Zellic

Prepared for
Luka Avbreht
Iztok Kavkler
Flare Network

Prepared by
Nipun Gupta
Weipeng Lai
Zellic

October 20, 2025

Flare FAssets
Smart Contract Patch Review

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. Introduction

We were asked to review two minor patches to Flare FAssets from October 16th to October 17th, 2025, which fixed the two issues described further in Section 2. ↗.

1.1. Scope

The engagement involved a review of the following targets:

Flare FAssets Contracts

Type	Solidity
Platform	EVM-compatible
Target	Only changes between 0abdb6c4...a70e8211
Repository	https://github.com/flare-foundation/fassets ↗
Version	`a70e821193c686f8721c46121d37991d7521d478`

Programs	`fasset/contracts/agentOwnerRegistry/implementation/*`
	`fasset/contracts/agentVault/**/*`
	`fasset/contracts/assetManager/**/*`
	`fasset/contracts/assetManagerController/**/*`
	`fasset/contracts/collateralPool/**/*`
	`fasset/contracts/coreVaultManager/**/*`
	`fasset/contracts/diamond/**/*`
	`fasset/contracts/fassetToken/**/*`
	`fasset/contracts/flareSmartContracts/**/*`
	`fasset/contracts/ftso/**/*`
	`fasset/contracts/governance/**/*`
	`fasset/contracts/userInterfaces/**/*`
	`fasset/contracts/utils/**/*`

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

Pedro Moura
Engagement Manager
pedro@zellic.io

The following consultants were engaged to conduct the assessment:

Nipun Gupta
Engineer
nipun@zellic.io

Weipeng Lai
Engineer
weipeng.lai@zellic.io

1.2. Disclaimer

2. Patch Review

This section documents the codebase changes made to address two issues identified during the Code4rena audit competition. The review focused solely on verifying the fixes for these findings. Both issues have been resolved, and the applied changes do not introduce any new security concerns.

1. An agent could anticipate the payment reference that would be required in a future block and could make transfers on the underlying chain. The agent could then submit the redemption request before the underlying transaction is proven illegal. If the redemption request is made before the payment, the transfer could not be proven illegal and the redemption could not be confirmed.
2. If the address of `WNAT` is modified by the asset updater in the Flare Time Series Oracle (FTSO), the address on the asset manager could be updated by anyone, but the address on the collateral pool could only be updated by the agent. If an agent does not update this address in a timely manner, it might have caused some accounting issues.

2.1. Payments before the first redemption block can be challenged

The fix for the first issue implements the following change in the ChallengesFacet.sol facet:

Diff between 0abdb6c4...1d389c2f

```
Redemption.Request storage redemption
    = state.redemptionRequests[redemptionId];
// Redemption must be for the correct agent, must not be rejected and
// only statuses ACTIVE and DEFAULTED mean that redemption is still missing a
    payment proof.
// We do not check for timestamp of the payment, because on UTXO chains
    legal payments can be
// Payments must not be made before the current underlying block when
    redemption was requested.
// We do not check that the payment is not too late, because on UTXO chains
    legal payments can be
// delayed by arbitrary time due to high fees and cannot be canceled, which
    could lead to
// unnecessary full liquidations.
bool redemptionActive = redemption.agentVault == _agentVault && Redemptions.
    isOpen(redemption);
bool redemptionActive = redemption.agentVault == _agentVault
    && Redemptions.isOpen(redemption)
    && _payment.data.responseBody.blockNumber >= redemption.firstUnderlyingBlo
        ck;
require(!redemptionActive, MatchingRedemptionActive());
```

The value of `redemptionActive` will now only be true if for an open redemption request the payment made on the underlying block is on or after the `firstUnderlyingBlock` of the redemption request. This allows proving the requests that were made before the `firstUnderlyingBlock` as illegal and thus liquidating the agent, therefore mitigating the issue.

The fix was implemented in commit `1d389c2f` ↗.

2.2. The `upgradeWNatContract` function must now be called by governance or executor

The fix for the second issue implements the following change in the AgentCollateralFacet.sol facet.

Diff between 746f1ef2...a70e8211

```
* NOTE: may only be called by the agent vault owner.
*/
function upgradeWNatContract(
    address _agentVault
    uint256 _start,
    uint256 _end
)
    external
    // no emergency pause check to allow changing collateral token
    onlyAgentVaultOwner(_agentVault)
    onlyImmediateGovernanceOrExecutor
{
    (address[] memory agentVaults,) = Agents.getAllAgents(_start, _end);
    for (uint256 i = 0; i < agentVaults.length; i++) {
        _upgradeWNatContract(agentVaults[i]);
    }
}

function _upgradeWNatContract(
    address _agentVault
```

```
)
    private
{
    Agent.State storage agent = Agent.get(_agentVault);
    AssetManagerState.State storage state = AssetManagerState.get();
```

The modifier is changed from `onlyAgentVaultOwner(_agentVault)` to `onlyImmediateGovernanceOrExecutor`, thus allowing the call to be made only by the immediate governance or the governance executor, thus mitigating the issue.

The fix was implemented in commit a70e8211 ↗.

3. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

3.1. Outdated comments

The `upgradeWNatContract` function now requires governance or the governance executor to invoke it, yet the in-line comments in both AgentCollateralFacet and `IIAssetManager` still state that only the agent vault owner may call it:

```
/**
 * When current pool collateral token contract (WNat) is replaced by the method
   setPoolWNatCollateralType,
 * pools don't switch automatically. Instead, the agent must call this method
   that swaps old WNat tokens for
 * new ones and sets it for use by the pool.
 * NOTE: may only be called by the agent vault owner.
 */
function upgradeWNatContract(
    uint256 _start,
    uint256 _end
)
    external
    onlyImmediateGovernanceOrExecutor
{
    // [...]
}


/**
 * When current pool collateral token contract (WNat) is replaced by the method
   setPoolWNatCollateralType,
 * pools don't switch automatically. Instead, the agent must call this method
   that swaps old WNat tokens for
 * new ones and sets it for use by the pool.
 * NOTE: may only be called by the agent vault owner.
 */
function upgradeWNatContract(
    uint256 _start,
    uint256 _end
) external;
```

We recommend updating the comments to reflect the access-control requirements.