VeeFriends Smart Contracts v2 Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Giovanni Di Siena

Assisting Auditors

Blckhv

Slavcheww

October 15, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

This deployment of the VeeFriends `VFTokenC` is intended as a migration from the legacy contract in which previously burned tokens are minted by the admin on the new chain and then burned once again. All other burn mechanics and existing functionality should continue to work as expected.

5 Audit Scope

The audit scope was limited to:

```
contracts/accesscontrol/AccessControlVFExtension.sol
contracts/accesscontrol/OwnableVFExtension.sol
contracts/token/ERC721VFC.sol
contracts/VFTokenC.sol
```

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the VeeFriends Smart Contracts v2 smart contracts provided by VeeFriends. In this period, a total of 4 issues were found.

This review yielded 2 low and 1 informational findings, along with 1 gas optimization. The low issues are related to broken supply/balance invariants due to an edge case in batch burning and inconsistent state updates when tokens are sent directly to the burn address. Overall, the codebase and its test suite were found to be robust and well developed.

Summary

Project Name	VeeFriends Smart Contracts v2
Repository	smart-contracts-v2
Commit	44d1ff9f87b7...
Audit Timeline	Oct 12th - Oct 15th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	1
Gas Optimizations	1
Total Issues	4

Summary of Findings

[L-1] Total supply and balance invariants can be broken by burning the same token id multiple times within a given batch operation	Resolved
[L-2] Inconsistent state updates when tokens are burned and/or transferred directly to the `DEAD_ADDRESS`	Resolved
[I-1] Superfluous unchecked block can be removed	Resolved
[G-1] Unused internal functions should be removed to decrease bytecode size	Resolved

7 Findings

7.1 Low Risk

7.1.1 Total supply and balance invariants can be broken by burning the same token id multiple times within a given batch operation

Description: The `_burnBatch(uint256[])` function loops over all provided `tokenIds` and updates the relevant state; however, the ownership/approval logic is executed for all tokens in `_burnBatch(address,uint256[],bool)` before any state updates have occurred. As such, this fails to explicitly consider the scenario in which `DEAD_ADDRESS` is already the owner of a given `tokenId` if it is specified multiple times within the array and thus has been burned.

Due to the presence of an unchecked block in `totalSupply()`, the `burnCounter` can be increased arbitrarily by calling `burnBatch()` with multiple identical ids which can ultimately break the total supply invariant via underflow:

```
function totalSupply() public view returns (uint256) {
    unchecked {
        return _mintCounter - _burnCounter;
    }
}
```

This occurs because burning performs a transfer from the current owner address to the `DEAD_ADDRESS`, but only validates ownership at the beginning of execution. Instead, this logic should be executed for each id in the batch in turn after the approval/ownership/balance state from the previously burned token are updated. Since the ownership is updated to the `DEAD_ADDRESS` after the first loop iteration, subsequent executions perform transfers from the `DEAD_ADDRESS` to itself which also underflows when decrementing the balance despite the following assumption:

```
unchecked {
    // Cannot overflow, as that would require more tokens to be burned/transferred
    // out than the owner initially received through minting and transferring in.
    _balances[owner] -= 1;
}
```

Impact: The total supply and balance invariants are broken by batch burning the same token id multiple times.

Proof of Concept: The following test should be added to `VFTokenC.test.ts`:

```
it("burning non-existent tokens", async function () {
  const { vfTokenC, owner } = await connection.networkHelpers.loadFixture(
    fixtureWithMintedTokensForBurning
  );

  await vfTokenC.connect(owner).toggleBurnActive();

  const balanceBefore = await vfTokenC.balanceOf(owner.address);
  const deadBefore = await vfTokenC.balanceOf(DEAD_ADDRESS);
  const totalSupplyBefore = await vfTokenC.totalSupply();

  await expect(vfTokenC.connect(owner).burnBatchAdmin([102, 102, 102, 102]))
    .to.emit(vfTokenC, "Transfer")
    .withArgs(owner.address, DEAD_ADDRESS, 102);

  const balanceAfter = await vfTokenC.balanceOf(owner.address);
  const deadAfter = await vfTokenC.balanceOf(DEAD_ADDRESS);
  const totalSupplyAfter = await vfTokenC.totalSupply();

  console.log("Balance before:", balanceBefore.toString());
  console.log("Balance after:", balanceAfter.toString());

  console.log("Dead before:", deadBefore.toString());
  console.log("Dead after:", deadAfter.toString());
```

```
  console.log("Total Supply before:", totalSupplyBefore.toString());
  console.log("Total Supply after:", totalSupplyAfter.toString());

  // Tokens should now be owned by DEAD_ADDRESS
  expect(await vfTokenC.ownerOf(102)).to.equal(DEAD_ADDRESS);

  // Owner balance should have decreased by one
  expect(balanceAfter).to.equal(balanceBefore - 1n);

  // Total supply should have only decreased by one (violated)
  expect(totalSupplyAfter).to.equal(totalSupplyBefore - 1n);

  // DEAD_ADDRESS balance should have increased by one (violated)
  expect(deadAfter).to.equal(deadBefore + 1n);
});
```

Recommended Mitigation: Consider preventing duplicate ids in the array and/or validating ownership for each id in turn after each loop iteration rather than batching before any state updates have occurred.

VeeFriends: Fixed in commit dc834fa.

Cyfrin: Verified. Execution will now revert if the owner is already the dead address.

7.1.2 Inconsistent state updates when tokens are burned and/or transferred directly to the `DEAD_ADDRESS`

Description: The following state updates are performed when batch burning tokens:

```
    function _burnBatch(uint256[] calldata tokenIds) internal virtual {
        for (uint256 i; i < tokenIds.length; i++) {
            uint256 tokenId = tokenIds[i];
            address owner = ownerOf(tokenId);

            _beforeTokenTransfer(owner, DEAD_ADDRESS, tokenId, 1);

            // Clear approvals
@>          delete _tokenApprovals[tokenId];

            unchecked {
                // Cannot overflow, as that would require more tokens to be burned/transferred
                // out than the owner initially received through minting and transferring in.
@>              _balances[owner] -= 1;
            }

@>          _owners[tokenId] = DEAD_ADDRESS;

            emit Transfer(owner, DEAD_ADDRESS, tokenId);

            _afterTokenTransfer(owner, DEAD_ADDRESS, tokenId, 1);
        }

        unchecked {
@>          _burnCounter += tokenIds.length;
        }
    }
```

Note how while the `_balances[owner]` state is decremented, that of `DEAD_ADDRESS` is not incremented despite being granted ownership.

Now consider the direct transfer of a token to the `DEAD_ADDRESS`. While this is likely not desirable, there exists an asymmetry in that the balance this time will be updated while the `_burnCounter` state remains unchanged:

```
    function _transfer(
```

```
        address from,
        address to,
        uint256 tokenId
    ) internal virtual {
        if (to == address(0)) {
            revert ERC721VFTransferToTheZeroAddress();
        }

        if (ownerOf(tokenId) != from) {
            revert ERC721VFTransferFromIncorrectOwner(from, tokenId);
        }

        _beforeTokenTransfer(from, to, tokenId, 1);

        // Clear approvals from the previous owner
        delete _tokenApprovals[tokenId];

        unchecked {
            // `_balances[from]` cannot overflow for the same reason as described in `_burn`:
            // `from`'s balance is the number of token held, which is at least one before the current
            // transfer.
            // `_balances[to]` could overflow in the conditions described in `_mint`. That would require
            // all 2**256 token ids to be minted, which in practice is impossible.
            _balances[from] -= 1;
@>          _balances[to] += 1;
        }
@>      _owners[tokenId] = to;

        emit Transfer(from, to, tokenId);

        _afterTokenTransfer(from, to, tokenId, 1);
    }
```

Impact: State is partially corrupted when burning and/or transferring tokens directly to `DEAD_ADDRESS`.

Recommended Mitigation: Increment `_balances[DEAD_ADDRESS]` when tokens are burned. Additionally consider restricting direct token transfers to the `DEAD_ADDRESS`, or at least ensure this edge case is handled to remain consistent with the expected burn mechanics.

VeeFriends: Fixed in commits dc834fa and ed976c6.

Cyfrin: Verified. The dead address balance is now incremented on burning and direct transfers are no longer permitted.

7.2 Informational

7.2.1 Superfluous unchecked block can be removed

Description: Unchecked blocks in which no arithmetic operation are performed are superfluous and can be removed:

```
function totalMinted() public view returns (uint256) {
    unchecked {
        return _mintCounter;
    }
}

function totalBurned() public view returns (uint256) {
    unchecked {
        return _burnCounter;
    }
}
```

Recommended Mitigation:

```
function totalMinted() public view returns (uint256) {
-   unchecked {
        return _mintCounter;
-   }
}

function totalBurned() public view returns (uint256) {
-   unchecked {
        return _burnCounter;
-   }
}
```

VeeFriends: Fixed in commit ed976c6.

Cyfrin: Verified.

7.3 Gas Optimization

7.3.1 Unused internal functions should be removed to decrease bytecode size

Description: The internal functions `_safeMint()`, `_safeMintBatch()`, `_airdrop()`, `_mintBatch()`, and `_burn()`, do not appear to be used and it is understood that there is not currently any requirement for additional functionality to be added to `VFTokenC`. The contracts are not upgradeable, so a separate deployment would be needed for any future upgrade, meaning that these functions should most likely be excluded from the current release to decrease the bytecode size.

VeeFriends: Fixed in commits dc834fa and ed976c6.

Cyfrin: Verified.