Zellic

Prepared for
Luka Avbreht
Iztok Kavkler
Flare Network

Prepared by
Weipeng Lai
Kuilin Li
Zellic

September 23, 2025

FAssets

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 September 16th to September 23rd, 2025. During this engagement, Zellic reviewed FAssets'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:

- Did the updates introduce any problems, bugs, exploits, or unusual behavior?

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.

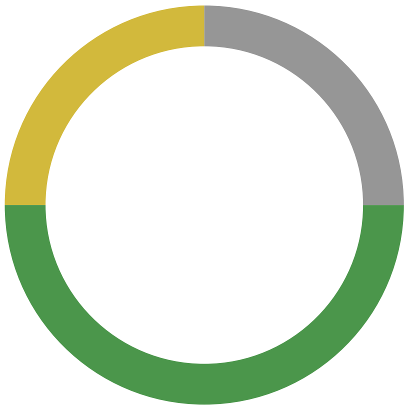
1.4. Results

During our assessment on the scoped FAssets contracts, we discovered four findings. No critical issues were found. One finding was of medium impact, two were of low impact, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Flare Network in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	1
🟩 Low	2
⬜ Informational	1



2. Introduction

2.1. About FAssets

Flare Network contributed the following description of FAssets:

> FAssets bring non-smart contract assets like XRP into DeFi — securely, scalably, and with full custody retained.
>
> The FAsset contracts are used to mint assets on top of Flare. The system is designed to handle chains which don't have (full) smart contract capabilities, although it can also work for smart contract chains. Initially, FAsset system will support XRP native asset on XRPL. At a later date BTC, DOGE, add tokens from other blockchains will be added.
>
> The minted FAssets are secured by collateral, which is in the form of ERC20 tokens on Flare/-Songbird chain and native tokens (FLR/SGB). The collateral is locked in contracts that guarantee that minted tokens can always be redeemed for underlying assets or compensated by collateral. Underlying assets can also be transferred to Core Vault, a vault on the underlying network. When the underlying is on the Core Vault, the agent doesn't need to back it with collateral so they can mint again or decide to withdraw this collateral.
>
> Two novel protocols, available on Flare and Songbird blockchains, enable the FAsset system to operate:
>
> - FTSO contracts which provide decentralized price feeds for multiple tokens.
> - Flare's FDC, which bridges payment data from any connected chain.

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.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

FAssets Contracts

Type	Solidity
Platform	EVM-compatible

Target	Only changes between 1b6a12f...fe40c97
Repository	https://gitlab.com/flarenetwork/fassets/fassets ↗
Version	`fe40c9777635499b8246eb66fdb546c5a1657326`

Programs	`assetManagerController/implementation/AssetManagerController.sol`
	`assetManager/facets/EmergencyPauseFacet.sol`
	`assetManager/facets/EmergencyPauseTransfersFacet.sol`
	`assetManager/library/data/AssetManagerState.sol`
	`fassetToken/implementation/FAsset.sol`
	`collateralPool/implementation/CollateralPool.sol`
	`assetManager/facets/AssetManagerBase.sol`
	`assetManager/facets/AgentCollateralFacet.sol`
	`assetManager/facets/AgentVaultManagementFacet.sol`
	`assetManager/facets/AvailableAgentsFacet.sol`
	`assetManager/facets/ChallengesFacet.sol`
	`assetManager/facets/CoreVaultClientFacet.sol`
	`assetManager/facets/LiquidationFacet.sol`
	`assetManager/facets/MintingDefaultsFacet.sol`
	`assetManager/facets/MintingFacet.sol`
	`assetManager/facets/RedemptionConfirmationsFacet.sol`
	`assetManager/facets/RedemptionDefaultsFacet.sol`
	`assetManager/facets/RedemptionRequestsFacet.sol`
	`assetManager/facets/UnderlyingBalanceFacet.sol`
	`assetManager/facets/CollateralTypesFacet.sol`
	`assetManager/facets/SettingsManagementFacet.sol`
	`assetManager/facets/SystemInfoFacet.sol`
	`assetManager/library/AgentUpdates.sol`
	`assetManager/library/CollateralTypes.sol`
	`assetManager/library/Liquidation.sol`
	`assetManager/library/LiquidationPaymentStrategy.sol`
	`assetManager/library/Minting.sol`
	`assetManager/library/SettingsInitializer.sol`
	`assetManager/library/data/CollateralTypeInt.sol`

2.4. Project Overview

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

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:

Weipeng Lai
Engineer
weipeng.lai@zellic.io ↗

Kuilin Li
Engineer
kuilin@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 16, 2025	Kick-off call
September 16, 2025	Start of primary review period
September 23, 2025	End of primary review period

3. Detailed Findings

3.1. Duration budget exhaustion prevents higher-severity pauses

Target	EmergencyPauseFacet		
Category	Business Logic	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The `emergencyPausedTotalDuration` state tracks cumulative time under emergency pause and enforces a global time limit on pauses. The `_calcPauseEndTime` function caps each new pause so the total does not exceed `maxEmergencyPauseDurationSeconds`.

```
function _calcPauseEndTime(uint256 _duration)
    private view returns (uint256 _endTime, uint256 _totalDuration) {
    AssetManagerState.State storage state = AssetManagerState.get();
    AssetManagerSettings.Data storage settings = Globals.getSettings();
    uint256 currentPauseEndTime = Math.max(state.emergencyPausedUntil,
    block.timestamp);
    uint256 projectedStartTime =
        Math.min(currentPauseEndTime - state.emergencyPausedTotalDuration,
    block.timestamp);
    uint256 maxEndTime = projectedStartTime
    + settings.maxEmergencyPauseDurationSeconds;
    _endTime = Math.min(block.timestamp + _duration, maxEndTime);
    _totalDuration = _endTime - projectedStartTime;
}
```

The `emergencyPausedTotalDuration` resets to zero if `emergencyPauseDurationResetAfterSeconds` has elapsed since the previous pause ended when `emergencyPause` is called:

```
function emergencyPause(EmergencyPause.Level _level, bool _byGovernance,
    uint256 _duration)
    external
    onlyAssetManagerController
{
    // [...]
    if (state.emergencyPausedUntil
    + settings.emergencyPauseDurationResetAfterSeconds <= block.timestamp) {
        state.emergencyPausedTotalDuration = 0;
    }
    // [...]
}
```

The current code upgrades introduce a three-level hierarchy of pauses:

```
library EmergencyPause {
    enum Level {
        // Pause is not active.
        NONE,
        // Prevent starting mint, redeem,
    liquidation and core vault transfer/return.
        START_OPERATIONS,
        // Everything from START_OPERATIONS,
    plus prevent finishing or defulating already started mints and redeems.
        FULL,
        // Everything from FULL, plus prevent FAsset transfers.
        FULL_AND_TRANSFER
    }
}
```

Moreover, previously, `emergencyPausedTotalDuration` tracked only nongovernance-triggered pauses. After the upgrade, it records durations triggered by both governance and nongovernance.

The issue is that the same `emergencyPausedTotalDuration` is used for different pause levels and initiators. Because the total cannot exceed `maxEmergencyPauseDurationSeconds` and does not reset within the `emergencyPauseDurationResetAfterSeconds` window,

- if a lower-severity pause is triggered (such as `START_OPERATIONS`) for a long period and consumes most of the budget, then any subsequent escalation to `FULL` or `FULL_AND_TRANSFER` within the `emergencyPauseDurationResetAfterSeconds` window will be limited by the remaining budget rather than the requested duration.
- if a nongovernance pause consumes most of the budget, the governance's new pause in the `emergencyPauseDurationResetAfterSeconds` window will be limited by the remaining budget rather than the requested duration.

Impact

A higher-severity pause may lack sufficient duration if lower-severity pauses have consumed the budget. Similarly, a governance pause may lack sufficient duration if nongovernance-triggered pauses have consumed the budget. An attacker could exploit this by waiting until lesser incidents deplete the duration budget before launching a more severe attack.

Recommendations

We recommend resetting `emergencyPausedTotalDuration` to zero when escalating to a higher pause level to ensure the full duration is available for the higher-severity pause. We also recommend tracking duration separately for pauses triggered by governance and nongovernance.

Remediation

This issue has been acknowledged by Flare Network, and fixes were implemented in the following commits:

- `c437fdaa` ↗
- `f218a710` ↗

Flare Network provided the following note regarding the remediation:

> We again allow governance to pause for as long as it wants and we have also re-added resetEmergencyPauseTotalDuration method. But to prevent governance START_OPERATIONS pause blocking higher level pauses by bots, and to do it without introducing inconsistencies, we had to make larger changes: we now track governance and trigger bot ("external") pauses independently. Only external pauses have total time limit. The effective pause level is the higher of the two.

3.2. Minor pauses block challenge operations

Target	ChallengesFacet		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The challenge functions in ChallengesFacet are modified by `notEmergencyPaused`:

```
function illegalPaymentChallenge(
    IBalanceDecreasingTransaction.Proof calldata _payment,
    address _agentVault
)
    external
    notEmergencyPaused
    nonReentrant
{
    // [...]
}

function doublePaymentChallenge(
    IBalanceDecreasingTransaction.Proof calldata _payment1,
    IBalanceDecreasingTransaction.Proof calldata _payment2,
    address _agentVault
)
    external
    notEmergencyPaused
    nonReentrant
{
    // [...]
}

function freeBalanceNegativeChallenge(
    IBalanceDecreasingTransaction.Proof[] calldata _payments,
    address _agentVault
)
    external
    notEmergencyPaused
    nonReentrant
{
    // [...]
}
```

These functions are disabled when the contract is paused at the least severe emergency pause level — `START_OPERATIONS`:

```
modifier notEmergencyPaused {
    _checkEmergencyPauseNotActive(EmergencyPause.Level.START_OPERATIONS);
    _;
}
```

At this pause level, agents can still operate on the native chain and potentially misbehave. Consequently, challengers cannot submit challenges against misbehaving agents when only a minor pause is active.

Impact

Challengers cannot challenge misbehaving agents during less severe pauses (`START_OPERATIONS`), delaying enforcement until the pause is lifted.

Recommendations

We recommend allowing these challenges during minor pauses. Specifically, change the modifier from `notEmergencyPaused` to `notFullyEmergencyPaused`.

Remediation

This issue has been acknowledged by Flare Network, and a fix was implemented in commit `b37765ee` ↗.

3.3. Emergency pause does not block agent-setting updates

Target	AgentSettingsFacet		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

Agents can use `announceAgentSettingUpdate` to schedule and `executeAgentSettingUpdate` to apply setting changes:

```
function announceAgentSettingUpdate(
    address _agentVault,
    string memory _name,
    uint256 _value
)
    external
    onlyAgentVaultOwner(_agentVault)
    returns (uint256 _updateAllowedAt)
{
    // [...]
}

function executeAgentSettingUpdate(
    address _agentVault,
    string memory _name
)
    external
    onlyAgentVaultOwner(_agentVault)
{
    // [...]
}
```

Neither function includes the `notEmergencyPaused` modifier. Agents can announce and execute setting updates while the contract is emergency-paused.

Impact

Agents can change agent settings during an emergency pause, potentially undermining the pause's intent. Although timelocks restrict timing, execution remains possible if the window occurs during the pause.

Recommendations

We recommend adding the `notEmergencyPaused` modifier to both functions to prevent setting updates during emergency pauses.

Remediation

This issue has been acknowledged by Flare Network.

3.4. Potential queue manipulation via permissionless `consolidateSmallTickets`

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

Description

When the lot size increases, many tickets smaller than one lot may remain in the queue. In extreme cases, this prevents redemptions if none of the first `maxRedeemedTickets` tickets contains at least one lot.

The current upgrade introduces a `consolidateSmallTickets` function to address this issue. The function consolidates the redemption queue starting from `_firstTicketId` and iterates up to `maxRedeemedTickets` times, converting small tickets to dust or consuming dust to increase ticket values.

```
function consolidateSmallTickets(
    uint256 _firstTicketId
)
    external
    notEmergencyPaused
    nonReentrant
{
    AssetManagerState.State storage state = AssetManagerState.get();
    uint256 maxRedeemedTickets = Globals.getSettings().maxRedeemedTickets;
    uint64 firstTicketId = _firstTicketId != 0 ? _firstTicketId.toUint64() :
    state.redemptionQueue.firstTicketId;
    uint64 ticketId = firstTicketId;
    for (uint256 i = 0; i < maxRedeemedTickets; i++) {
        if (ticketId == 0) break;   // end of queue
        RedemptionQueue.Ticket storage ticket
    = state.redemptionQueue.getTicket(ticketId);
        // in the first run of the loop, we must validate that passed
    _firstTicketId is valid
        require(i > 0 || ticket.agentVault != address(0), InvalidTicketId());
        uint64 nextTicketId = ticket.next;
        // this will convert small tickets to dust or consume some dust to
    actually increase the ticket
        Redemptions.removeFromTicket(ticketId, 0);
        ticketId = nextTicketId;
    }
    emit IAssetManagerEvents.RedemptionTicketsConsolidated(firstTicketId,
    ticketId);
}
```

The issue is that this function is permissionless and might be used by agents to influence redemption-queue ordering after a lot size increase. For example, after a lot size increase, suppose an agent has two one-lot tickets — one within the first `maxRedeemedTickets` (range A) and another near the end (range B).

Depending on call order, if someone first calls `consolidateSmallTickets` starting in range A and then in range B, the ticket in A could be removed and its value added to the ticket in B. If the agent instead calls starting in range B first, then in range A, the ticket in B could be removed and its value added to the ticket in A, giving the agent a more favorable queue position.

Impact

Agents can manipulate redemption-queue ordering to their advantage after lot size increases.

Recommendations

We recommend restricting `consolidateSmallTickets` to governance-only access.

Remediation

This issue has been acknowledged by Flare Network, and a fix was implemented in commit `7758bcfc ↗`.

4. 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.

4.1. Ensure absence of invalid tokens before removing collateral-token validation

Commit `e7723fa0` ↗ removed collateral-token deprecation and agent vault token switching. After the upgrade, the `_getCollateralAmount` function in the Liquidation library and `currentLiquidationFactorBIPS` in LiquidationPaymentStrategy no longer check whether collateral tokens are invalid; they assume all input tokens are valid.

These functions implicitly assume that no invalid collateral tokens exist at the time of contract upgrade. If this assumption proves incorrect, invalid collateral tokens can be passed to these functions, which process them as valid tokens. Therefore, ensuring no invalid collateral tokens exist before this upgrade is critical.

5. System Design

This provides a description of the high-level components of the system that were changed by the in-scope commits, including details about what changes were implemented and how those changes affect the critical invariants or constraints of the system.

Not all modified components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. Addition of a three-stage emergency pause feature

Description

An emergency pause feature was added to the system. The system can now be paused at three levels of increasing severity.

The `START_OPERATIONS` pause level prevents starting operations without preventing finishing them. This means that mints, redemptions, liquidations, and core-vault actions cannot be started, but ones that have already been started can still be finished. Since these operations have time limits that would not be paused by the emergency pause, this allows the protocol to gently come to a pause without disrupting normal users.

The `FULL` pause level, in addition to encompassing all of the `START_OPERATIONS` level, also pauses the completion of operations. This pause level is intended to halt an ongoing attack, and successful resumption is likely to affect pending operations negatively if their time limits are exceeded. During the pause, the finishing or defaulting of mints and redemptions are all paused.

The `FULL_AND_TRANSFER` pause level, in addition to encompassing all of the `FULL` level, also pauses all transfers of FAssets. This means that FAssets will not accrue transfer fees, and the balances of FAssets will not change during the halt.

The EmergencyPauseFacet facet previously only supported a single level of pause, which had limited duration when not applied by governance. This facet was substantially changed in order to implement the multilevel pauses described above. If the system was paused by the governance, then the nongovernance pausers can only increase the pause level and/or the duration of the pause.

Also, a maximum total pause duration limit was added to the EmergencyPauseFacet, which applies to both the governance and the nongovernace pauses. This maximum limits the duration of a pause and is only reset after a configurable amount of `emergencyPauseDurationResetAfterSeconds` has passed.

Test coverage

Tests were added to the EmergencyPause integration test to check actions against each pause level.

- The `START_OPERATIONS` pause level is tested against initiating minting, redemption, liquidation, affected asset-manager operations, and affected collateral-pool operations.

- The `FULL` pause level is tested against affected asset-manager operations, including executing minting, reporting a payment default, finishing redemptions in various ways, exiting or destroying agents, and so on.
- The `FULL_AND_TRANSFER` pause level is tested against transfers only.

Attack surface

During each pause level, the attack surface of the system changes. For instance, if an attacker is anticipating the initiation of a `FULL` pause and has the ability to reorder transactions, they can cause the completion of a target operation to fail. However, this is an intended effect of that level of pause.

5.2. Removal of collateral deprecation

Description

Previously, this system included logic for deprecating specific collateral types. After a collateral has been marked as deprecated, it would still be included in liquidation calculations during the transitory period, but agents would be encouraged to switch to a new collateral before the time elapses for it to not count as collateral anymore.

After the changes reviewed in this audit, the ability for agents to switch their collateral type via the `switchVaultCollateral` function in the AgentCollateralFacet was completely removed. Since agents cannot switch collaterals, deprecating a collateral type becomes meaningless.

This change includes mostly the removal of logic and functions, but it also includes an adjustment to the liquidation logic to remove the special cases where a deprecated postexpiration collateral type is worth zero in collateral and cannot be used for payment.

Test coverage

Tests corresponding to `switchVaultCollateral` and collateral deprecation were removed.

Attack surface

The attack surface was strictly reduced by this change.

5.3. Automatic reduction of core-vault redemption lots

Description

Previously, the CoreVaultClientFacet required that the redeemed quantity of lots be less than the currently available core-vault lots in order to accept a redemption request.

This limitation was removed in order to allow for the core vault to be cleared. An added comment indicates that the off-chain core vault will reduce the redeemed lots in this case to the amount that is actually available.

Test coverage

The affected tests were modified to check that the process reverted with `InsufficientFunds` rather than the now-removed `NotEnoughAvailableOnCoreVault` error.

Attack surface

The attack surface now includes core-vault redemption requests that are larger than the core vault's available balance. However, based on the documentation, the core vault should now behave sensibly if such a request was made. Since the correct operation of the core vault itself is not in scope for this audit, this does not change the attack surface.

5.4. Addition of a limit to the amount of NAT equivalent burned by a request

Description

The `_burnVaultCollateral` function in the MintingDefaultsFacet swaps vault collateral for NAT before burning the NAT, in order to implement a cost to the agent in the case where the vault collateral is not NAT. Because the system only wishes to burn NAT, the agent is forced to purchase the collateral for NAT at the current price, and then that NAT is burned.

Previously, the amount of burned collateral was strictly set to the amount of reserved collateral that is relevant to the operation. However, since prices may have changed between the initiation of the operation and the burn, this means that, in times of market volatility, the request may burn more than its fair share of the vault collateral, even including the pool fees.

In order to ensure that this burn does not have unlimited access to vault collateral, this change implements a cap so that if it would exceed its fair share, then the system reduces both the amount of vault collateral transferred to the agent for the swap and the amount of NAT burned.

Test coverage

Integration tests were added to validate both the cases where prices have not moved enough to hit this cap and where prices moved substantially to hit the cap.

Attack surface

The attack surface is unchanged.

5.5. Addition of a permissionless function to consolidate small tickets

Description

The `consolidateSmallTickets` function was added to the RedemptionRequestsFacet that allows anyone to consolidate small tickets by either adding them to the agent's dust or removing from the agent's dust to create a larger ticket.

After successful consolidation, the function emits a new `RedemptionTicketsConsolidated` event to record and broadcast the range of tickets that were consolidated.

Test coverage

Integration tests were added to cover both the case of successful redemption-ticket consolidation and the case of unsuccessful consolidation due to an invalid provided ticket ID.

Attack surface

The attack surface of the system was increased to include this new permissionless function. Since the function does not behave differently depending on the sender and only takes in a ticket ID, the additional attack surface is very limited.

5.6. Minor changes

Description

Several minor changes were implemented in this diff.

- The lot redemption logic was changed to opportunistically assume that there is a ticket to redeem. This makes it so that the logic does not have to check if the redemption queue is empty. The helper function `_redeemFirstTicket` now returns a separate `bool queueEmpty` if the queue is actually empty.
- The payment challenge reward, which was previously denominated in NAT WEI, is now in units of USD. This change also modified the speed limit on governance changes to this parameter to reflect the new unit name and size.
- The liquidation collateral-amount calculation now uses the minimum price between the trusted and nontrusted price-feed results, rather than only using the nontrusted result.
- Additional checks were added to the SettingsInitializer to ensure that settings items corresponding to deprecated features are not mistakenly set.
- A getter view function was added to enumerate the list of authorized nongovernance emergency-pause senders.
- The `RedemptionRequiresClosingTooManyTickets` error threshold was adjusted to allow the case where the `requiredFAssets` is exactly equal to the maximum agent redemption.

Test coverage

No substantial logic was changed by these modifications, so test coverage remains unchanged.

Attack surface

The attack surface is unchanged by these modifications.

6. Assessment Results

During our assessment on the scoped FAssets contracts, we discovered four findings. No critical issues were found. One finding was of medium impact, two were of low impact, and the remaining finding was informational in nature.

6.1. 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.