OpenZeppelin | security

Sentora Firelight Vault Audit

July 21, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-06-16 To 2025-06-20
Languages	Solidity

Total Issues	9 (8 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	3 (3 resolved)
Low Severity Issues	3 (3 resolved)
Notes & Additional Information	2 (1 resolved)
Client Reported Issues	1 (1 resolved)

Scope

OpenZeppelin audited the firelight-protocol/firelight-core repository at commit 509038.

In scope were the following files:

```
contracts
├── lib
│   ├── Checkpoints.sol
└── FirelightVault.sol
└── FirelightVaultStorage.sol
```

A final review of commit 386dbde was performed after the fixes for all issues had been merged to confirm that no additional issues had been introduced. This commit included an additional renaming change that was also reviewed to ensure no unintended issues were introduced.

System Overview

The `FirelightVault` contract is an upgradeable, ERC-4626-compatible vault which manages the deposits and redemptions of an ERC-20 asset. It implements a delayed withdrawal request system where users must wait at least one full period before claiming their assets. When a withdrawal request is made, the receiver gains withdrawal shares for the next period which corresponds to the assets withdrawn.

The vault uses the Transparent Upgradeable Proxy pattern, with storage gaps reserved for safe upgrades. It implements role-based access control and includes functionality for pausing, blacklisting, and enforcing deposit limits. Changes in balances, total supply, and total assets are recorded through timestamp-based checkpoints. These checkpoints can later be queried for the historical state of the contract.

Security Model and Trust Assumptions

The `mint` function increases the total shares of the vault without increasing the total assets, which decreases the value of all the shares held by the current shareholders. Conversely, the `burn` function decreases the share supply which increases the value of shares. Shares cannot be immediately redeemed. Instead, the user must submit a withdrawal request and wait at least a `periodDuration` until it can be claimed.

The `FirelightVault` contract does not have any function that can update the `periodDuration` value. Making the period duration updatable will require significant changes to the contract logic because any change to the `periodDuration` could delay ongoing withdrawals or make withdrawal requests impossible to finalize.

Privileged Roles

Throughout the codebase, the following privileged roles were identified:

- `DEFAULT_ADMIN_ROLE` can grant and revoke all other roles.
- `MINTER_ROLE` can mint an arbitrary amount of shares to any address.
- `BURNER_ROLE` can burn shares owned by any user.
- `DEPOSIT_LIMIT_UPDATE_ROLE` can update the global deposit limit at any time.
- `BLACKLIST_ROLE` can blacklist addresses, preventing them from interacting with core functions. It can also remove addresses from the blacklist.
- `PAUSE_ROLE` can pause or unpause the `FirelightVault` contract.

Medium Severity

M-01 Burn Can Be Sandwiched

The `burn` function of the `FirelightVault` contract reduces the owner's shares and decreases `totalShares`, but does not reduce `totalAssets`. This results in an artificial increase in the share price which can be exploited. An attacker could front-run the `burn` function with a large deposit and immediately submit a withdrawal request at the inflated share price, effectively capturing unearned gains once the withdrawal is processed.

Consider modifying the `burn` function so that it transfers a proportional amount of assets to a `recipient`.

Update: *Resolved in pull request #7 and commit 386dbde.*

M-02 Period Duration Cannot Be Changed

The `FirelightVault` contract is intended to have an updatable `periodDuration`. This update will be made by calling a function which, as of the current contract commit, does not exist, but is expected to be added either before the update is needed or shortly thereafter. However, increasing the `periodDuration` can lead to issues with already-initiated withdrawal requests. This is because the `currentPeriod` function will begin underreporting the period number due to the longer duration, which will cause delays in withdrawals as a result of the `currentPeriod` check in `claimWithdrawal`.

Moreover, it can potentially lead to a loss of funds. If a user had requested a withdrawal in a previous period and, after the `periodDuration` change, the recalculated `currentPeriod` still matches the earlier request period, the user may attempt to withdraw again. In such cases, their shares would be burned during the call to `_requestWithdraw`, but the subsequent call to `claimWithdraw` would revert due to the `isWithdrawClaimed` check.

Consider keeping `periodDuration` constant throughout the lifetime of the vault to prevent inconsistencies such as the one discussed above.

Update: *Resolved in pull request #6 at commit 1433252.*

M-03 Missing Blacklist Check Can Lock Funds

A user can initiate a transfer to another user through the `withdraw` and `redeem` functions. However, there is no check to ensure that the receiver is not blacklisted. If funds are withdrawn or redeemed to a blacklisted address, they will become irretrievable as the `claimWithdraw` function will fail due to the blacklist check on the sender.

Consider adding a validation step in both the `withdraw` and `redeem` functions to ensure that the receiver is not blacklisted.

Update: *Resolved in pull request #5 at commit 68a9dd2.*

Low Severity

L-01 Rounding in Favor of User in `_requestWithdraw`

In the `_requestWithdraw` function of the `FirelightVault` contract, the asset-to-share conversion for `sharesWithdraw` rounds up. Usually, rounding up when converting to shares during a withdrawal is correct. However, in this case, `withdrawShares` are being minted to the user and, therefore, overestimating these shares would be rounding in favor of the user.

Consider rounding down instead of rounding up when converting to shares in the `_requestWithdraw` function.

Update: *Resolved in pull request #4 at commit c4ee01c.*

L-02 Incorrect Claim Timing in Comment

The comments above the `withdraw` and `redeem` functions state that withdrawals can be claimed in the period immediately following the one in which the request had been made. However, claims are only allowed starting from the period after the next full period. The current period plus the entire next period must pass before claiming is allowed.

Consider updating the comments to reflect the claim timing accurately.

Update: *Resolved in pull request #3 at commit 750b5f5.*

L-03 Missing Docstrings

Within `FirelightVaultStorage.sol`, multiple instances of missing docstrings were identified:

- The `DEPOSIT_LIMIT_UPDATE_ROLE` state variable
- The `MINTER_ROLE` state variable
- The `BURNER_ROLE` state variable
- The `BLACKLIST_ROLE` state variable
- The `PAUSE_ROLE` state variable

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #2 at commit dec2a9e.*

Notes & Additional Information

N-01 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, multiple instances of contracts without a security contact were identified:

- The `FirelightVault` contract
- The `FirelightVaultStorage` abstract contract

- The `Checkpoints` library

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #1 at commit 8de5adf.*

N-02 Incomplete SPDX License Identifiers

Throughout the codebase, multiple instances of files having incomplete SPDX license identifiers were identified:

- `FirelightVault.sol`
- `FirelightVaultStorage.sol`

To avoid legal issues regarding copyright and follow best practices, consider changing the SPDX license identifiers to comply with the Solidity documentation.

Update: *Acknowledged, will resolve. The Sentora team stated:*

> *We are going to use BSL, but the final specification file will be added later.*

Client Reported

CR-01 Limit Functions Not Overridden to Enforce Deposit/Withdraw Restrictions

The `FirelightVault` inherits from `ERC4626`, which includes the functions `maxDeposit`, `maxMint`, `maxWithdraw`, and `maxRedeem`. However, none of these functions are currently overridden to account for deposit limits, user block status, or whether the contract is paused. Consider overriding them to return values that correctly reflect these constraints.

Update: *Resolved in pull request #9 at commit ddb821b.*

Conclusion

The Firelight Vault enables deposits and redemptions of assets in exchange for shares, with a withdrawal delay that is based on a specified duration. The state of the contract can be queried on-chain, with state changes tracked through checkpoints. Privileged roles can mint and burn shares, blacklist accounts, and pause the contract. Three medium-severity issues were identified, along with some low- and note-severity issues. The Sentora team is appreciated for their responsiveness and assistance during the audit.