Unstoppable Security Review

Reviewers
Américo Júnior, Security Auditor
September 26, 2022

1 Executive Summary

Over the course of 5 days in total, Unstoppable engaged with Américo Júnior to review Unstoppable.

We found a total of 4 issues with Unstoppable.

Repository	Commit
Unstoppable	commithash

Summary

Type of Project	TYPE
Timeline	Sep 24, 2022 - Sep 26, 2022
Methods	Manual Review
Documentation	High
Testing Coverage	High

Total Issues

Critical Risk	0
High Risk	2
Medium Risk	1
Low Risk	1
Gas Optimizations and Informational	0

Contents

2 Unstoppable

Unstoppable is a damnn vulnerable defi challenge, there's a lending pool with a million DVT tokens in balance, offering flash loans for free.

3 Introduction

Unstoppable offers flash loans of DVT tokens for free.

The focus of the security review was on the following:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of brink according to the specific commit by a three person team. Any modifications to the code will require a new security review.

4 Findings

4.1 High Risk

4.1.1 Dangerous strict equalities

Severity: High

Context: `UnstoppableLender.sol#L40`

Description: Use of strict equalities that can be easily manipulated by an attacker.

If an attacker transfer DVT token to the contract without using the `depositTokens()` function, the poolBalance doesn't change and the assert at line 40 returns false, consequently it's impossible to flash loan.

```
contract UnstoppableLender {
    ...
    function depositTokens(uint256 amount) external nonReentrant {
        require(amount > 0, "Must deposit at least one token");
        // Transfer token from sender. Sender must have first approved them.
        damnValuableToken.transferFrom(msg.sender, address(this), amount);
        poolBalance = poolBalance + amount;
    }
    ...
    function flashLoan(uint256 borrowAmount) external nonReentrant {
        ...
        // Ensured by the protocol via the `depositTokens` function
        assert(poolBalance == balanceBefore);

        damnValuableToken.transfer(msg.sender, borrowAmount);
        ...
    }
    ...
}
```

Recommendation:

```
+Don't use strict equality to determine if a pool has same balance.
- ...
```

4.1.2 Reentrancy

Severity: High

Context: UnstoppableLender.sol#L26-31

Description: A state variable is changed after a contract uses call.value. The attacker uses a fallback function—which is automatically executed after Token is transferred from the targeted contract—to execute the vulnerable function again, before the state variable is changed. Abusing this vulnerability I created an exploit that makes 2 deposits, and only 1 is updated in the pool balance, and that makes us break flash loan functionality.

```
contract UnstoppableLender {
    ...
    function depositTokens(uint256 amount) external nonReentrant {
        ...
        damnValuableToken.transferFrom(msg.sender, address(this), amount);
        poolBalance = poolBalance + amount;
        ...
    }
    ...
}

contract Attack {
    UnstoppableLender public unstoppableLender;

    constructor(address _unstoppableLenderAddress) {
        unstoppableLender = UnstoppableLender(_unstoppableLenderAddress);
    }

    fallback() external payable {
        unstoppableLender.depositTokens(10);
    }

    function attack() external payable {
        unstoppableLender.depositTokens(10);
        unstoppableLender.flashLoan(10);
    }
}
```

Recommendation:

```
+Ensure all state changes happen before calling external contracts.
- ...
```

4.2 Medium Risk

4.2.1 Incorrect versions of Solidity

Severity: Medium

Context: `UnstoppableLender.sol#L3`, `ReentrancyGuard.sol#L3` and `IERC20.sol#L3`

Description: `solc` frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex `pragma` statement.

Recommendation:

```
+Deploy with any of the following Solidity versions:
+ 0.5.16 - 0.5.17
+ 0.6.11 - 0.6.12
+ 0.7.5 - 0.7.6
+ 0.8.4 - 0.8.7 Use a simple pragma version that allows any of these versions.
↪  Consider using the latest version of Solidity for testing.
- ...
```

4.3 Low Risk

4.3.1 Unchecked transfer

Severity: Low

Context: UnstoppableLender.sol#L26-31 and UnstoppableLender.sol#L33-48

Description: The return value of an external transfer/transferFrom call is not checked

```
contract UnstoppableLender {
    ...
    function depositTokens(uint256 amount) external nonReentrant {
        ...
        damnValuableToken.transferFrom(msg.sender, address(this), amount);
        ...
    }
    ...
    function flashLoan(uint256 borrowAmount) external nonReentrant {
        ...
        damnValuableToken.transfer(msg.sender, borrowAmount);
        ...
    }
    ...
}
```

Recommendation:

```
+Ensure that the transfer/transferFrom return value is checked.
- ...
```