Zellic

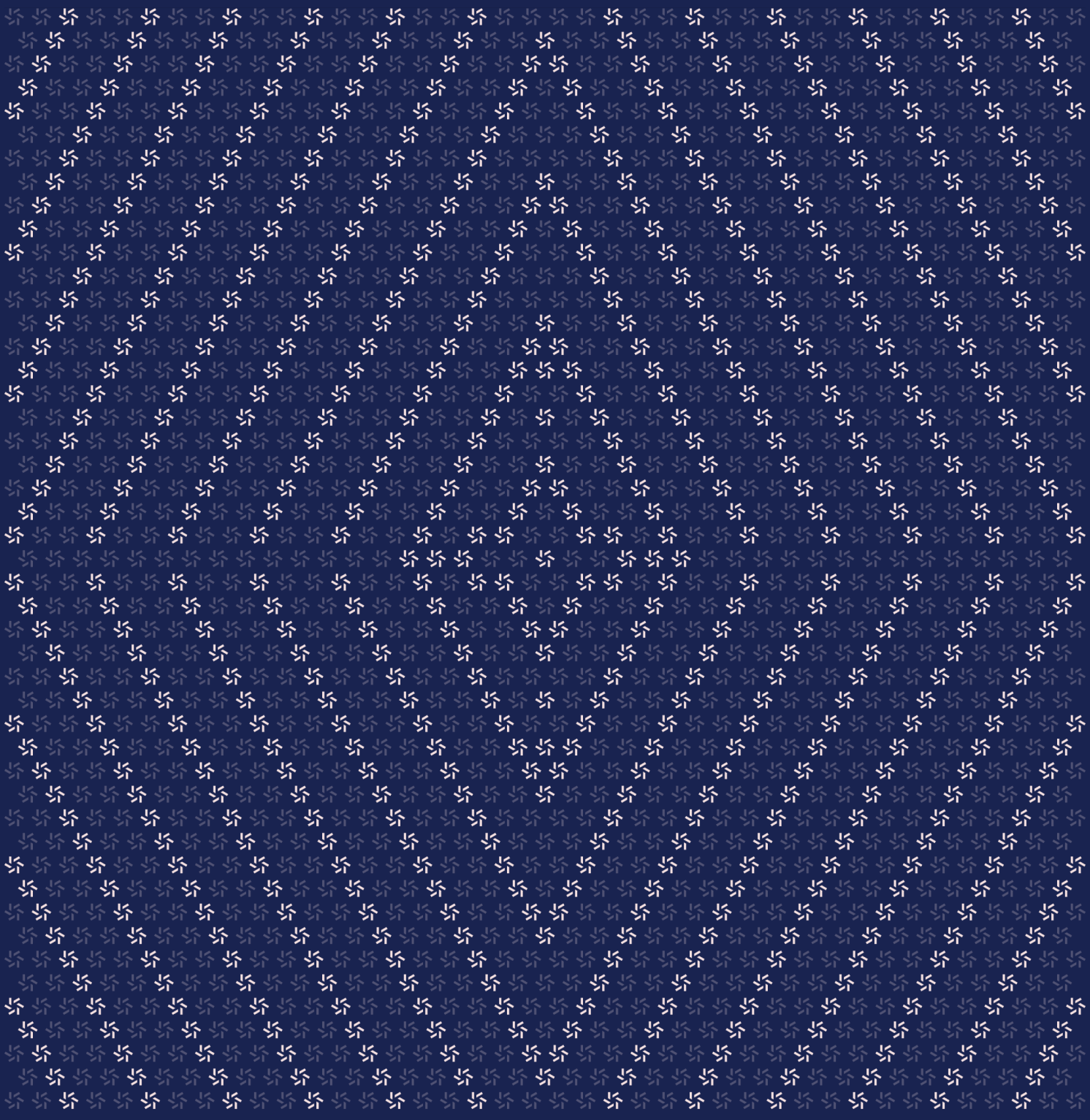
Prepared for
Luka Avbreht
Flare Network

Prepared by
Evan Hwang
Weipeng Lai
Zellic

April 23, 2026

Smart Accounts Diff

Smart Contract 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 Flare Network from April 15th to April 16th, 2026. During this engagement, Zellic reviewed the Smart Accounts diff'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 question:

- Is there a way for users to interact improperly with the system, assuming FDC works as expected?

1.3. Non-goals and Limitations

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

- Front-end components
- Infrastructure relating to the project
- Key custody

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 Smart Accounts diff contracts, we discovered one finding, which was informational in nature.

Breakdown of Finding Impacts

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

2. Introduction

2.1. About Smart Accounts

Flare Network contributed the following description of Smart Accounts:

> Smart Accounts provide an account abstraction layer for Flare's EVM that allows XRPL addresses to control and interact with EVM accounts directly, without leaving the XRPL environment.

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:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

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:

Smart Accounts Diff Contracts

Type	Solidity
Platform	EVM-compatible
Target	Only changes between 6bc781a...6b937e0
Repository	https://github.com/flare-foundation/flare-smart-accounts ↗
Version	`6b937e0fad6c89753db66a4ae5c2461be62c4e70`
Programs	`contracts/**/*.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.5 person-days. The assessment was conducted by two consultants over the course of two calendar days.

Contact Information

The following project manager was associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

The following consultants were engaged to conduct the assessment:

Evan Hwang
Engineer
evan@zellic.io

Weipeng Lai
Engineer
weipeng.lai@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 15, 2026	Start of primary review period
April 16, 2026	End of primary review period

3. Detailed Findings

3.1. Inconsistent native-value funding expectations in PersonalAccount

Target	PersonalAccount		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

PersonalAccount enforces different assumptions about who supplies native value across its execution paths.

In `reserveCollateral()` and `redeemFXrp()`, the call requires sufficient inbound `msg.value`, so an executor that submits too little native value causes the transaction to revert. This makes those flows explicitly executor funded.

By contrast, `executeUserOp()` does not require the inbound `msg.value` to cover the total native value forwarded by the calls it executes. Instead, it simply performs each call with the requested `value`:

```
function executeUserOp(
    IPersonalAccount.Call[] calldata _calls
)
    external payable
    onlyController nonReentrant
{
    for (uint256 i = 0; i < _calls.length; i++) {
        (bool success, bytes memory result) = _calls[i].target.call{
            value: _calls[i].value
        }(_calls[i].data);
        if (!success) {
            revert CallFailed(i, result);
        }
    }
}
```

In the `0xFF` memo-instruction flow, the amount of native value forwarded into `PersonalAccount::executeUserOp()` is whatever the direct-minting executor supplied as `msg.value`. If the executor supplies less native value than the user operation forwards, the shortfall is covered by the PersonalAccount's existing balance.

While this behavior may be intended, it is inconsistent with the stricter funding model used elsewhere in PersonalAccount and may create incorrect expectations about who bears the

native-value cost of execution. A user may assume, by analogy to `reserveCollateral()` and `redeemFXrp()`, that the executor is expected to fund the native value used by `executeUserOp()`. That assumption is incorrect. For example, if a memo-authorized operation sends 10 FLR, the executor can submit the direct-minting transaction with zero `msg.value`. If the PersonalAccount already holds 10 FLR, the operation will still succeed and consume the account's existing balance.

Impact

If users assume that the executor is expected to fund the native value used by `executeUserOp()`, they may authorize operations under an incorrect cost model. In practice, an executor can submit the direct-minting execution with insufficient or zero `msg.value`, and the operation can still succeed by consuming the PersonalAccount's preexisting balance. This can lead to unexpected depletion of prefunded native value in a user's personal account.

Recommendations

Clearly document that `executeUserOp()` does not require the executor to supply the total outbound native value for the calls it executes and that execution may instead draw from the PersonalAccount's existing balance.

Remediation

This issue has been acknowledged by Flare Network.

4. Patch Review

This section documents notable and minor changes applied to the in-scope smart contracts between commit `6bc781a5` and commit `6b937e0f`.

4.1. Notable changes

The following are notable changes made to the codebase.

- **The swap system was removed.** All Uniswap V3 swap functionality was removed, including swap facets, swap libraries, and swap parameter management.
- **The `mintedFAssets` memo mint flow was added.** The `mintedFAssets()` function was added as an XRPL direct-minting entry point called by the AssetManager. It processes XRPL memo instructions (`0xFF`, `0xE0`, `0xE1`, `0xE2`, `0xD0`, and `0xD1`) and uses `MemoInstructions` (ERC-7201 storage) for nonce, executor lock, and ignore-memo and replacement-fee state.
- **In PersonalAccount, `executeUserOp` was added.** PersonalAccount now includes `executeUserOp(Call[] calldata _calls)` as a generic multicall execution path. This path is controller-only and reentrancy-protected, and it reverts atomically if any internal call fails.

4.2. Minor changes

The following are minor changes made to the codebase.

- The `isTransactionIdUsed` check was moved from `InstructionsFacet` to `MemoInstructionsFacet`.
- Vault typing migrated from raw `uint8` values to an `IVaultsFacet.VaultType` enum, improving type safety across facets and libraries.
- PersonalAccount now supports generic `executeUserOp` multicalls and receiver hooks (`ERC721`, `ERC1155`, `ERC1363`) with `ReentrancyGuardTransient`.
- Pausing was added as an emergency stop for instruction execution.

5. Assessment Results

During our assessment on the scoped Smart Accounts diff contracts, we discovered one finding, which was informational in nature.

5.1. 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.

Where Zellic states that a finding has been addressed or fixed in one or more specific commits, this indicates only that those commits introduce changes that, in our assessment, address the finding relative to the codebase version originally reviewed. This does not imply that Zellic reviewed other changes contained in those commits, the overall state of the codebase at those commits, or the interplay between remediation measures for different findings.

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.