Naive Receiver Security Review

Reviewer
Américo Júnior, Security Auditor
October 3, 2022

1 Executive Summary

Over the course of 3 days in total, Damn Vulnerable DeFi engaged with Américo Júniorto review Naive Receiver.

I found a total of 1 issue with Naive Receiver.

Repository	Commit
Naive Receiver	commithash

Summary

Type of Project	TYPE
Timeline	Sep 27, 2022 - Sep 29, 2022
Methods	Manual Review
Documentation	High
Testing Coverage	High

Total Issues

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

Contents

2 Naive Receiver

Naive Receiver is a damnn vulnerable defi challenge, There's a lending pool offering quite expensive flash loans of Ether, which has 1000 ETH in balance.

3 Introduction

Naive Receiver offers quite expensive flash loans of ETH. 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 one person. Any modifications to the code will require a new security review.

4 Findings

4.1 High Risk

4.1.1 Lack of validation of the amount to be borrowed

Severity: High

Context: `NaiveReceiverLenderPool.sol#L21-L41`

Description: Using the function `flashLoan` is possible to set borrower address and the amount to borrow, the borrower address have to be a `FlashLoanReceiver` contract address deployed by the user, and the amount can be any, because there is no validation for the amount to be borrowed.

If an attacker makes an flash loan setting victim `FlashLoanReceiver` contract address and `amount` 0, the victim will borrow 0 ETH and pay 1 ETH as fee. Due to the lack of any restrictions, the attacker can do this as many times as he wants until he steals all the ETH from the victim's contract.

```
contract NaiveReceiverLenderPool {
    ...
    uint256 private constant FIXED_FEE = 1 ether; // not the cheapest flash
    loan
    ...
    function flashLoan(address borrower, uint256 borrowAmount) external
    nonReentrant {

        uint256 balanceBefore = address(this).balance;
        require(balanceBefore >= borrowAmount, "Not enough ETH in pool");


        require(borrower.isContract(), "Borrower must be a deployed contract");
        // Transfer ETH and handle control to receiver
        borrower.functionCallWithValue(
            abi.encodeWithSignature(
                "receiveEther(uint256)",
                FIXED_FEE
            ),
            borrowAmount
        );

        require(
            address(this).balance >= balanceBefore + FIXED_FEE,
            "Flash loan hasn't been paid back"
        );
    }
    ...
}
```

Recommendation:

```
+ Implement validation for the amount to be borrowed, must be greater than
  zero.
+ Validate if the person making the flash loan owns the `FlashLoanReceiver`
  contract.
- ...
```