Unstoppable Audit Report

Prepared by Ahmad Faraz

Auditing Protocol: `Unstoppable`

Date: July 29, 2025

Contents

1 Disclaimer

I Ahmad Faraz makes every effort to understand the protocol and identify vulnerabilities within the given timeframe but holds 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	High	Medium	Low
High	High	High/Medium	Medium
Medium	High/Medium	Medium	Medium/Low
Low	Medium	Medium/Low	Low

3 Audit Details

3.1 Scope

The audit covered the following files:

- `./src/`
- `UnstoppableMonitor.sol`
- `UnstoppableVault.sol`

4 Protocol Summary

The `Unstoppable` protocol implements a tokenized vault using the `ERC-4626` standard. The vault is preloaded with 1,000,000 DVT tokens and is designed to offer flash loans free of charge during a grace period (a predefined end timestamp). To validate its design in a semi-public environment, the developers launched a beta on testnet, where they monitor vault activity using an `UnstoppableMonitor` contract. This monitor regularly requests flash loans to verify the liveness of the protocol.

4.1 Roles

- **Vault:** Holds ERC tokens. Lend a flash loan.
- **Monitor:** Requesting for flash loan.

5 Executive Summary

The `Unstoppable` smart contract protocol audited by Ahmad Faraz. The audit revealed a total of 3 issues, classified as follows.

Severity Level	Issue Count
High	3
Total	**3**

6 Findings

6.1 High Severity

6.1.1 [H-1] Potential Denial of Service in `UnstoppableVault`

- **Description:**The `UnstoppableVault` protocol is designed to provide flash loans to `UnstoppableMonitor`. Within the `UnstoppableVault::flashLoan` function, a strict check ensures that `convertToShares(totalSupply) != balanceBefore`, requiring the vault to maintain an equal number of assets to the total supply of shares per the `ERC4626` vault standard. The `UnstoppableVault` does not directly handle token deposits. If a malicious user sends tokens directly to the contract, it breaks the invariant enforced in `UnstoppableVault::flashLoan`, causing the function to always revert.

```
1  function flashLoan(IERC3156FlashBorrower receiver, address _token,
       uint256 amount, bytes calldata data) external returns (bool) {
2      if (amount == 0) revert InvalidAmount(0);
3      if (address(asset) != _token) revert UnsupportedCurrency();
4
5      uint256 balanceBefore = totalAssets();
6  @>  if (convertToShares(totalSupply) != balanceBefore) revert
7      InvalidBalance();
8
9      ERC20(_token).safeTransfer(address(receiver), amount);
10     uint256 fee = flashFee(_token, amount);
11
12     if (receiver.onFlashLoan(msg.sender, address(asset), amount,
           fee, data) != keccak256("IERC3156FlashBorrower.onFlashLoan")
           ) {
13         revert CallbackFailed();
14     }
15     ERC20(_token).safeTransferFrom(address(receiver), address(this)
           , amount + fee);
16     ERC20(_token).safeTransfer(feeRecipient, fee);
17
18     return true;
19 }
```

- **Impact:** Violating this condition renders the vault inoperable, preventing all users from obtaining flash loans. While the vault owner can adjust tokens and shares, the function remains susceptible to DoS attacks.

- **Cause:** Strict check `if (convertToShares(totalSupply) != balanceBefore)` .
- **Recommended Mitigation:** Implement a `deposit` function in `UnstoppableVault` to allow users to deposit tokens, ensuring that the vault's shares are updated to match the number of tokens deposited.
- **Proof of Concept:**
  Normal User:
  1. Calls `flashLoan`.
  2. The condition `convertToShares(totalSupply) != balanceBefore` is checked.
  3. The `flashFee‘ is calculated.`
  4. The flash loan is granted.
  5. The flash loan is recovered.

  Malicious User:
  1. Calls `ERC4626::deposit(uint256 assets, address receiver)`.
  2. The condition `convertToShares(totalSupply) != balanceBefore` causes reverts for all users.

```
 function testDoSFlashLoanByDirectTransfer() public {
    // 1. Attacker directly transfers 1 token to the vault
    vm.startPrank(attacker);
    token.mint(attacker, 1);
    token.transfer(address(vault), 1);
    vm.stopPrank();

    // 2. A legitimate user tries to take a flash loan
    vm.startPrank(address(this));
    token.mint(address(this), 20);
    token.approve(address(vault), 20);
    vault.deposit(20, address(this));

    vm.expectRevert();
    vault.flashLoan(IERC3156FlashBorrower(monitor), address(token),
        10, "");
    vm.stopPrank();
}
```

6.1.2 [H-2] `UnstoppableMonitor::onFlashLoan` Reverts on Zero Fee

- **Description:**The `UnstoppableVault` protocol is designed to provide flash loans to `UnstoppableMonitor`, which can request loans before or after the `end` period. However, the `UnstoppableMonitor` contract restricts itself to non-zero fees, reverting with an `UnexpectedFlashLoan()` error, even in cases where the monitor is eligible for a zero-fee flash loan during the grace period.

```
function onFlashLoan(address initiator, address token, uint256
    amount, uint256 fee, bytes calldata) external returns (bytes32)
    {
    if (initiator != address(this) || msg.sender != address(vault)
        || token address(vault.asset()) ||
@>    fee != 0
    ){
```

```
        revert UnexpectedFlashLoan();
    }

    ERC20(token).approve(address(vault), amount);
    return keccak256("IERC3156FlashBorrower.onFlashLoan");
}
```

- **Impact:** The `UnstoppableMonitor` blocks itself from taking flash loans within the 'end' period when the fee is zero.
- **Cause:** `fee != 0` revert with an `UnexpectedFlashLoan()`
- **Recommended Mitigation:** Remove the `fee != 0` check to allow zero-fee flash loans. Alternative logic can be implemented to manage this behavior.

```
function onFlashLoan(address initiator, address token, uint256
    amount, uint256 fee, bytes calldata) external returns (bytes32)
    {
    if (initiator != address(this) || msg.sender != address(vault)
        || token != address(vault.asset()) ||
-   fee != 0
    ){
     revert UnexpectedFlashLoan();
    }
    ERC20(token).approve(address(vault), amount);
    return keccak256("IERC3156FlashBorrower.onFlashLoan");
}
```

- **Proof of Concept:**
  1. Monitor initiates a flash loan.
  2. Condition `block.timestamp < end && _amount < maxFlashLoan(token)` is met.
  3. The `fee` is set to `0`. 4. `receiver.onFlashLoan(@params)` is called.
  5. `UnstoppableMonitor` reverts due to the `fee != 0` check.

```
function testFlashLoan() public {
    vm.startPrank(address(monitor));
    vault.flashLoan(IERC3156FlashBorrower(monitor), address(token),
        10, "");
    vm.stopPrank();
}
```

6.1.3 [H-3] `UnstoppableVault` Fails to Recover Fees

- **Description:**When `UnstoppableMonitor` takes a flash loan from `UnstoppableVault`, the `UnstoppableMonitor::onFlashLoan` function does not approve sufficient tokens for `UnstoppableVault` to recover both the loan amount and the fee,particularly when flash loans are requested after the grace period, where a non-zero fee applies.

```
function onFlashLoan(address initiator, address token, uint256
    amount, uint256 fee, bytes calldata) external returns (bytes32)
    {
```

```
2       if (initiator != address(this) || msg.sender != address(vault)
            || token != address(vault.asset()) || fee != 0) {
3           revert UnexpectedFlashLoan();
4       }
5
6   @>      ERC20(token).approve(address(vault), amount);
7       return keccak256("IERC3156FlashBorrower.onFlashLoan");
8   }
```

- **Impact:** `UnstoppableVault` cannot recover its fee from `UnstoppableMonitor`, which causes transactions to revert with an `ERC20InsufficientAllowance` error.
- **Cause:** `UnstoppableMonitor` not allowing `Unstoppable` to get fee back `ERC20(token).approve(address(vault), amount)`
- **Recommended Mitigation:** Update `UnstoppableMonitor::onFlashLoan` to approve `amount + fee` to allow `UnstoppableVault` to recover both the loan and the fee.

```
1   function onFlashLoan(address initiator, address token, uint256
        amount, uint256 fee, bytes calldata) external returns (bytes32)
        {
2       if (initiator != address(this) || msg.sender != address(vault)
            || token != address(vault.asset()) || fee != 0) {
3           revert UnexpectedFlashLoan();
4       }
5
6   -     ERC20(token).approve(address(vault), amount);
7   +     ERC20(token).approve(address(vault), amount + fee);
8       return keccak256("IERC3156FlashBorrower.onFlashLoan");
9   }
```

- **Proof of Concept:**
  Run the code with proper setup

```
1   function testFlashLoan() public {
2       vm.startPrank(address(monitor));
3       vault.flashLoan(IERC3156FlashBorrower(monitor), address(token),
            10, "");
4       vm.stopPrank();
5   }
```