SHERLOCK

Security Review For Fluent

Collaborative Audit Prepared For: **Fluent**
Lead Security Expert(s): **dobrevaleri**
ParthMandale
Date Audited: **February 9 - February 11, 2026**

Introduction

The Fluent Network is an Ethereum-based Layer 2 blockchain designed to support blended execution across multiple virtual machines within a single execution environment.

Fluent enables developers to build and deploy on-chain applications using the Ethereum Virtual Machine (EVM), Solana Virtual Machine (SVM), and WebAssembly (Wasm), allowing applications written for different virtual machines and programming languages to interoperate atomically on one chain.

This architecture is powered by rWasm, Fluent's unified execution layer, which maintains compatibility with existing developer tooling while reducing fragmentation across blockchain ecosystems. By minimizing execution tradeoffs, Fluent allows developers to use the most appropriate language and virtual machine for each workload without sacrificing composability or security.

Fluent is also designed as a reputation-native execution layer, supporting applications that depend on long-term user behavior and contribution quality. This is enabled through Prints, Fluent's reputation and social capital data layer, and Fluent Connect, a native application that demonstrates how reputation can be used for identity, onboarding, and access control across on-chain applications.

The BLEND token is the native ecosystem token of the Fluent Network and is used to support network operations and participation, including transaction fees, execution and security incentives, builder staking, and governance. The BLEND token is the native ecosystem token of the Fluent Network and is used to support network operations and participation, including transaction fees, execution and security incentives, builder staking, and governance, as further described in Section 3 (Token and Token Distribution Information).

Scope

Repository: fluentlabs-xyz/blend-token

Audited Commit: 51dd554b3d37f029b8b53ca5d562b6dae31fddd4

Final Commit: dda48a47df573261c4e3d4e24f53d564c45c8deb

Files:

- script/DeployBlendToken.s.sol
- src/BlendToken.sol
- src/EIP3009Upgradeable.sol
- src/IEIP3009.sol
- test/BlendToken/AccessControl.t.sol
- test/BlendToken/Base.t.sol

- test/BlendToken/EIP2612Permit.t.sol
- test/BlendToken/EIP3009Auth.t.sol
- test/BlendToken/ERC20.t.sol
- test/BlendToken/Initialization.t.sol
- test/BlendToken/Minting.t.sol
- test/BlendToken/Multicall.t.sol
- test/BlendToken/Upgrades.t.sol
- test/fixtures/Signatures.sol
- test/invariant/BlendToken.invariant.t.sol

Final Commit Hash

dda48a47df573261c4e3d4e24f53d564c45c8deb

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	1	3

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: Cap Semantics Mismatch: `_cap` Enforces Live Supply, Not Lifetime Issuance

Source: https://github.com/sherlock-audit/2026-02-fluent-feb-9th/issues/16

Summary

`BlendToken` currently enforces `cap` as a maximum live supply, not a lifetime issuance ceiling. As confirmed by the developer here the intended tokenomics is “never mint more than X cumulatively” the current logic does not enforce that invariant.

Vulnerability Detail

In `BlendToken.sol`, both mint paths check against `totalSupply()` check: `if (newSupply > _cap) revert CapExceeded(_cap, newSupply);`:

```
mint: newSupply = totalSupply() + amount
mintBatch: newSupply = totalSupply() + total
```

`Burn` and `burnFrom` paths reduce `totalSupply()`

Because burn paths decrease `totalSupply`, a minter can mint to cap again, resulting in cumulative minted tokens exceeding the nominal cap over time. breaking the intended invariant.

Impact

The protocol intended cap as a lifetime issuance limit, but the current code breaks that invariant, creating a tokenomics integrity break as minters can mint beyond the expected lifetime cumulative ceiling.

Code Snippet

https://github.com/sherlock-audit/2026-02-fluent-feb-9th/blob/b3236ce5d12b6a11cd600d995fe3b5a294f57b87/blend-token/src/BlendToken.sol#L107-L108

https://github.com/sherlock-audit/2026-02-fluent-feb-9th/blob/b3236ce5d12b6a11cd600d995fe3b5a294f57b87/blend-token/src/BlendToken.sol#L126-L127

Tool Used

Manual Review

Recommendation

Track a separate lifetime issuance counter `mintedTotal` that only increments on mint and is never reduced by burn, and use that as `_cap` revert-check.

Issue L-1: Misordered InvalidPayee Arguments Cause Misleading Revert alert

Source: https://github.com/sherlock-audit/2026-02-fluent-feb-9th/issues/17

Summary

`receiveWithAuthorization` reverts with `InvalidPayee` arguments in reversed order, causing incorrect error interpretation by integrators, indexers, and debugging tools.

Vulnerability Detail

The error is declared as `error InvalidPayee(address expected, address actual);` But `EIP3009Upgradeable.sol` the revert is: `revert InvalidPayee(msg.sender, to);`

This maps: expected = msg.sender actual = to

which is opposite of the intended meaning. The expected payee is `to`, and the actual caller is `msg.sender`.

Impact

- incorrect monitoring/alert interpretation
- harder incident debugging
- integration mistakes in clients relying on structured custom error fields.

Code Snippet

https://github.com/sherlock-audit/2026-02-fluent-feb-9th/blob/b3236ce5d12b6a11cd600d995fe3b5a294f57b87/blend-token/src/EIP3009Upgradeable.sol#L95

Tool Used

Manual Review

Recommendation

Swap argument order in the revert site:

```
- if (to != msg.sender) revert InvalidPayee(msg.sender, to);
+ if (to != msg.sender) revert InvalidPayee(to, msg.sender);
```

Also update tests to assert the corrected argument ordering for InvalidPayee.

Issue L-2: Ambiguous CapExceeded Revert Used for Invalid Zero Cap

Source: https://github.com/sherlock-audit/2026-02-fluent-feb-9th/issues/18

Summary

`initialize` uses `CapExceeded(0, 0)` when `cap_ == 0`, which is semantically incorrect and can confuse deployment tooling and alerting systems.

Vulnerability Detail

In `BlendToken.sol` zero cap is checked as `if (cap_ == 0) revert CapExceeded(0, 0);` But `CapExceeded(cap, attemptedSupply)` is designed for "mint would exceed cap" situations, and not for "cap configuration is invalid"

Impact

monitoring can raise misleading incident labels.

Code Snippet

https://github.com/sherlock-audit/2026-02-fluent-feb-9th/blob/b3236ce5d12b6a11cd600d995fe3b5a294f57b87/blend-token/src/BlendToken.sol#L73

Tool Used

Manual Review

Recommendation

Introduce a dedicated configuration error and use it for `_cap == 0`:

```
error InvalidCap();
...
if (cap_ == 0) revert InvalidCap();
```

Keep CapExceeded only for genuine supply-over-cap checks during initialization/minting.

Also, update testcases accordingly.

Issue L-3: Storage gap size inconsistency between upgradeable contracts

Source: https://github.com/sherlock-audit/2026-02-fluent-feb-9th/issues/19

Description

The `BlendToken` contract uses a storage gap of 48 slots, while its inherited `EIP3009Upgradeable` contract uses a storage gap of 49 slots. Both contracts have 1 storage variable each, but their gap sizes differ.

In upgradeable contracts using the UUPS pattern, storage gaps reserve space for future storage variables that may be added in upgraded versions. The gap size determines how many new variables can be added without causing storage collisions with child contracts.

It is a good practice to have the same gap size in all upgradeable contracts.

Recommendation

Adopt a consistent gap size policy for all upgradeable contracts in the system. Set all gaps to 49 (or another chosen size) to ensure uniform upgrade capacity across the inheritance chain.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.