Switcheo Ethereum Contracts Security Audit

September 17th, 2021

@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, for any potssible inaccuracies in 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.

This document does not represent endorsement of any product or investment, and the authors do not endorse any product or investment related to this audit.

Summary

This document contains the results of the audit conducted on the Switcheo Tradehub Ethereum smart contracts, and related Poly Network smart contracts.

- **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.
- **Exploitable Vulnerabilities**: a list of actually found issues that could be exploited in the audited contracts.
- **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

Security review of smart contracts found at:

- https://github.com/polynetwork/eth-contracts (at commit 2b1cbe073e40a7bd26022d1cda9341b4780d07ee)
- https://github.com/Switcheo/switcheo-tradehub-eth (at commit 3191692ac5d06e19ad208aeb8d29c0dc5a92b030)
- https://github.com/Switcheo/switcheo-tradehub-zil (at commit 2a606a55de3b4504fdb50e3dff2f14441e136450)

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.

Serialization Attacks

Since transaction data is serialized and de-serialized within contract implementations in different chains, an issue in that area of the code could allow an attacker to bypass validations and force a contract to unlock more tokens than have been locked in the other chain. No issues related to that scenario were found.

Excess Minting of SWTH Tokens

Since SWTH tokens are burned and minted instead of simply locked and unlocked, an issue in validation logic could lead to minting of funds without corresponding burning in a different chain. One bug relating to this scenario was found (see below).

Malleable Transactions

Locking funds from wallets happens when signed messages are sent to the LockProxy contracts, and the only the signatures are validated, not the sender of the message. That means if any parameters are not included in the hash signed by the user, it could be used to perform an operation in a way not intended by the user who signed it. An attack related to this scenario was identified (same as in the scenario above).

Arbitrary Calls from CCM

The Switcheo LockProxy security model depends completely on the good functioning of the CCM. If a security issue in that contract allows for an attacker to send arbitrary calls to the LockProxy, tokens could be freely transferred by the attacker. Not issue related to this scenario was found.

Limitations

The Poly Network depends on the relayer software to register transactions in the Poly Chain, after observing cross-chain calls in the source chains.

That means any security issue that compromises the relayer software might lead to arbitrary CCM calls.

As the relayer software isn't in the scope of this audit, we must assume it is safe. Any potential security issue arising from vulnerabilities in the relayer software would not have been identified in this audit.

Exploitable vulnerabilities

Lack of validation of `callAmount` in LockProxy

The contract LockProxy has a critical issue, introduced in commit `37b243e5c7c1e372b592e6f3a0cbb6917b2a4650`.

The root cause of the issue lies in the following line of code (functions `_transferIn` and `_trasnferInFromWallet`):

```
require(transferred == _amount || _tokenAddress == swthToken, "Tokens transferred does not match the expected amount");
```

The second operand of the OR expression means the test isn't performed if the token being transferred is the Swicheo Token. That check is necessary because the user calling `lockWithWallet` or `lock` can choose any amount `callAmount` that will be passed as argument to the transfer.

The above line is the only validation that the amount actually transferred matches the amount needed. That means the transaction can be completed with arbitrarily small amounts (e.g. 0) of SWTH being transferred, and producing an event that seems to lock any arbitrarily large amount of SWTH.

The corresponding contract in another chain can be used to mint as many tokens as the attacker wants -- as the Switcheo Token contracts grant the LockProxy the privilege of minting tokens on calls to transfer/transferFrom.

It seems that the corresponding Zilliqa contracts aren't vulnerable to this attack as they don't include the `callAmount` parameter.

Another vector related to this issue is the possibility of front-running a user transaction, and running the same operation (with the same signature), but modifying the `callAmount` parameter to burn more tokens than necessary from the victim.

Suggested Remedy: in the case the token being transferred is SWTH, a test for `callAmount == amount` should be required.

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.

Replay Protection of LockProxy Transaction

The function `lockFromWallet` in the contract `LockProxy` does not include the chain id in the hash of the operation signed by the user.

While in most scenarios the token address (`_tokenAddress`), and target asset address (`_toAssetHash`) uniquely identify the source and destination chains, there might be scenarios when that isn't the case, in particular when considering chains that use different rules for generating contract addresses.

Suggestion: include the original chain id in the hash to be signed.

Conclusion

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

The vulnerabilities found were reported, and remedies suggested.