CD SECURITY

AUDIT REPORT

Token Locker
May 2023

Introduction

A time-boxed security review of the **TokenLocker** smart contract was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts.

About **TokenLocker**

The Token Locker smart contract is designed to provide users with a secure way to lock their ERC20 tokens in a holder contract. The smart contract charges a small fee for the service, which is paid in ETH. Users can lock their tokens for a specified period of time and unlock them after that. They can also extend the lock period of their tokens.

Threat Model

Roles & Actors

- Users - able to lock their tokens and extend the lock period.
- Owner - withdrawing contract's fees

Security Interview

Q: What in the protocol has value in the market?

A: The ERC20 tokens locked by the user and the fees in the form of ETH.

Q: What is the worst thing that can happen to the protocol?

A: If the smart contract is put into DoS state or if the holder contracts are drained by an attacker.

Q: In what case can the protocol/users lose money?

A: If the `tokenHolder` contract is not approved to transfer all of the locked tokens, the user will not receive the full amount of tokens.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

Scope

The following smart contracts were in scope of the audit:

- `tokenlocker.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issues
- Medium: 2 issues
- Low: 3 issues
- Informational: 6 issues

Findings Summary

ID	Title	Severity
[M-01]	Usage of non-standard ERC20 tokens might lead to stuck funds	Medium
[M-02]	Use `call` instead of `transfer` when sending ETH	Medium
[L-01]	Use `abi.encodeCall` instead of `abi.encodeWithSelector`	Low
[L-02]	CEI pattern is not followed	Low
[L-03]	Remove `receive` and `fallback` functions	Low
[I-01]	Prefer battle-tested code over reimplementing common patterns	Informational
[I-02]	Unclear error message	Informational
[I-03]	Missing event emission	Informational
[I-04]	NatSpecs are incomplete	Informational
[I-05]	Prefer Solidity Custom Errors over `require` statements with strings	Informational

ID	Title	Severity
[I-06]	Lock `pragma` to specific version	Informational

Detailed Findings

[M-01] Usage of non-standard ERC20 tokens might lead to stuck funds

Severity

Impact: High, because tokens will be left stuck in `tokenlocker.sol`

Likelihood: Low, because there aren't many such ERC20 tokens

Description

The `transferFrom` method of `ERC20` is used across the contract, but do not check if the returned `bool` values are `true`. This is problematic, because there are tokens on the blockchain which actually do not revert on failure but instead return `false` (example is `ZRX`). If such a token is used and a transfer fails, the tokens will be stuck in the smart contract forever.

Recommendations

Use the `SafeERC20` library from `OpenZeppelin` and change the `transferFrom` call to a `safeTransferFrom` call instead.

Discussion

CLIENT

Acknowledged - corrected:

- `safeTransfer` and `safeTransferFrom` instead of `transfer` and `transferFrom`.

[M-02] Use `call` instead of `transfer` when sending ETH

Severity

Impact: Medium, because if the owner of `tokenlocker.sol` is a smart contract or a specific multisig, the transaction may fail

Likelihood: Medium, because there is a big chance that the deployer of the `TokenLocker` contract is a smart contract or a specific multisig wallet

Description

The `getServiceFee` function uses the `transfer` method of `address payable` to withdraw the contract fees to the owner. The owner address is possible to be a smart contract that has a `receive` or `fallback` function that takes up more than the 2300 gas which is the limit of `transfer` or a specific multi-sig wallet, so usage of `transfer` is discouraged.

Recommendations

Use a `call` with value instead of `transfer`.

Discussion

The client stated that the deployer of the contract will be EOA, but nevertheless he corrected it because it is a simple fix and you are sure that no problems will occur with that if you decide to deploy the smart contract with a specific multisig or smart contract in the future.

CLIENT

Acknowledged - corrected.

[L-01] Use `abi.encodeCall` instead of `abi.encodeWithSelector`

The problem with `abi.encodeWithSelector` is that the compiler does not check whether the supplied values actually match the types expected by the called function. `abi.encodeCall` which is similar to `abi.encodeWithSelector`, just that it does perform these type checks. Note that `abi.encodeCall` is available from version `pragma solidity 0.8.11`.

Discussion

CLIENT

Acknowledged - corrected:

- The current version of the contract is using `abi.encodeCall` as we recommended

[L-02] CEI pattern is not followed

The Checks-Effects-Interactions pattern is not followed inside `lockTokens` function. Not following this pattern is often the case for reentrancy attack, however we couldn't find a way to exploit this for now. Move the `transferFrom` method at the end of the function. You can read more about the CEI pattern here.

Discussion

CLIENT

Acknowledged - corrected.

[L-03] Remove `receive` and `fallback` functions

The smart contract is not expected to receive ETH apart from the fee which is collected in `lockTokens`. Therefore, `receive` and `fallback` should be removed. It will be better if a transaction fails when a user sends ETH to the contract by mistake instead of storing it in the contract.

Discussion

CLIENT

Acknowledged - corrected:

- Both functions are removed from the contract.

[I-01] Prefer battle-tested code over reimplementing common patterns

Instead reimplementing your own `IERC20` interface, use the one provided by OpenZeppelin, since it is well tested and optimized.

Discussion

CLIENT

Acknowledged.

[I-02] Unclear error message

The error message is unclear: `require(msg.value == 2e16, "Fee")`. Consider changing it to `"Insufficient fee amount"`.

Discussion

CLIENT

Acknowledged - corrected.

[I-03] Missing event emission

All functions which are external in the contract are state changing but do not emit an event which might not be good for off-chain monitoring. Consider declaring a proper events and emit them on state change.

Discussion

CLIENT

Acknowledged.

[I-04] NatSpecs are incomplete

@notice, @param and @return fields are missing throughout the contract. NatSpec documentation is essential for better understanding of the code by developers and auditors and is strongly recommended. Please refer to the NatSpec format and follow the guidelines outlined there.

Discussion

CLIENT

Acknowledged - corrected.

[I-05] Prefer Solidity Custom Errors over `require` statements with strings

Using Solidity Custom Errors has the benefits of less gas spent in reverted transactions, better interoperability of the protocol as clients of it can catch the errors easily on-chain, as well as you can give descriptive names of the errors without having a bigger bytecode or transaction gas spending, which will result in a better UX as well. Remove all `require` statements and use Custom Errors instead.

Discussion

CLIENT

Acknowledged - corrected.

[I-06] Lock `pragma` to specific version

Always use a stable pragma to be certain that you deterministically compile the Solidity code to the same bytecode every time. The project is currently using a floatable version. Furthermore, consider using a newer version of the compiler as the latest available version is `0.8.19`.

Discussion

CLIENT

Acknowledged - corrected.