Automating Security Test-cases using DevSecOps approach for AWS Serverless application with WebSockets

MSc Industrial Internship
Cyber Security

Deven Ahlawat
Student ID: x20214341

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Deven Ahlawat
Student ID:	x20214341
Programme:	Cyber Security
Year:	2022-2023
Module:	MSc Industrial Internship
Supervisor:	Vikas Sahni
Submission Due Date:	06/01/2023
Project Title:	Automating Security Test-cases using DevSecOps approach for AWS Serverless application with WebSockets
Word Count:	6399
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Deven Ahlawat
Date:	6th January 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Automating Security Test-cases using DevSecOps approach for AWS Serverless application with WebSockets

Deven Ahlawat
x20214341

Abstract

This research focused on the automation of security test cases for an AWS Serverless application with WebSockets using a DevSecOps approach. The study employed the popular web scanner Burp Suite Professional, along with a custom extension and Python, to automate the testing process. The research found that automation was possible for serverless applications using this approach. Results indicated that the automation process yielded similar results to manual testing for in-scope applications and successfully identified security issues. However, the study also observed that the automation process was limited to testing a set of predefined payloads and did not address potential vulnerabilities in HTTP headers. Overall, the research demonstrated the feasibility of using a DevSecOps approach for automating security testing of serverless applications with WebSockets.

1 Introduction

1.1 Research Background

Application development requires a lot of effort from the developers to server engineers to keep it running and troubleshoot regularly to minimise downtime. Companies are looking to improve this situation by adopting new containerisation solution to adopt a micro services based architecture which has been proved to be very effective. Serverless technology adoption has completely eliminated the need for developing a server to manage the application and need to secure it with every change and update. This allows the security team to pass on the responsibility to manage security threats to the infrastructure provider like AWS (Amazon Web Service), GCP and Azure Eismann et al. (2021) Wang et al. (2018).

This does not mean security vulnerabilities do not exist in these serverless application but the methodology employed to test these application is different which can be referred from the OWASP Top 10 checklist. The OWASP Top 10 is a widely-recognized list of the most critical web application security risks, developed by the Open Web Application Security Project (OWASP). Serverless applications may have a secure server managed by the Cloud Provider but vulnerable code can result in application-level vulnerabilities which allows the customer to employ the usual application pentesting guide updated in 2021 by OWASP OWASP (2021).

The given application by Akeero ran on AWS serverless utilising Lambda, DynamoDB and Application Gateway. Manual pentesting resulted that all the communication is happening using web-sockets instead of APIs which safeguarded the application from multiple attack vectors like session replay because the application was in constant sync with a randomly-generated unique identifier in every socket call. An attacker could attempt to test multiple payloads, while in case of a web-socket, contact synchronisation was observed which made it difficult to replay the request with the help of BurpSuite Professional or any other proxy.

WebSockets provide the client or server to establish a 'full-duplex' (two-way) communication channel, enabling the client and server to communicate asynchronously. WebSockets perform their initial upgrade handshake over HTTP, and thereafter all communication is conducted using TCP channels and frames. WebSocket provides bidirectional communication, security, functionality, and efficiency to web apps Fette and Melnikov (2011). However, WebSocket service providers can configure some security decisions made during service construction. All decisions affect WebSocket security, therefore, security testing is important for the same reasons as other web technologies Marin et al. (2022).

1.2 Problem Definition

It is not possible to do an automated security test on a WebSocket implementation using widely used online security testing tools like Burp Suite or OWASP ZAP. These tools are not designed for this purpose. This leaves it as a manual task that requires comprehensive understanding of how the implementations work, what the common security concerns in them are, and how they may be security tested Kuosmanen (2016) Marin et al. (2022). The objective of this research was to study Serverless WebSockets based application attacks and automate those test-cases using DevSecOps approach in the CICD pipeline.

RQ 1 - Is it possible to perform Dynamic Automated Security Assessment on Serverless WebSocket based application?
RQ 2 - How is the false positives rate for DAST scans for Serverless WebSocket based application as recorded by commercial webapplication scanners?

In order to accomplish the task at hand, it was necessary to get an understanding of the typical security flaws that are present in WebSocket implementations. Because problems might occur on numerous levels, it was imperative that proper scope be established to ensure that the problems were confined to WebSocket based application [1] alone. The reader, armed with prior knowledge of typical security flaws, is then expected to have an understanding of how these flaws might be tested, both in theory and in practice. For this purpose, an in-depth technical understanding of the WebSocket protocol security and the web security testing tools is required.

A sample AWS serverless application infrastructure is depicted in Figure Figure 1 which highlights that native AWS services are used in this architecture which almost minimises if not eliminates the risk of regular updates to the supporting infrastructure Marin et al. (2022).

[1]Akeero Serverless

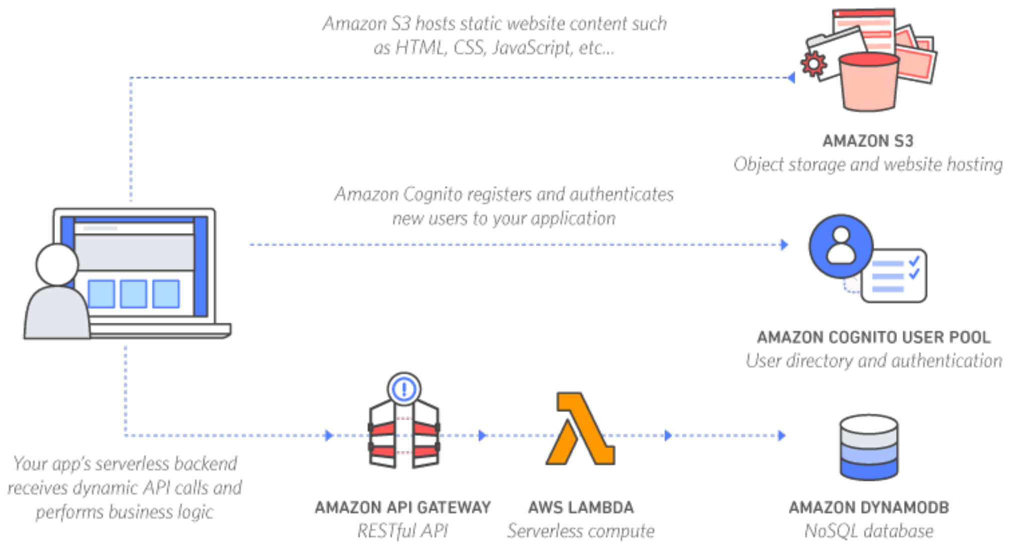

Amazon S3 hosts static website content such as HTML, CSS, JavaScript, etc...
AMAZON S3
Object storage and website hosting
Amazon Cognito registers and authenticates new users to your application
AMAZON COGNITO USER POOL
User directory and authentication
Your app's serverless backend receives dynamic API calls and performs business logic
AMAZON API GATEWAY
RESTful API
AWS LAMBDA
Serverless compute
AMAZON DYNAMODB
NoSQL database

Figure 1: AWS Serverless application infrastructure

2 Related Work

This paper discussed the recent works on security automation using two approaches including SAST and DAST. Sub-section 1 contains the related works on SAST 2.1 and sub-section 2 discussed recent methods researched on DAST 2.2.

2.1 SAST

Open-source solution have been very effective in terms of security vulnerabilities finding in the code during the development stages of application. Zampetti et al. (2017) compares multiple open-source technologies available on the internet to find issues in the code Nguyen-Duc et al. (2021). Though executing all of them could be a challenge and the author has not made much tweaks on softwares which can increase their efficiency by finding more issues and reducing false-positives. This paper also focuses mostly on Java based application with less tools for Python. In the recent year more tools have evolved like SemGrep and Snyk which offers free scanning upto a limit with promising results Tomas et al. (2019).

Akujobi (2021) suggests a model to compare the improvement in Security in Continuous Integration pipelines. This paper suggests that SAST tools are not optimised enough and to perform at their best ability which can be improved by involving a person with development background to assist with the right policies suggesting that most tools are run with default configuration and policies. This also suggests a need to focus on optimising the result, which was also suggested by Garg and Sengamedu (2022) by stressing on code quality checks, and monitor issues which are applicable to the company environment mentioning the correct severity of the issues found.

2.2 DAST

Rajapakse et al. (2022) discusses multiple challenges faced while adoption the DevSecOps approach mentioning about the absence of the vulnerabilities relating to the hosted environment like containers which are often overlooked by most tools.

Petukhov and Kozlov (2008) discussed the use of dynamic analysis and penetration testing to detect security vulnerabilities in web applications and discussed it's need as the number of reported vulnerabilities have been increasing dramatically. These were mostly reported with help of security scanner like Acunetix. They then described dynamic analysis as a method of testing web applications by simulating real-world user behavior and interactions, which could help to identify vulnerabilities that might not be detected using static analysis techniques. This paper also highlighted that most of the reported vulnerabilities arise due to improper input validation and lack of testcases. One strength of this paper was that it provided a detailed description of the methodology used in the case study, which helps to make the research more transparent and replicable.

Albahar et al. (2022) This paper discussed about multiple tools that were used to perform web application scanning with a comparative analysis of their efficiency. One of the strengths of this paper was its empirical approach, which allowed the authors to provide concrete data on the performance of the different tools. The authors also clearly outlined the methodology of their study, including the selection of the tools and the vulnerable web applications used in the testing. This established the features and the variety offered by multiple scanners including both open-source and commercial scanner.

Qasaimeh et al. (2018) Conducted in 2017 to compare multiple web application scanners for their efficiency found BurpSuite to be less efficient as compared to other tools like ZAP, Nessus, NetSparker and Acunetix which had above 90% accuracy rate. These results were compared with Albahar et al. (2022) which was more recent taking into consideration the updates all tools have received over the years. One strength of this paper is its focus on black box scanners, which are commonly used in real-world scenarios and can be evaluated in a more practical manner. The authors also provide a detailed description of their evaluation method, including the security standards used and the criteria for evaluating the scanners. The paper provides a useful approach for evaluating the accuracy of black box web application scanners and sheds light on the limitations and strengths of different tools. However, more research is needed to fully understand the effectiveness of these scanners in a wider range of scenarios.

Gojare et al. (2015) This paper showed Selenium's capabilities to test web-application without the need of human interference with the help of a headless browser. Selenium also allowed for proxing the traffic through and made it possible to scan WebSocket based application. Holmes and Kellogg (2006) demonstrated that Selenium test scripts can be developed in table format and then later coded into any desired programming language such as Java, and Python. This represents code export capability that will eventually invoke the respective test scripts on the specified browsers.

3 Methodology

The methodology for this research has been devised using Research Onion Fig 2 ref Saunders et al. (2007) HARP model which has been discussed below.

The choices of methods used in this research were based on the research philosophy of positivism, which emphasizes the importance of empirical evidence and objective obser-

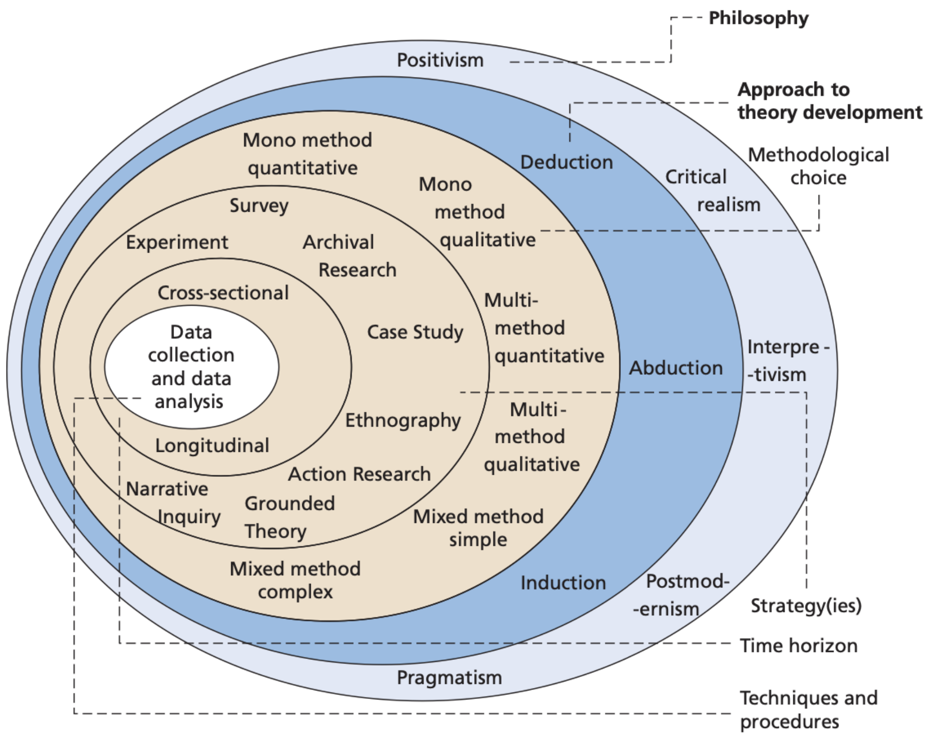


Figure 2: Research Onion

vation in the research process. This philosophy informed the decision to use a deductive research approach, as the research started with a broad set of hypotheses and aimed to narrow down to specific conclusions through the analysis of data.

Mono-method qualitative approach was used for this research because this type of research was focused on understanding a particular issue of Serverless applications which have multiple limitation to allow for an automated DAST.

In terms of research strategy, a case study approach was used in this research in order to examine the specific context of the AWS Serverless application with WebSockets and its implementation using the DevSecOps approach. This strategy allowed for in-depth analysis of the application and its security practices, and provided an opportunity to understand the unique challenges and solutions that were encountered during the implementation process.

The research choices made included the selection of Burp Suite Professional and Selenium WebDriver as the primary tools for the study, as well as the decision to use a case study approach. The case-study approach was specifically chosen because this research dealt with the challenges faced during the automation of web-socket based Serverless applications.

In this research, a longitudinal time horizon method was employed which involved collecting data on the security of the Akeero Serverless application with WebSockets over a period of time, and analyzing this data to identify any trends or changes. This process included regularly conducting security assessments using a combination of automated and manual testing methods, and tracking the results over time to identify any changes or trends in the application's security posture.

The techniques and procedures employed in the research was inspired by the SDLC

(Software Development Life Cycle) Waterfall Model with necessary changes. In Waterfall Model, the development is split into multiple phases, the output of one step feeds the next phase consecutively Model (2015). The guiding steps are mentioned in Fig 3 establishing requirement analysis and coverage followed by automation steps and evaluation.

This sections provides a detailed methodology that has been followed to achieve this research.

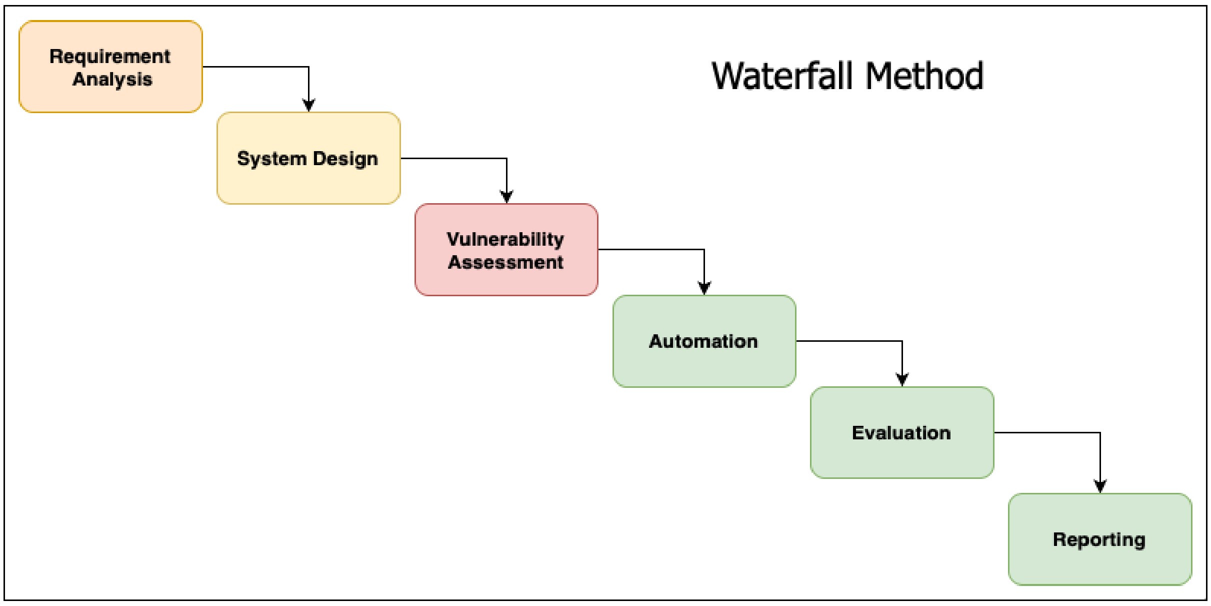


Figure 3: Project methodology steps

3.1 Requirement Analysis

The first step in the research methodology had been to define the application scope and walk-through of all its functionalities to identify different elements that could be vulnerable. After documenting all the features offered by the application, a manual pentest (penetration test) of the security test cases for the scoped web application was performed. This had involved manually testing the application for common vulnerabilities such as SQL injection, cross-site scripting (XSS), and insecure direct object references (IDOR) using a variety of manual testing tools and techniques. Testcases were defined and documented with the help of OWASP Top 10 (version 2021) vulnerabilities[2].

Once the manual pentest had been completed, the traffic for the web application had been proxied using a tool such as Burp Suite Professional in order to analyze the traffic and identify any additional security issues. The proxying process had involved intercepting and capturing the traffic between the client and server, allowing the security issues to be identified and analyzed in greater detail.

3.2 System Design

In this research, system design was restricted to the internal AWS infrastructure provided by Akeero with an EC2 Kali instance along with the vulnerability assessment licensed

[2]https://owasp.org/Top10/

scanner. This step also included the programming langauge and framework to work with for the automation.

3.3 Vulnerability Assessment

The results of the proxying process had been assessed in order to identify any security issues or vulnerabilities that may have been present in the web application. This assessment process had provided insight into the types of data and requests being sent between the client and server, allowing the identification and analysis of potential security issues such as injection attacks or cross-site scripting (XSS). The vulnerabilities that were identified had been carefully reviewed to ensure that they were properly captured and highlighted in the report. This had involved cross-verifying the reported vulnerabilities manually against the results of the proxying process to ensure their accuracy. The impact and certainty of each vulnerability had also been evaluated to determine its potential risk to the web application and the Akkero environment. This careful assessment of the vulnerabilities had helped to identify and prioritize the most critical issues that needed to be addressed in order to improve the security of the web application. Special attention had been paid to the evaluation of the vulnerabilities by the proxy tool and their applicability as per the actual CVSS score [3] in the Akkero environment in order to determine the most appropriate course of action for addressing them.

3.4 Automation

3.4.1 Testcases Automation

Once the manual pentest and proxying process had been completed, the next step was to automate the security test cases using a tool such as Selenium WebDriver. This involved writing automation scripts in a programming language such as Python in order to automate the testing process. The automation process was made headless by using Selenium to run the automation scripts without the need for a visible browser window. This allowed the automation scripts to run in the background, providing a more memory efficient and uninterrupted testing process. Post automation, comparison was made with the original results generated by the manual pentest to ensure unit testing and accuracy of the script.

3.4.2 Proxy Automation

In addition to automating the security test cases, the process of proxying the traffic had to be automated in order to start the proxy before the scripts started execution and end the process after the testing was complete. This was aimed at reducing the CPU and RAM overhead and prevent the host machine from becoming slower. This step also ensured that proper clean up was done after the script execution was performed, allowing for a fresh scan every time the script executed.

3.5 Reporting

Once the automated security test cases had been completed and the proxying process was automated, the next step was to automate the generation of the security report.

[3]https://nvd.nist.gov/vuln-metrics/cvss

This involved using Python scripts to analyze the results of the automated testing and proxying processes and generate a report detailing any security issues or vulnerabilities that were identified. The generated report was aimed to have the brief information about the issues intended for all stakeholders, this information comprised of Issue Type, Description, Severity and Likelihood. The final step in the research methodology was to deliver the security report to the appropriate stakeholders. This involved emailing the report to the relevant individuals or departments within the organization, as well as any external parties that were involved in the testing process.

4 Design Specification

To demonstrate the efficiency of DAST, Burp Suite Professional was used in passive scanning mode as a proxy to analyze traffic. The efficiency also relied on the number of test cases from the OWASP Top 10 that this research was able to replicate, which were compared with the manual pentesting approach. Overall, effective DAST for websocket-based web applications is important because it can help identify and prevent security vulnerabilities that could be exploited by malicious actors, protecting the web application and its users from potential security breaches. Fig 4 shows the overview of the implementation. Inputs to the automation script are fed from after comparing the results to increase the efficiency.

Table 1: OWASP Top 10 2021 vulnerabilities

Broken Access Control
Cryptographic Failures
Injection
Insecure Design
Security Misconfiguration
Vulnerable and Outdated Components
Identification and Authentication Failures
Software and Data Integrity Failures
Security Logging and Monitoring Failures
Server-Side Request Forgery

Manual pentesting was performed using the BurpSuite Professional licensed to Akeero which has many capabilities to test an application. It offers a range of capabilities for performing automated and manual testing, as well as network-level testing. One key capability of Burp Suite Professional is its ability to perform web application vulnerability scanning to identify potential vulnerabilities using automated tools. It also includes manual testing tools such as an HTTP request editor and a web application spider.

In Table 2 system configuration used has be mentioned.

As websockets were involved in the Akeero application, some capabilities of BurpSuite wont be utilised like Repeater and Intruder as the application doesn't allow replay of requests. To tackle this and perform those checks, automation was used in addition to the existing vulnerabilities checks.

The pentesting was done by using BurpSuite Professional in the Manual approach, where it was made to run in the Active mode along with the Passive scanning mode. This was the preferred choice after analysing its capabilities as compared to OWASP ZAP.

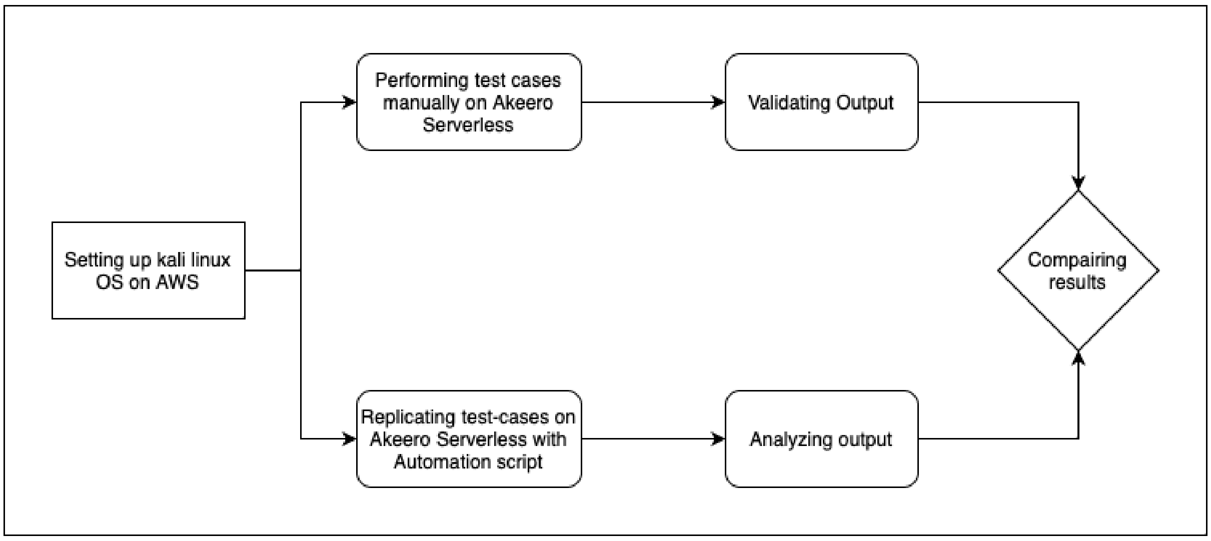

Setting up kali linux OS on AWS
Performing test cases manually on Akeero Serverless
Validating Output
Replicating test-cases on Akeero Serverless with Automation script
Analyzing output
Compairing results

Figure 4: Flowchart overview of the project

Table 2: Preferred configuration for the host

Property	**Description**	**Version**
Platform	AWS	n/a
Instance Type	Type T	t2.medium
Operating System	Kali	2022
Programming Language	Python	3.10
RAM	4GB	n/a
CPU	2vcore	n/a

Headless execution of these tools was preferred to reduce overhead on processor make the execution possible in linux systems without GUI installed. This testing was performed in Kali Linux (6.0.0-kali5-cloud-amd64) deployed on AWS EC2 instance with *t2.medium*[4] specification with 2 vcores and 4 GB of RAM. BurpSuite Professional v2022.2.5 released in December 2022 in the JAR format because of the headless capability only exclusive to jar file.

Python was used to automate the pentest flow with Selenium WebDriver and Chrome. Selenium was chosen because it allows for cross-browser implementation and greater flexibility with element operation using Python [5]. Python 3.10 was used for this implementation, along with the external packages selenium (v4.7.2), pyotp (v2.8.0), and smtplib. PyOTP was used to handle the mandatory multi-factor authentication using Time-Based One-Time Password (TOTP) implemented on the Akeero Serverless.

A custom Burp Suite extension was developed in Python, which was used to extract the passive issues reported by the BurpScanner and save them to a file. The saved file was analyzed by the "main.py" Python script to create an overview of the findings and mail it, along with the original report (exported by the extension).

[4]https://aws.amazon.com/ec2/instance-types/

[5]https://brightdata.com/blog/proxy-101/puppeteer-vs-selenium

5 Implementation

5.1 Scope Definition

Scope was determined by opening the Akeero website[6] and signing up for their serverless application as a normal user. As Akeero uses a unique subdomain for each user, the application URL in scope of this research was *https://testings-human-sand-dollar-cfb6.free.app.akeero.com/*. A complete walk-through of the application was carried out to check all its functionalities and identify vulnerable elements which included input fields including text, images, files, etc.

Post the application walk through, list of features were documented to identify vulnerabilities from the OWASP Top 10 (release 2021) that would be applicable to the scoped in application. Akeero serverless mostly included the following input methods -

- Text based input - These inputs were found to have accepted Alphabet, Numbers and Special Characters as input.
- File upload - accepted only JSON file types less than 1 MB.
- URL rewriting - This was completely user-controlled which allowed all text input.

With the above conditions, the list of applicable vulnerabilities that could be tested are listed in Table 3 followed by the justification for each category.

Table 3: OWASP Top 10 2021 vulnerabilities

Broken Access Control	Partial
Cryptographic Failures	Partial
Injection	Applicable
Insecure Design	NA
Security Misconfiguration	Applicable
Vulnerable and Outdated Components	Applicable
Identification and Authentication Failures	NA
Software and Data Integrity Failures	NA
Security Logging and Monitoring Failures	NA
Server-Side Request Forgery	Applicable

Broken Access Control could be partially tested after the login was successful because of third party application usage for login and signup called Auth0 by Okta [7] which was out-of-scope. Test-cases included -

1. Trying to access projects from different account
2. Trying to access settings restricted to premium user plan
3. Trying to add users (restricted to premium users)
4. Trying to delete projects from different account

[6]https://akeero.com

[7]https://auth0.com

Cryptographic Failures could also be partially tested relying on the capabilities of BurpSuite. Major checks included use of latest encryption algorithms, valid certificates, SSL version used, etc. Cryptographic checks are performed during the Secure Code review which is more credible as the algorithm used in clearly mentioned in the application code.

*Injection attack*s are a type of security vulnerability that can occur when an attacker is able to send malicious input to an application, which is then executed by the application. This can allow the attacker to gain unauthorized access to sensitive data, execute arbitrary commands, or manipulate the application in other ways OWASP (2021). Test-cases for injection attacks included -

1. SQL injection attacks: These attacks involve injecting malicious SQL code into an application in order to gain unauthorized access to sensitive data stored in a database.

2. LDAP injection attacks: These attacks involve injecting malicious LDAP statements into an application in order to gain unauthorized access to sensitive data stored in an LDAP directory.

3. OS command injection attacks: These attacks involve injecting malicious OS commands into an application in order to execute arbitrary commands on the server.

4. XML injection attacks: These attacks involve injecting malicious XML code into an application in order to manipulate or access sensitive data stored in an XML document.

5. HTML injection attacks: These attacks involve injecting malicious HTML code into an application in order to manipulate the appearance or behavior of the application.

6. XSS (Cross-Site Scripting) attack: These involves injecting malicious JavaScript code into a web application in the context of a victim's browser, allowing the attacker to gain unauthorized access to sensitive data or manipulate the appearance or behavior of the application.

Insecure Design could not be tested as it involves security checks in Secure Software Development Cycle and abuse case checks which are out of scope for this assessment.

Security Misconfiguration is defined as a vulnerability that occurs when web application security is not properly configured. This can include issues such as default accounts with weak passwords, unnecessary services that are enabled and exposed, or unpatched vulnerabilities. Security misconfiguration can allow attackers to gain unauthorized access to the system, steal sensitive data, or execute malicious code. This type of vulnerabilities are easily detected by burp as it extracts version number of applications and components being used in the application to identify them against any known vulnerability. Burp also highlights version number detected in the Dashboard tab using an extension called Software Version Reporter[8] which covers the *Vulnerable and Outdated Components* category from OWASP Top 10 [9]

Identification and Authentication Failures was defined as the vulnerabilities in the authentication implementation. In this case, Akeero uses a outsourced service as mentioned

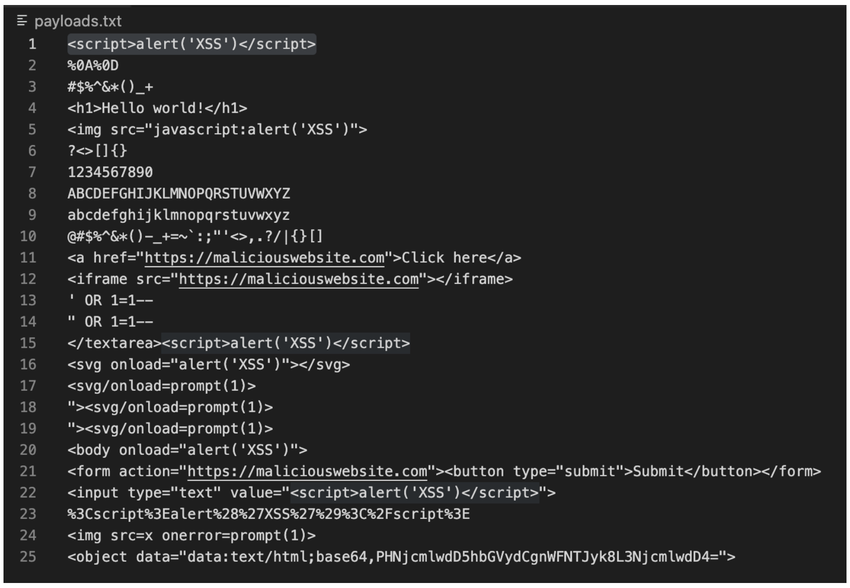
[8]https://portswigger.net/bappstore/ae62baff8fa24150991bad5eaf6d4d38

[9]https://portswigger.net/support/using-burp-to-test-for-the-owasp-top-ten

above, which is out of scope for DAST. *Software and Data Integrity Failures* was out-of-scope of this implementation as this was covered mainly in CI pipeline with different tools and scripts. *Security Logging and Monitoring Failures* was out-of-scope because it is mainly to do with back-end logging and alerting systems.

Server-Side Request Forgery is a type of attack that involves an attacker causing a server to perform a request on their behalf. This can allow the attacker to access sensitive information, bypass security controls, or execute arbitrary code on the server. Testcases to check for SSRF was included in the payload file.

The final payload file consists of payloads as shown in Figure 5



```
payloads.txt
<script>alert('XSS')</script>
%0A%0D
#$%^&*()_+
<h1>Hello world!</h1>
<img src="javascript:alert('XSS')">
?<>[]{}
1234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
@#$%^&*()-_+=~`:;"'<>,.?/|{}[]
<a href="https://maliciouswebsite.com">Click here</a>
<iframe src="https://maliciouswebsite.com"></iframe>
' OR 1=1--
" OR 1=1--
</textarea><script>alert('XSS')</script>
<svg onload="alert('XSS')"></svg>
<svg/onload=prompt(1)>
"><svg/onload=prompt(1)>
"><svg/onload=prompt(1)>
<body onload="alert('XSS')">
<form action="https://maliciouswebsite.com"><button type="submit">Submit</button></form>
<input type="text" value="<script>alert('XSS')</script>">
%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E
<img src=x onerror=prompt(1)>
<object data="data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4=">
```

Figure 5: Payload file for the DAST

5.2 Vulnerability Assessment

Manually perform a pentest (penetration test) of the security test cases for the scoped web application with multiple payloads from PayloadAllThings repository[10] which is a carefully curated list of all the vulnerabilities payloads.

Manual Pentest was performed on a different subdomain[11] by setting up the proxy on a Firefox browser and running BurpSuite Professional for scanning. Payloads were entered in different input fields as shown in Fig 6. This subdomain was also used to check if the two environments were properly isolated and were not able to access each other's resources by trying to access projects via it's id.

Test for common vulnerabilities such as SQL injection, cross-site scripting (XSS), and insecure direct object references (IDOR) was done by manually inputting them in the input fields individually.

[10]https://swisskyrepo.github.io/PayloadsAllTheThingsWeb/

[11]https://accounttwos-personal-mongoose-e9e9.free.app.akeero.com/

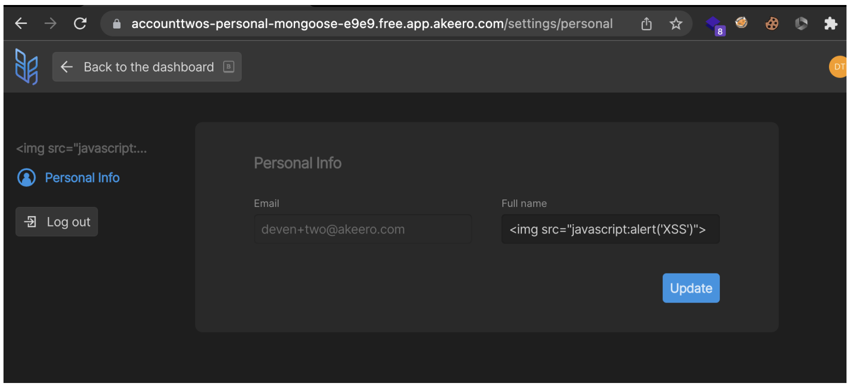
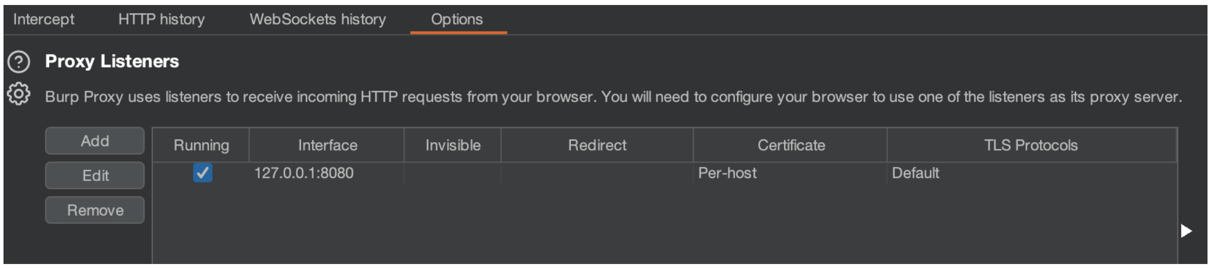


Figure 6: Sample input field test with a payload during manual pentest

5.3 Testcases Automation

Automation was done with the help of Selenium WebDriver on scoped in subdomain as mentioned above, it was connected with chromium browser to execute the test cases from the *payload.txt* file. BurpSuite proxy was setup as shown in the Fig 7 and 8 to route all the traffic for scanning.

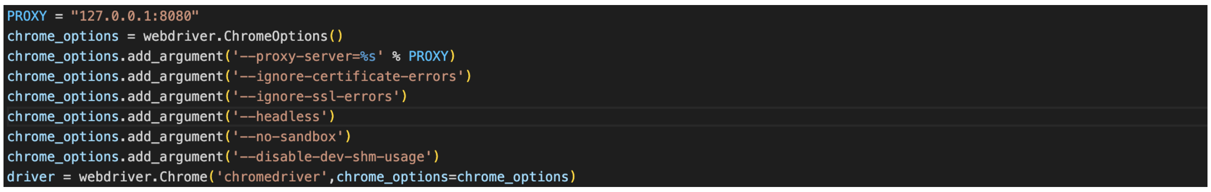


Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

Running	Interface	Invisible	Redirect	Certificate	TLS Protocols
✓	127.0.0.1:8080			Per-host	Default

Figure 7: BurpSuite Proxy configuration

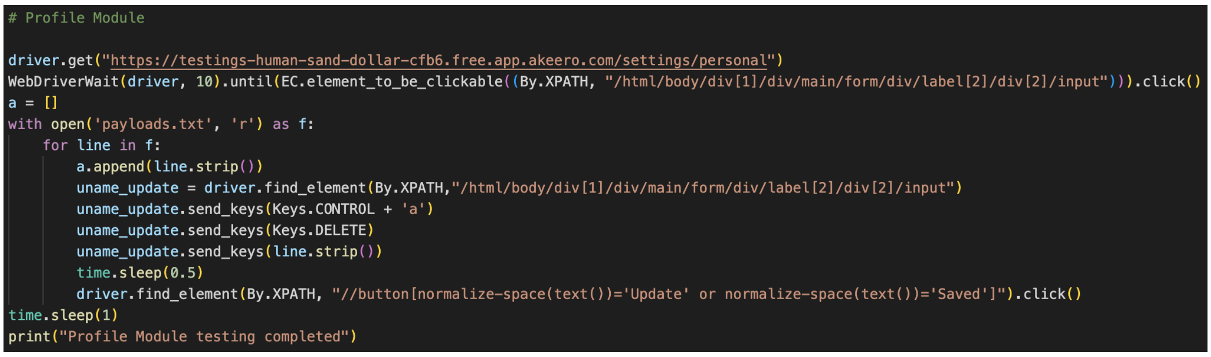


```
PROXY = "127.0.0.1:8080"
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--proxy-server=%s' % PROXY)
chrome_options.add_argument('--ignore-certificate-errors')
chrome_options.add_argument('--ignore-ssl-errors')
chrome_options.add_argument('--headless')
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')
driver = webdriver.Chrome('chromedriver',chrome_options=chrome_options)
```

Figure 8: Proxy and Browser configuration

Automation code was written with Python and modules were defined to run the automation in different phases testing each module for easy management and complete coverage as shown in fig 9.

Comments were used to highlight section beginning and necessary output was printed to logging and debug purpose. Next step was to remove the GUI (Graphical User Interface) and make this headless which was done with the help of Selenium WebDriver variable *chrome_options* with the parameter --*headless* as shown in Fig 8.



```
# Profile Module

driver.get("https://testings-human-sand-dollar-cfb6.free.app.akeero.com/settings/personal")
WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.XPATH, "/html/body/div[1]/div/main/form/div/label[2]/div[2]/input"))).click()
a = []
with open('payloads.txt', 'r') as f:
    for line in f:
        a.append(line.strip())
        uname_update = driver.find_element(By.XPATH,"/html/body/div[1]/div/main/form/div/label[2]/div[2]/input")
        uname_update.send_keys(Keys.CONTROL + 'a')
        uname_update.send_keys(Keys.DELETE)
        uname_update.send_keys(line.strip())
        time.sleep(0.5)
        driver.find_element(By.XPATH, "//button[normalize-space(text())='Update' or normalize-space(text())='Saved']").click()
time.sleep(1)
print("Profile Module testing completed")
```

Figure 9: Profile module testing code snippet

5.4 Proxy Automation

Automation of BurpSuite Professional was done to start itself and the process of proxying traffic to start the proxy before the scripts start execution and end the process after the testing was complete. This was done with the help of JAR file of the BurpSuite which had an option to start it in headless mode with the required extensions as per the configuration files included. The process was started with the help of *subprocess* Python library. This was done to reduce CPU and RAM overhead to prevent the host machine from becoming slower processing GUI while allowing it run on linux machine without a GUI.

Ensure proper clean up was done at the beginning of the code when burp was initialised to avoid overlapping of results or corrupting the scan with last scan project. The burp process was closed at the end of the script to allow for a fresh scan every time the script executes.

5.5 Reporting

A custom Burp Suite extension was developed to export the issues reported by Burp's scanner module and save them to a file. This extension was entirely code in Python with the below code snippet (Fig 11) to fetch details about an issue which included Issue Name, Issue Details, Severity and Confidence using the Burp Guide[12]

Develop a custom Burp Suite extension in Python to extract passive issues reported by the BurpScanner and save them to a file Analyze the saved file using the main.py Python script to prepare an overview of the findings Mail the overview and original report (file exported by the extension)

6 Evaluation

Evaluation of this research was to answer multiple RQs which were discussed in the implementation while keeping in mind the pentesting benchmark to scan for all the applicable OWASP Top 10 vulnerabilities in the DAST process to ensure effectiveness.

[12]https://portswigger.net/burp/extender/writing-your-first-burp-suite-extension

6.1 Automation

Another objective was to check if automation of the WebSocket based application could be done via DevSecOps approach with CICD integration which was proved to be possible starting with the initiation phase to trigger the execution of script until the reporting part that generated a report along with a summary that was mailed to the stakeholders at the end of the scan with the help of Python. Figure 10 illustrates the different integration to make this automation possible in the AWS infrastructure of Akeero that comprised of the trigger by GitHub Actions to SSH into the EC2 instance that had a burp installation and python script on it. The python script invoked the burp and chrome initiation to start the execution of test-cases from the payload file on different modules of Akeero application. The report generation was a challenge as BurpSuite Professional does not provide a functionality to automate the report generation which was solved with the help of a custom BurpSuite extension, code snippet of which is shown in Fig 11. The python script then analysed those issues to prepare a summary with unique issue names and emailed it to the stakeholders. A sample email received is shown in Fig 12.

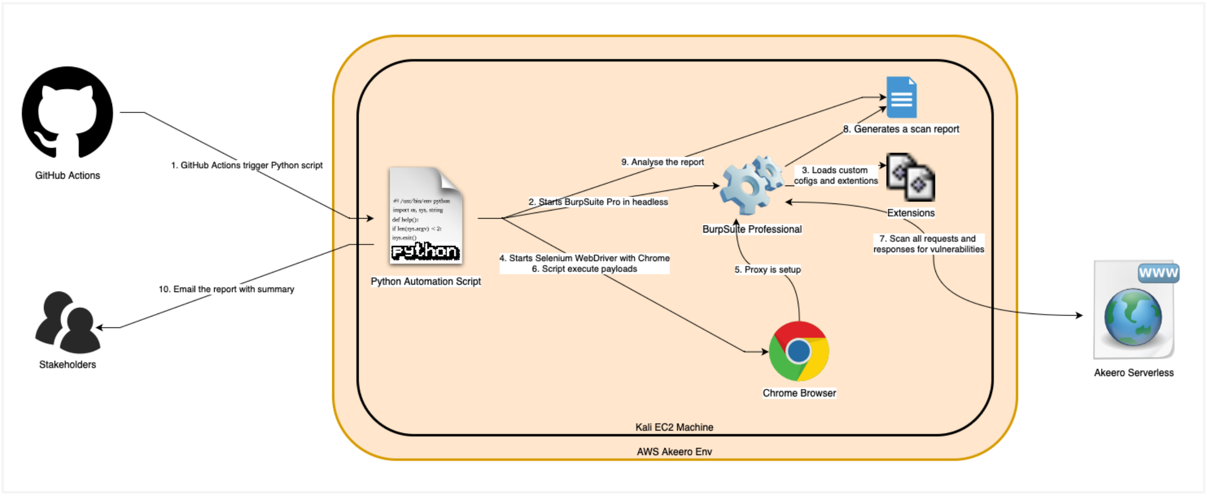

GitHub Actions
1. GitHub Actions trigger Python script
Stakeholders
10. Email the report with summary
#! /usr/bin/env python
import os, sys, string
def help():
if len(sys.argv) < 2:
sys.exit()
python
Python Automation Script
9. Analyse the report
2. Starts BurpSuite Pro in headless
4. Starts Selenium WebDriver with Chrome
6. Script execute payloads
8. Generates a scan report
3. Loads custom cofigs and extentions
Extensions
BurpSuite Professional
7. Scan all requests and responses for vulnerabilities
5. Proxy is setup
Chrome Browser
WWW
Akeero Serverless
Kali EC2 Machine
AWS Akeero Env

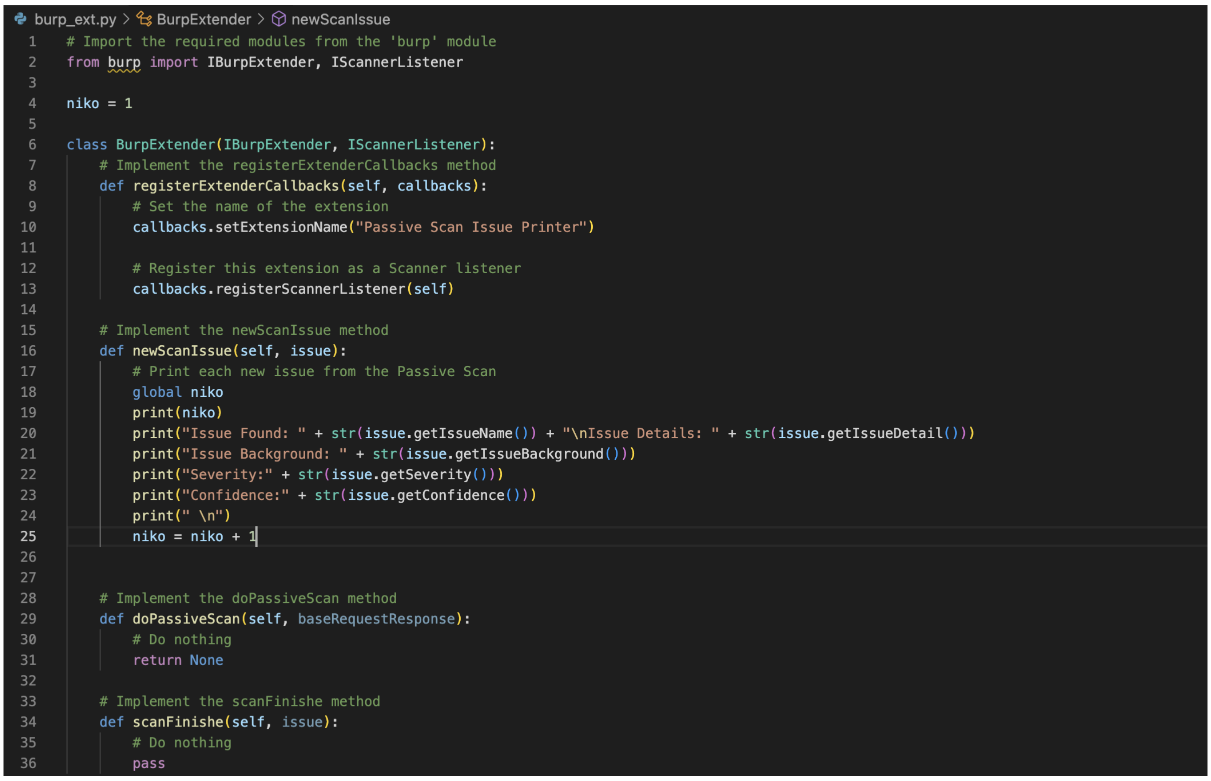
Figure 10: Project workflow model for automation

6.2 Issues Testing

As observed from the python script, it can be observed that the payload file was properly read and all the inputs were tested as shown in the sample Fig 6 with the output and it's respective implementation code shown in Fig 9. This helps us conclude the following checks against the OWASP Top 10 vulnerabilities checklist in the Table 4. Figure 13 shows the output from the scanner which added up to 25 unique issues while the total number reported was 27 with 2 duplicates, that were found while testing for the applicable vulnerabilities with automation.

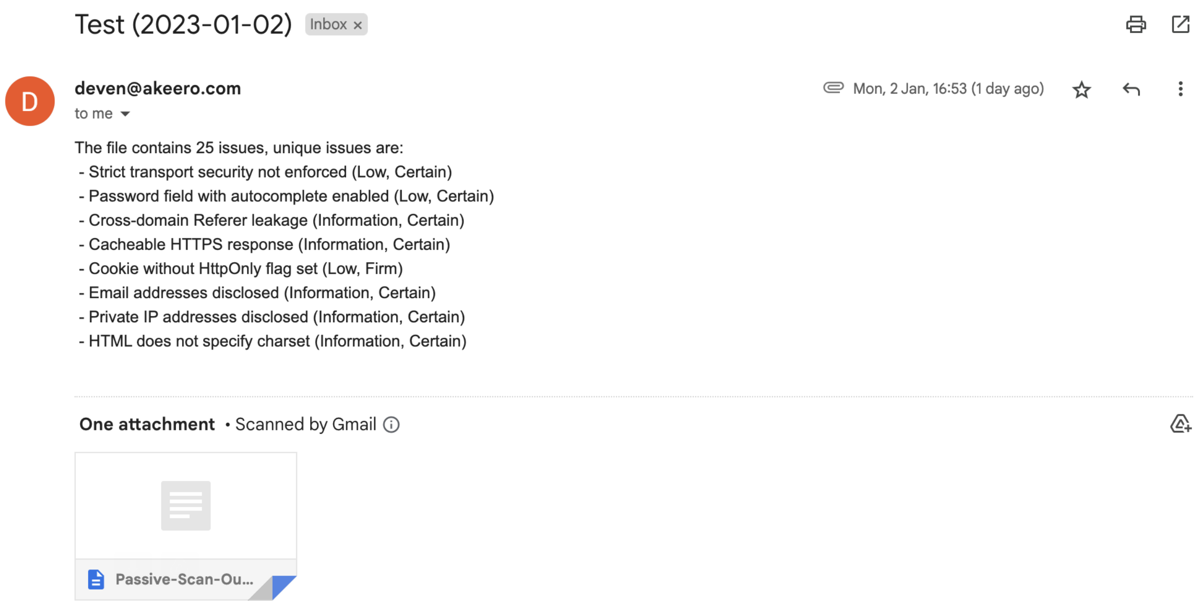
6.3 Accuracy

The results were verified to be in line with the manual approach which had a much higher time complexity and effort. Because this case study focused on testing if the test cases



```python
# Import the required modules from the 'burp' module
from burp import IBurpExtender, IScannerListener

niko = 1

class BurpExtender(IBurpExtender, IScannerListener):
    # Implement the registerExtenderCallbacks method
    def registerExtenderCallbacks(self, callbacks):
        # Set the name of the extension
        callbacks.setExtensionName("Passive Scan Issue Printer")

        # Register this extension as a Scanner listener
        callbacks.registerScannerListener(self)

    # Implement the newScanIssue method
    def newScanIssue(self, issue):
        # Print each new issue from the Passive Scan
        global niko
        print(niko)
        print("Issue Found: " + str(issue.getIssueName()) + "\nIssue Details: " + str(issue.getIssueDetail()))
        print("Issue Background: " + str(issue.getIssueBackground()))
        print("Severity:" + str(issue.getSeverity()))
        print("Confidence:" + str(issue.getConfidence()))
        print(" \n")
        niko = niko + 1


    # Implement the doPassiveScan method
    def doPassiveScan(self, baseRequestResponse):
        # Do nothing
        return None

    # Implement the scanFinishe method
    def scanFinishe(self, issue):
        # Do nothing
        pass
```

Figure 11: Custom BurpSuite Extension to generate report

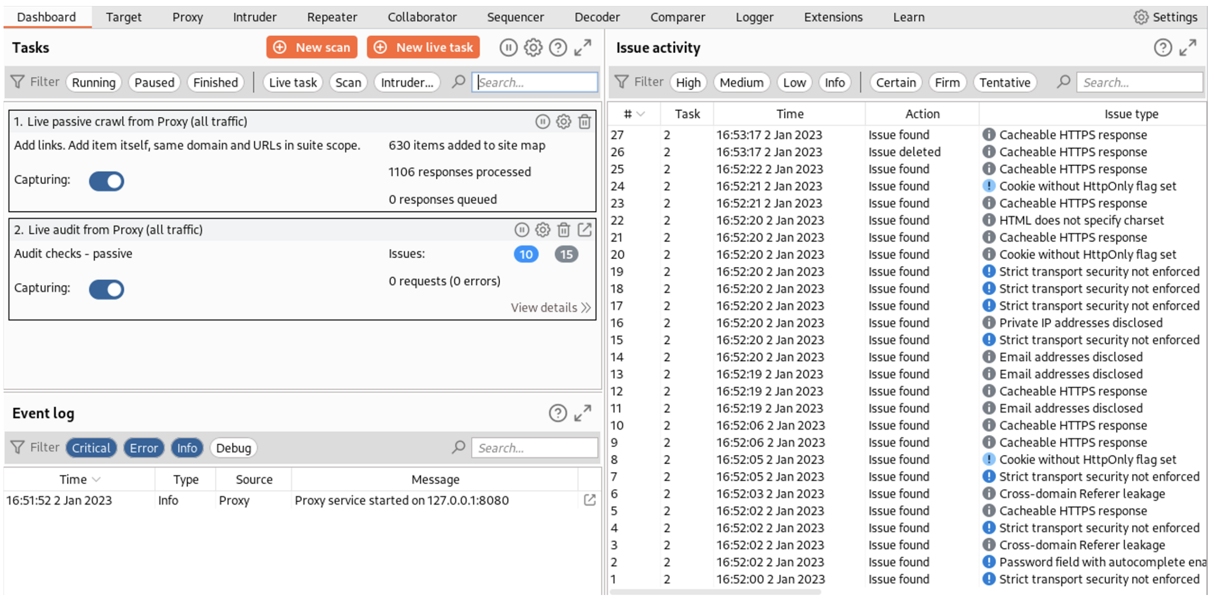


The file contains 25 issues, unique issues are:
- Strict transport security not enforced (Low, Certain)
- Password field with autocomplete enabled (Low, Certain)
- Cross-domain Referer leakage (Information, Certain)
- Cacheable HTTPS response (Information, Certain)
- Cookie without HttpOnly flag set (Low, Firm)
- Email addresses disclosed (Information, Certain)
- Private IP addresses disclosed (Information, Certain)
- HTML does not specify charset (Information, Certain)

Figure 12: Sample email received by the stakeholders

Table 4: OWASP Top 10 2021 vulnerabilities outcome

Broken Access Control	Partial	Tested
Cryptographic Failures	Partial	Tested
Injection	Applicable	Tested
Insecure Design	NA	NA
Security Misconfiguration	Applicable	Tested
Vulnerable and Outdated Components	Applicable	Tested
Identification and Authentication Failures	NA	NA
Software and Data Integrity Failures	NA	NA
Security Logging and Monitoring Failures	NA	NA
Server-Side Request Forgery	Applicable	Tested



1. Live passive crawl from Proxy (all traffic)
Add links. Add item itself, same domain and URLs in suite scope.
Capturing:
630 items added to site map
1106 responses processed
0 responses queued

2. Live audit from Proxy (all traffic)
Audit checks - passive
Capturing:
Issues: 10 15
0 requests (0 errors)

Time	Type	Source	Message
16:51:52 2 Jan 2023	Info	Proxy	Proxy service started on 127.0.0.1:8080

#	Task	Time	Action	Issue type
27	2	16:53:17 2 Jan 2023	Issue found	Cacheable HTTPS response
26	2	16:53:17 2 Jan 2023	Issue deleted	Cacheable HTTPS response
25	2	16:52:22 2 Jan 2023	Issue found	Cacheable HTTPS response
24	2	16:52:21 2 Jan 2023	Issue found	Cookie without HttpOnly flag set
23	2	16:52:21 2 Jan 2023	Issue found	Cacheable HTTPS response
22	2	16:52:20 2 Jan 2023	Issue found	HTML does not specify charset
21	2	16:52:20 2 Jan 2023	Issue found	Cacheable HTTPS response
20	2	16:52:20 2 Jan 2023	Issue found	Cookie without HttpOnly flag set
19	2	16:52:20 2 Jan 2023	Issue found	Strict transport security not enforced
18	2	16:52:20 2 Jan 2023	Issue found	Strict transport security not enforced
17	2	16:52:20 2 Jan 2023	Issue found	Strict transport security not enforced
16	2	16:52:20 2 Jan 2023	Issue found	Private IP addresses disclosed
15	2	16:52:20 2 Jan 2023	Issue found	Strict transport security not enforced
14	2	16:52:20 2 Jan 2023	Issue found	Email addresses disclosed
13	2	16:52:19 2 Jan 2023	Issue found	Email addresses disclosed
12	2	16:52:19 2 Jan 2023	Issue found	Cacheable HTTPS response
11	2	16:52:19 2 Jan 2023	Issue found	Email addresses disclosed
10	2	16:52:06 2 Jan 2023	Issue found	Cacheable HTTPS response
9	2	16:52:06 2 Jan 2023	Issue found	Cacheable HTTPS response
8	2	16:52:05 2 Jan 2023	Issue found	Cookie without HttpOnly flag set
7	2	16:52:05 2 Jan 2023	Issue found	Strict transport security not enforced
6	2	16:52:03 2 Jan 2023	Issue found	Cross-domain Referer leakage
5	2	16:52:02 2 Jan 2023	Issue found	Cacheable HTTPS response
4	2	16:52:02 2 Jan 2023	Issue found	Strict transport security not enforced
3	2	16:52:02 2 Jan 2023	Issue found	Cross-domain Referer leakage
2	2	16:52:02 2 Jan 2023	Issue found	Password field with autocomplete ena
1	2	16:52:00 2 Jan 2023	Issue found	Strict transport security not enforced

Figure 13: Burp Scanner Output after testing was completed

tested in manual approach could be automated using a scanner, this was proven to be true with some exception highlighted in Section 7 and the percentage of false positive reported by the application was less than 1% for the application in-scope while there were issues that were found to be falsely rated as per their rated severity vs actual severity which would depend on different applications as per the compliance requirements and the security posture of an organisation.

6.4 Time Complexity

Time complexity of the manual pentesting approach was indefinite as it required human intervention and effort which cannot be determined, while in the automated testing the time complexity was found to be dependent of the number of payloads in the dataset (n) resulting in the O(n) time complexity.

6.5 Discussion

The use of a popular web scanner, Burp Suite Professional, with a custom extension and Python programming language allowed for the automation of test cases for the serverless application. The automation process was observed to yield the same results as manual testing for in-scope applications, although only limited payloads were tested in the automated scan. One limitation of this research was that issues related to HTTP headers, which could also potentially be vulnerable, were not considered in the scope of the study.

An additional limitation of this research was that it did not consider multiple benchmarks such as SANS 25 or compliance standards when evaluating the security of the AWS Serverless application with WebSockets. This could potentially impact the validity and reliability of the findings, as these benchmarks and standards may have identified additional vulnerabilities or areas for improvement that were not considered in the study. It is important to consider multiple sources of information and standards when conducting security research in order to ensure a comprehensive and thorough evaluation of the system's security posture.

7 Conclusion and Future Work

This research suggests that the use of DAST for automated security testing was successful in identifying vulnerabilities in the in-scope applications. The DAST was able to replicate the results of manual testing, indicating that it is a reliable approach for identifying security issues. However, the limited payloads that were tested in the automated scan may have resulted in the DAST missing some vulnerabilities that could have been identified with a more comprehensive testing approach. Some limitations of this research are -

1. File Upload vulnerabilities were not tested and were out-of-scope as the application only accepted specific file types.
2. This research was limited to Akeero Serverless application but the approach can be used for other web-socket based application to perform automation.
3. Open Redirect vulnerabilities were not tested in this research to reduce execution time of the testing as manual testing did not yield any security issues.
4. DAST on HTTP headers was not done as HTTP requested belonged to third party services and were therefore out-of-scope.

As the research did not consider above mentioned issues in the limited scope, which could be a potential weakness in the security of the applications. Overall, the use of DAST for automated security testing appears to be a valuable approach, but it should be used in conjunction with other testing methods in order to achieve a more comprehensive understanding of the security posture of the applications. There is a need to research further on websocket applications security automation focusing on the following challenges -

1. Lack of a generalised framework to test all or any web-socket based applications with more than 90% coverage.
2. Expand the testing scope to automate more vulnerabilities from OWASP as well as other standards like SANS 25 covering more CWEs like File Upload vulnerabilities and Open Redirect.

References

Akujobi, J. C. (2021). *A model for measuring improvement of security in continuous integration pipelines: Metrics and four-axis maturity driven devsecops (mfam)*, Master's thesis, University of Twente.

Albahar, M., Alansari, D. and Jurcut, A. (2022). An empirical comparison of pen-testing tools for detecting web app vulnerabilities, *Electronics* **11**(19): 2991.

Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., Abad, C. L. and Iosup, A. (2021). Serverless applications: Why, when, and how?, *IEEE Software* **38**(1): 32–39.

Fette, I. and Melnikov, A. (2011). The websocket protocol, *Technical report*.

Garg, P. and Sengamedu, S. H. (2022). Synthesizing code quality rules from examples, *Proc. ACM Program. Lang.* **6**(OOPSLA2).
URL: *https://doi.org/10.1145/3563350*

Gojare, S., Joshi, R. and Gaigaware, D. (2015). Analysis and design of selenium webdriver automation testing framework, *Procedia Computer Science* **50**: 341–346.

Holmes, A. and Kellogg, M. (2006). Automating functional tests using selenium, *AGILE 2006 (AGILE'06)*, IEEE, pp. 6–pp.

Kuosmanen, H. (2016). Security testing of websockets.

Marin, E., Perino, D. and Di Pietro, R. (2022). Serverless computing: a security perspective, *Journal of Cloud Computing* **11**(1): 1–12.

Model, W. (2015). Waterfall model, *Luettavissa: http://www. waterfall-model. com/. Luettu* **3**.

Nguyen-Duc, A., Do, M. V., Hong, Q. L., Khac, K. N. and Quang, A. N. (2021). On the adoption of static analysis for software security assessment–a case study of an open-source e-government project, *computers & security* **111**: 102470.

OWASP (2021). Owasp top 10-2021.

Petukhov, A. and Kozlov, D. (2008). Detecting security vulnerabilities in web applications using dynamic analysis with penetration testing, *Computing Systems Lab, Department of Computer Science, Moscow State University* pp. 1–120.

Qasaimeh, M., Shamlawi, A. and Khairallah, T. (2018). Black box evaluation of web application scanners: Standards mapping approach, *Journal of Theoretical and Applied Information Technology* **96**(14): 4584–4596.

Rajapakse, R. N., Zahedi, M., Babar, M. A. and Shen, H. (2022). Challenges and solutions when adopting devsecops: A systematic review, *Information and Software Technology* **141**: 106700.

Saunders, M., Lewis, P. and Thornhill, A. (2007). Research methods, *Business Students 4th edition Pearson Education Limited, England* .

Tomas, N., Li, J. and Huang, H. (2019). An empirical study on culture, automation, measurement, and sharing of devsecops, *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pp. 1–8.

Wang, L., Li, M., Zhang, Y., Ristenpart, T. and Swift, M. (2018). Peeking behind the curtains of serverless platforms, *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 133–146.

Zampetti, F., Scalabrino, S., Oliveto, R., Canfora, G. and Di Penta, M. (2017). How open source projects use static code analysis tools in continuous integration pipelines, *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 334–344.