EthSign

EthSign

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen r@osec.io

Woosun Song procfs@osec.io

Contents

01 | Executive Summary

Overview

EthSign engaged OtterSec to perform an assessment of the `Tokentable-unlock` program. This assessment was conducted between September 7th and September 12th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 4 findings in total.

In particular, we identified an issue regarding using tx.origin in a constructor to assign ownership of the contract, which is unsafe and may result in adverse effects (OS-ESN-ADV-00). Additionally, we highlighted the possibility of allowing the takeover of contracts by invoking the initialize function (OS-ESN-ADV-01) and another issue regarding setting the custom fee to zero, potentially resulting in confusion (OS-ESN-ADV-02).

We also recommended certain changes following best coding practices and improved code readability (OS-ESN-SUG-00).

02 | Scope

The source code was delivered to us in a git repository at github.com/EthSign/tokentable-unlock-contract/. This audit was performed against commit a988710.

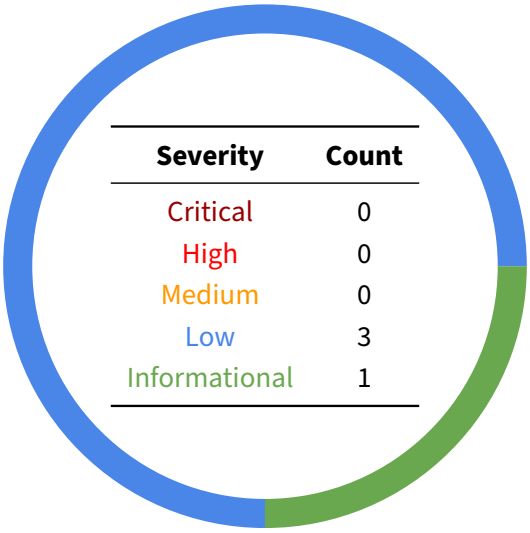
A brief description of the programs is as follows:

Name	Description
tokentable-unlock	TokenTable is a platform designed for the management of token ownership, facilitating users in the execution, monitoring, and enforcing agreements utilized by a project to distribute their tokens. It empowers users to create, endorse, and carry out a range of token investment contracts and token grant agreements.

03 | Findings

Overall, we reported 4 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	3
Informational	1

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-ESN-ADV-00	Low	Resolved	In `TTUV2BeaconManager`'s constructor, `tx.origin` is assigned as owner, which may have detrimental effects on security and robustness.
OS-ESN-ADV-01	Low	Resolved	`initialize` may be called for implementation contracts, resulting in a takeover.
OS-ESN-ADV-02	Low	Resolved	It is infeasible to set custom fees to zero, and attempting to do so will result in counter-intuitive consequences.

OS-ESN-ADV-00 [low] | Usage of Of Tx.Origin

Description

In the constructor of `TTUV2BeaconManager`, the contract owner is set to `tx.origin`, which should be avoided.

TTUV2BeaconManager.sol SOLIDITY

```
contract TTUV2BeaconManager is Ownable {
    [...]
    constructor(
        address unlockerImpl,
        address futureTokenImpl,
        address trackerTokenImpl
    ) {
        [...]
        // solhint-disable-next-line avoid-tx-origin
        transferOwnership(tx.origin);
    }
```

The usage of `tx.origin` leaves the contract vulnerable to phishing-like attacks. `tx.origin` is the address of the transaction initiator and not necessarily the internal message sender. If a malicious actor is capable of encapsulating the constructor call within its transaction, the owner will be set to the actor instead of the constructor invoker.

The usage of `tx.origin` also hinders robustness, as it is incompatible with existing standards, such as account abstraction (ERC4337), where the transaction initiator is not an externally owned account.

Remediation

Utilize `msg.sender` in replace of `tx.origin`.

Patch

Fixed in 2b32e53.

OS-ESN-ADV-01 [low] | Initializable Implementation Contracts

Description

Under the current implementation of the following contracts:

- `TokenTableUnlockerV2`
- `TTFutureTokenV2`
- `TTTrackerTokenV2`

It is possible to call `initialize` on the implementation contract, as the `isInitialized` flag is set to 0 by default. Calling `initialize` on the implementation contract may allow the caller to set the owner to `msg.sender`, effectively resulting in a takeover.

At current time, the takeover has no security implications, but it may be exploited if `UUPSUpgradable` is added. In this case, an attacker may `selfdestruct` the implementation contract and brick the proxy.

Remediation

Prevent `initialize()` from being invoked on the contracts by adding the following constructor to Hub.

SOLIDITY

```
constructor() {
    _disableInitializers();
}
```

Patch

Fixed in ef24820.

OS-ESN-ADV-02 [low] | Ambiguity In Fee Setter

Description

Setting a custom fee to zero in `TTUFeeCollector` is not viable, as the value zero is used as an indicator of an uninitialized user rather than an indication of a zero fee. Additionally, the current setter and getter design is ambiguous, as `setCustomFee(x,0)` will succeed but its effects are counter-intuitive, as fees default to `defaultFeesBips` in `getFee` instead of becoming zero.

contracts/core/TTUFeeCollector.sol SOLIDITY

```
function setCustomFee(
    address unlockerAddress,
    uint256 bips
) external onlyOwner {
    _customFeesBips[unlockerAddress] = bips;
    emit CustomFeeSet(unlockerAddress, bips);
}

function getFee(
    address unlockerAddress,
    uint256 tokenTransferred
) external view override returns (uint256 tokensCollected) {
    uint256 feeBips = _customFeesBips[unlockerAddress] == 0
        ? defaultFeesBips
        : _customFeesBips[unlockerAddress];
    tokensCollected = (tokenTransferred * feeBips) / BIPS_PRECISION;
}
```

This ambiguity may lead users to inadvertently incur unexpected fees.

Remediation

Utilize a sentinel value to indicate zero fee. Also, document this behavior to prevent any potential confusion.

Patch

Fixed in 34efdcd.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-ESN-SUG-00	Suggestions regarding best practices and improved code readability.

OS-ESN-SUG-00 | Code Maturity

Description

For clarity and conciseness, it is worth considering a refinement in selector expressions, specifically those resembling the format of `TTTrackerTokenV2(address(0)).initialize.selector`. A more concise and equally effective alternative exists, which involves simplifying this expression to `TTTrackerTokenV2.initialize.selector`.

Remediation

Simplify selector expressions as specified above.

Patch

Fixed in bf313c4.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings section.

Critical	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
High	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
Medium	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
Low	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.