Safe

Token Locking

by Ackee Blockchain

13.3.2024

Contents

1. Document Revisions

1.0	Final report	7.3.2024
1.1	Fix review	13.3.2024

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Wake is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzz testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzz tests.

2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don’t jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven’t found a way to exploit the issue given the time constraints, it might be marked as a ”Warning” or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review team

Member's Name	Position
Štěpán Šonský	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Revision 1.0

Safe engaged Ackee Blockchain to perform a security review of the Safe protocol with a total time donation of 2 engineering days in a period between March 4 and March 7, 2024, with Štěpán Šonský as the lead auditor.

The audit was performed on the commit `f467abf` and the scope was the following:

- SafeTokenLock.sol
- TokenRescuer.sol

We began our review using static analysis tools, including Wake. We then took a deep dive into the logic of the contracts. For testing and fuzzing, we have involved Wake testing framework (see Appendix C). During the review, we paid special attention to:

- ensuring the arithmetic and logic of the system are correct,
- ensuring nobody can steal the funds,
- detecting possible reentrancies in the code,
- checking access controls are not too relaxed or too strict,
- looking for common issues such as data validation.

Our review resulted in 4 findings, all with the Warning severity and represent rather robustness recommendations than potential vulnerabilities. The overall code quality is solid, contracts contain perfect NatSpec documentation and description comments of gas optimizations such as unchecked math or downcasting.

Ackee Blockchain recommends Safe:

- change contract state before external calls,
- disable renouncing ownership,
- use safe transfers,
- add zero-address check.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

The fix review was performed on commit `ac32f77` with a result of 2 issues fixed according to our recommendations, 1 issue acknowledged, and 1 issue invalidated.

See Revision 1.1 for the system overview of the codebase.

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Fix*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
W1: State change after external call	Warning	1.0	Fixed
W2: Renounce ownership	Warning	1.0	Fixed
W3: Unsafe transfers	Warning	1.0	Acknowledged
W4: Missing zero-address validation	Warning	1.0	Invalidated

Table 2. Table of Findings

5. Report revision 1.0

5.1. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

Contracts

Contracts we find important for better understanding are described in the following section.

SafeTokenLock.sol

The core contract of the project. It inherits from the `TokenRescuer` contract and implements the `ISafeTokenLock` interface. The address that deploys the contract becomes the initial owner. The contract `SafeTokenLock` itself contains 3 external, state-changing operations:

- Lock - Transfers the specified `amount` of Safe Token to the contract.
- Unlock - Unlocks the specified `amount` for withdrawal. Every unlocked amount is stored as a slot, to be withdrawn at once after `COOLDOWN_PERIOD`.
- Withdraw - Withdraws/transfers the unlocked Safe Tokens from the contract to the `msg.sender` address if the `COOLDOWN_PERIOD` is passed. Withdrawal can be limited using `maxUnlocks` parameter, which determines the maximum number of unlocks which passed the `COOLDOWN_PERIOD`.

Aside from these external state-changing functions, there are external view functions `getUserTokenBalance`, `getUser`, and `getUnlock`. Rescuing the Safe Token from the contract is disabled using `rescueToken` overriding. The contract also contains the OpenZeppelin `Ownable2Step` logic through the `TokenRescuer` inheritance.

TokenRescuer.sol

The `TokenRescuer` contract allows the owner to rescue any ERC-20 token from the contract to the `beneficiary` address. The contract inherits from the OpenZeppelin `Ownable2Step`.

Actors

This part describes actors of the system, their roles, and permissions.

Owner

The owner has the following extra privileges in the system. He can rescue ERC-20 tokens from the `SafeTokenLock` contract, except the Safe Token. He can transfer (using two-step) ownership to another address and he can renounce the ownership.

User

The user (any EOA or contract) can interact with the system using the `lock`, `unlock` and `withdraw` functions, described above in 5.1.1.1.

5.2. Trust Model

From the user's perspective, the system can be considered trustless. The owner does not have any control over the Safe Token holdings stored in the `SafeTokenLock` contract and does not have any other privilege that represents a trust issue. The project relies on a trusted Safe Token contract and it is not designed to be used with other tokens.

ackee | blockchain security

W1: State change after external call

Impact:	Warning	Likelihood:	N/A
Target:	SafeTokenLock.sol	Type:	Best practices

Description

The function `SafeTokenLock.lock` changes the value of `_users[msg.sender].locked` after the `IERC20.transferFrom` external call. Although the Safe Token contract is trusted, this approach generally can lead to reentrancies and should be avoided in any circumstances. In case the token contract would be another token, then the `transferFrom` function could have access to an inconsistent state using `SafeTokenLock.getUserTokenBalance` function, which reads the `_users[holder].locked` value.

```
IERC20(SAFE_TOKEN).transferFrom(msg.sender, address(this), amount);
_users[msg.sender].locked += amount;
```

Recommendation

Always change the contract state before the external calls when possible.

```
_users[msg.sender].locked += amount;
IERC20(SAFE_TOKEN).transferFrom(msg.sender, address(this), amount);
```

Fix 1.1

Fixed. The operations order was changed according to CEI pattern.

[Go back to Findings Summary]

W2: Renounce ownership

Impact:	Warning	Likelihood:	N/A
Target:	SafeTokenLock.sol	Type:	Best practices

Description

The contract `SafeTokenLock` inherits the logic from OpenZeppelin `Ownable`, including the `renounceOwnership` function which permanently and irreversibly removes the contract owner, therefore nobody would be able to rescue tokens from the contract anymore.

Recommendation

Decide, if this feature is intended and planned to be used. If not, disable this unwanted function by overriding it with an empty body or revert.

```
function renounceOwnership() public override onlyOwner {

}
```

Fix 1.1

Fixed. The `renounceOwnership` function is overridden and reverting.

[Go back to Findings Summary]

W3: Unsafe transfers

Impact:	Warning	Likelihood:	N/A
Target:	SafeTokenLock.sol	Type:	Best practices

Description

The contract `SafeTokenLock` uses unsafe `transferFrom` and `transfer` functions, in the `lock` and `withdraw` functions, which can introduce various issues with non-standard ERC-20 tokens in combination with unchecked return values.

```
IERC20(SAFE_TOKEN).transferFrom(msg.sender, address(this), amount);
```

```
IERC20(SAFE_TOKEN).transfer(msg.sender, uint256(amount));
```

Recommendation

Although the protocol uses the trusted Safe Token implementation, we recommend using the `SafeERC20` library as a good practice. The library is already used in the `TokenRescuer` contract.

Fix 1.1

Acknowledged.

> As the token used in lock and withdraw is the trusted SafeToken, we deem any change to that unnecessary.
>
> — Safe

[Go back to Findings Summary]

W4: Missing zero-address validation

Impact:	Warning	Likelihood:	N/A
Target:	SafeTokenLock.sol	Type:	Data validation

Description

The contract `SafeTokenLock` lacks the zero-address validation of `safeToken` parameter in the constructor.

Recommendation

Add the `safeToken` zero-address validation into the constructor for better robustness.

```
if (safeToken == address(0)) revert InvalidSafeToken();
```

Fix 1.1

Invalidated.

> As the token address is checked for its totalSupply, adding another check to ensure it is not Zero Address is not required.
>
> — Safe

[Go back to Findings Summary]

6. Report revision 1.1

6.1. System Overview

Updates and changes we find important for fix review.

Contracts

Updates in contracts that modify behavior against the previous revision.

SafeTokenLock.sol

The `SafeTokenLock` newly overrides the `renounceOwnership` function and reverts with the error `RenounceOwnershipDisabled`.

Actors

Updates regarding actors of the system, their roles, and permissions.

Owner

The owner lost the ability to renounce ownership of the `SafeTokenLock` contract.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Safe: Token Locking, 13.3.2024.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

Appendix C: Wake outputs

C.1. Tests

The following fuzz test flow was developed to test the contract logic and balance invariants. After 100k runs, no inconsistency was found and the system behaves like it's described.

```
@flow()
def flow_locking(self) -> None:
    default_chain.set_next_block_timestamp(default_chain.blocks[
"latest"].timestamp + (random_int(1,24) * DAY))
    default_chain.mine()

    if(random_bool()):
        user = self.random_user()
        amount = random_int(0, 10) * ONE
        self.token.approve(self.locking.address, amount, from_= user)

        if(amount > self.token.balanceOf(user)):
            with must_revert('ERC20: transfer amount exceeds balance'):
                self.locking.lock(amount, from_ = user)
        else:
            if(amount > 0):
                self.locking.lock(amount, from_ = user)
            else:
                with must_revert(SafeTokenLock.InvalidTokenAmount):
                    self.locking.lock(amount, from_ = user)

    if(random_bool()):
        amount = random_int(0, 10) * ONE
        user = self.random_user()

        if(amount > 0):
            if(self.locking.getUser(user).locked >= amount):
                self.locking.unlock(amount, from_ = user)
            else:
                with must_revert(SafeTokenLock.UnlockAmountExceeded):
                    self.locking.unlock(amount, from_ = user)
        else:
            with must_revert(SafeTokenLock.InvalidTokenAmount):
```

```
            self.locking.lock(amount, from_ = user)

    if(random_bool()):
        user = self.random_user()
        self.locking.withdraw(maxUnlocks= random_int(0, 10), from_ = user)
```

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://twitter.com/AckeeBlockchain