DOYENSEC

Security Auditing Report

Canary Tokens OSS

Prepared for: **Thinkst Applied Research**
Prepared by: **Viktor Chuchurski, Francesco Lacerenza**
Date: **07/22/2024**

Table of Contents

Revision History

Version	Date	Description	Author
1	04/29/2024	First release of the final report	Viktor Chuchurski, Francesco Lacerenza
2	04/30/2024	Peer review	Luca Carettoni
3	07/19/2024	Retesting	Viktor Chuchurski
4	07/22/2024	Peer review before external publication	Anthony Trummer

Contacts

Company	Name	Email
Thinkst Applied Research	Marco Slaviero	marco@thinkst.com
Doyensec, LLC	John Villamil	john@doyensec.com
Doyensec, LLC	Luca Carettoni	luca@doyensec.com

Executive Summary

Overview

Thinkst Applied Research engaged Doyensec to perform a security assessment of their Canary Tokens OSS. The project commenced on 04/22/2024 and ended on 04/26/2024 requiring two (2) security researchers. The project resulted in nine (9) findings of which one (1) was rated as medium severity.

The project consisted of a manual web application security assessment.

Testing was conducted remotely from Doyensec's EMEA and US offices.

Scope

Through meetings with Thinkst Applied Research the scope of the project was clearly defined. The agreed upon assets are listed below:

- https://<REDACTED>.com
- https://<REDACTED>.net

The testing took place in a testing environment using the latest version of the software at the time of testing. In detail, this activity was performed on the following releases:

- **canarytokens**
  - `846c6a063a008042627de189f673a7efc47c7d40`
- **canarytokens-docker**
  - `684069b02959a58e7276de14f34e14202079e6ac`

Scoping Restrictions

During the engagement, Doyensec did not encounter any difficulties in testing the application.

The Thinkst team was very responsive in debugging the issues that surfaced during the test, ensuring a smooth assessment.

Findings Summary

Doyensec researchers discovered and reported nine (9) vulnerabilities in the Canary Token OSS component.

While most of the issues were departures from best practices and low-severity flaws, Doyensec identified one (1) issue rated as medium severity.

It is important to reiterate that this report represents a snapshot of the environment's security posture at a point in time.

The findings included the possibility to exploit different types of denial of service attacks. The worst case reported involved the exploitation of a known issue in the "python-multipart" library (`CAN-Q224-1`) to block the service. Moreover, best practices and least privilege principle violations (`CAN-Q224-2`, `CAN-Q224-3`) were reported in the AWS tokens infrastructure. Finally, low-impact Cross-Site-Scripting (XSS) vulnerabilities (`CAN-Q224-5`, `CAN-Q224-6`) were identified in multiple canaries.

Overall, the security posture of the Internet-facing APIs was found to be in line with industry best practices.

Recommendations

The following recommendations are proposed based on studying Thinkst's security posture and the vulnerabilities discovered during this engagement.

Short-term improvements

- Work on mitigating the discovered vulnerabilities. You can use **Appendix B** - Remediation Checklist to make sure that you have covered all areas

Long-term improvements

- As per the current threat model, the analyzed OSS is applying canary-based authentication, leaving unauthenticated access to the dashboard and the canaries creation operation. The tests evidenced multiple issues exploitable against the dashboard during the creation step (see `CAN-Q224-9`, `CAN-Q224-8` and `CAN-Q224-7`). Implement additional opt-in authorization mechanisms to protect features in the dashboard related to limited resources or SSRF opportunities

Methodology

Overview

Doyensec treats each engagement as a fluid entity. We use a standard base of tools and techniques from which we built our own unique methodology. Our 30 years of information security experience has taught us that mixing offensive and defensive philosophies is the key to standing against threats. Thus we recommend a *white-box* approach combining dynamic fault injection with an in-depth study of the source code to maximize the ROI on bug hunting.

During this assessment, we have employed standard testing methodologies (e.g., OWASP Testing guide recommendations), as well as custom checklists, to ensure full coverage of both code and vulnerability classes.

Setup Phase

Thinkst Applied Research provided access to the online environment and access to relevant source code repositories via GitHub.

In addition to the online environment, Doyensec compiled and ran the application locally, using instructions in the repository itself.

Tooling

When performing assessments, we combine manual security testing with state-of-the-art tools in order to improve efficiency and efficacy of our effort.

During this engagement, we used the following tools:

- Burp Suite
- VisualStudio Code
- Curl, netcat and other Linux utilities

Web Application and API Techniques

Web assessments are centered on the data sent between clients and servers. In this realm, the principle audit tool is Burp Suite. However, we also use a large set of custom scripts and extensions to perform specific audit tasks. We focus on authorization, authentication, integrity and trust. We study how data is interpreted, parsed, stored, and relayed between producers and consumers.

We subvert the client with malicious data through reflected and DOM based Cross Site Scripting and by breaking assumptions in trust. We test the server endpoints for injection style flaws including, but not limited to, SQL, template, XML, and command injection flaws. We look at each request and response pair for potential Cross Site Request Forgery and race conditions. We study the application for subtle logic issues, whether they are authorization bypasses or insecure object references. Session storage and retrieval is scrutinized and user separation is thoroughly tested.

Web security is not limited to popular bug titles. Doyensec researchers understand the goals and needs of the application to find ways of breaking the assumed control flow.

Source Code Auditing

Source code reviews are critical to understand the true state of our clients' applications. Removing the layers of abstraction and focusing on the raw application code allows us to obtain an unobstructed view of vulnerabilities, which could otherwise go undetected. Unlike black-box testing, which can be impeded by things like network equipment, security devices and services, rate limits, hard to find routes or inputs, code obfuscation and/or minimization and the fear of creating downtime, source code reviews reveal the true software quality. This gives our clients the confidence to deploy their software anywhere, knowing we've found the hidden vulnerabilities.

We pride ourselves on hiring engineers who not only break applications, but have experience building them as well. This differentiates Doyensec from many other consulting firms, as it allows us to immerse ourselves in the applications we test and create a more comprehensive threat model, ultimately revealing more impactful issues and in greater numbers.

Our methodology consists of several stages outlined in the table below. During this process, we typically begin by thoroughly reviewing the code to understand the composition of the application, its uses, data flows and its authentication and authorization structures. This provides the context needed to evaluate any potential bugs we might encounter in later steps.

Next, we use our custom threat model to trace inputs through the code and look for problem areas. These could be anything from typical bugs like injection style vulnerabilities and authorization bypasses, or subtle business-logic flaws and unsafe function usages, which are not as easily detected via automation. We also examine the frameworks used within the application to ensure they are configured as securely as possible for the given context.

Only once we really understand the application, will we deploy our custom scripts, created by our team over many years of experience. These parse the code and identify hotspots, where we have seen vulnerabilities manifested in the past. Our scripts typically identify coding errors and misconceptions about functionality during development and are the product of bespoke security research by our team.

Finally, we leverage a curated set of customized open-source tools to complete our analysis. We start with those specifically designed for the application's languages and components, eventually moving into more generalized tools. After assessing the application with these tools, we manually validate any findings they report and filter out the noise, delivering only actionable results to our clients.

Project Findings

The table below lists the findings with their associated ID and severity. The severity ranking and vulnerability classes are defined in **Appendix A** at the end of this document. The vulnerability class column groups the entry into a common category, while the status column refers to whether the finding has been fixed at the time of writing.

This table is organized by time of discovery. The issues at the top were found first, while those at the bottom were found last. Presenting the table in this fashion has a number of benefits. It inherently shows the path our auditing took through the target and may also reveal how easy or difficult it was to discover certain findings. As a security engagement progresses, the researchers will gain a deeper understanding of a target which is also shown in this table.

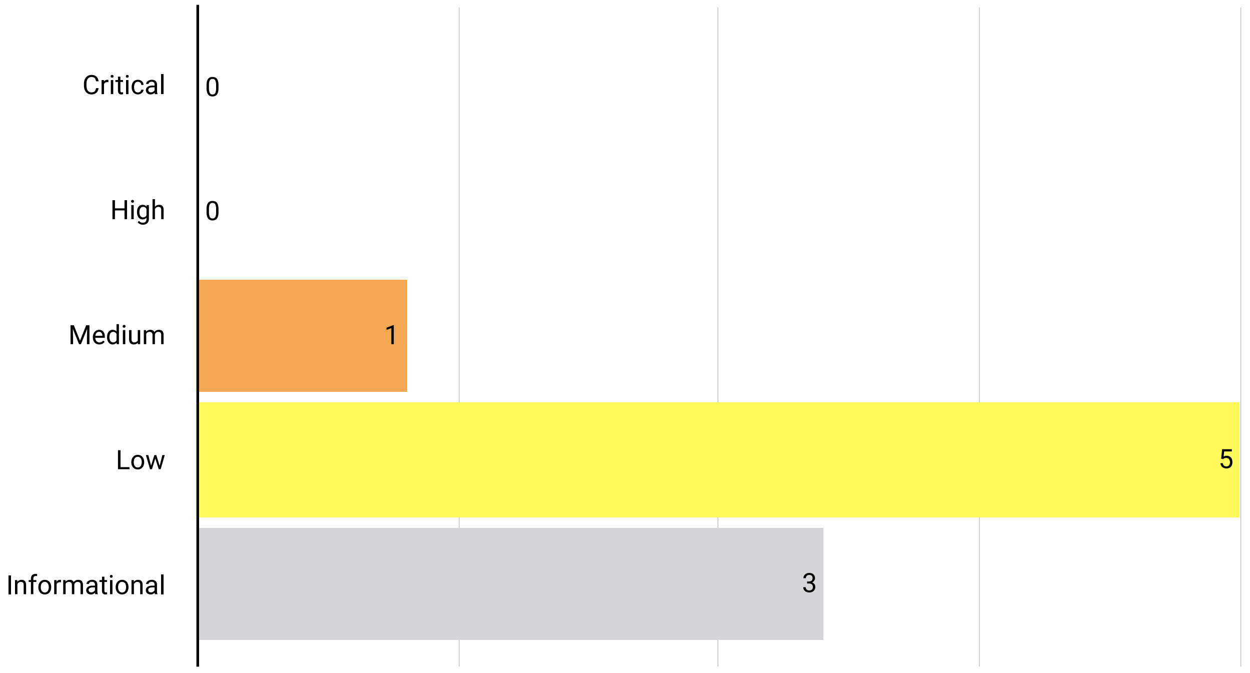
Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
CAN-Q224-1	**ReDoS via Outdated python-multipart Library**	**Denial Of Service (DoS)**	**Medium**	**Closed**
Comment	The issue was mitigated by updating the vulnerable library to the latest version.			
CAN-Q224-2	**Overprivileged AWSProcessTokenLogsRole Lambda Role**	**Insecure Design**	**Low**	**Closed**
Comment	The `AWSProcessTokenLogsRole` role was updated to allow decryption of specific AWS resources only.			
CAN-Q224-3	**Unencrypted Lambda Environment Variables**	**Insecure Design**	**Informational**	**Risk Accepted**
Comment	Risk was accepted. **Thinkst's comment**: "The Lambda function runs in a single-purpose AWS account, limiting the impact of unauthorized access to unencrypted environment variables."			
CAN-Q224-4	**Missing Authorization in create_user_api_tokens**	**Insufficient Authorization**	**Informational**	**Risk Accepted**
Comment	Risk was accepted. **Thinkst's comment**: "The Lambda function is accessible through a random hostname, making discoverability and direct access difficult for attackers."			
CAN-Q224-5	**Stored Cross-Site Scripting on "Cloned Site" Token**	**Cross-Site Scripting (XSS)**	**Low**	**Closed**
Comment	The issue was present in the application's old UI, which is no longer available.			
CAN-Q224-6	**Stored Cross-Site Scripting in "Slow Redirect" Token Page**	**Cross-Site Scripting (XSS)**	**Informational**	**Closed**

ID	Title	Vulnerability Class	Severity	Status
Comment	The issue has been mitigated by forcing the HTTP(S) protocol for redirect URLs.			
CAN-Q224-7	**Potential Denial of Service via Unlimited Creation of "AWS" Canary Tokens**	**Denial Of Service (DoS)**	**Low**	**Risk Accepted**
Comment	Risk was accepted. **Thinkst's comment**: Addressing the issue would require introducing "… a complex user model, which will introduce significantly more risk to us…"			
CAN-Q224-8	**Potential Denial of Service via Unlimited Creation of "Web Image" Canary Tokens**	**Denial Of Service (DoS)**	**Low**	**Risk Accepted**
Comment	Risk was accepted. **Thinkst's comment**: Addressing the issue would require introducing "… a complex user model, which will introduce significantly more risk to us…"			
CAN-Q224-9	**Blind SSRF via Token Webhook**	**Server-Side Request Forgery**	**Low**	**Closed**
Comment	The SSRF was mitigated by updating the application to use the "advocate" library to make HTTP requests.			

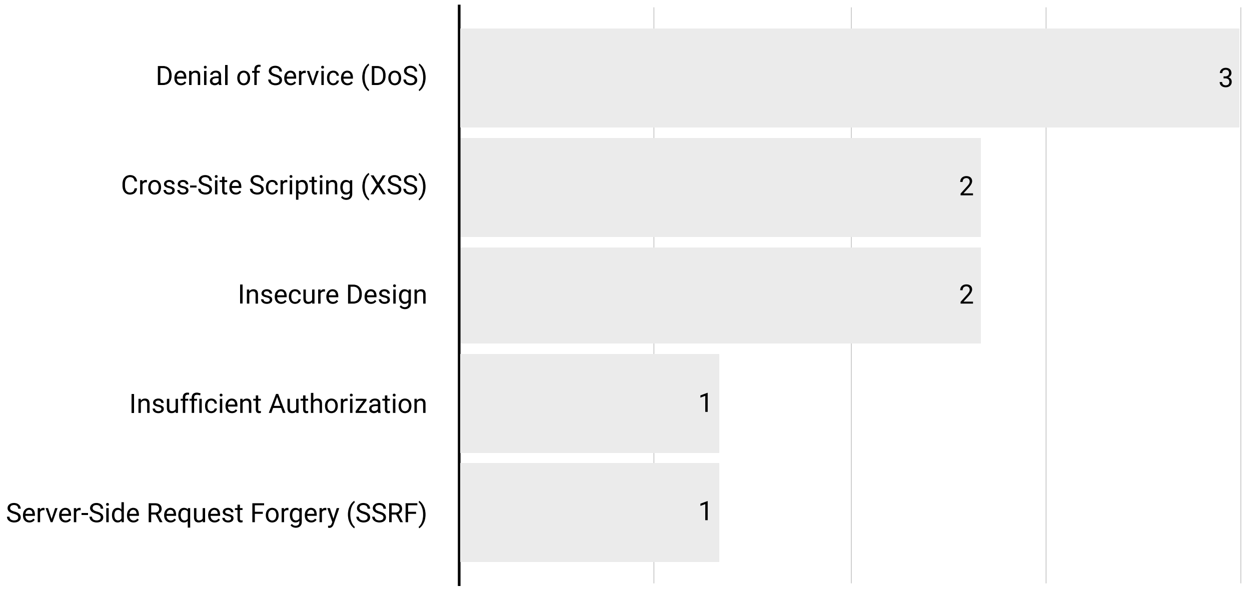
Findings per Severity

The table below provides a summary of the findings per severity.



Findings per Type

The table below provides a summary of the findings per vulnerability class.



CAN-Q224-1. ReDoS via Outdated “python-multipart” Library	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	Application Dependencies
Status	Closed

Description

While reviewing the Canary Tokens OSS source code, Doyensec discovered some dependencies are affected by known vulnerabilities. More specifically, the “python-multipart”[1] dependency was found to be affected by a regular expression (Regex) denial-of-service (DoS) attack or ReDoS.

Namely, parsing additional parameters supplied via the value of the `Content-Type` HTTP header will lead to the library using a vulnerable regex. If a malicious value is supplied, the library will try to match the value using the vulnerable regex, which will exhaust system resources causing the application to stall.

Reproduction Steps

To verify the issue, make the following HTTP request to the application:

```
POST /d3aece8093b71007b5ccfedad91ebb11/generate HTTP/1.1
Host:<REDACTED>.com
Content-Type: application/x-www-form-urlencoded

email=viktor%2b1%40gmail.com&webhook_url=http%3a//
<REDACTED>.com%3ffast&redirect_url=http%3a//
<REDACTED>.com%3fredirect&memo=123&token_type=fast_redirect
```

Note the time it takes for the application server to respond. Next, reply to the same request with the following `Content-Type` header value:

```
Content-Type: application/x-www-form-urlencoded; !="\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\
```

Note that the time to produce a response has significantly increased. During testing, the above payload resulted in a 5 second response time. That can be further increased by adding more back-slashes (`\`) to the header value.

1 https://github.com/Kludex/python-multipart

Impact

Medium. If an attacker gains access to the application, abusing this issue will allow them to perform a denial-of-service attack on the system. The attack will use a large amount of the system's resources, rendering it unresponsive and unable to process canary token callbacks.

In extreme cases, the attack may lead to a full system crash.

Complexity

Low. Successful exploitation only requires access to the application and basic knowledge of web application security. Because the application's source code is public, discovering the vulnerability is also trivial. Overall, we consider the complexity to be low.

Remediation

Update the "python-multipart" dependency to the latest secure version.

To mitigate the vulnerability, update the "python-multipart" library to the latest know secure version. At the time of testing, that is version 0.0.9.

Resources

- Google, "OSV Scanner"
  https://github.com/google/osv-scanner

- Kludex-GitHub Advisory, "Content-Type Header ReDoS",
  https://github.com/Kludex/python-multipart/security/advisories/GHSA-2jv5-9r88-3w3p

CAN-Q224-2. Overprivileged AWSProcessTokenLogsRole Lambda Role	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	canarytokens-Feature_branch_New_UI/aws-token-infra/awsid.tf:336
Status	Closed

Description

Implementing the principle of least privilege, for roles and policies used by service instances in an infrastructure, is crucial for security. Roles and policies are typically used by AWS services, EC2 instances, Lambda functions, and other resources to access other AWS resources and services, while accomplishing tasks dictated by internal application logic.

The `AWSProcessTokenLogsRole` role was defined with permissions that extend beyond its immediate operational requirements, thus introducing unnecessary risk to the system.

In particular, the role has the `kms:Decrypt` permission on any (“*”) resource, hence allowing arbitrary decryption within the account.

As a result, a potential attacker who compromises the reported roles or users will be able to execute any possible action, in the targeted environment, by exploiting the escalation patterns. Consequently, the privacy and integrity of sensitive data and the platform availability could be fully compromised.

Reproduction Steps

N/A. This is a source code finding.

Impact

High. An attacker with AWS IAM knowledge could easily exploit the loose permission to exfiltrate other keys in the infrastructure.

Complexity

High. As currently implemented, the attacker is more likely to be an internal threat actor.

Remediation

We recommend applying resource limitation on the `kms:Decrypt` **operation on the** `AWSProcessTokenLogsRole` **role. New canary keys should be either tagged or named to restrict their namespace within the role.** By doing so, the role will not be exploitable to decrypt arbitrary KMS objects in the account.

CAN-Q224-3. Unencrypted Lambda Environment Variables

Severity	**Informational**
Vulnerability Class	Insecure Design
Component	AWS Lambda CreateUserAPITokens
Status	Risk Accepted

Description

Lambda functions often utilize environment variables to store sensitive configuration information such as API keys, database credentials, or other secrets. However, it has been observed that some environment variables within Lambda functions are stored without encryption, posing a significant security risk. Unencrypted environment variables can be exposed, potentially leading to data breaches or unauthorized access to critical resources.

In particular, the AWS lambda function `CreateUserAPITokens` stores the `SLACK_WEBHOOK_URL` as an environment variable with default encryption settings.

As a security best practice, variables containing secrets should be encrypted with a custom AWS KMS key and with encryption in transit enabled.

Reproduction Steps

The exposure of the environment variables can be confirmed from any AWS user with `GetFunctionConfiguration` capabilities.

Example command:

```
❯ aws lambda get-function-configuration --function-name CreateUserAPITokens --region
us-east-1
{
    "FunctionName": "CreateUserAPITokens",
    "FunctionArn": "arn:aws:lambda:us-east-1:<REDACTED>:function:CreateUserAPITokens",
    "Runtime": "python3.9",
    "Role": "arn:aws:iam::<REDACTED>:role/AWSTokenRole",
    "Handler": "lambda_function.lambda_handler",
    "CodeSize": 1508,
    "Description": "",
    "Timeout": 60,
    "MemorySize": 128,
    "LastModified": "2024-04-23T15:45:20.000+0000",
    "CodeSha256": "9Sr/G3hUpMJBgBWk24Drq0Av2wPjB8qmPU+Uq2jllbM=",
    "Version": "$LATEST",
    "Environment": {
        "Variables": {
            "SLACK_WEBHOOK_URL": “REDACTED_PLAINTEXT_WEBHOOK”
        }
```

Impact

Medium. An attacker compromising the lambda execution or an internal threat actor with `GetFunctionConfiguration` capabilities could read the plaintext environment variables, exposing the Slack webhook.

Complexity

High. The attacker needs to either be able to exploit the function and obtain RCE or be an internal actor with `GetFunctionConfiguration` permissions.

Remediation

As an infrastructure security best practice, AWS Secrets Manager with a custom key and encryption in transit should be applied while storing sensitive environment variables.

Resources

- Amazon Web Services, “Securing environment variables”
  https://docs.aws.amazon.com/lambda/latest/dg/configuration-envvars.html

CAN-Q224-4. Missing Authorization in create_user_api_tokens	
Severity	**Informational**
Vulnerability Class	Insufficient Authorization
Component	canarytokens/aws-token-infra/awsid.tf:149
Status	Risk Accepted

Description

API Gateway functions often serve as entry points for various services and applications, and without adequate authorization controls, they become vulnerable to unauthorized access by malicious actors. In the token infrastructure, an AWS API Gateway function was deployed without proper authorization mechanisms in place, allowing unauthorized access.

See the definition for the `create_user_api_tokens` endpoint at `canarytokens/aws-token-infra/awsid.tf:149`

```
[REDACTED]…
resource "aws_api_gateway_method" "create_user_api_tokens" {
   rest_api_id   = aws_api_gateway_rest_api.create_user_api_tokens.id
   resource_id   = aws_api_gateway_resource.create_user_api_tokens.id
   http_method   = "ANY"
   authorization = "NONE"
}
```

The missing authorization allows external attackers to call the endpoint and potentially exhaust the AWS keys quota as described in `CAN-Q224-7`.

Reproduction Steps

N/A

Impact

Medium. The attacker could abuse the function to exhaust the number of AWS Keys in the account, de facto preventing the creation of new legit users.

Complexity

High. The attacker needs to be able to enumerate, guess, or exfiltrate the actual AWS Gateway URL for the unauthorized function.

Remediation

Utilize authentication mechanisms such as AWS Identity and Access Management (IAM) policies, API keys, AWS Cognito, or OAuth tokens to authenticate and authorize users accessing the API Gateway function.

Resources

- Amazon Web Services, “Controlling and managing access to a REST API in API Gateway” https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-control-access-to-api.html

CAN-Q224-5. Stored Cross-Site Scripting on "Cloned Website" Token

Severity	**Low**
Vulnerability Class	Cross-Site Scripting (XSS)
Component	canarytokens/templates/manage_new.html:743
Status	Closed

Description

Cross-Site Scripting (also referred to as XSS) occurs when a web application accepts malicious code (usually JavaScript) as input from an attacker, which is subsequently executed in a victim's browser. Since the browser executes the code in the victim's session context, it allows the attacker to access any cookies or session data retained by the browser. It is also possible to hijack the browser itself. The attacker may also modify arbitrary content on the page presented to the user. The attack is possible because a browser, by default, cannot distinguish between the malicious code mentioned above and legitimate code from the web server.

When the application accepts and saves an attacker's payload to persistent storage, which can later be served to victims through normal usage of the application, we categorize the vulnerability as *stored* XSS (as opposed to *reflected*). In this particular case, the attack may target any user of the platform who can see the malicious content. For this reason, we consider the vulnerability's severity higher than in the reflected case.

When a "cloned website" canary token's details are displayed on the application's old UI, the following code is used to place the generated canary JavaScript code in its HTML container:

```
$('#result_cloned_website_obfuscated')
  .append(obfuscateClonedWebJs(decodeClonedSiteJs(`{{canarydrop.get_cloned_site_j
avascript(force_https)}}`)));
```

This snippet is part of a Jinja template (`canarytokens/templates/manage_new.html`) used to render the entire web page. The canary JavaScript is rendered on the page using the double bracket (`{{`) directive, which will perform appropriate output HTML encoding. However, the JavaScript is rendered in a `script` tag context, making the applied encoding irrelevant and allowing for XSS.

Reproduction Steps

Use the following steps to reproduce the issue:

1. Navigate in the application and choose to create a new "Cloned Website" token
2. Supply ``any.domain`+alert(document.location)+` `` as the cloned website's URL and verify that the token was successfully created
3. Navigate to the token's management page
4. Verify that an alert dialog was shown with the domain of the application

Impact

Low. If successfully exploited, the issue can allow malicious JavaScript code to be executed in the context of the victim's sessions. Since the attack does not require user interaction, apart from visiting the token's management page, we consider the severity as higher than the reflected case.

While arbitrary JavaScript execution will allow the attacker to view the token's history, and in specific scenarios allow them to delete or disable the token, as the creator of the token, they would already have access to that functionality. Overall, we consider the impact to be low.

Complexity

Medium. Finding and exploiting the issue requires basic knowledge of web application security. The open source nature of the application significantly increases the likelihood of discovering the vulnerability.

Successful exploitation also requires a degree of social engineering to get the victim to navigate to the malicious token's page. One way of doing this is by supplying the victim's email when creating the token and subsequently triggering it. This will result in an email with a link to the token's page being emailed to the victim. The email will be sent by the application itself, increasing the likelihood of the being visited.

Overall, we consider the complexity to be low to medium.

Remediation

XSS vulnerabilities can only be prevented with a combination of:

- Context-aware output escaping/encoding,
- Strict user input validation and sanitization, filtering meta-characters[2] from user input,
- Validating that URLs dynamically created using user-controlled data, (e.g., `HREF`, `IFRAME`, etc.) only allow the intended schemes (e.g., http:, https:) and forbid specifying the `javascript:` scheme,
- Proper implementation and configuration of the Content Security Policy,
- Proper implementation of the `X-XSS-Protection` header, and/or
- Setting the `HTTPOnly` flag on sensitive cookies

Perform input validation and the appropriate output encoding on the `clonedsite` parameter.

In this particular scenario, on the application level, the vulnerability can be mitigated in multiple ways:

- Implement URL validation on the "cloned website" parameter, and reject any invalid values
- Implement appropriate output encoding for the JavaScript context using Jinja's "tojson"[3] filter

2 https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

3 https://jinja.palletsprojects.com/en/3.0.x/templates/#jinja-filters.tojson

Resources

- OWASP, "Cross-site Scripting (XSS)"
  https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

- OWASP, “XSS Prevention Cheat Sheet"
  https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.md

CAN-Q224-6. Stored Cross-Site Scripting in "Slow Redirect" Token Page	
Severity	**Informational**
Vulnerability Class	Cross-Site Scripting (XSS)
Component	canarytokens/templates/ browser_scanner.html:501
Status	Closed

Description

Cross-Site Scripting (also referred to as XSS) occurs when a web application accepts malicious code (usually JavaScript) as input from an attacker, which is subsequently executed in a victim's browser. Since the browser executes the code in the victim's session context, it allows the attacker to access any cookies or session data retained by the browser. It is also possible to hijack the browser itself. The attacker may also modify arbitrary content on the page presented to the user. The attack is possible because a browser, by default, cannot distinguish between the malicious code mentioned above and legitimate code from the web server.

When the application accepts and saves an attacker's payload to persistent storage, which can later be served to victims through normal usage of the application, we categorize the vulnerability as *stored* XSS (as opposed to *reflected*). In this particular case, the attack may target any user of the platform who can see the malicious content. For this reason, we consider the vulnerability's severity higher than in the reflected case.

The "slow redirect" canary token generates a webpage which will ultimately navigate the user to a user-supplied URL. Navigation is performed by the following snippet:

```
{% if redirect_url %}
window.location = '{{redirect_url}}';
{% endif %}
```

The value of the `redirect_url` doesn't undergo any input validation and is rendered on the page using HTML output encoding, which is incorrect within a `script` tag context. This allows for arbitrary JavaScript code to be supplied and executed - then the token is triggered.

Reproduction Steps

Use the following steps to reproduce the issue:

1. Navigate in the application and choose to create a new "Slow Redirect" token
2. Supply `javascript:alert(document.location)` as the redirect URL and verify that the token was successfully created
3. Navigate to the token's canary URL
4. Verify that an alert dialog was shown with the domain of the application

Impact

N/A

Complexity

Low. Access to the application and basic knowledge of web application security is required to find and exploit this issue. The open source nature of the application significantly increases the likelihood of discovering the vulnerability.

Remediation

XSS vulnerabilities can only be prevented with a combination of:

- Context-aware output escaping/encoding,
- Strict user input validation and sanitization, filtering meta-characters[4] from user input,
- Validating that URLs dynamically created using user-controlled data, (e.g., `HREF`, `IFRAME`, etc.) only allow the intended schemes (e.g., http:, https:) and forbid specifying the `javascript:` scheme,
- Proper implementation and configuration of the Content Security Policy,
- Proper implementation of the `X-XSS-Protection` header, and/or
- Setting the `HTTPOnly` flag on sensitive cookies

Perform input validation and the appropriate output encoding on the `redirect_url` parameter.

In this particular scenario, on the application level, the vulnerability can be mitigated in multiple ways:

- Implement URL validation on the "cloned website" parameter, and reject any invalid values
- Implement appropriate output encoding for the JavaScript context using Jinja's "tojson"[5] filter

Resources

- OWASP, "Cross-site Scripting (XSS)"
  https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

- OWASP, "XSS Prevention Cheat Sheet"
  https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.md

4 https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

5 https://jinja.palletsprojects.com/en/3.0.x/templates/#jinja-filters.tojson

CAN-Q224-7. Potential Denial of Service via Unlimited Creation of "AWS" Canary Tokens

Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	AWS Keys Canary
Status	Risk Accepted

Description

As described in the Canary documentation:

```
The AWS API token provides you with a set of AWS API keys. Leave them in private
code repositories, leave them on a developer's machine, or anywhere else API keys
would be expected. An attacker who stumbles onto them will believe they are the
keys to your cloud infrastructure. If they are used via the AWS API at any point,
you will be alerted.
```

According to the current threat model, Canary tokens creation is not restricted and the dashboard is exposed as an unauthenticated service. Consequently, the creation of AWS keys could easily reach the limit allowed for the AWS quota, set by the customer or Thinkst, in the account setup.

As a result, an attacker with access to the public dashboard could exhaust the AWS keys for the account and prevent the creation of legitimate canaries or users' keys for the target account.

Reproduction Steps

In order to exhaust the quota limit, it is sufficient to automate the creation of AWS Canary Keys. As an example, it would be sufficient to use the Intruder functionality of Burp Suite Proxy.

Since the testing environment was linked to the production AWS account used by Thinkst, Doyensec did not reproduce it to avoid availability issues for the legitimate users.

Impact

Medium. Attackers can prevent legitimate users from obtaining a valid key or AWS key canary.

Complexity

Low. It is sufficient to find the exposed Canary dashboard (unauthenticated) and automate the creation of AWS key canaries. Basic web hacking skills are required.

Remediation

Provide an option to restrict the number of AWS Key tokens.

Consider providing an option for users to configure a limit of the token type or a secret key to be submitted during the creation. Requiring a configurable secret key to create tokens could be applied to other canaries with limited resources (see `CAN-Q224-8`).

Resources

- Amazon Web Services, “IAM and AWS STS quotas” https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_iam-quotas.html

CAN-Q224-8. Potential Denial of Service via Unlimited Creation of “Web Image” Canary Tokens

Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	canarytokens/frontend/app.py:1534, :1607
Status	Risk Accepted

Description

When a user creates a new “Web Image” canary token, they are prompted to upload an image file. The application will perform validation on the uploaded file, verifying that it has one the the allowed file extensions and that its size is lower than the application’s pre-configured limit:

```
if len(filebody) > frontend_settings.MAX_UPLOAD_SIZE:
    max_size = str(frontend_settings.MAX_UPLOAD_SIZE / (1024 * 1024))
    raise HTTPException(
        status_code=400,
        detail=f"File too large. File size must be < {max_size} MB.",
    )
```

If the file passes validation, it is stored on disk under a random file path. While required in the creation request, the file is not used for creating or triggering the token.

The lack of any upper limits on the number of “Web Image” tokens created or the number of images stored, can allow attackers to create a large number of tokens, filling up the application’s disk space, rendering the application unresponsive and potentially unable to create new tokens, due to a lack of memory.

Reproduction Steps

The issue can be replicated by replaying the “Web Image” token creation request a large number of times and verifying that all images have been stored on disk and no limit on the number of created tokens was reached:

```
POST /d3aece8093b71007b5ccfedad91ebb11/generate HTTP/1.1
Host:<REDACTED>.com
Content-Type: multipart/form-data;
boundary=---------------------------279696661313754475963624687173

-----------------------------279696661313754475963624687173
Content-Disposition: form-data; name="email"

viktor+1@doyensec.com
-----------------------------279696661313754475963624687173
Content-Disposition: form-data; name="webhook_url"
```

```
<WEBHOOK_ADDRESS>
-----------------------------27969666131375447596362468717З
Content-Disposition: form-data; name="memo"

demo_memo
-----------------------------279696661313754475963624687173
Content-Disposition: form-data; name="web_image"; filename="duckie.jpg"
Content-Type: image/jpeg

<IMAGE CONTENTS>
-----------------------------279696661313754475963624687173
Content-Disposition: form-data; name="token_type"

web_image
-----------------------------279696661313754475963624687173--
```

Impact

Potentially high. If successfully exploited, flooding the application with a large number of uploads will fill up its disk space. In the scenario where the disk is full, the application may become unable to process incoming token callbacks.

Complexity

High. While performing the attack only required access to the application, successful exploitation is based on the pre-defined file size limits and the disk space allocated to the application.

Remediation

Provide an option to limit the number of created “Web Image” tokens.

Consider providing an option for users to configure an upper limit of created “Web Image” tokens, which will allow them to protect themselves against any denial-of-service attempts.

Resources

- Cloudflare, “What is a denial-of-service (DoS) attack?”
  https://www.cloudflare.com/en-gb/learning/ddos/glossary/denial-of-service

CAN-Q224-9. Blind SSRF via Token Webhook	
Severity	**Low**
Vulnerability Class	Server-Side Request Forgery (SSRF)
Component	Canaries with *webhook_url* support
Status	Closed

Description

A Server Side Request Forgery (SSRF) attack describes the ability of an attacker to create network connections from a vulnerable web application to the internal network and other Internet hosts. Frequently, an SSRF vulnerability is used to attack internal services located behind a firewall and not directly accessible from the Internet.

In the Canary solution, the *webhook_url* feature (supported by multiple canaries) could can be leveraged to initiate an HTTP(S) connection and potentially gather information about the internal infrastructure of the application. For instance, this attack can be used to invoke internal unprotected webhooks or reach internal API endpoints.

The SSRF in question is considered to be “blind”, since the attacker does not receive the full response body. However, numerous techniques exist to infer results by either using timing or DNS requests[6]. For example, the attacker may infer its success or failure by comparing the web application latency on different requests and detect if there was a reply, or not, from an internal remote host.

Reproduction Steps

Perform a `GET` request to the `/generate` endpoint, specifying the URL of a controlled web server (e.g., Burp Suite’s Collaborator or a standard web server with full requests logging) in the `webhook_url` parameter.

After triggering the vulnerable functionality, you can observe the request hitting the external endpoint:

```
POST /test HTTP/1.1
Host: <BURP_SUITE_COLLABORATOR_INSTANCE_ID>.oastify.com
User-Agent: python-requests/2.31.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
content-type: application/json
Content-Length: 414

{"channel": "HTTP", "token_type": "azure_id", "src_ip": "127.0.0.1", "src_data":
null,  "token":  "a+test+token",  "time":  "2024-04-26  13:09:41  (UTC)",  "memo":
```

6 https://lab.wallarm.com/blind-ssrf-exploitation/

```
"Congrats! The newly saved webhook works", "manage_url": "http://example.com/
test/url/for/webhook", "additional_data": {"src_ip": "1.1.1.1", "useragent":
"Mozilla/5.0...", "referer": "http://example.com/referrer", "location": "http://
example.com/location"}}
```

Note the particular `User-Agent`, which demonstrates that the request has been made by the vulnerable web application.

This endpoint also accepts private IP addresses, opening up the possibility for a `Cross Site Port Attack (XSPA)`, which allows an attacker to enumerate services used by the web application, or exposed by the victim server or neighboring servers, by conducting a port scan from the perspective of the vulnerable host.

To perform an XSPA attack, it is sufficient for an attacker to issue a batch of `GET` requests to the correct endpoint, specifying an internal target (e.g., `http://domain.internal`) followed by the desired port number.

The feasibility of this attack was confirmed by measuring significant differences in response times for requests for valid and invalid open ports (e.g., `127.0.0.1:8080` vs. `127.0.0.1:1234`).

Impact

Medium. By leveraging this vulnerability, an attacker can gain information about the local system, internal network and potentially machines in adjacent networks. The ability to issue arbitrary requests to internal endpoints may also cause unwanted interactions with internal systems.

Complexity

Low. An attacker just needs to abuse an already existing functionality offered by the web application. No mitigation has been put in place to mitigate this issue. Since the dashboard is exposed and unauthenticated by design, the attacker has high chances of discovering it and exploiting the issue.

Remediation

The application could leverage the "advocate[7]" library. Usually SSRF protections involve the resolution of the hostname to an IP address and then checking whether the IP address belongs to a private network (RFC 1918). Since the webhook feature of a Canary instance could be legitimately used to call internal monitoring services to propagate an alert, such mitigation techniques needs to be carefully evaluated for this application. For instance, the creation of canaries with web-hooks pointing to internal services should be authorized (e.g., shared key set in the instance configuration files).

Resources

- OWASP, "Server Side Request Forgery"
  https://www.owasp.org/index.php/Server_Side_Request_Forgery

7 https://pypi.org/project/advocate/

Appendix A - Vulnerability Classification

Vulnerability Severity	**Critical**
	High
	Medium
	Low
	Informational
Vulnerability Class	Components With Known Vulnerabilities
	Covert Channel (Timing Attacks, etc.)
	Cross Site Request Forgery (CSRF)
	Cross Site Scripting (XSS)
	Denial of Service (DoS)
	Information Exposure
	Injection Flaws (SQL, XML, Command, Path, etc)
	Insecure Design
	Insecure Direct Object References (IDOR)
	Insufficient Authentication and Session Management
	Insufficient Authorization
	Insufficient Cryptography
	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
	Race Condition
	Security Misconfiguration
	Server-Side Request Forgery (SSRF)
	Unrestricted File Uploads
	Unvalidated Redirects and Forwards
	User Privacy
	Time-of-Check to Time-of-Use (TOCTOU)
	Insecure Deserialization

Appendix B - Remediation Checklist

The table below can be used to keep track of your remediation efforts inside this report. Mark the boxes when a fix has been implemented for the vulnerability.

☑	**CAN-Q224-1. ReDoS via Outdated "python-multipart" Library** Update the "python-multipart" dependency to the latest secure version
☑	**CAN-Q224-2. Overprivileged AWSProcessTokenLogsRole Lambda Role** We recommend applying resource limitation on the `kms:Decrypt` operation on the `AWSProcessTokenLogsRole` role. New canary keys should be either tagged or named to restrict their namespace within the role
☐	**CAN-Q224-3. Unencrypted Lambda Environment Variables** As an infrastructure security best practice, AWS Secrets Manager with a custom key and encryption in transit should be applied while storing sensitive environment variables
☐	**CAN-Q224-4. Missing Authorization in create_user_api_tokens** Utilize authentication mechanisms such as AWS Identity and Access Management (IAM) policies, API keys, AWS Cognito, or OAuth tokens to authenticate and authorize users accessing the API Gateway function
☑	**CAN-Q224-5. Stored Cross-Site Scripting on "Cloned Website" Token** Perform input validation and the appropriate output encoding on the `clonedsite` parameter
☑	**CAN-Q224-6. Stored Cross-Site Scripting in "Slow Redirect" Token Page** Perform input validation and the appropriate output encoding on the `redirect_url` parameter
☐	**CAN-Q224-7. Potential Denial of Service via Unlimited Creation of "AWS" Canary Tokens** Provide an option to restrict the number of AWS Key tokens. Consider providing an option for users to configure a limit of the token type or a secret key to be submitted during the creation. Requiring a configurable secret key to create tokens could be applied to other canaries with limited resources (e.g. see `CAN-Q224-8`)
☐	**CAN-Q224-8. Potential Denial of Service via Unlimited Creation of "Web Image" Canary Tokens** Provide an option to limit the number of created "web image" tokens. Consider providing an option for users to configure an upper limit of created "web image" tokens, which will allow them to protect themselves against any denial-of-service attempts
☑	**CAN-Q224-9. Blind SSRF via Token Webhook** The application could leverage the "advocate" library

When done patching the listed vulnerabilities, many clients find it worthwhile to perform a retest. During a retest, Doyensec researchers will attempt to bypass and subvert all implemented fixes. Retests usually take one or two days. Please reach out if you'd like more information on our retesting process.