Integrating Advanced Security Features into Canarytokens for Enhanced Security in Kubernetes

University of Turku
Department of Computing, Faculty of Technology
Master's Degree Programme in Information and Communication Technology
Cybersecurity
June 2024
Tushar Avinash Wagh

Supervisors:
Dr. Antti Hakkala (University of Turku)
Dr. Tahir Mohammad (University of Turku)

UNIVERSITY OF TURKU
Department of Computing, Faculty of Technology

Tushar Avinash Wagh: Integrating Advanced Features into Canary Tokens for Enhanced Security in Kubernetes

Master's Degree Programme in Information and Communication Technology, 82 p., 9 app. p.
Cybersecurity
June 2024

Kubernetes has become the standard for orchestration and management of container-based applications, in an environment that is rapidly evolving due to cloud computing and containerization. Nevertheless, the complexity and distributed nature of Kubernetes introduces new security issues including identification of unauthorized access attempts and compromised credentials. This study examines the use of Canarytokens in Kubernetes environments as a seamless trickster-like defense measure designed to impersonate sensitive data.

This primarily aims at bringing out new advanced add-on security features that will enhance Canarytokens capabilities by addressing their current challenges such as non-integration with SIEM systems, manual response/remediation procedures, limited alert forwarding options, offline working etc.

It is a comprehensive study of the Canarytokens open-source project that was followed by designing, developing and implementing innovative solutions. These included integration of Canarytokens with SIEM systems to enhance visibility and correlation; automated response and remediation mechanisms triggered by Canary Token alerts; alternative methods for receiving alerts; and enabling offline working capabilities. To demonstrate their effectiveness, the proposed solutions are empirically tested and evaluated in a live Kubernetes environment.

The research findings contribute towards improving the security position of Kubernetes deployments by providing organizations with an advanced approach to securing their containerized environments through employment of these sophisticated features offered by the Canarytokens system. The developed modular add-ons offer practical and innovative solutions thereby allowing organizations to effectively use the Canarytokens according to their technology needs.

Keywords: Kubernetes, Canarytokens, Security, Threat Detection, IaC, SIEM Integration, Automated Response, Incident Logging, Offline Monitoring

Contents

List of Figures

List of acronyms

API Application Programming Interface

APT Advanced persistent threat

AWS Amazon Web Services

DLP Data Loss Prevention

DNS Domain Name System

HTTP/S Hypertext Transfer Protocol/Secure

IaC Infrastructure as Code

IDS Intrusion Detection Systems

SIEM Security Information and Event Management

SMTP Simple Mail Transfer Protocol

SSL/TLS Secure Sockets Layer/Transport Layer Security

UBA User Behavior Analytics

URL Uniform Resource Locator

WAF Web Application Firewall

1 Introduction

Not so long ago, in 2015, the Canarytokens open source project came into existence on GitHub, changing the scenario of advanced threat monitoring in the cyber security industry [1].

Similar to Canarytokens, in the computing domain, Kubernetes has exceptionally emerged as the industry standard for automating the deployment, scaling, and management of containerised applications [2]. However, the decentralised structure of Kubernetes is prone to complexity. Owing to its complexity and increasing popularity, the security issues related to its Kubernetes environment are also on the rise.

Any instances of unauthorised access or compromised credentials are one of the major concerns for today's Kubernetes infrastructure. Canarytokens, a security tool developed by Thinkst Applied Research, have gained popularity as a proactive approach to detecting and dealing with such kind of potential threats [1]. The purpose of these tokens is to mimic sensitive data, such as API keys, passwords, or other privileged credentials. These objects are used as a security measure. They serve as an stealthy active alert mechanism for the security teams when unauthorised individuals try to access or steal them.

1.1 Motivation

The motivation behind this research is driven by the potential of the Canarytokens open source project to serve advanced threat monitoring needs that can meet the security monitoring requirements of the Kubernetes infrastructure owing to its ever-growing complexity and constantly changing threat landscape. With the increasing adoption of containerisation and microservices, it becomes essential to prioritise strong security measures in order to safeguard sensitive data and workloads [3]. Canarytokens provide a potential solution for identifying and addressing credential misuse. However, their efficiency can be improved by incorporating specialised features that meet specifically to the needs of Kubernetes environments.

By integrating advanced features into Canarytokens, organisations can enhance their understanding of potential risks, streamline response procedures, and fortify the overall security of their Kubernetes deployments. The focus of this research is to investigate and create new features for improving the functionality and addressing the existing limitations of Canarytokens. The ultimate goal is to offer a more advanced and proactive strategy for safeguarding Kubernetes clusters using Canarytokens.

1.2 Problem statement

Although Canarytokens are increasingly being used as a security tool, their utilisation in Kubernetes systems frequently lacks advanced functionality and interoperability with pre-existing security frameworks. The current state of Canarytokens in Kubernetes presents four primary challenges:

First, the insufficient integration with SIEM systems. The integration of Canary Token alerts with SIEM systems can offer improved visibility, correlation, and auto-

mated response capabilities. However, this integration is often lacking or not fully developed [4].

Second, a manual response and remediation. When a Canary Token is activated, organisations often depend on human intervention to investigate and address the issue. This process can be slow and ineffective, particularly in extensive Kubernetes deployments [5].

Third, a limited alerts forwarding capabilities. The current version of Canarytokens can only send alerts through email and does not have integration capability with third-party monitoring solutions [6].

Fourth, the lack of offline alert forwarding functionality. Canary tokens require internet connectivity to activate and send alerts, leaving them ineffective in offline environment [7].

1.3 Research question

In order to tackle the challenges outlined in the problem statement, this study aims to provide a comprehensive response to the following overarching questions:

1. How to integrate the Canarytokens server with the SIEM systems that can forward the realtime alert logs to the same?

2. What are the mechanisms that can be implemented to automate the first line of defensive response to restrict the attacker from receiving an alert after triggering a Canary token?

3. What are the ways beyond an email for receiving such alerts after triggering Canary tokens?

4. How to make the Canary tokens work and receive alerts even when offline?

1.4 Research objective

The ultimate objective of this project is to tackle the research questions by enhancing the existing Canarytokens projects with advanced features that can overcome its limitations.

To achieve the desired outcome, the author strategies to address several key initiatives. The author will initiate the research by studying the workings and architecture of the Canarytokens open source project. Next, they will investigate and analyse potential solutions to address the limitations in the existing Canarytokens project, as discussed earlier in the problem statement. At the end, they will develop such technical solutions and demonstrate their effectiveness in a live Kubernetes environment.

1.5 Scope of the work

The scope of this research is focused on providing realistic and innovative solutions to address the limits of Canarytokens in Kubernetes environments, allowing organisations to use these modular add-ons efficiently based on their technology and needs.

The author aims to cover several key areas in this research. Beginning with a comprehensive study and evaluation of the existing Canarytokens open-source project, focusing on its architecture, working principles, and current limitations. This will provide a solid foundation for understanding the project and identifying areas for improvement.

In addition, the research will investigate the most effective ways to tackle the identified challenges. These include the need for better integration with Security Information and Event Management (SIEM) systems, improving response and re-

mediation processes, enhancing alert forwarding capabilities, and enabling offline functionality. Addressing these challenges is crucial for improving the overall effectiveness of the Canarytokens project.

Later, the design, development, and implementation of technical solutions will follow. This will include integration with SIEM systems for enhanced visibility, correlation, and automated response. Automated response and remediation mechanisms will be developed for alerts triggered by Canarytokens. Alternative methods for receiving alerts, such as webhooks or integrations with third-party monitoring solutions, will be explored. Additionally, enabling offline working capabilities for Canarytokens to function and receive alerts without internet access will be addressed.

Moreover, practical tests and evaluations of the proposed solutions and modular add-ons will be conducted in a real-world Kubernetes environment to demonstrate their effectiveness and verify their functionality. This step is essential for ensuring that the developed solutions work as intended in a practical setting.

Finally, the research findings, including the technical implementation details, will be documented comprehensively. This documentation will serve as a valuable resource for future research and development in this area.

1.6 Structure of the thesis

The thesis is structured as follows:

Chapter 2 provides an in-depth introduction to Canarytokens, as well as descriptions of their use cases, alternatives, and restrictions. This chapter also sheds insight on the distinctive characteristics of Canarytokens, which explains why they are better to other similar alternatives. Later, this chapter discusses the use of Canarytokens in securing Kubernetes environments and how they can strengthen the security further.

Chapter 3 illustrates the practical operation of the Canarytokens in comprehensive detail, explaining each step in the procedure.

Chapter 4 describes the architecture of the Canarytokens open source project (Docker version), focusing on the technical components of the Canarytokens.

Chapter 5 focuses on the development of advanced security features for Canarytokens to secure the Kubernetes environment.

Chapter 6 demonstrates the implementation of a technical solution that enables the practical use of the developed advanced security features for Canarytokens.

Chapter 7 presents the results of testing the developed solution in a live Kubernetes environment.

Chapter 8 revisits the research questions, discussing and answering them. This chapter also discusses the contribution of this research work to Kubernetes security and to the Canarytokens project as a whole, as well as its limitations.

Chapter 9 conveys a summary, concluding remarks, and prospective future research directions.

2 Literature review

2.1 Introduction to Canarytokens

Canarytokens or honeytokens also known as tripwires are identification markers created to sense and announce unauthorized access or misuse of sensitive data, systems or resources [8]. Essentially, these decoys mimic as real assets in an organization's infrastructure [9]. If a canary token is interacted with by the attacker, it will trigger an alert that can be immediately investigated and responded to by security teams [10].

Canarytokens were introduced based on the traditional analogy of canaries in a coal mine where miners would carry caged birds underground. If the bird stopped singing or died, it indicated the presence of toxic gases and resulted into immediate evacuation [11]. Thus, Canarytokens are like early warning systems that indicate potential security breaches or malicious activities [12].

Some forms of Canary tokens include fake API keys, database entries, URLs, email addresses and files which appear valuable but are monitored for any interaction. These type of tokens generate alerts when accessed thus informing security if there is potential breach. Canary tokens deploy easily and are widely adaptable which makes them versatile instruments for use in defense strategy as their strength lies in their adaptability being easy to deploy on many targets anywhere anytime [13].

2.2 Use Cases of Canarytokens

Canarytokens have a wide range of applications in the field of anti-persistence and threat hunting which can assist organizations in proactively identifying possible threats. Some use cases are highlighted below.

In the realm of threat hunting, Canarytokens serve as a powerful augmentation to traditional security controls. They can be strategically placed across an organization's assets to monitor for hidden dangers or stealthy activities. By embedding these tokens in various locations, security teams can identify indicators of compromise and pinpoint potential weaknesses within the infrastructure [14]. When an attacker interacts with a Canarytoken, it triggers an alert, thereby revealing the path the attacker might take. This proactive measure allows security professionals to map out and anticipate an attacker's movements, enhancing their ability to defend critical systems and data [5].

Canarytokens also play a crucial role in anti-persistence measures. Adversaries often employ persistence mechanisms to maintain access to compromised systems over extended periods. These mechanisms enable them to execute malicious actions even after initial detection and blocking efforts [15]. By incorporating Canarytokens into sensitive data, configurations, or system artifacts, organizations can disrupt these persistence techniques [16]. Any attempt by an adversary to interact with a Canarytoken triggers an alert, allowing security teams to implement timely countermeasures and remediation strategies. This not only helps in neutralizing the immediate threat but also hinders the adversary's ability to establish long-term control over the system [17].

Furthermore, Canarytokens are instrumental in detecting data exfiltration attempts. Data exfiltration, or the unauthorized transfer of data out of an organiza-

tion, poses a significant threat to sensitive information [18]. By placing Canarytokens within critical repositories, documents, or databases, companies can monitor for unauthorized access attempts. When an adversary tries to exfiltrate data containing a Canarytoken, an alert is triggered, notifying security teams of the breach. This immediate alert allows for rapid response and investigation, helping to prevent data loss and mitigate the impact of the breach [14].

Canarytokens are equally effective in monitoring for insider threats. Insider threats, which originate from within the organization, can be challenging to detect and mitigate. By embedding Canarytokens within sensitive data, systems, or applications accessible to insiders, organizations can identify malicious or unintentional actions that may lead to data exposure or system compromise [14]. These tokens provide a silent yet effective monitoring mechanism that alerts security teams to suspicious activities, enabling them to take corrective actions before significant damage occurs [19].

2.3 Canarytokens alternatives and their limitations

Aside from Canarytokens, there are other security measures and techniques that exist with the intention of detecting and mitigating threats. Nevertheless, these alternatives may have some disadvantages or shortcomings compared to the use of Canarytokens [20]:

(i) **Honeypots:** Honeypots are decoy systems or resources designed to attract and deceive attackers, allowing security teams to study their techniques and gather intelligence [17]. Although honeypots may help identify threats and understand them, they may involve significant resource requirements with high

maintenance cost and limited detection abilities only within the confines of particular systems or environments [21].

(ii) **Intrusion Detection Systems (IDS):** In addition, the IDS solutions watch system activities as well as network traffic in search of malicious activities such as behaviors related to certain known attacks [16]. Moreover, these signatures may sometimes fail to detect advanced or new threats since the systems are based on predefined rules. Also, the alerting capability might be overactive resulting into alert fatigue and causing important situations go unnoticed by the analysts [21].

(iii) **Data Loss Prevention (DLP):** This solution works by watching data flows and applying predefined policies that aim at preventing unauthorized disclosure or exfiltration of sensitive information [15]. DLP might not necessarily detect advanced persistence techniques or insider threats making it ineffective in some cases. Moreover, deployment and maintenance costs are high due to resource usage [19].

(iv) **User Behavior Analytics (UBA):** Behavioral analytics is one of the many types of security solutions that use UBA tools where user's behavior is analyzed in order detect anomalies which could point towards any breach or misuse by detecting unusual patterns [17]. However, UBA relies heavily on accurate baseline profiles which may introduce false positive results, especially in dynamic or complex environments [18].

Unlike the other alternative methods, Canarytokens help organizations uncover different types of targeted threats such as APTs, insider attacks as well as data extraction attempts [16]. Besides, depending on various assets and environments where they are situated, Canarytokens represent a comprehensive and cost-effective mitigation strategy for companies [20].

2.4 Why Canarytokens are a better option?

Canarytokens have several benefits over other security measures, which makes them the preferred choice for organizations that want to improve their threat detection and mitigation capabilities [20].

One of the primary benefits of Canarytokens is their proactive threat detection capability. Unlike traditional security measures that often rely on reactive approaches or pre-set guidelines, Canarytokens continuously monitor for unauthorized access or misuse. This proactive methodology allows organizations to stay ahead of emerging threats and respond swiftly to potential incidents. By embedding Canarytokens within various assets, companies can identify and address security breaches at the earliest stages, significantly reducing the potential damage caused by such incidents [17].

Canarytokens also offer high accuracy and low false positives, a critical advantage over many other security tools. They only generate alerts when explicitly interacted with or accessed, minimizing the number of false positives that can overwhelm security teams. This precision ensures that the alerts generated by Canarytokens are genuine and require immediate attention, enabling security personnel to focus their efforts on real threats rather than wasting time on false alarms [19]. This high accuracy improves the overall efficiency of an organization's security operations.

The versatility and scalability of Canarytokens make them suitable for any organization, regardless of its size or complexity. Canarytokens can be easily deployed across various assets, environments, and systems, including databases, applications, networks, and even physical documents or devices [16]. This flexibility allows organizations to integrate Canarytokens into their existing infrastructure seamlessly, providing comprehensive coverage without requiring significant changes to their op-

erational setup.

Another significant advantage of Canarytokens is their stealthy nature, which makes them difficult for adversaries to detect. Canarytokens are designed to blend seamlessly into their environment, making them indistinguishable from real assets. This stealthiness enhances their effectiveness in detecting and monitoring malicious activities without alerting the attackers. By remaining undetected, Canarytokens can provide continuous surveillance and early warning of potential security breaches [22].

Cost-effectiveness is another key benefit of Canarytokens. Deploying Canarytokens requires fewer resources compared to alternative security solutions such as honeypots or extensive monitoring systems. This cost-efficiency makes Canarytokens particularly suitable for organizations operating on limited budgets or with constrained resources [23]. Despite their lower resource requirements, Canarytokens still deliver robust security benefits, making them an attractive option for many organizations.

Lastly, Canarytokens are complementary to existing security measures, fitting effortlessly into an organization's current security architecture without causing disruptions. They enhance and augment other security controls already in place, providing an extra layer of defense [24]. By working alongside traditional security measures, Canarytokens help to create a more comprehensive and resilient security posture, ensuring that organizations are better equipped to detect and mitigate threats.

2.5 Kubernetes and its security issues

Kubernetes, an open-source container orchestration platform, has revolutionized the way applications are deployed, managed, and scaled in cloud environments [25]. Despite its widespread adoption and advanced capabilities, Kubernetes is not immune to security challenges. One primary concern is its complex architecture, which includes various components such as the API server, etcd, scheduler, and controller manager [25]. Each of these components, if not properly secured, can become a potential entry point for attackers. For instance, the API server, which is the central management entity of Kubernetes, needs stringent access controls to prevent unauthorized access and potential data breaches [26].

Another significant security issue in Kubernetes is the management of secrets. Kubernetes uses secrets to store sensitive information like passwords, tokens, and keys. However, if these secrets are not adequately protected, they can be easily accessed by malicious actors [27]. The default storage of secrets in etcd in an unencrypted format further exacerbates this problem. Additionally, the RBAC (Role-Based Access Control) mechanism, while powerful, can be misconfigured, leading to excessive permissions being granted to users or service accounts, which can be exploited to perform unauthorized actions within the cluster [28].

Network security within a Kubernetes cluster also presents challenges. Kubernetes relies on networking plugins to manage communication between pods, services, and nodes. However, if these plugins are not configured correctly, they can lead to vulnerabilities such as man-in-the-middle attacks or unauthorized network access [29]. Moreover, the use of containers themselves introduces security concerns, such as vulnerabilities in the container images, runtime security risks, and the potential for container escape, where an attacker could gain access to the host system from within a container [30].

2.6 Canarytokens efficacy in addressing Kubernetes security problems

Canarytokens offer a proactive approach to security by acting as tripwires that alert administrators to unauthorized or suspicious activities within a system [31]. In the context of Kubernetes, Canarytokens can be strategically integrated to enhance security and address the aforementioned issues. By deploying Canarytokens within a Kubernetes cluster, administrators can gain early warnings of potential security breaches, allowing for rapid response and mitigation [32].

One effective application of Canarytokens in Kubernetes is the protection of sensitive information such as secrets. Canarytokens can be embedded within Kubernetes secrets, and any unauthorized attempt to access these secrets would trigger an alert. This early detection mechanism helps administrators to quickly identify and respond to potential breaches before they can cause significant damage [27]. Furthermore, Canarytokens can be placed in configuration files, container images, and network traffic, providing comprehensive coverage and monitoring of the entire Kubernetes environment [33].

In terms of network security, Canarytokens can be used to monitor network traffic and detect unauthorized access attempts. By placing tokens in critical network paths or within specific pods and services, any attempt to access these components without proper authorization will trigger an alert. This helps to identify and block unauthorized network activities, thereby preventing potential attacks such as man-in-the-middle or lateral movement within the cluster [30].

Additionally, Canarytokens can be integrated into the Kubernetes API server and other critical components to detect and alert on suspicious activities [34]. For instance, a Canarytoken can be configured to trigger when there are unusual API

requests or attempts to escalate privileges. This provides an additional layer of security by monitoring and protecting the core components of the Kubernetes architecture [29].

Overall, Canarytokens offer a versatile and effective solution for enhancing Kubernetes security. By providing early detection of unauthorized activities, they enable administrators to respond swiftly to potential threats, thus mitigating risks and strengthening the overall security posture of the Kubernetes environment [29].

3 Working of Canarytokens

This section outlines each phase in detail, emphasizing practical working of the Canarytokens. Canary tokens deployment follows a structured process consisting of five primary phases:

1. Token Generation
2. Token Customization (Optional)
3. Strategic placement of a token
4. Monitoring
5. Receiving alerts

3.1 Token Generation

Token generation is the initial phase wherein a canary token is created using a web application. This process can be executed either through a private server or an official Canarytokens service. For demonstration, the tokens are generated using a private server accessible at `https://canary.fi/generate`, hosted with the IP address `21.159.12.101` and the monitoring domain `alert.canary.fi`. Alternatively, Thinkst Canary's official service at `https://canarytokens.org/generate` may also be utilized as needed.

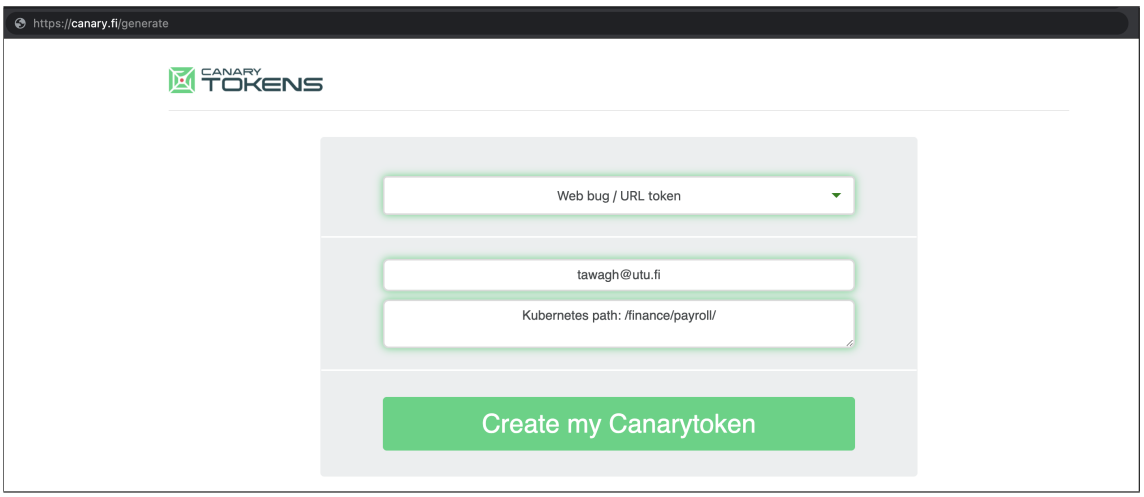


Figure 3.1: Canarytokens web applications front page

Upon visiting `https://canary.fi/generate`, the user is greeted with a web interface as shown in the Figure 3.1, offering various types of tokens such as URL tokens, DNS tokens, Microsoft Office files, kubeconfig, Azure tokens, AWS keys, MySQL dumps, etc. For illustration, a URL token was chosen. The user must provide an email address to receive alerts and a description to identify the token's context upon triggering. Once these details are entered, the URL token is generated successfully and can be copied for deployment.

Example URL token:

```
https://alert.canary.fi/static/to8adxao41m1rhxubo0ivpzs8/index.html
```

3.2 Token Customization (Optional)

While tokens can be used forthrightly after they have been produced, their customization can make them more attractive to possible attackers. For instance, modifying the URL token to look more legitimate increases its possibilities of being accessed. Such modification can be done as:

```
https://alert.canary.fi/admin/to8adxao41m1rhxubo0ivpzs8/payment.js
```

It is essential to keep the domain and hash (`gopz8puzx4uwujyo1x7gnztbp`) genuine and limit customization to the path and filename for maintaining token's functionality.

3.3 Strategic Placement of a Token

To deploy effective Canary tokens, they must be placed strategically where unauthorized visitors may access them. These locations should be selected with care so that they are tempting to intruders while at the same time being unnoticeable enough not to unintentionally trigger by authorized users [35]. For example, putting a token in an Apache web server directory makes it highly detectable without impinging on normal operations:

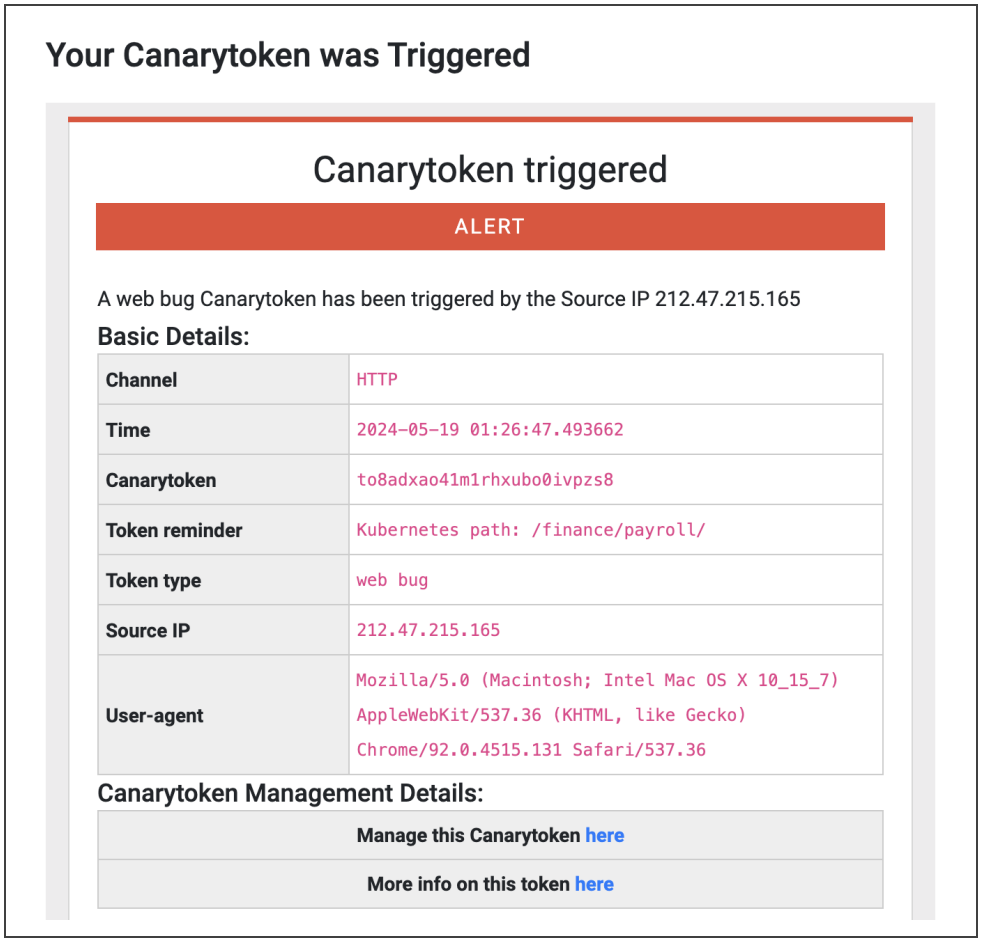
Placement path: `/usr/src/apache2/finance/payroll/`

3.4 Monitoring

The canary token server serves dual purposes: it provides a web interface for generating tokens and monitors these tokens for triggered alerts. The server's preconfigured monitoring mechanism continuously tracks the activity of the generated tokens from the moment of their creation, as it is crucial for immediate identification and response towards any attempt for unauthorized entry into our systems.

3.5 Receiving Alerts

When an attacker accesses a canary token, an alert is triggered and sent to the user's email. This alert includes detailed information about the event, such as the



Your Canarytoken was Triggered

Canarytoken triggered

ALERT

A web bug Canarytoken has been triggered by the Source IP 212.47.215.165

Basic Details:

Channel	HTTP
Time	2024-05-19 01:26:47.493662
Canarytoken	to8adxao41m1rhxubo0ivpzs8
Token reminder	Kubernetes path: /finance/payroll/
Token type	web bug
Source IP	212.47.215.165
User-agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36

Canarytoken Management Details:

Manage this Canarytoken here
More info on this token here

Figure 3.2: Email alert for a triggered Canary token

attacker's IP address, the time of access, and browser details. An example of such an alert is depicted in figure 3.2.

This email alert provides comprehensive information necessary for further investigation and mitigation of the unauthorized access attempt.

4 Architecture of Canarytokens

To comprehend the incorporation of advanced features into this project, it is imperative first to understand the architecture of Canarytokens. The Canarytokens open-source project offers two versions of its releases available on their github page: the standard version and the Docker version. The Docker version is the recommended [1], and it will be the focus of our discussion.

The architecture of Canarytokens can be categorized into three main components:

1. Nginx Web Server
2. Redis & Data Storage
3. Configuration files

Each of these components will be discussed in greater detail as follows:

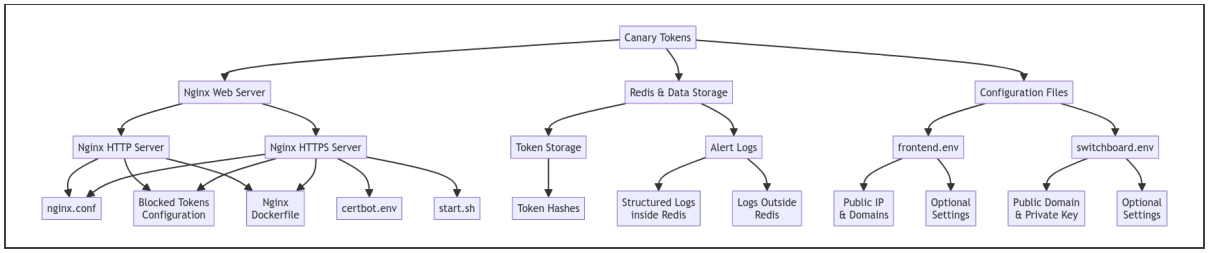


Figure 4.1: Architecture of the Canarytokens open-source project

4.1 Nginx Web Server

Nginx is a high-performance, modular web server capable of handling various protocols and tasks, including proxying, caching, serving static files, load balancing, SSL, and streaming [36].

The Canarytokens project separates the HTTP and HTTPS configurations into different directories for better organization. The HTTP configuration resides in the `nginx` directory, while the HTTPS configuration is in the `certbot-nginx` directory.

4.1.1 Nginx HTTP Server

The Nginx HTTP server configuration, located in the `nginx` directory, consists of three key files: the Nginx configuration file (`nginx.conf`), a Docker configuration file (`Dockerfile`), and a configuration file for blocked Canary tokens (`blocked_tokens.conf`) in the `conf.d` directory.

➢ **Nginx Configuration File (nginx.conf)**

The `nginx.conf` file exhibits several critical server features:

♦ Efficient file handling and compression:

```
gzip on;
gzip_HTTP_version 1.0;
gzip_proxied any;
gzip_min_length 500;
gzip_disable "MSIE [1-6]\.";
gzip_types text/plain text/xml text/css text/comma-separated
   -values text/javascript application/x-javascript
   application/atom+xml;
```

This configuration enhances performance by utilizing efficient file handling and compression while ensuring that files are served with the correct MIME types, thereby improving client compatibility and proper file handling [36].

- Proxying connections to application servers:

```
location = / {
proxy_pass HTTP://frontend:8082/;
proxy_redirect off;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Host $server_name;
```

This snippet configures Nginx as a reverse proxy, forwarding requests to an internal service running on `http://frontend:8082/` [37].

- Adding basic authentication:

```
auth_basic "Basic Auth Restricted Canrytokens";
auth_basic_user_file /etc/nginx/.htpasswd;
```

This configuration adds a basic authentication layer, securing access to the public-facing Canary token site. User credentials are stored in the `.htpasswd` file [38].

➢ **Blocked tokens configuration**

```
# location ^~ /token_path_prefix_to_block {
# access_log off;
# log_not_found off;
```

```
# return 444;
# }
```

This configuration snippet provides the capability to block specific tokens, extending the token management functionality. The code is commented out by default, as there are no pre-configured blacklisted tokens [37].

➢ **Nginx Dockerfile**

```
FROM nginx
MAINTAINER Marco Slaviero <marco@thinkst.com>
LABEL Description="This image provides the HTTP proxy for
   Canarytokens"
Vendor="Thinkst Applied Research" Version="1.3"
COPY nginx.conf /etc/nginx/nginx.conf
COPY .htpasswd /etc/nginx/.htpasswd
COPY conf.d/blocked_tokens.conf /etc/nginx/conf.d/blocked_tokens
   .conf
```

This `Dockerfile` creates a Docker image based on the Nginx web server, specifically tailored to serve as an HTTP proxy for Canarytokens, which are tools for creating and managing various types of honeypot tokens [37].

4.1.2 Nginx HTTPS Server

In addition to the features offered by the HTTP server, the HTTPS server supports encrypting traffic using SSL/TLS. Canarytokens utilize SSL certificates from "Let's Encrypt" (a nonprofit organization that provides TLS certificates to the requested websites) via a tool called `Certbot` [39].

`certbot.env` configuration file:

```
MY_DOMAIN_NAME=alert.canary.ee
EMAIL_ADDRESS=tushar.wagh@utu.fi
```

To request a new SSL certificate, the project includes a `certbot.env` configuration file in the root directory, which requires configuration with the domain name and an email address.

`start.sh` shell script:

```
echo "Starting nginx and lets encrypt setup using"
_args=""
_server_names=""
if [ "x${MY_DOMAIN_NAME}" != "x" ]; then
echo "Domain : $MY_DOMAIN_NAME"
_args=" -d ${MY_DOMAIN_NAME} -d www.${MY_DOMAIN_NAME}"
_server_names="${MY_DOMAIN_NAME} www.${MY_DOMAIN_NAME} "
fi
if [ "x${MY_DOMAIN_NAMES}" != "x" ]; then
echo "Domains : $MY_DOMAIN_NAMES"
for domain in $MY_DOMAIN_NAMES; do
_args="${_args} -d ${domain}"
_server_names="${_server_names} ${domain}"
done
fi
echo "Email : $EMAIL_ADDRESS"
sed -i "s/___server_names___/$_server_names/g" /etc/nginx/nginx.conf
sleep 5
```

```
nginx
sleep 5
certbot --nginx ${_args} --text --agree-tos --no-self-upgrade --keep
   --no-redirect --email $EMAIL_ADDRESS -v -n
nginx -s stop
sleep 3
nginx -g "daemon off;"
```

This `start.sh` script in the `certbot-nginx` directory uses these details to request a new certificate.

4.2 Redis & Data Storage

Redis (Remote Dictionary Server) is a freely available storage system that is used for storing and retrieving data. Even though it can be used as a cache, or as an in-memory key-value store, or as a message broker that can be fault tolerant [40]. As a result, Canarytokens Project uses Redis to store and manage tokens.

Once generated, these tokens and logs are maintained in the `dump.rdb` file within the data directory. Each new token has its own hash called `canarydrop:unique_canary_token_hash` which enables distinction of various tokens prompting alerts. Such hashes can be retrieved by using the command `keys canarydrop:*`, where it displays all created canaries together with their hashes [41].

An alert in form of logs or email will be stored when a canary token triggers. Detailed information on a particular token within Redis container can be gathered using the `hgetall canarydrop:unique_canary_token_hash` command. While there are structured logs inside the Redis container, there exist logs outside the container at `/data/appendonlydir/` saved in `appendonly.aof.1.base.rdbfile`. The logs inside the Redis container are structured, while those outside are restricted to sudo

privileges.

Example of Redis container alert logs following a canary token trigger:


```
"triggered_list"
"{\"1515986709.981106\": {\"src_ip\": \"118.37.205.148\", \"geo_info
   \": {\"loc\": \"59.4370,24.7535\", \"org\": \"AS3327 CITIC
   Telecom CPC Netherlands B.V.\", \"city\": \"Tallinn\", \"country
   \": \"EE\", \"region\": \"Harjumaa\", \"ip\": \"212.47.215.165\",
    \"timezone\": \"Europe/Tallinn\", \"postal\": \"10111\"}, \"
   is_tor_relay\": false, \"input_channel\": \"HTTP\", \"useragent
   \": \"Mozilla/5.0 (Windows NT 10.0; rv:123.0) Gecko/20100101
   Firefox/123.0\", \"request_headers\": {\"Host\": \"ganadmin.
   hostix.ee\", \"X-Real-Ip\": \"212.47.215.165\", \"X-Forwarded-For
   \": \"212.47.215.165\", \"X-Forwarded-Host\": \"_\", \"Connection
   \": \"close\", \"User-Agent\": \"Mozilla/5.0 (Windows NT 10.0; rv
   :123.0) Gecko/20100101 Firefox/123.0\", \"Accept\": \"text/html,
   application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
   ,*/*;q=0.8\", \"Accept-Language\": \"en-US,en;q=0.5\", \"Accept-
   Encoding\": \"gzip, deflate\", \"DNT\": \"1\", \"Upgrade-Insecure
   -Requests\": \"1\"}, \"request_args\": {}}}"
```


4.3 Configuration Files

Canarytokens project distributes two crucial configuration files: `frontend.env` and `switchboard.env`. These files define essential parameters for handling token requests, monitoring, and ensuring secure communication.

4.3.1 frontend.env configuration file

The `frontend.env` file contains the configuration settings for handling requests coming after a token is triggered. In the field `CANARY_PUBLIC_IP` the IP address needs to define where the Canarytokens web application will be running while facing the internet. The field `CANARY_DOMAINS` has to be specified with one or more domain addresses where we want to get the alerts after triggering tokens. Whereas, inside a `CANARY_NXDOMAINS` field, at least one non-existing domain should be specified in order to get alerts, particularly for PDF tokens. There are also a couple of settings that are optional and can be defined to increase the functionality offered by the Canarytokens project.

Contents from frontend.env file:

```
CANARY_PUBLIC_IP=51.159.12.201
CANARY_DOMAINS=alert.canary.fi
CANARY_NXDOMAINS=alert.canary.fi
LOG_FILE=frontend.log
```

4.3.2 switchboard.env configuration file

The `switchboard.env` configuration file focuses on comprehensive settings for configuring a canary deployment system, which contains a `CANARY_PUBLIC_DOMAIN` field, which should be specified with the same domain as `CANARY_DOMAINS` from the `frontend.env` file. The other important field to specify is `CANARY_WG_PRIVATE_KEY_SEED`, which is a crucial component for generating private keys and ensuring secure communication via WireGuard. Such a private seed key for wireguard can be generated by running the command `dd bs=32 count=1 if=/dev/urandom2>/dev/null | base64`.

There are also some of the optional settings present, such as network settings, alert configurations, third-party integrations, and logging mechanisms. Such optional settings are key considerations for integrating canary deployment with existing cloud infrastructure and services, such as Mailgun, Sendgrid, SMTP, and Sentry [37].

Contents from switchboard.env file:

```
CANARY_PUBLIC_DOMAIN=ganadmin.hostix.ee
CANARY_WG_PRIVATE_KEY_SEED=
   xTVexTkZgU5LZJtsDXPARYDnuzrjnEXsz6DFmNSEY74=
LOG_FILE=switchboard.log


# Optional Settings
#CANARY_CHANNEL_DNS_IP=
#CANARY_CHANNEL_DNS_PORT=
#CANARY_CHANNEL_HTTP_PORT=


# Logging Settings
#CANARY_SWITCHBOARD_LOG_SIZE=
#CANARY_SWITCHBOARD_LOG_COUNT=
#ERROR_LOG_WEBHOOK=
```

5 Development

This chapter will showcase the development of the aforementioned three advanced security features for the Canarytokens open source project:

1. **Feature 1**: Capturing realtime incident logs from Redis
2. **Feature 2**: Incident Response Using IP-Based Blocking in WAF
3. **Feature 3**: Offline tokens monitoring and incident logging

5.1 Feature 1: Capturing realtime incidents from Redis

The realtime logs from Redis can be listened to on default port 6379. A Python program is needed to be developed to listen to this port, which will capture and store these logs in a separate log file locally. It should also continuously monitor and process incidents, ensuring that new events are logged efficiently. Moreover, the repeatedly occurring duplicate logs should also be discarded.

5.1.1 Python Program Overview

The following program named `ct2rds.py`, will listen to Redis log port 6379 while capturing and storing these logs in a separate `incident.log` file inside the new

`/logs` folder. It will also continuously monitor and process incidents, ensuring that new events are logged efficiently.

The program will integrate various Python modules, including `os`, `redis`, `time`, `json`, `datetime`, and `logging`, to retrieve incident data, process it, and log it into a file. The operational logic, including timestamp management and data processing from Redis, is examined to understand its functionality and effectiveness.

The program comprises two main components:

(i) **Setup:** Logging configuration and function definitions.

(ii) **Main Logic:** The main loop that manages the retrieval and processing of incident data from Redis.

5.1.2 Working and Breakdown of the program

This program combines various Python modules, such as `os`, `redis`, `time`, `json`, `datetime`, and `logging`. It retrieves incident data using the same modules, then processes it before storing it in a file. In order to apprehend its functionality and effectiveness, this study examines operational logic, including timestamp management and processing of Redis' data.

```
logging.basicConfig(filename='logs/incident.log', level=logging.INFO)
```

As showcased in the above part, when the program starts up, logging is initialized. `logs/incident.log` would record any incident logs at an `INFO` level. This configuration ensures that all informational messages, including incident logs, are monitored and processed more efficiently.

Key Sections

a. Helper Functions

Within the program, two helper functions were introduced to manage timestamps, as the timestamp obtained from the logs is in Unix format. These helper functions are:

- Current Timestamp: Returns the current Unix timestamp.

```
def current_timestamp():
    return int(time.time())
```

- Convert Timestamp to Datetime: Converts a Unix timestamp to a human-readable datetime string.

```
def convert_timestamp_to_datetime(timestamp):
    return datetime.datetime.fromtimestamp(timestamp).strftime
        ('%Y-%m-%d %H:%M:%S')
```

b. Incident Processing Function

The core function `process_incidents` handles the retrieval and logging of incidents:

```
def process_incidents(redis_client, timestamp, processed_timestamps):
    token_keys = redis_client.keys('canarydrop:*')

    for token_key in token_keys:
        triggered_list_key = f"{token_key} triggered_list"
        triggered_list = redis_client.hget(token_key, "triggered_list
            ")
```

```
    if triggered_list:
        triggered_list = json.loads(triggered_list)
        for incident_timestamp, incident_data in triggered_list.
            items():
            incident_datetime_str = convert_timestamp_to_datetime(
                float(incident_timestamp))

            if incident_timestamp not in processed_timestamps and
                float(incident_timestamp) > timestamp:
                log_entry = f"{incident_datetime_str}\n{json.dumps(
                    incident_data, indent=4)}\n\n"
                logging.info(log_entry)
                print(f"Logged incident:\n{log_entry}")
                processed_timestamps.add(incident_timestamp)
```

This function performs the following steps:

(i) Retrieve Tokens: This fetches all keys matching the pattern `canarydrop:*`.

(ii) Process Triggered List: For each token retrieve and process its `triggered_list`.

(iii) Log Incidents: Convert incident timestamps to readable format and log incidents if they are new and unprocessed.

c. Main Functions

The main function orchestrates the program's operation:

```
def main():
    start_timestamp = current_timestamp()
    processed_timestamps = set()
```

```
redis_host = os.environ.get("redis_host")
redis_port = os.environ.get("redis_port", 6739)
redis_client = redis.StrictRedis(host=redis_host, port=redis_port
    , decode_responses=True)


try:
    while True:
        process_incidents(redis_client, start_timestamp,
            processed_timestamps)
        time.sleep(20)
except KeyboardInterrupt:
    print("program terminated by user.")
```

The crucial elements include:

(i) Initialization: Captures the current timestamp and initializes a set for processed timestamps.

(ii) Redis Connection: Connects to the Redis server using environment variables at port 6739.

(iii) Continuous Monitoring: Enters an infinite loop where incidents are processed every 20 seconds.

5.1.3 Operational Flow

This section i.e. operational flow of a program efficiently describes process and log incident data. It covers initialization steps, such as setting up logging and connecting to Redis, the continuous loop for retrieving and processing incidents, and the detailed logging of each new incident.

Initialization:

- The program initializes logging and sets up helper functions.
- The main function records the start timestamp and connects to Redis.

Incident Processing Loop:

- The program enters a continuous loop, calling `process_incidents` every 10 seconds.
- It retrieves incident data from Redis, processes unlogged incidents, and updates the log file.

Logging:

- Each new incident is logged with a timestamp and detailed JSON data.
- Logged incidents are marked to avoid reprocessing.

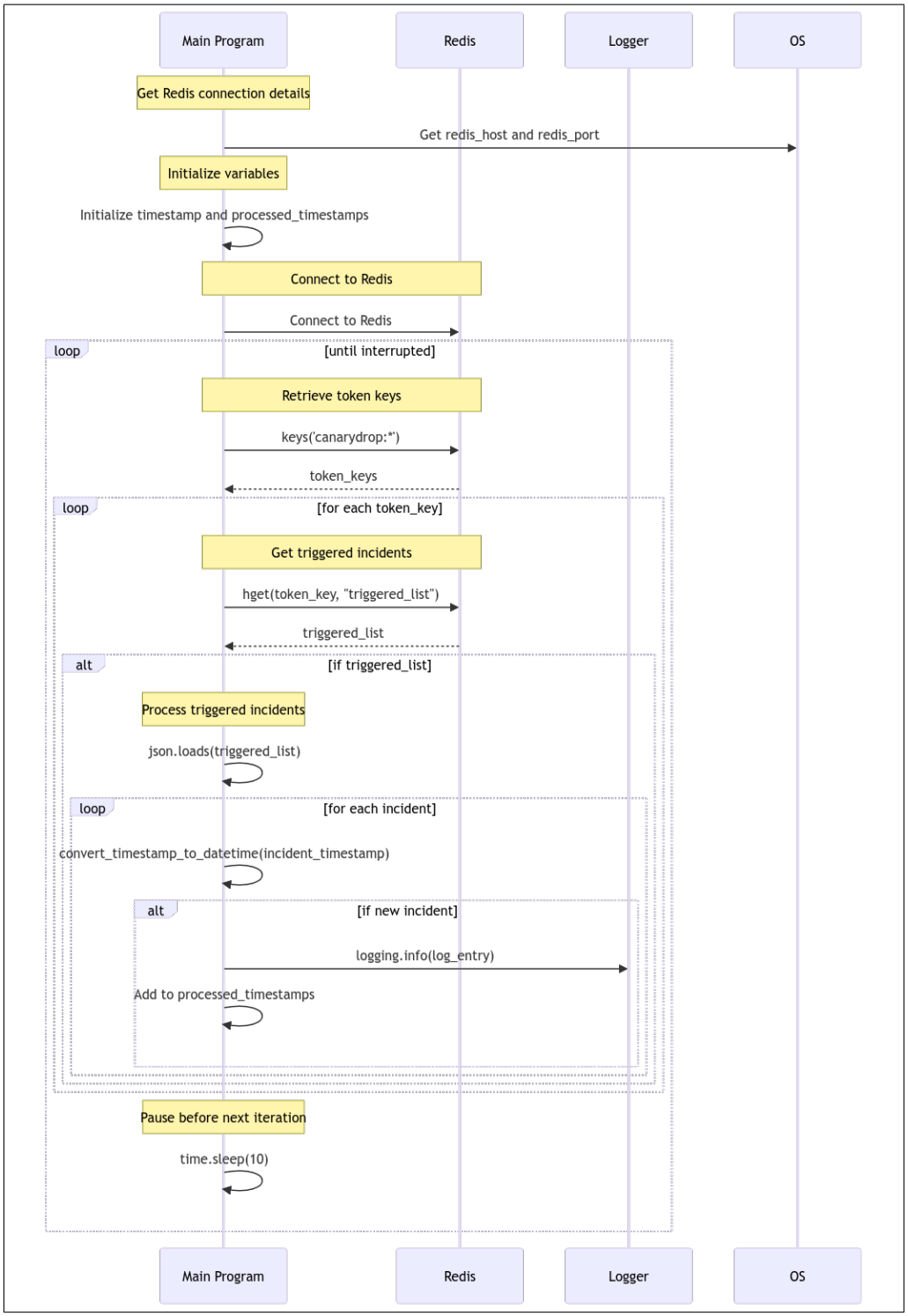


Figure 5.1: Operational flow diagram for the Feature 1 python program

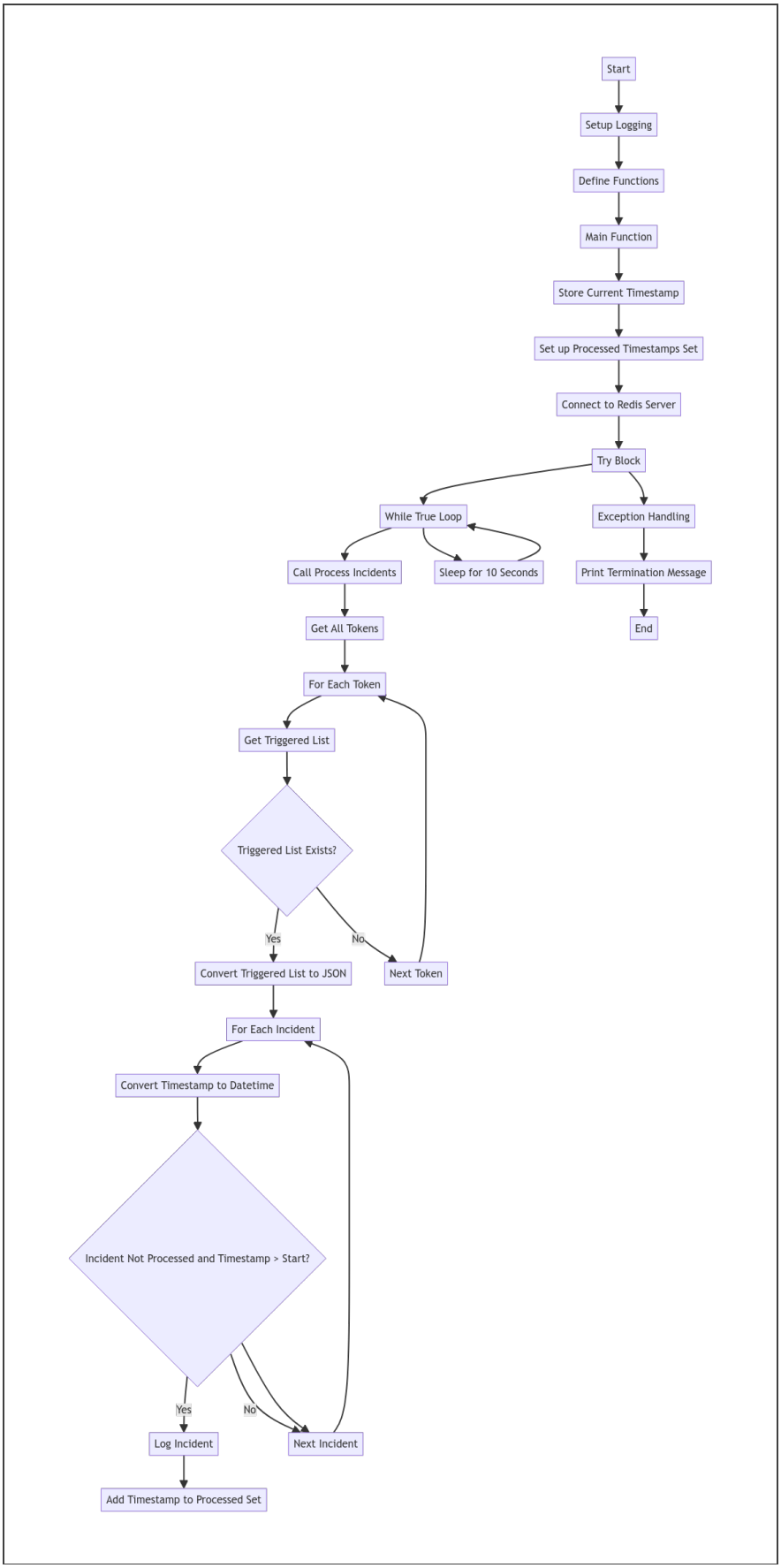


Figure 5.2: Flowchart for the Feature 1 python program

5.2 Feature 2: Realtime incident response in WAF

In the presence of rapidly evolving threats, relying on traditional manual responses is often too slow. IaC can be used to automate incident response, including updating firewall rules, thus significantly reducing the time it takes to mitigate an attack. In order to achieve this, a program is required that can parse log files for malicious IP addresses and subsequently update a Terraform configuration to block these IPs by using WAF's API. In this manner, via automation of the process of adding malicious IPs to a blocklist, the system must have an inbuilt mechanism that prompts it to respond promptly to any threats enhancing the security landscape.

5.2.1 Python Program Overview

This `waf.py` Python program keeps monitoring a certain log file for new IP addresses and then modifies a Terraform configuration file according to them while executing various Bash commands that lead to the application of these changes. Once the tokens are triggered and written into the `incident.log` file itself, we need to parse this `incident.log` file so as to get any regex-based IP addresses associated with such triggers, which should then be blocked at the WAF level by updating dynamic firewall policies based on detected incidents using WAF's API calls only. Thereby, any newly detected IP address will be added to the infrastructure as per Terraform.

The Python script performs the following tasks:

(i) Monitors a log file for IP addresses.

(ii) Updates a Terraform configuration file with the newly detected IP addresses.

(iii) Executes a series of Bash commands to apply the updated Terraform configuration.

Following are the Key Components of this Python script:

- **Log File Path:** The script monitors a specific log file for new IP addresses.
- **Terraform Configuration Path:** The script updates a specific Terraform configuration file (`main.tf`).
- **Regex Pattern:** A regular expression pattern used to extract IP addresses from the log file.
- **Bash Commands:** A list of commands to initialize, plan, and apply Terraform configurations.

5.2.2 Working and Breakdown of the program

Key Sections

a. Defining Paths and Patterns

```
\url{log_file_path} = \url{/path/to/incident/log/file/incident.log}
\url{main_tf_path} = \url{/path/to/terraform/main.tf}
\url{ip_pattern} = \url{r'"ip":\s*"([^"]*)"'}
```

The paths to the log file and Terraform configuration file are defined, along with the regex pattern for extracting IP addresses.

b. Extracting IP Addresses from Log File

```
def extract_ips_from_log():
ips = set()
with open(log_file_path) as file:
    for line in file:
        matches = re.findall(ip_pattern, line)
```

```
            ips.update(matches)
    return ips
```

The function `extract_ips_from_log()` reads the log file line by line, applies the regex pattern to find IP addresses, and stores them in a set to ensure uniqueness.

c. Updating Terraform file

```
def update_main_tf(ips):
with open(main_tf_path, 'r') as file:
    lines = file.readlines()


for i, line in enumerate(lines):
    if 'ips' in line:
        indent = len(line) - len(line.lstrip())
        ips_str = ', '.join(f'"{ip}"' for ip in ips)
        new_line = f'{" " * indent}"ips": [{ips_str}]\n'
        lines[i] = new_line
        break


with open(main_tf_path, 'w') as file:
    file.writelines(lines)
```

This function reads the Terraform configuration file, identifies the line containing IP addresses, updates it with the new IPs, and writes the changes back to the file.

d. Main Loop

```
def main():
previous_ips = set()
```

```
while True:
    ips = extract_ips_from_log()
    new_ips = ips - previous_ips
    if new_ips:
        update_main_tf(ips)
        execute_bash_commands()
        previous_ips = ips
    time.sleep(15)
```

This is the main function that orchestrates the entire workflow of automating responses after threat detection. This main loop runs indefinitely, checking for new IPs every 15 seconds. If new IPs are detected, it updates the Terraform configuration `main.tf` file and applies the changes.

5.2.3 Operational Flow

This section explains the initialization of necessary modules and commands, the looping mechanism for extracting and checking IP addresses, and the execution of bash commands to update the firewall using Infrastructure as Code (IaC) with Terraform. This process ensures efficient and automated management of firewall rules.

Initialization:

- Import the required modules.
- Define paths and patterns.
- Define Bash commands.
- Initialize Previous IPs Set.

IP address Processing Loop:

- The program enters a continuous loop, calling `extract_ips_from_log` every 15 seconds.
- It retrieves IP address from `incident.log` file, checks if the given IP is already present inside `main.tf` Terraform file, if yes then discards the duplicate entries, if no then updates to the `main.tf` Terraform script.

Command Execution:

- If the program detects adding of new IP's to the `main.tf` Terraform script, it starts executing the series of bash commands.
- This commands are designed to call the WAF's API, connect them by validating the credentials and update the firewall blacklist using IaC containing inside the `main.tf` Terraform script.

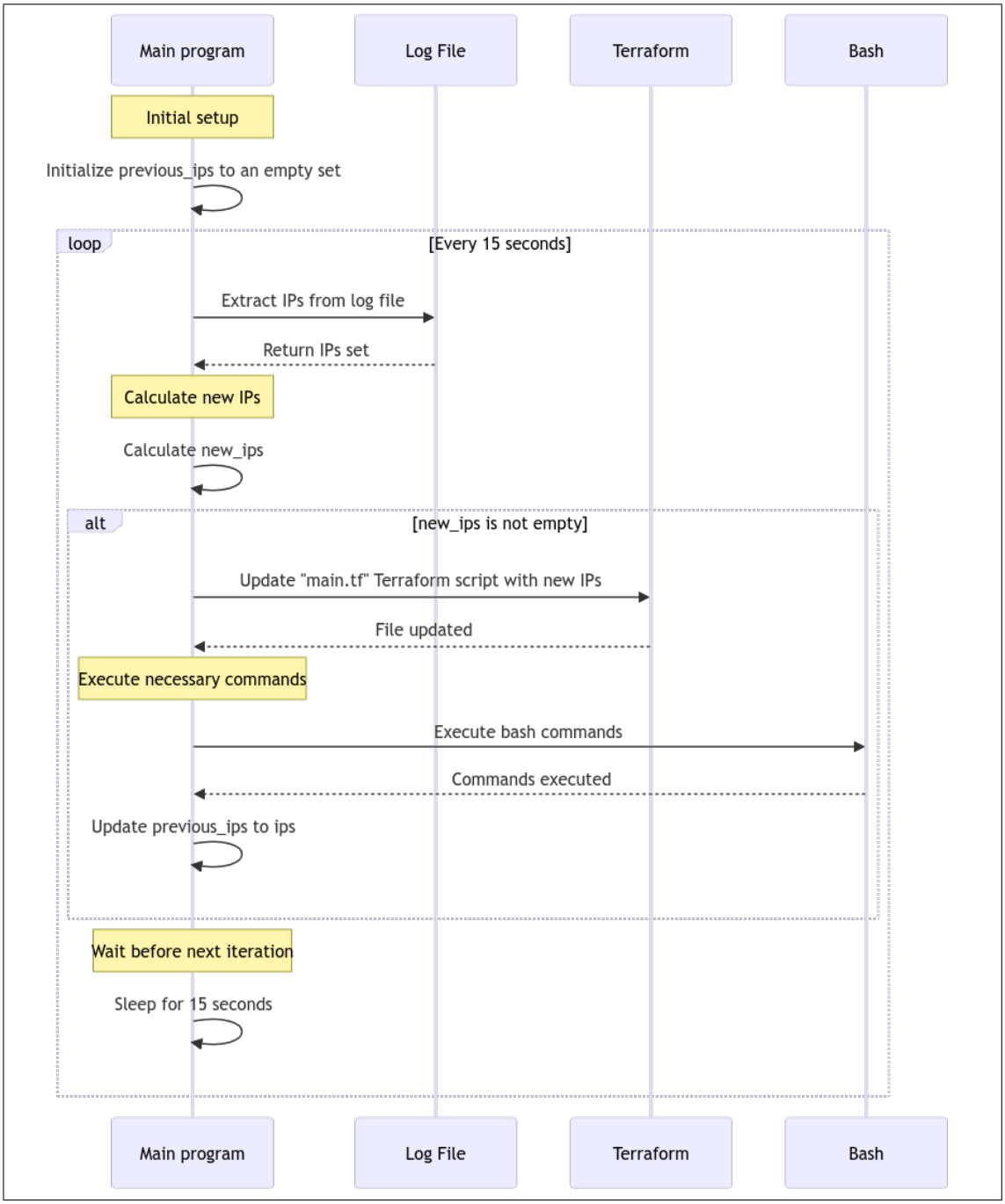


Figure 5.3: Operational flow diagram for the Feature 2 Python program

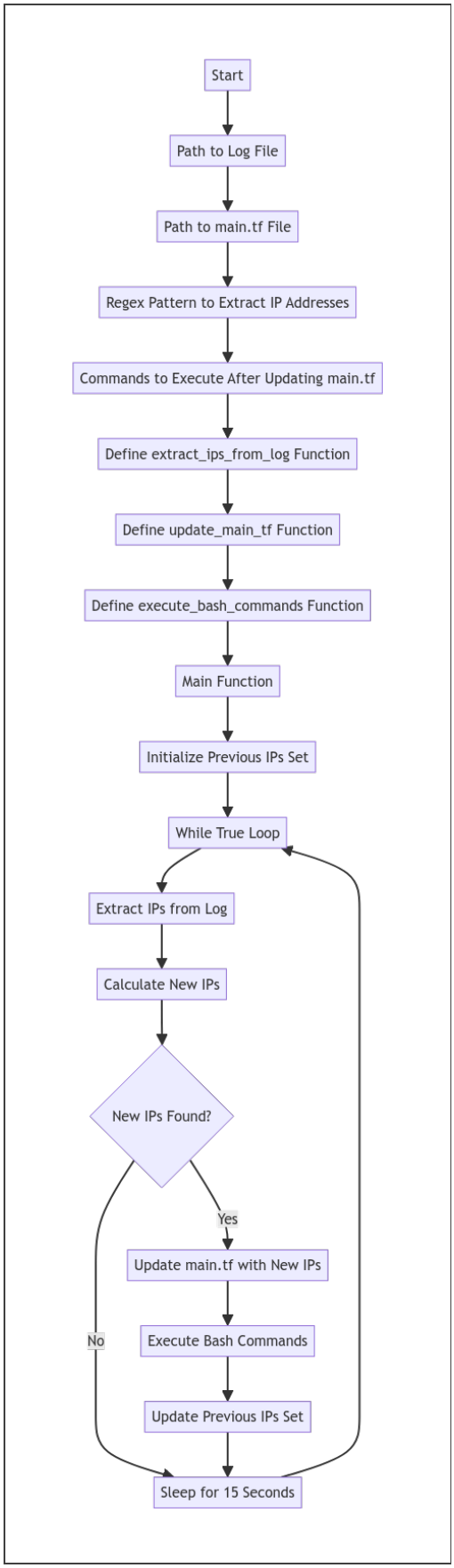


Figure 5.4: Flowchart for the Feature 2 Python program

5.3 Feature 3: Offline tokens monitoring

As the Canarytokens have a limited capability of gathering indecent data, they are only available online. As a result, a solution has to be developed that can also trigger the tokens when accessed even while offline and store the incident logs as well. To gain such capability, we can leverage the capabilities of the inotifywait utility to track token changes and log these events into a specified log file for any operations on them or while accessed.

5.3.1 Shell Script Overview

This bash script named `offline.sh` monitors specific files for various events such as access, modification, and deletion, logging these activities to a centralized log file. This analysis covers the script's components, operational flow, and effectiveness in achieving its objectives.

The script consists of following key sections:

(i) **Path to the Monitored Files:** Lists the files to be monitored.

(ii) **Log File Initialization:** Ensures the log file exists.

(iii) **Duplicate Entry Check:** Function to prevent redundant log entries.

(iv) **Monitoring Function:** Uses `inotifywait` to monitor and log file events.

(v) **Execution Flow:** Waits before starting the monitoring process.

5.3.2 Working and Breakdown of the script

Key Sections

a. Paths to the Monitored Files

```
files=(
"example/path/to/a/token1"
"/other/path/to/a/token2"
)
```

This array includes various critical tokens whose activities are to be tracked.

b. Log File Initialization

```
log_file="/home/bb/offline/operations.log"
touch "$log_file"
```

The script ensures the log file exists, creating it if necessary. This step prevents errors if the log file is missing.

c. Duplicate Entry Check

```
check_duplicate_entry() {
local datetime=$1
local user=$2
local operation=$3
local path=$4

if grep -q "$datetime.*$user.*$operation.*$path" "$log_file";
   then
    return 1
```

```
else
    return 0
fi
}
```

The `check_duplicate_entry` function prevents logging redundant entries. This function checks if an entry with the same timestamp, user, operation, and file path already exists in the log file.

d. File Monitoring Function

```
monitor_files() {
inotifywait --monitor --format "%e %w%f" \
    -e access -e close_write -e modify -e move -e create -e
       delete -e delete_self -e move_self -e attrib \
    "${files[@]}" | \
while read -r events file; do
    datetime=$(date "+%Y-%m-%d %H:%M:%S")
    user=$(whoami)
    hostname=$(hostname)

    if check_duplicate_entry "$datetime" "$user@$hostname" "
       $events" "$file"; then
        echo "$datetime $user@$hostname $events $file" >> "
           $log_file"
    fi
done
}
```

The `monitor_files` function utilizes `inotifywait` to monitor file events.

The two key points included here are:

(i) **Monitoring Setup:** `inotifywait` is configured to monitor multiple events (e.g., access, modification, deletion) on the specified files.

(ii) **Event Handling Loop:** The loop captures events, fetches the current date, time, user, and hostname, checks for duplicates, and logs new events.

5.3.3 Operational Flow

The operational outlines the flow of a program that monitors and processes file events for IP address management. It details initialization by defining files to monitor and ensuring log file existence, setting up monitoring with inotifywait, and executing commands to check for duplicate IP entries. It also describes logging new events with relevant details.

Initialization:

- Defines the list of files to monitor.
- Ensures the log file exists.

Monitoring Setup:

- `inotifywait` is used to monitor specified events on the files.
- It retrieves IP address from `incident.log` file, checks if the given IP is already present inside `main.tf` Terraform file, if yes then discards the duplicate entries, if no then updates to the `main.tf` Terraform script.

Command Execution:

- For each event, the script checks for duplicate entries.
- New events are logged with a timestamp, user, operation, and file path.

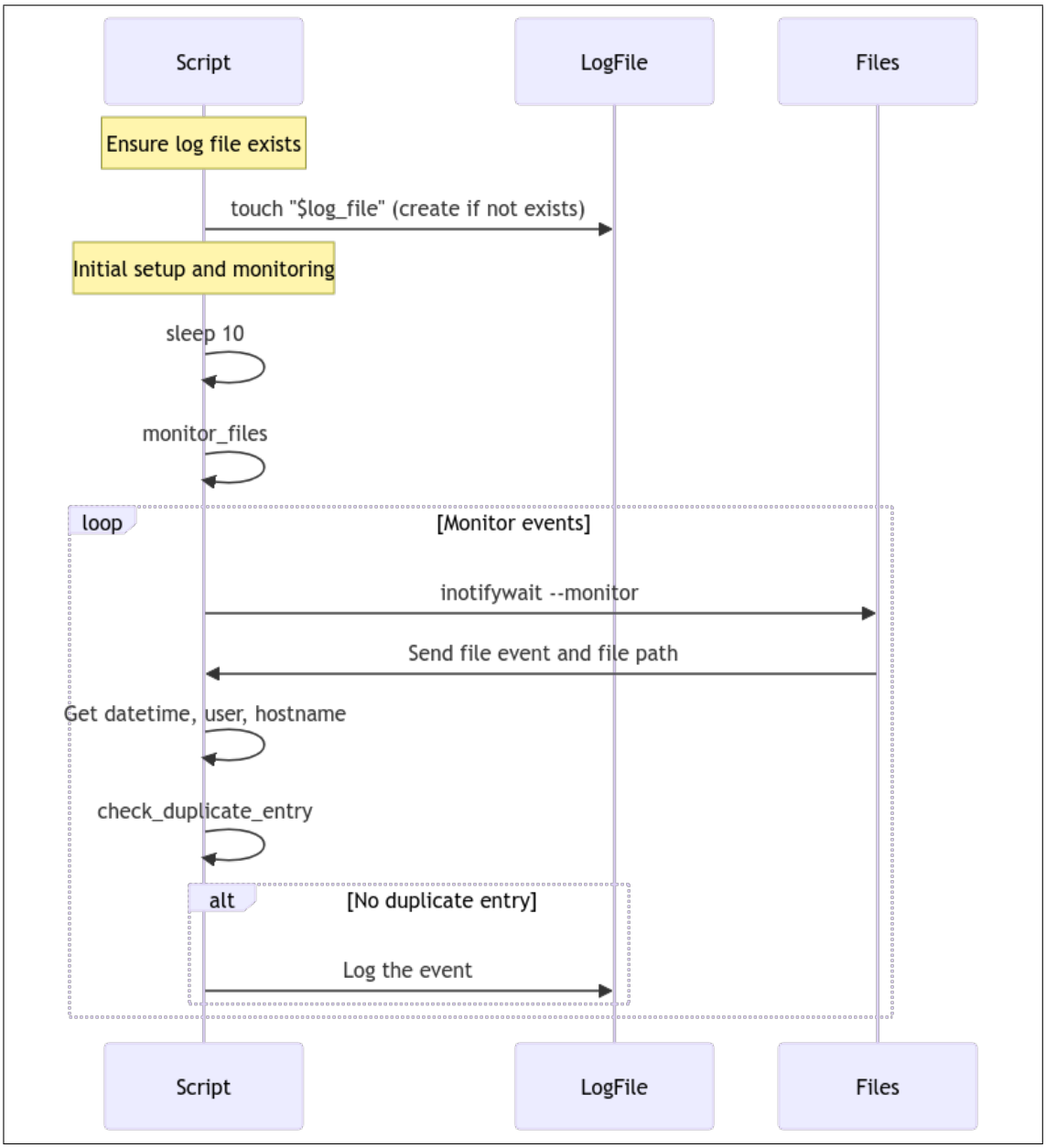


Figure 5.5: Operational flow diagram for the Feature 3 Shell script

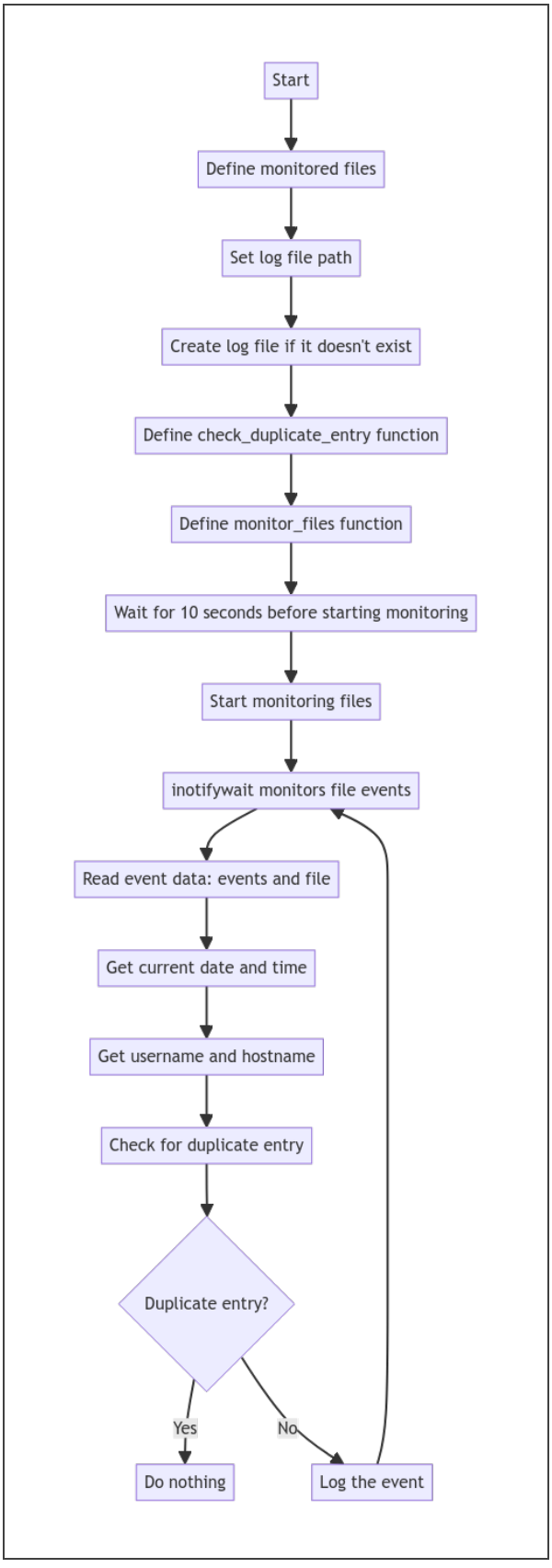


Figure 5.6: Flowchart for the Feature 3 Shell script

6 Implementation

This chapter provides a detailed explanation of the implementation of all the advanced functionalities with Canarytokens. This implementation will go incrementally, focusing on each functionality individually.

6.1 Implementation of Feature 1

The implementation of Feature 1 encompasses a series of tasks aimed at setting up a Canarytokens server, integrating it with a Dockerized `ct2rds` Python program, and deploying a Wazuh agent for enhanced monitoring and real-time alerting.

The primary steps include:

1. Creating a new Canarytokens server.
2. Building docker image named `ct2rds` for `ct2rds.py` program.
3. Integrating `ct2rds` image with the Canarytokens Docker image.
4. Starting the Canarytokens server.
5. Deploying and configuring the `Wazuh` agent.
6. Writing a `Wazuh` decoder.
7. Forwarding logs to a third-party platform via a webhook.

6.1.1 Creating a new Canarytokens server

Pre-requisites:

(i) **Canarytokens Official Docker Repository:** Cloned from the Canarytokens GitHub repository.

(ii) **Dedicated Server:** Debian 12 was used to host the Canarytokens server.

(iii) **Public IP Address and Domain:** IP address `11.159.12.101` and domain `www.canary.fi` are used to serve the frontend.

(iv) **Alert Domain Names:** A dedicated domain `alerts.canary.fi` for general alerts and `nx.canary.fi` for non-existent domain alerts for PDF tokens.

(v) **Wireguard Private Key Seed:** Generated using the shell command- `dd bs=32 count=1 if=/dev/urandom2>/dev/null | base64`.

Configuring the Canarytokens server settings:

As per the instructions from the `README.md` file, the server settings are configured using the pre-requisites. Mainly, five files had been configured: `frontend.env`, `switchboard.env`, and `certbot.env` from the Canarytokens root directory, while `nginx.conf` and the nginx-related `Dockerfile` file from `/certbot-nginx` directory.

6.1.2 Building docker image "ct2rds" for ct2rds.py program

The following steps are taken to create a Docker image for the `ct2rds` Python program:

(i) Installed Docker and ran the Docker service using official instructions from `https://docs.docker.com/engine/install/debian/` on the Debian server.

(ii) A Docker image with a Python execution environment was required to run the `ct2rds.py` program inside the Docker container. To create a Docker container, a `Dockerfile` was created inside the same directory containing the `ct2rds.py` program, which was in the `/rds_logs` directory. A `requirements.txt` file was also defined to install the necessary Redis packages.

Contains of the `ct2rdsimage` related `Dockerfile`:

```
FROM python:3.8
WORKDIR /usr/src/app
COPY . .
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
RUN mkdir logs
CMD ["python", "./ct2rds.py"]
```

(iii) The Docker image for the `ct2rds.py` program with the `ct2rds` name was built by executing the following command inside the shell:

```
docker build -t ct2rds .
```

6.1.3 Integrating ct2rds image with the Canarytokens Docker image

Integration of `ct2rds` Docker image with the Canarytokens Docker image was done by calling the `ct2rds` container inside the `common-services.yml`, which was a primary YAML configuration file of the Canarytokens Docker image as:

```
ct2rds:
restart: always
image: ct2rds
```

```
    links:
        - redis
    volumes:
        - ./logs:/usr/src/app/logs
    environment:
        - redis_host=redis
        - redis_port=6379
    container_name: ct2rds
```

In addition, the ct2rds container service from `common-services.yml` was called inside the `docker-compose-letsencrypt.yml` file from the Canarytokens root directory, so the `ct2rds` container can start along with the other Canarytokens containers:

```
    ct2rds:
        extends:
                file: common-services.yml
        service: ct2rds
```

6.1.4 Starting the Canarytokens server

The integrated Canarytokens containers are built and started using the following commands after visiting the `canarytokens-docker` directory:

```
docker compose -f docker-compose-letsencrypt.yml build
docker compose -f docker-compose-letsencrypt.yml up
```

6.1.5 Deploying and configuring the Wazuh agent

The Wazuh agent was deployed on the Canarytokens server using the commands:

```
curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | gpg --no-
   default-keyring --keyring gnupg-ring:/usr/share/keyrings/wazuh.
   gpg --import && chmod 644 /usr/share/keyrings/wazuh.gpg

echo "deb [signed-by=/usr/share/keyrings/wazuh.gpg] https://packages.
   wazuh.com/4.x/apt/ stable main" | tee -a /etc/apt/sources.list.d/
   wazuh.list

apt-get update

WAZUH_MANAGER="10.0.0.2" apt-get install wazuh-agent
```

The `incident.log` file was forwarded to the `Wazuh` agent by modifying the `ossec.conf` file present at ‘`/var/ossec/etc/`‘ as:

```
   <localfile>
   <log_format>syslog</log_format>
   <location>/home/tushar/canarytokens-docker/logs/incident.log</
      location>
   </localfile>
```

6.1.6 Writing a Wazuh Decoder

A custom decoder was created for parsing the `incident.log` file.

A typical log of an incident from the `incident.log` file:

```
INFO:root:2024-05-23 08:53:15: {'src_ip': '12.7.21.65', 'geo_info':
   {'loc': '59.4370,24.7535', 'org': 'AS3327 CITIC Telecom', 'city':
    'Tallinn', 'country': 'EE', 'region': 'Harjumaa', 'ip':
```

```
'12.7.21.65', 'timezone': 'Europe/Tallinn', 'postal': '10111'}, '
is_tor_relay': False, 'input_channel': 'HTTP', 'useragent': '
Mozilla/5.0 (Windows NT 10.0; rv:123.0) Gecko/20100101 Firefox
/123.0', 'request_headers': {'Host': 'eg.alert.ee', 'X-Real-Ip':
'12.7.21.65', 'X-Forwarded-For': '12.7.21.65', 'X-Forwarded-Host
': '_', 'Connection': 'close', 'User-Agent': 'Mozilla/5.0 (
Windows NT 10.0; rv:123.0) Gecko/20100101 Firefox/123.0', 'Accept
': 'text/html;q=0.9,*/*;q=0.8', 'Accept-Language': 'en-US,en;q
=0.5', 'Accept-Encoding': 'gzip, deflate', 'DNT': '1', 'Sec-Gpc':
 '1', 'Upgrade-Insecure-Requests': '1'}, 'request_args': {}}
```

The ‘Wazuh‘ decoder for decoding the ‘incident.log‘ file:

```
<decoder name="canary_token">
  <prematch>^\w+\p\w+\p\d+\p\d+\p\d+\s\d+\p\d+\p\d+\p</prematch>
</decoder>
<decoder name="canary_token_child">
  <parent>canary_token</parent>
  <regex offset="after_parent">^\s\p\p\.+\p\p\s(\w+)\p</regex>
  <order>status</order>
</decoder>
<decoder name="canary_token_child">
  <parent>canary_token</parent>
  <regex offset="after_parent">^\s\p\.+\p\p\s\p(\.+)\p\p\s\p\.+\p\p\s
     \p\p\.+\p\p\s\p\.+\s\p\w+\p\p\s\p(\w+)\p\p\s\p\w+\p\p\s\p(\w+)\p
     \p\s\p\w+\p\p\s\p(\.+)\s\p\w+\p\p\s\p(\.+)\s\p\w+\p\p\s\p(\.+)\s
     \p\w+\p\p\s\p\.+\s\p\w+\p\p\s\p\.+</regex>
  <order>connectiontype,city,country,region,hostname,timezone,
```

```
    ipaddress</order>
</decoder>
```

A corresponding `Wazuh` rule was defined to use this decoder:

```
<group name="canary_token">
  <rule id="60023" level="12">
    <decoded_as>canary_token</decoded_as>
    <description>Canary Token Incident</description>
    <options>no_full_log</options>
  </rule>
</group>
```

6.1.7 Forwarding realtime logs to a third-party platform via Webhook

To forward real-time logs to `Slack`, a webhook was created as per Slack's official documentation. The webhook was integrated into the `Wazuh` server configuration:

```
<integration>
  <name>slack</name>
    <hook_url>https://hooks.slack.com/services/T9HDYGH7RL/
       Hyt5wud7eyH74Fjr74hdb7a</hook_url>
  <alert_format>json</alert_format>
  <level>10</level>
  <group>canary_token</group>
</integration>
```

With this, the Canarytokens server was established, and feature 1 was implemented.

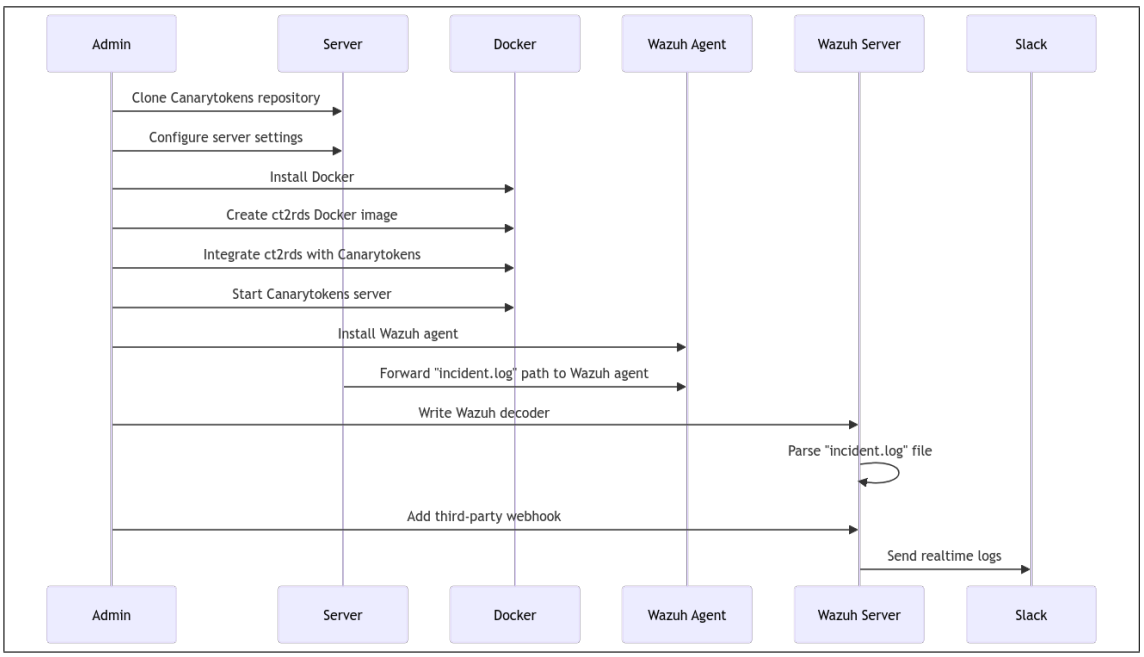


Figure 6.1: Operational flow diagram for Feature 1 implementation

6.2 Implementation of Feature 2

To implement feature 2, the following tasks were performed:

1. Preparing a Kubernetes pod.
2. Generating and distributing Canary tokens.
3. Adding the active Canary tokens path to the `offline.sh` shell script.
4. Installing `offline.sh` shell script to the Kubernetes pod.
5. Forwarding `offline.sh` logs file path to the `Wazuh agent`.
6. Writing `Wazuh` Decoders and Rules for Kubernetes.

6.2.1 Preparing a Kubernetes Pod

A pre-configured live Kubernetes pad named `kubemonitor` was selected for monitoring potential compromises.

6.2.2 Generating and distributing Canary tokens

Tokens can be generated by visiting the Canarytokens server's frontend at `www.canary.fi/generate` as shown in Figure 6.2.

To monitor the Kubernetes cluster, some of the tokens were shortlisted, which are URL tokens, Microsoft Excel tokens, `Kubeconfig` tokens, and email tokens.

In order to distribute tokens within a `kubemonitor` cluster, it was necessary to have shell access. To acquire shell access to the `kubemonitor`, execute the following command:

```
kubectl exec -it $(kubectl get pods -n kubemonitors | grep
   kubemonitor | cut -d ' ' -f 1) -n kubemonitors bash
```

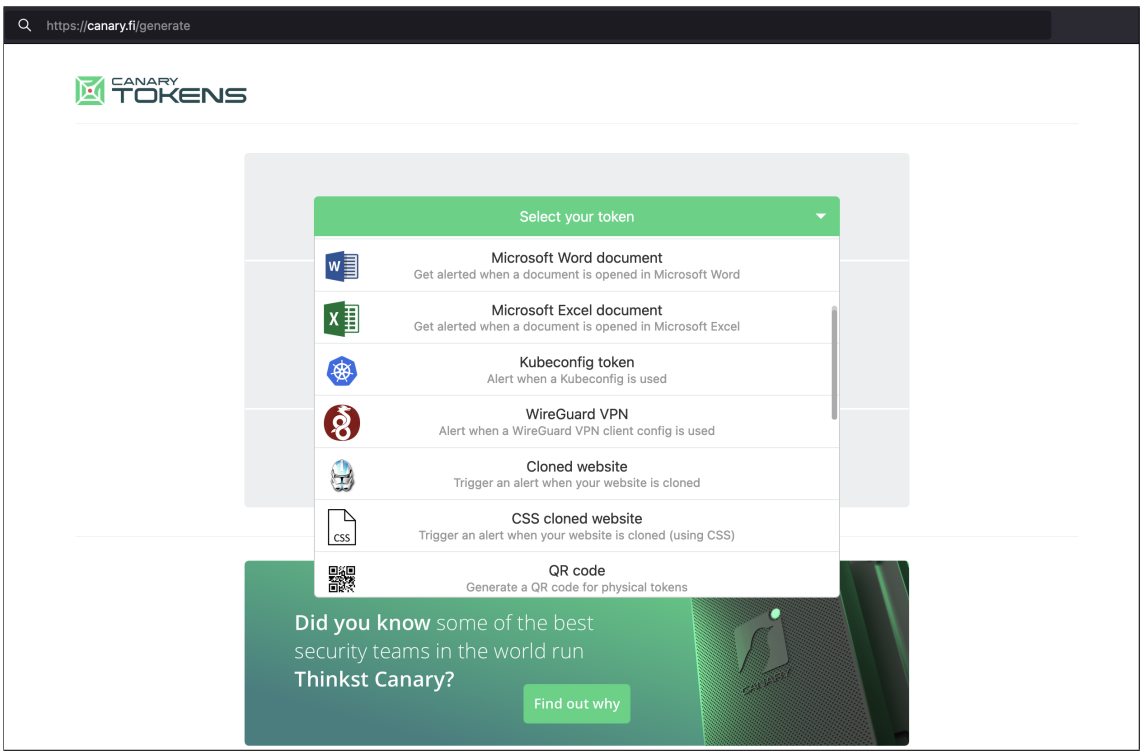


Figure 6.2: Canarytokens front page for generating tokens

After gaining the shell access, the newly generated Canary tokens have been strategically placed inside the `kubemonitor` pod, which was being monitored. The newly generated URL token, for example, `http://alert.canary.fi/stuff/fi9c3-mqjcpwayavupjkntp8ff/payments.js` was strategically placed inside the `sensitive_links.md` file under the `/admin` directory inside the `apache` home folder. Some of the `Kubeconfig` tokens are directly placed inside the`/home/kubemonitor/tokens` directory with unique names like `utu_kubeconfig`. While some Microsoft Excel tokens are placed in multiple locations inside sensitive directories like `/payments` with the file name `client_detail.xlsx`, the list of email tokens was placed inside a specific file named `authorized_list.md` inside the `/okta/admin` directory.

6.2.3 Adding tokens path to the offline.sh shell script

Paths of the tokens are updated in the `offline.sh` script:

```
files=(
    "/home/kubemonitor/tokens/utu_kubeconfig"
    "/home/kubemonitor/payments/client_detail.xlsx"
    "/okta/admin/authorised_list.md"
)
```

6.2.4 Installing the offline.sh Shell Script inside the Kubernetes Pod

While installing the `offline.sh` script inside the pod, the log file path created by this script should be forwarded to the `Wazuh agent` to receive the alerts on third-party platforms. To do so, a pre-installed `Wazuh agent` on the nodes parent to this pod was taken into consideration, as in present the `Wazuh agent` lacks proper support for installing at pod level. Thus, to deploy the shell script inside a pod and ensure it can write logs to a specific path on the Kubernetes node, a hostPath volume in the pod manifest can be used to mount a directory from the host (node) to the pod. This enables the script running inside the pod to access and write logs to the node's file system.

Moreover, a Docker image has been generated from the `offline.sh` shell script, and it was stored inside the locally accessible harbor instance.

A manifest file that defines a pod that mounts `/home/tushar/offline` from the host node to `/offline` inside the container:

```
apiVersion: v1
kind: Pod
metadata:
  name: offline-pod
```

```
spec:
  containers:
  - name: log-monitoring-container
    image: #harbour_image_path
    volumeMounts:
    - mountPath: /offline
      name: offline-volume
  volumes:
  - name: offline-volume
    hostPath:
      path: /home/tushar/offline
      type: Directory
  restartPolicy: OnFailure
```

A ConfigMap was created to store the Shell script to mount it inside the pod:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: offline-script
data:
  log_monitor.py: |
    # (Contents of the offline.py script here)
```

Both `ConfigMap` and `DaemonSet` configuration had been applied using the following kubernetes commands:

```
kubectl apply -f offline-script.yaml
kubectl apply -f offline-pod.yaml
```

6.2.5 Forwarding log file path to the Wazuh agent

The `offline_incident.log` file path has been forwarded to the `Wazuh agent` by adding the following configuration to the `ossec.conf` file present at `/var/ossec/etc/` inside the Kubernetes node as:

```
<localfile>
    <log_format>syslog</log_format>
    <location>/home/tushar/offline/offline_incident.log</location>
</localfile>
```

6.2.6 Writing Wazuh Decoder and Rule for Kubernetes

The typical log of an incident from the `offline_incident.log` file looks like:

```
2024-05-21 12:54:18 tushar@sd-166573 ACCESS /home/tushar/offline/
    tokens/passwords.txt
```

For decoding the `offline_incident.log` file, the default `Wazuh` decoder was used.

The corresponding `Wazuh rule`:

```
    <group name="kube_apiserver_log">
      <rule id="100200" level="10">
        <decoded_as>kube_apiserver_log</decoded_as>
        <description>Unauthorized access attempt</description>
        <options>no_full_log</options>
      </rule>
    </group>
```

With this, the offline Canary tokens monitoring capabilities had been integrated in to the Kubernetes pid, and feature 2 has been successfully implemented.

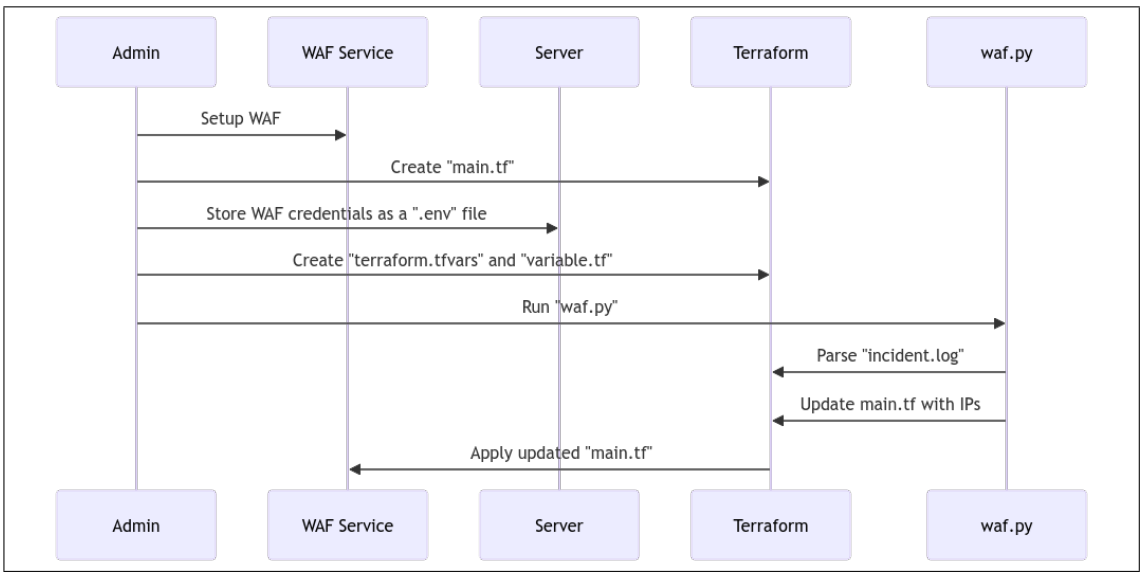


Figure 6.3: Operational flow diagram for Feature 2 implementation

6.3 Implementation of Feature 3

Feature 3 focuses on automating the security analysis of JSON logs generated by the Canarytokens server and forwarding pertinent alerts to Slack via a webhook. This involves:

1. Setting-up WAF
2. Creating an Terraform IaC file named `main.tf`
3. Running the `waf.py` Python script

6.3.1 Setting-up WAF

A pre-configured WAF service from Imperva was used.

6.3.2 Creating an Terraform IaC for Imperva WAF

To add the attacker's IP to the WAF blacklist dynamically, an IaC `main.tf` was created using Terraform with the content as:

```
terraform {
  required_providers {
    incapsula = {
      source = "imperva/incapsula"
      version = "3.22.0"
    }
  }
}


provider "incapsula" {
  api_id = var.api_id
```

```
  api_key = var.api_key
}


resource "incapsula_policy" "Canarytokens_IPs" {
  name = "Canarytokens_IPs"
  enabled = true
  policy_type = "ACL"
  policy_settings = <<POLICY
    [
      {
        "settingsAction": "BLOCK",
        "policySettingType": "IP",
        "data": {
          "ips": []
        }
      }
    ]
  POLICY
}
```

Here, the IP addresses logged inside the file `incident.log` will be parsed by the `waf.py` Python script and then added to the `ips` field inside the `main.tf` terraform file. To securely store the Imperva WAF credential, a separate `.env` file was created storing such credentials. And, to map these credentials from `.env` file to the `main.tf` file, two files, i.e. `terraform.tfvars` file and `variable.tf` file was created.

Contents of the `terraform.tfvars` file:

```
api_id = var.api_id
```

```
api_key = var.api_key
```

Contents of the variable.tf file:

```
variable "api_id" {
  description = "The API ID for Incapsula"
  type = string
}
variable "api_key" {
  description = "The API key for Incapsula"
  type = string
  sensitive = true
}
```

6.3.3 Running the waf.py Python script

The Python script was scheduled to run periodically every 15 seconds in the background using the following command:

```
python3 waf.py &
```

With this, the implementation of feature 3 was completed.

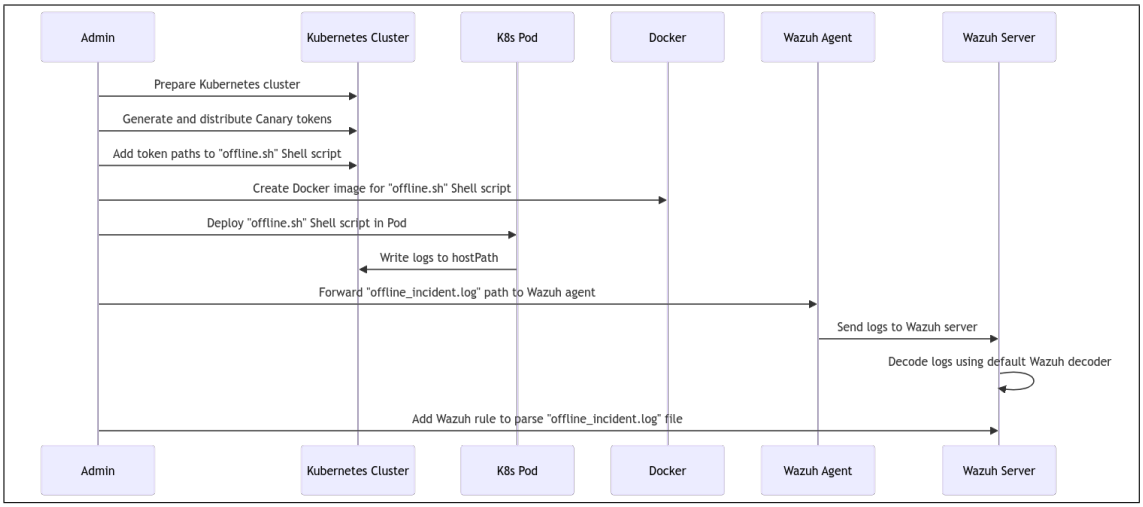


Figure 6.4: Operational flow diagram for Feature 3 implementation

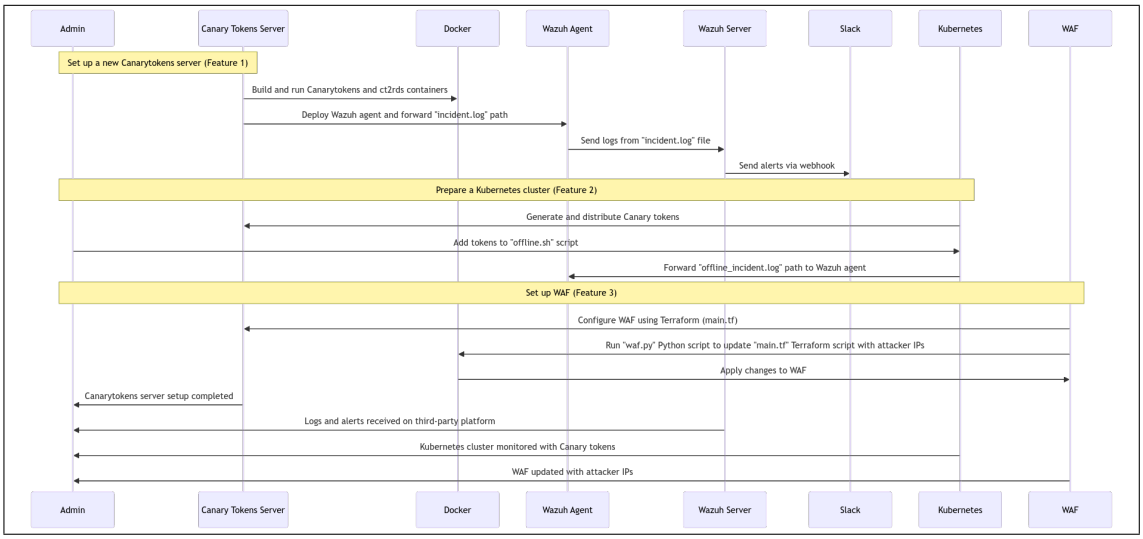


Figure 6.5: Operational flow diagram for the implementation of all three features

7 Results

In order to check the functionality of the solution and demonstrate its capabilities, a test lab was prepared. The vulnerable Kubernetes pod is being developed along with the implementation of the canarytokens with their newly added advanced security features. As part of this work, the sensitive data for the kubernetes and permissions is to be exploited by the attacker.

7.1 Preparing the test environment

To prepare the lab, the following steps should be taken:

1. Build a local Canarytokens server by installing the Docker image, which is integrated with the advanced security features that we have witnessed in the implementation section.

2. Generate the Canary tokens from the Canarytokens frontend located at the local server, which is running on localhost at port 80.

3. At strategic locations, deploy the newly generated Canary tokens from the server inside the vulnerable Kubernetes pod.

4. Add the paths of these tokens from the pods to the `offline.sh` script. Install and run the script in the background inside the pod.

5. Set up WAF on a different server; here, the author uses Imperva SaaS WAF, to block the attackers IP address when detected for malicious activities after triggering the Canary tokens.

6. Write a Terraform script to build an IaaC so that the Canarytokens server and WAF can interact with each other.

7. dd the terraform commands to the `waf.py` Python script.

8. Run the `waf.py` Python script inside the Canarytokens server.

9. Create a new server with a Wazuh SIEM tool installed.

10. Inside the pod, install the Wazuh agent.

11. Forward the log file paths that are generated by the `offline.sh` Shell script and Redis to the Wazuh agent config file located in its root directory.

7.2 Examining the advanced features inside the prepared live Kubernetes environment

In order to provide the proof-of-concept, the two groups, each consisting of five users, were granted authorised access to the Kubernetes machine. This setup imitated a real-world situation where attackers manage to acquire unauthorised access to the machine and sustain persistence.

The first group of users were unaware of the existence of Canary tokens integrated into the provided Kubernetes pod. Although the second group of users were aware of the existence of the Canary tokens within the provided Kubernetes pod.

Both groups were assigned the objective of extracting the data to the greatest extent achievable in order to obtain valuable information about the organisation, similar to what an attacker would want to achieve. Furthermore, the group was not provided with any information on the legality or illegitimacy of the data included within the system, replicating the scenario of an ideal attacker and ideal machine.

In attack scenario 1, the Canarytokens server has been started, and the `waf.py` Python program is being used to seamlessly block the IP addresses of the attackers. The `offline.sh` shell script is exclusively used for scenario 2. This is done to demonstrate the clarity of the alerts received from each feature and to prevent duplicate alerts from both offline and online sources.

7.2.1 Attack Scenario 1

Initially, all the users from Group 1 were given access to the machine and given a signal to exfiltrate the data. After a certain period of time, Slack begins to receive and display the alerts individually from five distinct IP addresses, as outlined in figure 7.1.

The reason for triggering the tokens was that three of the users from group one copied the exfiltered data into their local machine and tried to open it while the local machine was connected to the internet. While some users directly opened the files inside the pods, which resulted in the triggering of the alerts.

Furthermore, the given IPs have been blocked automatically on `Imperva` WAF, restricting further access to the given attacker's IP as depicted in Figure 7.2. As the IPs were logged inside the `incident.log` file, the `waf.py` triggered the `main.tf` terraform file and added the attackers IPs to the Imperva WAF, blocking their further access to the pod.

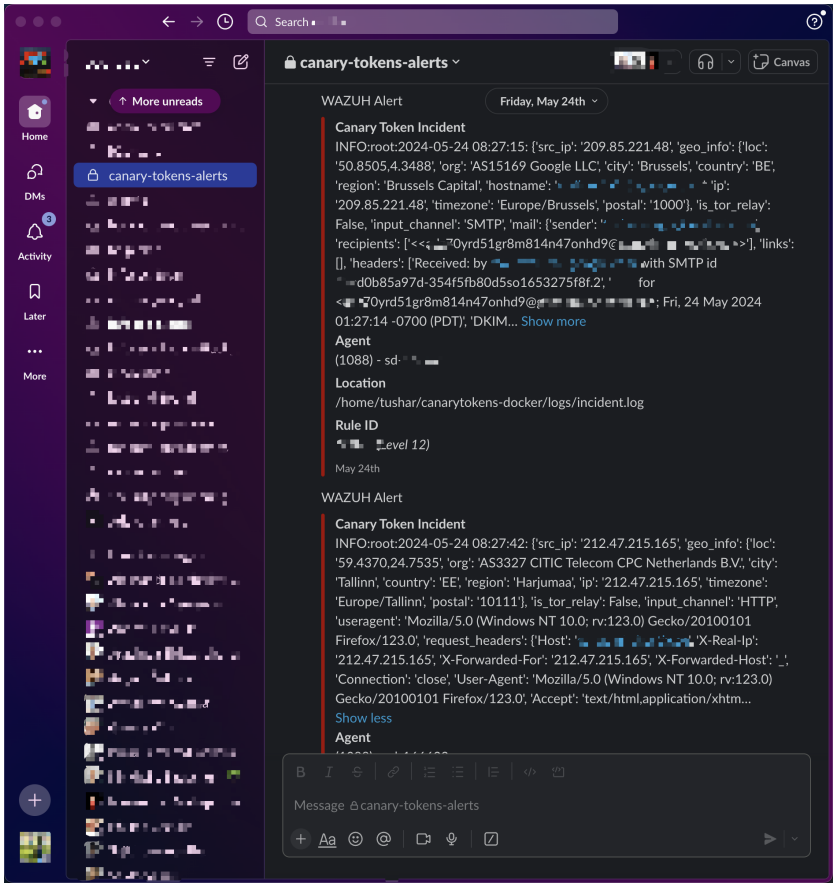


Figure 7.1: Realtime Wazuh alerts received on Slack after tokens triggered online

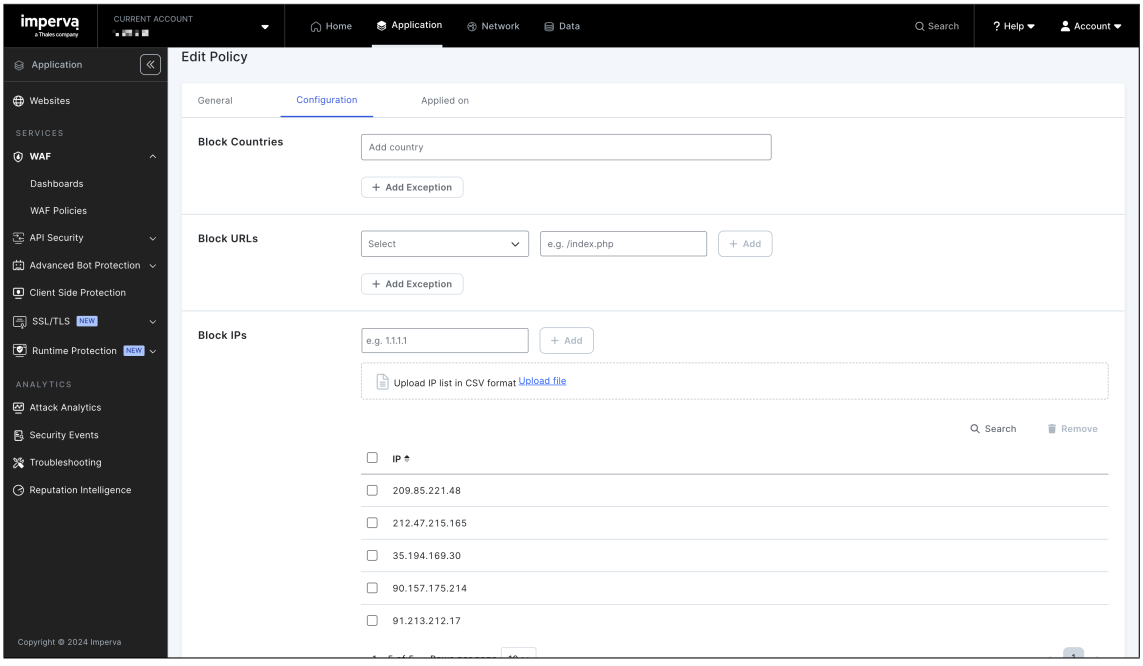


Figure 7.2: Attackers IP addresses blocked on Imperva WAF

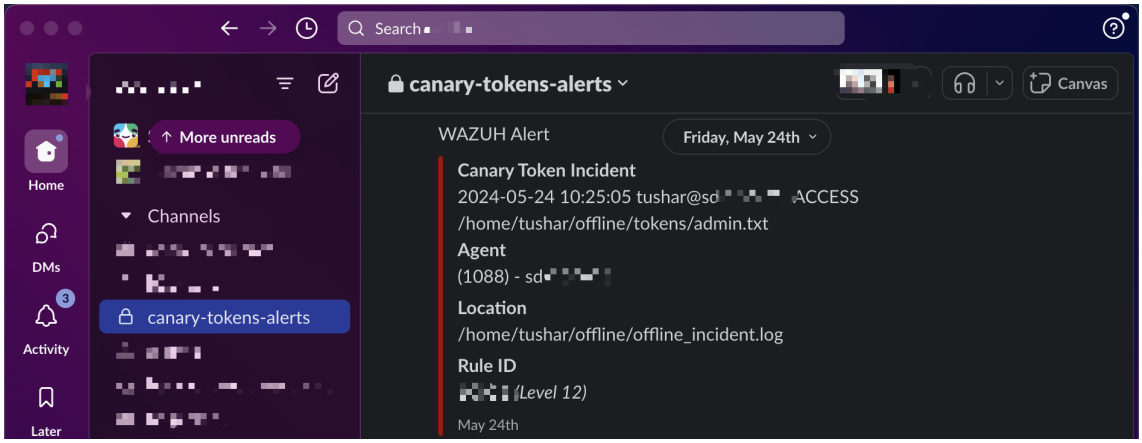


Figure 7.3: Realtime Wazuh alerts received on Slack after tokens triggered offline

7.2.2 Attack Scenario 2

Along with the previous features, for this scenario, the `offline.sh` shell script was also activated, actively monitoring the token files while running in the background.

Later, all the users from Group 2 were given the signal to access the machine and exfiltrate the data.

After a brief moment of time, Slack starts to notify the alerts one by one, as shown in Figure 7.3.

The reason for triggering the tokens was that all of the users from group two, who knew the Canary tokens were active inside this pod, copied the exfiltered data into their local machine and tried to operate on them in an offline environment, which is the biggest disadvantage with the Canary tokens.

As the offline.sh script was monitoring the token files, it captured all of the operations that occurred while the file was being copied within the pod. As the attackers were not aware of the legitimacy of the data, they ended up copying all the files, including the placed Canary tokens. As a result, the alerts had been set off and forwarded by the `offline.sh` shell script after triggering the Canary tokens.

7.3 Evaluation

The final evaluation on implementing and testing advanced security features in a Kubernetes environment using Canarytokens is highly commendable. The research successfully demonstrates the effectiveness of Canarytokens in detecting unauthorized access and exfiltration activities within a Kubernetes pod. By preparing a detailed test lab and conducting realistic attack scenarios, the study provides a robust proof-of-concept for the proposed security measures.

The test environment was meticulously set up to replicate real-world conditions, where attackers exploit sensitive data and sustain persistence. Key steps included building a local Canarytokens server, deploying Canarytokens within the Kubernetes pod, and integrating various security tools like Imperva SaaS WAF and Wazuh SIEM. These preparations ensured a comprehensive evaluation of the security features under both online and offline attack scenarios.

In the first attack scenario, where users were unaware of the Canarytokens, the system effectively identified and blocked malicious activities. The integration of waf.py with Terraform commands enabled automatic IP blocking via Imperva WAF, showcasing the practical applicability of the solution in real-time threat mitigation. The alerts received through Slack provided clear evidence of unauthorized data access attempts, validating the effectiveness of the Canarytokens and the automated response mechanism.

The second attack scenario, involving users aware of the Canarytokens, highlighted the challenges and limitations of the solution, particularly in offline environments. Despite this, the offline.sh script successfully monitored and captured all operations on the token files, proving the robustness of the implemented security measures. The study's design, which did not inform users about the legitimacy of

data, further emphasized the practical implications of the security setup, mimicking real attacker behavior.

Overall, the thesis presents a well-structured and comprehensive evaluation of advanced security features in a Kubernetes environment. The successful implementation and testing of Canarytokens, combined with automated threat response mechanisms, provide significant contributions to the field of cybersecurity. The insights gained from this research can inform future developments in securing containerized applications and enhancing incident response strategies.

8 Discussion

8.1 Revisiting the research questions

8.1.1 Research Question 1

How to integrate the Canarytokens server with the SIEM systems that can forward the real-time alert logs to the same?

There exists a custom Python script called `ct2rds.py`, which has been developed to keep track of the Redis database for new incident logs. This script listens to the Redis port number 6379 and captures the incident logs in realtime recording them in another file called `incident.log`. This file can further be sent to the SIEM system using either a log forwarder or an agent such as `Wazuh agent`.

To make integration easier, the Wazuh agent was deployed on the same server where Canarytokens are hosted. The path towards `incident.log` was adjusted within `ossec.conf` file of `Wazuh agent` enabling it to monitor and forward logs to the Wazuh server. Besides, there was also a need to create a custom decoder on Wazuh server that could rightly parse the `incident.log` file.

In addition, author aslo set up webhook integration into a third party platform (in this case, a dedicated `Slack` channel) so that real time alert logs from `Wazuh` server could be forwarded into it; hence, immediate notification and visibility of possible security incidents by Canarytokens.

8.1.2 Research Question 2

What are the mechanisms that can be implemented to automate the first line of defensive response to restrict the attacker from receiving an alert after triggering a Canary token?

A Python script, `waf.py`, was created to automate the first line of defensive response by restricting an attacker who triggered Canary token. It monitors `incident.log` file for any new IP addresses associated with triggered Canary tokens. To be more precise, this script perpetually checks for new IP addresses associated with triggered Canary tokens in the `incident.log` file. As new IPs are detected, the script updates a Terraform configuration file, `main.tf`, with the identified IP addresses.

The `main.tf` file outlines an IaC setup that interacts with a WAF service, i.e. `Imperva` is used in this case. The script then executes several Shell commands to initialize, plan, and apply the updated Terraform configuration, thereby adding these IP addresses that were detected on WAF's blocklist.

Through the leveraging of WAF APIs and Terraform's IaC capabilities, this automated mechanism ensures that any instance of a triggered Canary token's IP address is immediately restricted at the WAF level to deny further access to the attackers while minimizing potential risks.

8.1.3 Research Question 3

What are the ways beyond an email for receiving such alerts after triggering Canarytokens?

In addition to native support for email alerts in the Canarytokens project, other methods for receiving alerts were reviewed and implemented, including SIEM systems integration as well as using third-party platforms, among others, so as to obtain more flexibility and improve visibility.

One implementation involves forwarding alert logs from `Wazuh` to a dedicated `Slack` channel via webhook integration. This technique enables realtime notification and clear visibility for faster response time and situational awareness enhancement.

Further, SIEM integration contains more room for alert reception. This depends on the SIEM solution being used as well as its capabilities. Alerts can be sent to various destinations based on the capabilities of the SIEM system. Some of these destinations include email distribution lists, ticketing systems, or other monitoring platforms that will ensure the right teams are notified in time.

8.1.4 Research Question 4

How to make the Canarytokens work and receive alerts even when offline?

To facilitate offline monitoring and alert generation for Canarytokens, a shell script called `offline.sh` was created. With this script, it is possible to keep an eye on specific files for certain events using the `inotifywait` utility.

The `offline.sh` script is programmed to monitor specified Canary tokens within a target environment. Whenever any of these monitored tokens are accessed, their respective logs, such as timestamp, user details, performed operations, and file path, all together are registered in a dedicated log file, `offline_incident.log`.

In cases where an internet connection may not be available, such as a complete isolation from any outside networks, organizations may deploy this `offline.sh` script into their target environments to help them keep track of tokens and detect potential threats or unauthorized access attempts.

To integrate with existing security monitoring infrastructure, the path to the `offline_incident.log` file can be forwarded to a SIEM system like `Wazuh agent`, that can then forward them to other notification platforms for enhanced visibility.

8.2 Contribution

This research intends to make significant contributions in the area of Kubernetes security by improving the capabilities of Canarytokens as well as addressing their real-world challenges.

The key contribution of this work is making the Canary tokens work offline in an Kubernetes environment, which was a major hurdle. Additionally, this work aims to develop pre-configured modular add-ons, which are Canarytokens open source project and could be seamlessly integrated with any third-party solutions in order to have logging monitoring or alerting on any platforms that can be used by end user or organizations for improved Kubernetes security.

This modular add-on includes:

1. A Python program that can export the realtime logs from the Canarytokens server and writing to a dedicated incident file inside the local server directory. These logs can be easily customized based on various SIEM platforms and also monitored directly.

2. A Python program that parses the IP address from the incident log file and invokes an IaC Terraform script that will automatically block the attacker's IP in any WAF.

3. A Shell script that actively monitors Canary tokens placed at specific locations within a Kubernetes environment, this shell script also writes alert logs to a separate offline incident file for forwarding to SIEM solutions.

Moreover, these modular add-ons allow the use of webhooks to get real-time alerts updates from Canarytokens server on third-party platforms like Slack, among others.

8.3 Limitations

It is important to note that while the research and implementation have provided great insights and improvements on the open source project of Canarytokens, there are some limitations:

- The implementation focused on specific tools and platforms, such as Wazuh, Imperva WAF, Slack and Terraform. Working with other SIEM systems, WAFs, or IaC tools may require additional work to adapt by the organizations.
- The attacker may get root access to the pod where the offline monitoring script is located and running. If the attacker stops the offline monitoring program right before exfiltrating the data, the system will be unable to log the offline security events.
- The offline token monitoring solution used the `inotifywait` utility which may not be compatible with environments other than Linux.

9 Conclusion

9.1 Summary

The performed research aimed at advancing the existing Canarytokens open-source project by eliminating some limitations and adding refined capabilities, specifically designed for Kubernetes environments. Main contributions include the integration of Canarytokens with SIEM systems, automate defensive response, enabling alert forwarding using other methods and facilitating token monitoring when it is offline.

The research has involved several aspects, which include a comprehensive study on the Canarytokens project to identify its limitations as well as develop technical solutions that would address them. They were: 1) Real-time incident logs captured from the Redis, then delivered to SIEM systems; 2) Blocking attacker IP addresses automatically through WAF; and 3) Offline token monitoring and logging of incidents.

The implementation involved creating Python scripts, Shell scripts, and leveraging various tools such as Wazuh, Terraform, and Imperva WAF. The implemented solutions have been tested in a live Kubernetes environment to demonstrate their impact on improving the security posture of Kubernetes deployments.

9.2 Concluding Remarks

The research successfully addressed the stated research questions and objectives by developing modular add-ons that extend the capabilities of the Canarytokens project. This success in the developed solutions revealed how organizations could enhance their security and improve their overall incident response capabilities. These can help organizations stay updated on emerging threats and respond to any incidents within their Kubernetes deployments as soon as possible by integrating Canarytokens with SIEM systems, automating incident responses, having alternative alert delivery methods and enabling offline monitoring functionalities.

The integration into SIEM systems like Wazuh assisted in centralized monitoring and analysis of Canarytokens alerts that aid in correlating as well as prioritizing incidents effectively for security teams. Additionally, Slack webhook integration has been able to boost collaboration towards realtime situational awareness of alerts. Automated incident response techniques with the WAF service using Terrafom IaC showed the speed of automation in mitigating threats.

Furthermore, this project addressed some limitations that were experienced by previous Canarytokens designed through the development of alternative modes of alerts delivery and offline monitoring mechanisms. For example, organizations can now receive alerts through different channels based on their specific demands or even maintain visibility into possible risks even where there is no internet connection.

Throughout this research process and its implementation, valuable insights were gained on the challenges and complexities involved in securing Kubernetes environments. The need for continuous improvement and adaptation was highlighted during the integration of advanced features into the Canarytokens project so as to keep pace with the ever-evolving threat landscape.

9.3 Future Research Directions

While this research has made significant contributions to enhancing the capabilities of Canarytokens in Kubernetes environments, there are several areas for future exploration and improvement:

1. Developing a more robust and tamper-resistant solution for offline token monitoring:
    - Explore methods to secure the offline monitoring script and prevent unauthorized modifications or termination by attackers.
    - Investigate the use of virtualization or containerization technologies to isolate the monitoring process from the monitored environment.
2. Exploring alternative approaches for offline token monitoring:
    - Investigate file system monitoring techniques that are platform-agnostic and compatible with different operating systems and environments.
    - Develop a cross-platform solution that can be deployed across heterogeneous environments, including Windows, macOS, and various Linux distributions.
3. Extending the research to address other aspects of Kubernetes security:
    - Develop integrated solutions that combine Canarytokens with other security controls to provide a multi-layered defense strategy for Kubernetes environments.
    - Explore the integration of the developed solutions with existing Kubernetes security tools such as Falco, etc.

References

[1] "Thinkst canary". (2015), [Online]. Available: https://canary.tools/ (visited on 05/09/2024).

[2] E. Casalicchio and V. Perciballi, "Challenges and opportunities with kubernetes", in *2019 IEEE International Congress on Internet of Things (ICIOT)*, IEEE, 2019, pp. 84–89.

[3] X. Lyu and S. Randriamasy, "Kubernetes security best practices", in *Architecting Cloud Computing Solutions*. 2022, pp. 209–227.

[4] J. David, "Honeytokens: Deception for web defensive", en, SANS Institute, May 2019. [Online]. Available: https://www.sans.org/cyber-security-resources/analyst-papers/honeytokens-deception-for-web-defensive-38780.

[5] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective", *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018, ISSN: 0360-0300. [Online]. Available: https://doi.org/10.1145/3214305.

[6] T. Canary. "Canary token alert forwarding capabilities". (2023), [Online]. Available: https://help.canary.tools/hc/en-gb/articles/360002431478-i-want-to-integrate-my-siem-with-my-canaries (visited on 06/09/2024).

[7] C. Puasuareanu, A. Roy, and H. Debar, "Survey of new trends in attack graph analysis", *Communications Surveys and Tutorials, IEEE*, vol. 11, no. 3, pp. 27–45, 2009.

[8] J. DeBlasio, S. Savage, G. M. Voelker, and A. C. Snoeren, "Tripwire: Inferring internet site compromise", in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17, London, United Kingdom: Association for Computing Machinery, 2017, pp. 341–354, ISBN: 9781450351188. [Online]. Available: https://doi.org/10.1145/3131365.3131391.

[9] E. Levy. "The beginner's guide to honeytokens- aka canary tokens". (2024), [Online]. Available: https://www.securityengineering.dev/the-beginners-guide-to-honeytokens-aka-canary-tokens (visited on 06/10/2024).

[10] H. Holm and M. Karresand, "Honeytokens: A study of the effectiveness of deception resources for detection and investigation of malicious activities", *Journal of Information Warfare*, vol. 13, no. 3, pp. 38–52, 2018.

[11] G. A. Reale and B. Z. Loft, "Canarytokens: An old concept for a new world", *Scientific and practical cyber security journal (SPCSJ)*, pp. 66–68, 2019, ISSN: 2587-4667. [Online]. Available: https://journal.scsa.ge/wp-content/uploads/2019/04/3.1_8_gionathan_armando_reale.pdf (visited on 05/08/2024).

[12] T. Väisänen, "Categorization of cyber security deception events for measuring the severity level of advanced targeted breaches", en, European Conference on Software Architecture, Sep. 2017. [Online]. Available: https://doi.org/10.1145/3129790.3129805.

[13] L. Spitzner, "Honeypots: Catching the insider threat", in *Computer Security Applications Conference, Annual*, Los Alamitos, CA, USA: IEEE Computer

Society, Dec. 2003, p. 170. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CSAC.2003.1254322.

[14] V. Troia, *Hunting cyber criminals: A hacker's guide to online intelligence gathering tools and Techniques*. John Wiley & Sons, 2020, pp. 25, 501, 488.

[15] H. Holm, "Honeytokens: Deception techniques for detecting threats in modern cyber environments", in *Nordic Conference on Secure IT Systems*, Springer, Cham, 2021, pp. 3–18.

[16] M. C. Crichlow, "A study on blue team's opsec failures", M.S. thesis, University of Twente, 2020, p. 133. [Online]. Available: https://essay.utwente.nl/84945/ (visited on 05/09/2024).

[17] G. Weir, W. Lee, A. Bettaui, and J. Zheng, "Automatic deployment of canary tokens for detecting data misuse", in *2022 IEEE International Conference on Web Services*, IEEE, 2022, pp. 349–357.

[18] S. Bodmer, M. Kilger, G. Carpenter, and J. Jones, *Reverse Deception: Organized Cyber Threat Counter-Exploitation*. McGraw Hill, Aug. 2012, p. 464, ISBN: 978-1259061011.

[19] A. Almashhadani, M. Kaiiali, M. Carvalho, and N. Aziz, "Honeytokens: A comprehensive study of their deployment, detection, and response", in *2020 IEEE Conference on Application, Information and Network Security (AINS)*, IEEE, 2020, pp. 199–204.

[20] M. U. Rana, O. Ellahi, M. Alam, J. L. Webber, A. Mehbodniya, and S. Khan, "Offensive security: Cyber threat intelligence enrichment with counterintelligence and counterattack", *IEEE Access*, vol. 10, pp. 108 760–108 774, 2022.

[21] W. Fan, Z. Du, D. Fernández, and V. A. Villagrá, "Enabling an anatomic view to investigate honeypot systems: A survey", *IEEE Systems Journal*, vol. 12, no. 4, pp. 3906–3919, 2018.

[22] M. J. Carey and J. Jin, *Tribe of hackers blue team: Tribal knowledge from the best in defensive cybersecurity*. John Wiley & Sons, 2020, ISBN: 9781119643418.

[23] D. Bojović and J. T. Lygre, "To deceive or not deceive: Unveiling the adoption determinants of defensive cyber deception in norwegian organizations", M.S. thesis, University of Agder, 2023, pp. 47–52. [Online]. Available: `https://uia.brage.unit.no/uia-xmlui/bitstream/handle/11250/3080486/no.uia:inspera:143804570:99410028.pdf?sequence=1` (visited on 05/09/2024).

[24] H. Holm, M. Karresand, A. Vidström, and E. Westring, "Honeytokens: A deception mechanism to detect and investigate unauthorized access to sensitive data", in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, IEEE, 2017, pp. 194–196.

[25] Kubernetes.io. "What is kubernetes". (2014), [Online]. Available: `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/` (visited on 05/06/2024).

[26] G. Darwesh, J. Hammoud, and A. Vorobeva, "Security in kubernetes: Best practices and security analysis", vol. 2, pp. 63–69, Jun. 2022.

[27] S. Pal and T. Elhance, "Securing kubernetes: Challenges and mitigation techniques", in *Proceedings of the International Conference on Innovative Computing & Communications (ICICC)*, 2022, pp. 1–6.

[28] A. A. Tripathi, "Attacking and defending kubernetes", Ph.D. dissertation, Dublin Business School, 2024, p. 56. [Online]. Available: `https://esource.dbs.ie/server/api/core/bitstreams/62cbffaa-d0b8-4a95-8030-ef0b9093d1d2/content` (visited on 05/09/2024).

[29] P. Ray and J. Matta, "Securing kubernetes secrets with canary tokens", in *Proceedings of the ACM International Workshop on Software Supply Chains and Attack Surfaces*, 2021, pp. 20–27.

[30] D. Jain and R. Tripathi, "Securing kubernetes workloads with canary tokens", in *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*, IEEE, 2021, pp. 1–6.

[31] D. Sharma, A. Singh, S. Chitkara, and T. Sharma, "Honeypot networks in deception technology for iot devices", in *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, 2023, pp. 1156–1162.

[32] S. Beguelin and R. Eisenbarth Vaughan, *Detecting data breaches with canary tokens*, SANS Institute, 2020.

[33] Kubernetes.io. "Kubernetes components". (2022), [Online]. Available: `https://kubernetes.io/docs/concepts/overview/components/` (visited on 05/06/2024).

[34] M. Panagiotis, "Attack methods and defenses on kubernetes", Ph.D. dissertation, University of Piraeus (Greece), Jun. 2020, p. 91. [Online]. Available: `https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/12888/Mytilinakis_mte1822.pdf` (visited on 05/09/2024).

[35] Y. Diogenes and E. Ozkaya, *Cybersecurity - Attack and Defense Strategies: Counter modern threats and employ state-of-the-art tools and techniques to protect your organization against cybercriminals*. Packt Publishing Ltd, Dec. 2019, p. 634, ISBN: 978-1838827793.

[36] D. DeJonghe, *NGINX Cookbook: Advanced Recipes for High-Performance Load Balancing*. O'Reilly Media, Jun. 2022, p. 202, ISBN: 978-1098126247.

[37] T. Canary. "Github: Thinkst/canarytokens". (2015), [Online]. Available: `https://github.com/thinkst/canarytokens?tab=readme-ov-file` (visited on 05/04/2024).

[38] Apache.org. “Htpasswd: Manage user files for basic authentication (apache http server version 2.4)”. (2023), [Online]. Available: https://httpd.apache.org/docs/current/programs/htpasswd.html (visited on 05/04/2024).

[39] Let’s-Encrypt. “About certbot”. (2021), [Online]. Available: https://certbot.eff.org/pages/about (visited on 05/05/2024).

[40] Redis-Trust. “Redis as an in-memory data structure store quick start guide”. (2020), [Online]. Available: https://redis.io/docs/latest/develop/get-started/data-store/ (visited on 05/05/2024).

[41] T. Canary. “Github: Thinkst/canarytokens”. (2015), [Online]. Available: https://github.com/thinkst/canarytokens/wiki (visited on 05/04/2024).

Appendix A ct2rds.py program

```
\begin{lstlisting}
import os
import redis
import time
import json
import datetime
import logging


logging.basicConfig(filename='logs/incident.log', level=logging.INFO)


def current_timestamp():
    return int(time.time())


def convert_timestamp_to_datetime(timestamp):
    return datetime.datetime.fromtimestamp(timestamp).strftime('%Y-%m
       -%d %H:%M:%S')


def process_incidents(redis_client, timestamp, processed_timestamps):
    token_keys = redis_client.keys('canarydrop:*')
```

```
    for token_key in token_keys:
        triggered_list_key = f"{token_key} triggered_list"
        triggered_list = redis_client.hget(token_key, "triggered_list
            ")

        if triggered_list:
            triggered_list = json.loads(triggered_list)
            for incident_timestamp, incident_data in triggered_list.
                items():
                incident_datetime_str = convert_timestamp_to_datetime(
                    float(incident_timestamp))

                if incident_timestamp not in processed_timestamps and
                    float(incident_timestamp) > timestamp:
                    log_entry = f"{incident_datetime_str}: {
                        incident_data}"
                    logging.info(log_entry)
                    print(f"Logged incident: {log_entry}")
                    processed_timestamps.add(incident_timestamp)

def main():
    start_timestamp = current_timestamp()
    processed_timestamps = set()
    redis_host = os.environ.get("redis_host")
    redis_port = os.environ.get("redis_port",6739)
    redis_client = redis.StrictRedis(host=redis_host, port=redis_port
        , decode_responses=True)
```

```
    try:
        while True:
            process_incidents(redis_client, start_timestamp,
                processed_timestamps)
            time.sleep(20)

    except KeyboardInterrupt:
        print("Script terminated by user.")

if __name__ == "__main__":
    main()
\end{lstlisting}
```

Appendix B waf.py program

```
\begin{lstlisting}
import os
import re
import time
import subprocess


log_file_path = '/home/tushar/canarytokens-docker/logs/incident.log'


main_tf_path = '/home/tushar/waf/main.tf'


ip_pattern = r'"ip":\s*"([^"]*)"'


bash_commands = [
    'export $(grep -v "^#" .env | xargs)',
    'terraform init',
    'terraform plan -var "api_id=$API_ID" -var "api_key=$API_KEY"',
    'terraform apply -var "api_id=$API_ID" -var "api_key=$API_KEY" -
       auto-approve'
]
```

```
def extract_ips_from_log():
    ips = set()
    with open(log_file_path) as file:
        for line in file:
            matches = re.findall(ip_pattern, line)
            ips.update(matches)
    return ips


def update_main_tf(ips):
    with open(main_tf_path, 'r') as file:
        lines = file.readlines()

    for i, line in enumerate(lines):
        if 'ips' in line:
            indent = len(line) - len(line.lstrip())
            ips_str = ', '.join(f'"{ip}"' for ip in ips)
            new_line = f'{" " * indent}"ips": [{ips_str}]\n'
            lines[i] = new_line
            break

    with open(main_tf_path, 'w') as file:
        file.writelines(lines)


def execute_bash_commands():
    for command in bash_commands:
        subprocess.run(command, shell=True)
```

```
def main():
    previous_ips = set()
    while True:
        ips = extract_ips_from_log()
        new_ips = ips - previous_ips
        if new_ips:
            update_main_tf(ips)
            execute_bash_commands()
            previous_ips = ips
        time.sleep(15)


if __name__ == '__main__':
    main()
\end{lstlisting}
```

Appendix C offline.sh script

```
\begin{lstlisting}
#!/bin/bash


# Define monitored files
files=(
    "/path/to/a/token1"
    "/other/path/to/a/token2"
    "/another/path/to/a/token3"
    "/path/to/a/token4"
    "/path_to/a/token5"
)


log_file="/home/tushar/offline/offline_incident.log"


touch "$log_file"


check_duplicate_entry() {
    local datetime=$1
    local user=$2
    local operation=$3
```

```
    local path=$4

    if grep -q "$datetime.*$user.*$operation.*$path" "$log_file";
        then
        return 1
    else
        return 0
    fi
}

monitor_files() {
    inotifywait --monitor --format "%e %w%f" \
        -e access -e close_write -e modify -e move -e create -e
            delete -e delete_self -e move_self -e attrib \
        "${files[@]}" | \
    while read -r events file; do
        # Get the current date and time
        datetime=$(date "+%Y-%m-%d %H:%M:%S")
        # Get the username and hostname
        user=$(whoami)
        hostname=$(hostname)

        if check_duplicate_entry "$datetime" "$user@$hostname" "
            $events" "$file"; then
            echo "$datetime $user@$hostname $events $file" >> "
                $log_file"
        fi
```

```
    done
}


sleep 10


monitor_files
\end{lstlisting}
```