Zellic

SushiSwap FundsReturner

Smart Contract Security Assessment

April 20, 2023

Prepared for:

Jared Grey

SushiSwap

Prepared by:

William Bowling and Syed Faraz Abrar

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for SushiSwap from April 19th to April 20th, 2023. During this engagement, Zellic reviewed SushiSwap FundsReturner's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is there a way for users to claim redemption tokens more than once?
- Can the owner accidentally lock funds into the contract?
- Can the owner accidentally lose ownership of the contract?

1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.3 Results

During our assessment on the scoped SushiSwap FundsReturner contracts, we discovered one critical finding.

Additionally, Zellic recorded its notes and observations from the assessment for SushiSwap's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	0
Medium	0
Low	0
Informational	0

2 Introduction

2.1 About SushiSwap FundsReturner

SushiSwap FundsReturner is a redemption contract for users affected by the SushiSwap Router Processor 2 exploit.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review the contracts' external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the code base in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood.

There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

SushiSwap FundsReturner Contracts

Repository	https://github.com/LufyCZ/sushi-funds-returner
Version	sushi-funds-returner: `601433d81a311a68033a90c1954508f4a4c4abc0`
Program	• SushiFundsReturner
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-days. The assessment was conducted over the course of two calendar days.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

William Bowling, Engineer
vakzz@zellic.io

Syed Faraz Abrar, Engineer
faith@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

April 19, 2023	Start of primary review period
April 20, 2023	End of primary review period

3 Detailed Findings

3.1 Allowing users to claim for other addresses can lead to stolen funds

- **Target**: SushiFundsReturner
- **Category**: Coding Mistakes
- **Likelihood**: High
- **Severity**: Critical
- **Impact**: Critical

Description

The `claim()` function is defined as follows:

```
function claim(uint256 index, address account, uint256 amount,
    address token, bytes32[] calldata merkleProof) external { /* ... */ }
```

The function allows the caller to pass in the address of the `account` that will claim the tokens, and then the `amount` is transferred to that account.

Impact

The original vulnerability that was exploited allowed attackers to steal funds from users who had approved the SushiSwap Router Processor 2 contract. Currently there are 60 addresses that have not revoked approval to that contract.

Since the Router Processor 2 contract is not upgradable, the vulnerability still exists in that contract. An attacker can claim tokens for the 60 unrevoked addresses and then exploit the original vulnerability to steal the claimed tokens from the users yet again.

Recommendations

Consider replacing the use of the `account` parameter with `msg.sender`.

Remediation

This issue has been acknowledged by SushiSwap, and a fix was implemented in commit 1a50f5a0.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment.

4.1 Second preimage attacks on Merkle trees

The OpenZeppelin implementation of Merkle trees double hashes the leaf nodes to distinguish them from intermediate nodes in the tree. This prevents second preimage attacks, as an attacker can never find an input that would result in the same hash as an intermediate node.

In the `SushiFundsReturner` contract, the leaf nodes are hashed as follows:

```
function claim(uint256 index, address account, uint256 amount,
    address token, bytes32[] calldata merkleProof) external {
    // [ ... ]

    // Verify the merkle proof.
    bytes32 node = keccak256(abi.encodePacked(index, account, amount,
    token));
    require(MerkleProof.verify(merkleProof, merkleRoot, node),
    'MerkleDistributor: Invalid proof.');

    // [ ... ]
}
```

In this specific instance, since the output of `abi.encodePacked( ... )` will be a value that is greater than 64 bytes, it is impossible for this to result in a value that might match the concatenation of two intermediate hashes. Furthermore, `index` and `account` are essentially out of the user's control, so that makes the attack even more infeasible.

Even though this contract is safe from a second preimage attack, we find it important to mention that it is not hardened against it. With redeem/refund style contracts like this becoming more and more common amongst protocols, there is a nonzero chance that, in the future, a similar contract may be deployed where the single hashing of the leaf node will leave it vulnerable to a second preimage attack. This would allow an attacker to find a collision and steal user funds.

5 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1 Module: SushiFundsReturner.sol

Function: `claim(uint256 index, address account, uint256 amount, address token, byte[32][] merkleProof)`

This function allows a user to claim tokens if they can supply a valid Merkle proof to prove that the provided values were used in the creation of the Merkle tree.

Preconditions

- The contract is not frozen.
- The index has not already been claimed.

Inputs

- `index`
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Is used to generate the node hash.
- `account`
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Is used to generate the node hash and will be where the tokens are sent.
- `amount`
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Is used to generate the node hash and will be the amount of tokens that are sent.

- `token`
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Is used to generate the node hash and specifies which tokens will be sent.
- `merkleProof`
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Used to verify that the node hash (generated from the index, account, amount, and token inputs) is part of the Merkle tree defined by `merkleRoot`.

Branches and code coverage (including function calls)

Intended branches

- The Merkle proof is valid, and the correct tokens are transferred to the account.
  - ☑ Test coverage

Negative behavior

- A claim can be made only once.
  - ☑ Negative test
- The Merkle proof is invalid.
  - ☐ Negative test

Function call analysis

- `IERC20(token).transfer(account, amount)`
  - **What is controllable?** `token`, `account`, and `amount` are all controllable but have been validated against the Merkle root to ensure they are legitimate.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The tokens are not transferred, and the index is not marked as claimed.

Function: `freeze(bool _freeze)`

Allows for the contracts frozen status to be changed. If `frozen` is true, then no claims can be made.

Preconditions

- Only callable by the owner.

Inputs

- `_freeze`
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Determines if claims can be made or not.

Branches and code coverage (including function calls)

Intended branches

- The contract is frozen, and no more claims can be made.
  - ☐ Test coverage

Negative behavior

- The caller is not the owner.
  - ☐ Negative test

Function: `yoink(address token)`

Allows the owner of the contract to transfer every `token` owned by this contract to themselves.

Preconditions

- Only callable by the owner.

Inputs

- `token`
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: The token balance of this contract will be transferred to the caller.

Branches and code coverage (including function calls)

Intended branches

- The specified tokens are transferred to the owner.
  - ☑ Test coverage

Negative behavior

- The caller is not the owner.
  - ☐ Negative test

Function call analysis

- `ERC20(token).transfer`
  - **What is controllable?** The `token` address is fully controllable.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable.
  - **What happens if it reverts, reenters, or does other unusual control flow?** None of the tokens will be transferred.

6 Audit Results

At the time of our audit, the code was not deployed to mainnet Arbitrum.

During our audit, we discovered one critical finding. SushiSwap acknowledged the finding and implemented a fix.

6.1 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.