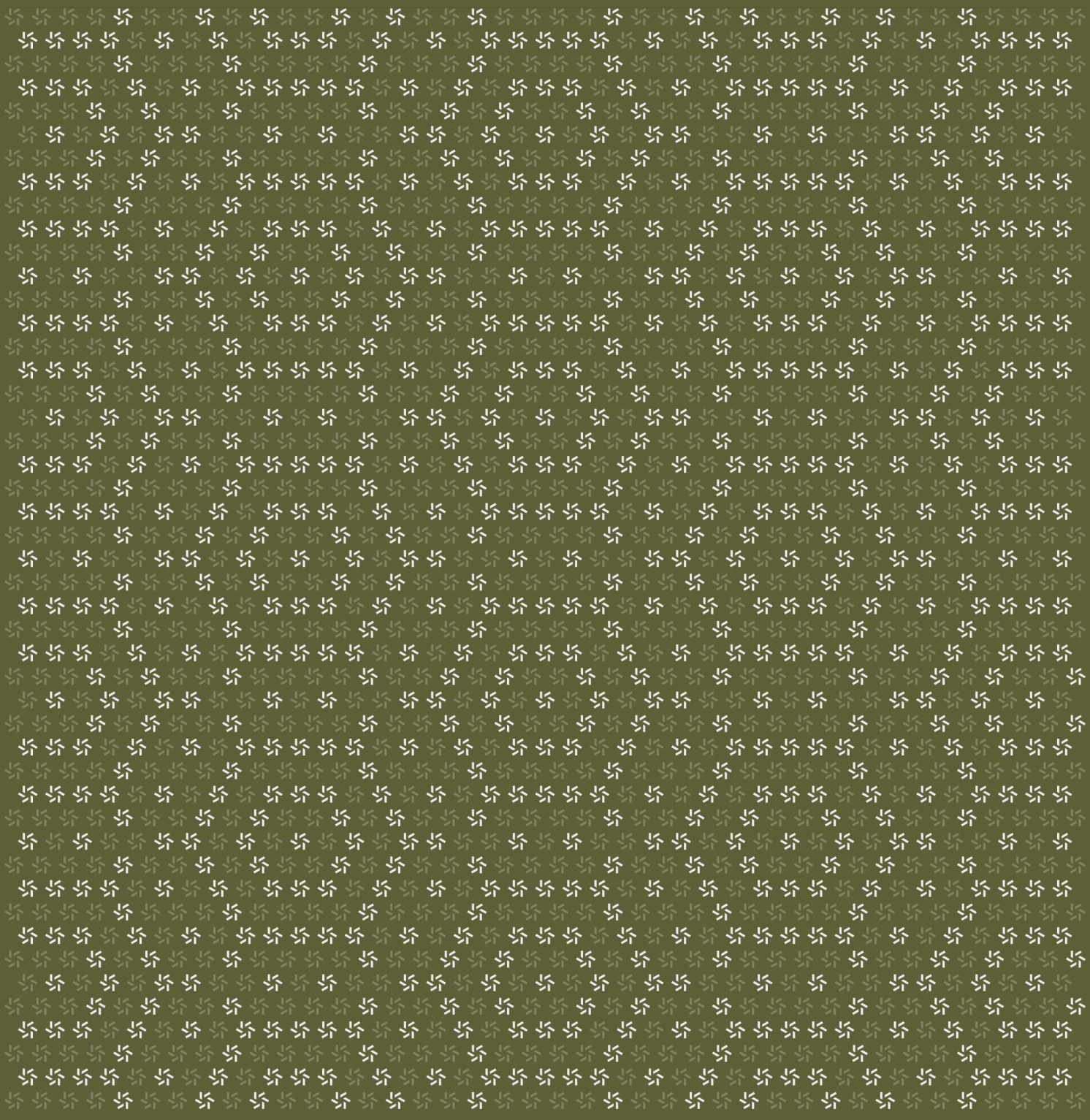
Zellic

Prepared for
Fuelet Wallet

Prepared by
SeungJun Kim
Philip Papurt
Zellic

October 15, 2024

Fuelet
Wallet Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Fuelet Wallet from September 30th to October 9th, 2024. During this engagement, Zellic reviewed Fuelet's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is authentication implemented in a secure and bug-free manner?
- Is Fuelet secure against common mobile vulnerabilities?
- Does the Fuelet app properly handle the confidentiality and integrity of user data?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- On-chain contracts
- Web-server components
- Third-party dependencies

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Fuelet files, we discovered four findings. One critical issue was found. One was of high impact and two were of medium impact.

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	1
Medium	2
Low	0
Informational	0

2. Introduction

2.1. About Fuelet

Fuelet Wallet contributed the following description of Fuelet:

> Non-custodial wallet on Fuel.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Unsafe code patterns. Many vulnerabilities in web-based applications arise from oversights during development. Missing an authentication check on a single API endpoint can critically impact the security of the overall application. Failure to properly sanitize and encode user input can lead to vulnerabilities like SQL injection, server-side request forgery, and more. We use both automated tools and extensive manual review to identify unsafe code patterns.

Business logic errors. Business logic is the heart of all web applications. We carefully review logic to ensure that the code securely and correctly implements the specified functionality. We also thoroughly examine all specifications for inconsistencies, flaws, and vulnerabilities, including for risks of fraud and abuse.

Integration risks. Web projects frequently have many third-party dependencies and interact with services and libraries that are not under the developer's control. We review the project's external interactions and identify potentially dangerous behavior resulting from compatibility issues and data inconsistencies.

Code maturity. We look for possible improvements across the entire codebase, reviewing violations of industry best practices and code quality standards. We suggest improvements to code clarity, documentation, and testing practices.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the

same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Fuelet Files

Type	Dart
Platforms	Web, iOS, Android
Target	fuelet_secure_layer
Repository	https://github.com/Fuelet/fuelet_secure_layer ↗
Version	`19851adc59fdb149bfbfcd7a42f7bda3b193403b`
Programs	`packages/fuelet_secure_layer`
Target	fuels-dart
Repository	https://github.com/Fuelet/fuels-dart ↗
Version	`7e4a223b3e2d08cf81a378b2899dc46ae51b11cf`
Programs	`packages`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.4 person-weeks. The assessment was conducted by two consultants over the course of 1.8 calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

SeungJun Kim
Engineer
seungjun@zellic.io

Philip Papurt
Engineer
philip@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 30, 2024	Start of primary review period
October 9, 2024	End of primary review period

3. Detailed Findings

3.1. Universal XSS in Fuelet dApp WebView

Target	fuelet		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The Fuelet iOS and Android apps use a WebView (using WebKit on iOS and Chromium on Android) that allows users to access dApps. The Fuelet app injects a JavaScript shim that allows JavaScript running in the WebView to interact with the native Fuelet app.

However, the WebView is vulnerable to universal XSS via injection in the `id` field in `sendResponseToJs` (fuelet/lib/presentation/browser/screens/state/browser_page_screen_state/browser_page_screen_bloc.dart):

```
Future<void> sendResponseToJs(
    {required String id,
    required String method,
    dynamic result,
    String? error}) async {
  final response = jsonEncode({
    'source': method,
    'result': result,
    'error': error,
  }).replaceAll('"', '\\"');

  _controller
      .runJavaScript('receiveResponseFromFlutter("$id", "$response");');
}
```

An attacker may exploit this to execute JavaScript on another origin by sending a request then navigating to the victim origin before the Fuelet wallet sends the response.

To test this vulnerability, executing `alert(origin)` on example.com, we can run the following JavaScript code:

```
location = '//example.com'
FueletWallet.postMessage(JSON.stringify({
  type: 'request',
  target: 'FuelContentScript',
```

```
  connectorName: 'Fuelet Wallet',
  request: {
    jsonrpc: '2.0',
    id: '",alert(origin),"',
    method: 'connect',
    params: {}
  }
}))
```

Impact

This issue allows any malicious website a user visits to compromise any other website loaded in the WebView.

Recommendations

The Fuelet team should

- ensure that `postMessage` responses are only sent to the origin that initiated the request, and
- properly escape both the `id` and `response` parameters via JSON encoding in `sendResponseToJs`.

Remediation

This issue has been acknowledged by Fuelet Wallet, and a fix was implemented in commit d686818 ↗.

3.2. Origin impersonation via URL username and elision

Target	fuelet		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

The Fuelet wallet dApp WebView displays the current URL in the address bar. However, the WebView does not properly

- elide the URL starting from the end of the hostname
- remove the `username:password@` component from the URL

This allows an attacker to craft a URL that when visited in the Fuelet WebView appears nearly identical to a legitimate URL.

The address bar is rendered in fuelet/lib/presentation/browser/screens/browser_page_screen.dart:

```
Text(
  maxLines: 1,
  overflow: TextOverflow.ellipsis,
  browserWebAddressVm?.description
          .formatWithMaxLength(25) ??
      "",
  style: NFTTypography.body2Medium.copyWith(
    color: theme.colorScheme.mainTextColor,
  ),
),
```

Impact

This issue allows any malicious website a user visits to impersonate another origin.

Recommendations

The Fuelet team should

- elide the URL starting from the end of the hostname, and
- remove the `username:password@` component from the URL.

Remediation

This issue has been acknowledged by Fuelet Wallet, and a fix was implemented in commit efdef49 ↗.

3.3. Optimizable PasswordManager check

Target	fuelet_secure_layer		
Category	Business Logic	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

The Fuelet wallet checks user passwords and PINs by first hashing it with SHA-256 then attempting to decrypt a known value with AES-GCM. This operation is highly optimizable and parallelizable, and the low-entropy PIN can be discovered by an attacker with the ciphertext in seconds.

This validation is performed in fuelet_secure_layer/packages/fuelet_secure_layer/lib/src/features/password/password_manager.dart:

```
const _secretToEncrypt = 'fuelet_secure_layer_secret_kmr_wpuOXFM4uaq3kym';

// ...

String _hashPassword(String password) {
  final bytes = utf8.encode(password);

  return sha256.convert(bytes).toString();
}

// ...

Future<String> _createPasswordSecret(String password) {
  final passwordHash = _hashPassword(password);

  return Aes256Gcm.encrypt(_secretToEncrypt, passwordHash);
}
```

Impact

This issue allows an attacker to quickly discover the user's password or PIN.

Recommendations

The Fuelet team should use a secure hash function such as Argon2 or Scrypt to hash the password.

Remediation

This issue has been acknowledged by Fuelet Wallet.

3.4. Insecure cloud-backup encryption

Target	fuelet_secure_layer		
Category	Business Logic	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

The Fuelet wallet backs up the user's private keys to Apple iCloud and Google Drive. It encrypts this data with the `aesPassword` value. However, this value is compiled (obfuscated) into the app and is shared between all users. Thus, the cloud backups are effectively stored unencrypted.

The password is hardcoded in fuelet/lib/env/env.dart:

```
@Envied(path: '.env', obfuscate: true)
abstract class Env {
  @EnviedField(varName: 'AES_PASSWORD')
  static final String aesPassword = _Env.aesPassword;
// ...
```

The cloud-backup encryption occurs in fuelet_secure_layer/packages/fuelet_secure_layer /lib/src/features/cloud_backup/entity/backup_accouts_dto.dart:

```
Future<String> toRawJson(String password) async {
  final Map<String, String> encryptedAccounts = {};

  for (var key in backupAccounts.keys) {
    encryptedAccounts[key] =
        await Aes256GcmUtils.encrypt(backupAccounts[key]!, password);
  }

  return jsonEncode(encryptedAccounts);
}
```

Impact

This issue allows an attacker with access to the user's Apple/Google account to compromise their Fuelet wallet.

Recommendations

The Fuelet team should encrypt cloud backups with the user's app password or add a new backup password that the user may enter.

Remediation

This issue has been acknowledged by Fuelet Wallet, and a fix was implemented in commit ea46b33 ↗.

4. Assessment Results

At the time of our assessment, the reviewed code was deployed.

During our assessment on the scoped Fuelet files, we discovered four findings. One critical issue was found. One was of high impact and two were of medium impact.

4.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.