Switcheo Ethereum Contracts Security Audit

December 10th, 2020 (*Updated December 17th, 2020*)

@lucash-dev

https://hackerone.com/lucash-dev

https://github.com/lucash-dev

Disclaimer

Security audits are inherently limited -- there is no possibility of strict proofs that a given system doesn't contain further vulnerabilities. The best that can be offered is a good faith effort to find as many vulnerabilities as possible within the resource constraints of the project.

Given the above, this audit report is provided "AS IS", without any guarantee of correctness or completeness. The author takes no responsibility for any loss or damage caused by the use, misuse or failure to use the information contained in this document, as well as for any information that might be missing from it (e.g. missed vulnerabilities).

By using the information contained in this report, you agree to do so at your own risk, and not to hold the authors liable for any consequences of such use.

Summary

This document contains the results of the audit conducted on the Switcheo Tradehub Ethereum smart contracts.

As *no immediately exploitable vulnerabilities* were found, this document will list the following information:

- **Methodology**: a general description of the methodology used to find possible issues.
- **Attack Scenarios**: a brief description of the main attack scenarios considered during this audit, even though no means of performing the attack were found.
- **Common Weaknesses**: a brief description of the common weaknesses that the code was searched for, though they weren't found in the audited code.
- **Possible Vulnerabilities**: a list of actually foud issues that might lead to vulnerabilities depending on factors external to the smart contracts being audited, depending on assumptions about contracts interacted with or other pieces of technology.

- **Improvement Suggestions**: a list of issues that, if corrected, might lead to increased security, though there is no evidence they lead to any vulnerability as they were found.

Scope

The scope of this audit are the contracts `Wallet.sol`, `LockProxy.sol`, and `BalanceReader.sol`, as found at https://github.com/Switcheo/switcheo-tradehub-eth, at the commit `6521a12d06c8dddc9d8186f24e081cb6bcf90022`.

Methodology

The audit was conducted manually by an experienced security researcher, by inspecting the code in two different ways:

- Systematic review of every function in the contracts, spotting missing best practices and logical flaws.
- Free exploration of interaction points and execution flows, searching for concrete attack vectors and invalid assumptions.

While the first approach mimics a more traditional code review, the second one tries to reproduce the kind of coverage you would obtain from a "bug bounty" program.

It's the author's belief that combining these approaches offer much higher likelihood of catching high-severity issues than using either individually.

Attack Scenarios

This section describes the main (non-exhaustive list) attack scenarios considered during the audit of the code. *No vulnerability* was found that could enable an attacker to perform them.

- **Unauthorized user can perform operations on wallets.**

  A malicious user, without proper authorization, is able to perform any operation on a wallet created by another user.

- **Tampering of Signed Messages**

  Messages authorizing transactions from a wallet are tampered, e.g. changing the "from" address or the amount, and are still accepted by the smart contract.

- **Preventing processing of legitimate operations (DoS)**

  A malicious user can change the state of the contracts in a way to prevent the processing of legitimage transactions, e.g. by marking a message as "seen" before it was processed.

- **Manager Contract can obtain user funds**

  The Cross-Chain Manager (CCM) contract is allowed to freely move any amount from user Wallets.

Common Weaknesses

This section describes some of the common weaknesses (non-exhaustive list) the code was searched for during the audit. *None of these weaknesses* were found in the code.

- **Incorrect Method Access Level**

  This weakness is found when a method in a contract was assigned the wrong access level (private/public/external) allowing an attacker to perform operations that shouln't have been authorized.

- **Lack of Sender Validation for Restricted Functionality**

  This weakness is found when methods that should be restricted fail to validate `msg.sender` and thus allow any user to access it's functionality.

- **Contract Reentrancy**

  This weakness is found when external/public methods that allow calls to user-specified methods are vulnerable to calling other methods in the original smart contract, leading to possible inconsitency in contract state that might be exploitable.

- **Improper Validation of Values Provided by Sender**

  This weakness happens when parameters to an external/public method are trusted to be correct without any validation and are used for determining the state or behaviour of the contract.

- **Improper Validation of Cryptographic Proofs**

  This weakness happens when authorization to certain operations is controlled by cryptographic proofs (e.g. signatures) provided by the message sender, yet the smart contract doesn't properly validate the proof provided.

- **Insecure Cryptographic Scheme used for Authorization**

  This weakness is found when authorization to certain operations is controlled by cryptographic proofs (e.g. signatures or pre-images) but the cryptographic scheme used can't in reality prove the legitimacy of the request. One example is a use of a weak hash function, or requiring a signature of a value that isn't a hash of a pre-image.

- **Lack of Replay Protection**

This weakness is found when the smart contract accepts multiple times the same cryptographic proof as authorization to perform an operation, leading to unauthorized users being able to perform again (replay) the operation.

- **Rounding Errors**

  This weakness happens when division operations aren't consistently rounded, leading to inconsistent contract state that might be exploitable.

Possible Vulnerabilities Found

This section describes issues that aren't exploitable directly in the contract, but can *possibly* result in a vulnerability depending on behavior of external pieces of software that interact with the contract.

Whether these need to be fixed or not will depend on evaluation of the external pieces of software, which is beyond the scope of this audit.

1. Ambiguous field in LockEvent

The event `LockEvent` contains a `fromAdress` field whose semantics can vary depending on whether the event was emitted by a call to the `lock` method or the `lockFromWallet` method.

In the first case it is necessarily the user who owned the funds originally, in the second it is whoever initiated the transaction (msg.sender).

If logic external to the EVM trusts this value to identify the original depositor, then it is vulnerable to fronting -- i.e. an attacker can send the legitimate lock data and signature and might succeed in having it processed thus generating an event in which the "fromAddress" point to the attacker, and possibly gaining access to funds.

Update by Switcheo Team: The Switcheo team has discussed the possible issue and determined that the event value is unused outside the EVM for any sort of authentication or ownership purpose, as the control of funds is simply passed to the receiver address / signatures on the sidechain.

Improvement Suggestions

This section contains suggestions for improvement of the contracts and issues that don't represent a significant security risk. We consider fixing those would benefit the legibility and maintainability of the contracts though.

1. Wrong error message in `_lock`

The error message "Fee amount cannot be greater than amount" as the code enforces the fee being strictly less than the amount.

Update by Switcheo Team: The Switcheo team acknowledges the issue, but considers it not significant to justify redeployment of the contract. It will be addressed if a new version is ever deployed.

2. Replicated logic

Methods `_callOptionalReturn` and `_isContract` are replicated accross multiple `.sol` files. Code maintainability would be improved if these were extracted into a library.

Update by Switcheo Team: The Switcheo team acknowledges the issue, but considers it not significant to justify redeployment of the contract. It will be addressed if a new version is ever deployed.

3. Variable naming

Multiple methods include variables names `assetHash` or similar. These variables don't hold the value of a hash of the asset though. Instead it directly holds the address to an ERC20 token. That naming might make it confusing to someone inspecting the contract, which might lead to errors that eventually cause a vulnerability.

Update by Switcheo Team: The Switcheo team acknowledges the issue, but considers it not significant to justify redeployment of the contract. It will be addressed if a new version is ever deployed.

Conclusion

This document described the methodology for conducting the audit of the code, the potential attacks and weaknesses considered.

No immediately exploitable vulnerablities were found in the audited code, but some potential issues and possible improvements are discussed.