOps infrastructure was the result.

There are many ideas to complete this project and implement it in production for daily usage of development life cycle which is the topic of the next chapter. How it is possible to make it more Agile, smarter and better, how it can help to eliminate external intruders and help developers to patch bugs as fast as possible.

## 5 FUTURE DEVELOPMENT POSSIBILITIES

Every system can be improved and made better. The important part of the development of an IT solution is to prioritize and decide what will need to be done first and what take the most, by now a DAST solution as part of DevSecOps has been developed.

Development and improvement of the Skinner is always the case but the whole system can work better, in this chapter the ideas of future development of the DevSecOps system is the topic. As one may heard this famous phrase: “hackers are always one step ahead of us”.

The ultimate goal is to change that phrase, with making security test Agile and with the use of all the new methodologies and technologies to try to get close to a performance of securing and defending the system that it may never be achieved before.

### 5.1 Joining several tools to work together

As described in the last chapter Skinner’s role is acting as a glue to connect and orchestrate different tools to achieve one goal, receiving security scan results from Burp Suite Pro, the tools that Skinner handles are Burp Suite Pro, Selenium and Gitlab Issue Tracker.

All of these works together to obtain scan reports and create new tickets for critical security issues but in practice with the security point of view it is just Burp Suite Pro scanner against the web application that does the heavy lifting of discovering security problems.

One of the main ideas for future development is adding more security tools for more sophisticated scans to the system like fuzzers, analyzers like security protocol analyzers and OS scanners.

All of these new tools can run during the pipeline, it is about being Agile in DevSecOps, so speed should not be sacrificed for more tools, automated security scanning is to find low hanging fruits and problems that can be found automatically.

Deep security testing for sophisticated problems is done by the security team, penetration testers and bug bounty hunters. When the time comes with proper testing, the decision needs to be made about either to run the new tool within Skinner during pipeline or collect the report from Skinner and continue testing separately after the pipeline finished running then update the monitoring tools with new reports and metrics.

The first tool to consider is about fuzzing, fuzz testing helps to discover various types of programming errors in software e.g. Buffer Overflow, the fuzzing tools are called fuzzers, they provide the software with numerous unexpected and invalid and random inputs, it looks for crashes and how software react by those inputs.

There are many fuzzers available, one of the candidates is Radamsa (25) the famous fuzzer by OUSPG (Oulu University Secure Programming Group) (26), many other tools like Mittn (23) are implemented it as a part of their solution, as developers of Radamsa defined it:

“Radamsa is a test case generator for robustness testing, a.k.a. a fuzzer. It is typically used to test how well a program can withstand malformed and potentially malicious inputs.

It works by reading sample files of valid data and generating interesting different outputs from them. The main selling points of Radamsa are that it has already found a slew of bugs in programs that actually matter, it is easily scriptable and easy to get up and running.”

Next important tool to consider is OSquery by Facebook (27), as its developers defined it:

“OSquery is an operating system instrumentation framework for OS X/macOS, Windows, and Linux.

The tools make low-level operating system analytics and monitoring both performant and intuitive. There are many additional continuous build jobs that perform dynamic and static analysis, test the package build process, rebuild dependencies from source, assure deterministic build on macOS and Linux, fuzz test the virtual tables, and build on several other platforms not included above. Code safety, testing rigor, data integrity, and a friendly development community is the primary goals.”

OSquery can be implemented to the environment to gather metrics about how the backend is performing.

## **5.2 Implementation of AI and Machine Learning**

Technology is getting more and more advanced in every aspect, every day. AI (Artificial Intelligence) and machine learning are in those advancement areas, these technologies recently had big impact on security industry too.

As read on the news about hackers using AI and machine learning to gather information and attack businesses and financial organizations (28), the only way to respond to these attacks is with the power AI and machine learning.

AI can be utilized in many ways, for instance, at the gateway of external application's interface by a WAF and a firewall, discovering the type of attacks and fetch metrics and trigger alarms at the pipeline which helps with categorization of false positive findings and the management of metrics.

In the current project the focus is to make the final report smarter, as different tools are being combined, each of them on each run generates tremendous amount of data to process mostly in the form of logs, metrics, artifacts and reports. These data need to be sent to its designated place, e.g. databases, dashboard or file servers for further processing.

Elastic search stack which already explained in section 2.4, and how it helps to make sense of data, the Elastic stack is a powerful tool and the best at what it is doing but it can even be better with additional machine learning toolset. HELK (A Hunting ELK) (29) is the Elastic stack with advanced analytic capabilities that enables machine learning (ML) within ELK stack.

HELK is an example of using tools like Spark, Hadoop and Kafka within ELK stack to make ELK to not only alert the teams with each event but also predicts what is going to happen next and decide which alert is critical and may damage the system.

### **5.3 Proactive defense and repair**

What is the end goal of DevSecOps with the power of AI and Machine learning? What need to be achieved is the production application works stable and new development code ship without glitches. This takes enormous amount of effort from all teams by working together, with the help and agility of DevSecOps and technologies that makes it happen it is possible to get close to this end goal.

The story of receiving attacks from intruders and blocking the traffic by admins, finding the problem, committing the fix by programmers so it ships, and production code get patched is an old story of software companies that occurs all the time every day in different shapes and forms. In these cases, the incident happens and manually reported to the developer team to work on a fix.

The idea is filling this gap between inbound traffic that makes a disturbance to the internal DevSecOps process and pipeline with the help of Machine learning, analytical power and software automation.

With the new developments the intruder that is either an adversary or a bug bounty hunter starts testing and attacking the production server from internet.

The AI-powered gateway WAF starts finding out the type of security issue and which part of the code it concerns then passes the information to the dashboard and internal teams to alarm and process further.

Next the AI-powered dashboard tries to find out if it is a false alarm or actually the intruder may end up breaking the system then in case of real attack it alarms the team which is responsible for that part of the code and creates a new ticket. After code fixed and committed to the Gitlab, the DevSecOps pipeline checks if the issue fixed and finally deploys the changes to the production server.

The code can be patched even before the attacker successfully hack the system and in their point of view the code may never had a problem and it all seems like false positive to them.

This is the end goal of DevSecOps and how it is possible to achieve a security nirvana in development and production process. This goal is plausible with the tools and procedures that explained in this thesis.

## 6 CONCLUSION

During this research, different scenarios and tools was tested to reach the aim and as the result, Skinner was developed and described in chapter 4, it connects all the parts of a dynamic security scan of the running application.

In chapter 1, agile development and how it impacts the way of software development and lifestyle of a software company has been explained. How it is being used by programmers, how operations department adopted it for their infrastructure management and how security teams can make use of it for making software and processes more secure.

Security concepts like DevSecOps was the topic in chapter 2, how it is possible to make use of the already running DevOps infrastructure to deploy and automate security tools and which type of tools can help security teams to make it possible, most importantly the cultural aspect of DevSecOps has been discussed and how without improving it even with the best tools in place the outcome can become a failed try.

All the introductions and concepts paved the road for the main tool of this thesis (Skinner), chapter 3 is about explaining the security tools can be used, How Burp Suite Pro works explained in detail and how it can help with the goal which leads to chapter 4, in this chapter the development of DAST solution was the topic and Skinner, how it runs in Gitlab CI/CD pipeline and how programmers, security team and management team can benefit from it.

The most important element of DevSecOps is human, all the tools which introduced help to do the job faster, more efficient and more precise, next important element to implement for a DevSecOps solution is the environment. Each company and each team have different habit, culture and infrastructure, there is no plug and play formula for DevSecOps that works everywhere, each environment needs to be understood separately for its own DevSecOps solution.



Throughout this thesis, a road map has been drawn and a practical example of a case in Liana Technologies has been discussed, the Skinner solution is just at the beginning of its challenging journey.

## REFERENCES

1. Oxford Online dictionary, Cited 10.05.2018

<https://en.oxforddictionaries.com/definition/agile>

2. Manifesto for Agile Software Development, Cited 10.05.2018

<http://agilemanifesto.org/>

3. Larman, Craig; Basili, Victor R. (June 2003), Cited 10.05.2018

<https://pdfs.semanticscholar.org/f9b3/ca89c69bacfade039c8be40762c6857bda11.pdf>

4. Debois, Patrickm, Cited 10.05.2018

<http://www.jedi.be/presentations/agile-infrastructure-agile-2008.pdf>

5. ISO/IEC 27034-1:2011, Cited 10.05.2018

<https://www.iso.org/standard/44378.html>

6. Gitlab enterprise source controll and CI/CD, Cited 12.05.2018

<https://about.gitlab.com/>

7. Perl6-Platform development environment management, Cited 12.05.2018

<https://gitlab.com/tavu/perl6-platform>

8. Vagrant environment management, Cited 12.05.2018

<https://www.vagrantup.com/>

9. Puppet automation system, Cited 14.05.2018

<https://github.com/puppetlabs/puppet>

10. Terraform infrastructure orchestration system, Cited 20.05.2018

<https://github.com/hashicorp/terraform>

11. Elastic stack (ELK), Cited 05.06.2018

<https://www.elastic.co/products>

12. Burp Suite Pro, Cited 20.06.2018

<https://portswigger.net/burp/>

13. OWASP ZAP project, Cited 20.06.2018

[https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

14. W3af project, Cited 20.06.2018

<http://w3af.org/>

15. Automated Audit using W3AF, Cited 03.07.2018

[https://www.owasp.org/index.php/Automated\\_Audit\\_using\\_W3AF](https://www.owasp.org/index.php/Automated_Audit_using_W3AF)

16. Carbonator plugin for Burp Suite Pro, Cited 10.07.2018

<https://portswigger.net/bappstore/e3a26fff8e1d401dade52f3a8d42d06b>

17. Burp Suite Pro Rest API from VMware, Cited 10.07.2018

<https://github.com/vmware/burp-rest-api>

18. Swagger tool to document API, Cited 10.07.2018

<https://swagger.io/>

19. Portswigger forum on developing new CI/CD tool for security scans, Cited 10.07.2018

<https://support.portswigger.net/customer/portal/questions/16672914-integration-of-burp-with-jenkins>

20. Selenium website, Cited 18.07.2018

<https://docs.seleniumhq.org/>

21. Facebook's php webdriver for Selenium, Cited 18.07.2018

<https://github.com/facebook/php-webdriver>

22. BDD-Security tool from Continuumsecurity, Cited 22.07.2018

<https://www.continuumsecurity.net/bdd-security/>

23. Mittn tool from F-secure, Cited 01.08.2018

<https://github.com/F-Secure/mittn>

24. Skinner tool for automatic security tests, Cited 01.08.2018

<https://github.com/LianaTech/skinner>

25. Radamsa Fuzz testing tool, Cited 05.08.2018

<https://gitlab.com/akihe/radamsa>

26. OUSPG (Oulu University Secure Programming Group), Cited 05.08.2018

<https://www.ee.oulu.fi/research/ouspg/>

27. OSquery by Facebook, Cited 08.08.2018

<https://osquery.io/>

28. Wired, Firewalls don't stop hackers, AI might, 27.09.2017, Cited 08.08.2018

<https://www.wired.com/story/firewalls-dont-stop-hackers-ai-might/>

29. HELK software (A hunting ELK), Cited 08.08.2018

<https://github.com/Cyb3rWard0g/HELK>

## APPENDICES

```
#-----  
---  
  
#                               W3AF AUDIT SCRIPT FOR WEB APPLICATION  
#-----  
---  
  
#Configure HTTP settings  
  
http-settings  
  
set timeout 30  
  
back  
  
#Configure scanner global behaviors  
  
misc-settings  
  
set max_discovery_time 20  
  
set fuzz_cookies True  
  
set fuzz_form_files True  
  
set fuzz_url_parts True  
  
set fuzz_url_filenames True  
  
back  
  
plugins  
  
#Configure entry point (CRAWLING) scanner  
  
crawl web_spider  
  
crawl config web_spider  
  
set only_forward False  
  
set ignore_regex (?i)(logout|disconnect|signout|exit)+  
  
back  
  
#Configure vulnerability scanners  
  
##Specify list of AUDIT plugins type to use
```

```
audit blind_sqli, buffer_overflow, cors_origin, csrf, eval, file_upload, ldapi, lfi,
os_commanding, phishing_vector, redos, response_splitting, sqli, xpath, xss, xst
```

```
##Customize behavior of each audit plugin when needed
```

```
audit config file_upload
```

```
set extensions
jsp,php,php2,php3,php4,php5,asp,aspx,pl,cfm,rb,py,sh,ksh,csh,bat,ps,exe
```

```
back
```

```
##Specify list of GREP plugins type to use (grep plugin is a type of plugin that
can find also vulnerabilities or informations disclosure)
```

```
grep analyze_cookies, click_jacking, code_disclosure, cross_domain_js, csp, di-
rectory_indexing, dom_xss, error_500, error_pages,
```

```
html_comments, objects, path_disclosure, private_ip, strange_headers,
strange_http_codes, strange_parameters, strange_reason, url_session,
xss_protection_header
```

```
##Specify list of INFRASTRUCTURE plugins type to use (infrastructure plugin is
a type of plugin that can find informations disclosure)
```

```
infrastructure server_header, server_status, domain_dot, dot_net_errors
```

```
#Configure target authentication
```

```
auth detailed
```

```
auth config detailed
```

```
set username admin
```

```
set password password
```

```
set method POST
```

```
set auth_url http://pcdom/dvwa/login.php
```

```
set username_field user
```

```
set password_field pass
```

```
set check_url http://pcdom/dvwa/index.php
```

```
set check_string 'admin'
```

```
set data_format username=%U&password=%P&Login=Login
```

```
back
```

```
#Configure reporting in order to generate an HTML report
```

```
output console, html_file
```

```
output config html_file
```

```
set output_file /tmp/W3afReport.html
```

```
set verbose False
```

```
back
```

```
output config console
```

```
set verbose False
```

```
back
```

```
back
```

```
#Set target informations, do a cleanup and run the scan
```

```
target
```

```
set target http://pccom/dvwa
```

```
set target_os windows
```

```
set target_framework php
```

```
back
```

```
cleanup
```

```
start
```