STRONGHOLD
SECURITY

Truflation Contracts Security Audit Report

May 16, 2024

Contents

Executive Summary

Title	Description
Client	Truflation
Project	truflation-contracts
Platform	Ethereum
Language	Solidity
Repository	https://github.com/truflation/truflation-contracts
Initial commit	010448ea069c0a1b762104a75aad345219c3f96e
Final commit	284afde20868012ca615ba812d28390acd50dbd9
Timeline	April 25 2024 - May 15 2024

Project Overview

Truflation is a decentralized service that tracks inflation. Truftoken is used for vesting, staking and voting.

Audit Scope

File	Link
VotingEscrowTruf.sol	VotingEscrowTruf.sol
TrufMigrator.sol	TrufMigrator.sol
ERC677Token.sol	ERC677Token.sol
TruflationToken.sol	TruflationToken.sol
TrufVesting.sol	TrufVesting.sol
VirtualStakingRewards.sol	VirtualStakingRewards.sol
StakingRewards.sol	StakingRewards.sol
TrufPartner.sol	TrufPartner.sol

Audit Methodology

General Code Assessment

The code is reviewed for clarity, consistency, style, and whether it follows code best practices applicable to the particular programming language used, such as indentation, naming convention, commented code blocks, code duplication, confusing names, irrelevant or missing comments, etc. This part is aimed at understanding the overall code structure and protocol architecture. Also, it seeks to learn overall system architecture and business logic and how different parts of the code are related to each other.

Code Logic Analysis

The code logic of particular functions is analyzed for correctness and efficiency. The code is checked for what it is intended for, the algorithms are optimal and valid, and the correct data types are used. The external libraries are checked for relevance and correspond to the tasks they solve in the code. This part is needed to understand the data structures used and the purposes for which they are used. At this stage, various public checklists are applied in order to ensure that logical flaws are detected.

Entities and Dependencies Usage Analysis

The usages of various entities defined in the code are analyzed. This includes both: internal usage from other parts of the code as well as possible dependencies and integration usage. This part aims to understand and spot overall system architecture flaws and bugs in integrations with other protocols.

Access Control Analysis

Access control measures are analyzed for those entities that can be accessed from outside. This part focuses on understanding user roles and permissions, as well as which assets should be protected and how.

Use of checklists and auditor tools

Auditors can perform a more thorough check by using multiple public checklists to look at the code from different angles. Static analysis tools (Slither) help identify simple errors and highlight potentially hazardous areas. While using Echidna for fuzz testing will speed up the testing of many invariants, if necessary.

Vulnerabilities

The audit is directed at identifying possible vulnerabilities in the project's code. The result of the audit is a report with a list of detected vulnerabilities ranked by severity level:

Severity	Description
● Critical	Vulnerabilities leading to the theft of assets, blocking access to funds, or any other loss of funds.
● High	Vulnerabilities that cause the contract to fail and that can only be fixed b y modifying or completely replacing the contract code.
● Medium	Vulnerabilities breaking the intended contract logic but without loss of fun ds and need for contract replacement.
● Low	Minor bugs that can be taken into account in order to improve the overall qu ality of the code

After the stage of bug fixing by the Customer, the findings can be assigned t he following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect it s security.
Acknowledged	The Customer took into account the finding. However, the recommendations wer e not implemented since they did not affect the project's safety.

Findings Summary

Severity	# of Findings
Critical	0
High	1
Medium	8
Low	7

ID	Severity	Title	Status
H-1	High	Slippage protection	Fixed
M-1	Medium	User funds may be blocked with the `migrate` function.	Fixed
M-2	Medium	No limits for variables	Fixed
M-3	Medium	Reentrancy risk	Fixed
M-4	Medium	Unexpected behavior of the `TrufPartner` contract with some tokens	Fixed
M-5	Medium	Possible stake zero points	Fixed
M-6	Medium	`RewardsDistribution` can stake less than needed	Fixed
M-7	Medium	A two-step ownership transfer	Fixed
M-8	Medium	The `owner` can't cancel a subscription	Fixed
L-1	Low	Additional checks	Fixed
L-2	Low	Variables can be declared as `immutable`	Fixed
L-3	Low	Unused imports	Fixed
L-4	Low	Pragma solidity version	Fixed
L-5	Low	The NatSpec is missing	Fixed
L-6	Low	The `SafeApprove` function is deprecated	Fixed

L-7	● Low	Unsafe cast	Fixed

Findings

Critical

Not Found

High

H-1 ● High Slippage protection Fixed

Description

TrufPartner.sol#L138-L147

The `amountBMin` variable is 0 for the `uniV2Router.addLiquidity` function. This variable controls slippage.

As the `uniswapV2` router documentation says:

`amountAMin` - Bounds the extent to which the B/A price can go up before the transaction reverts.

`amountBMin` - Bounds the extent to which the A/B price can go up before the transaction reverts.

Recommendation

We recommend adding a param for slippage control.

Client's commentary

Fixed in PR-56

Medium

M-1 ● Medium User funds may be blocked with the `migrate` function. Fixed

Description

TrufMigrator.sol#L48

TrufMigrator.sol#L58

TrufMigrator.sol#L63

Multiple leaves can use the same user addresses (or the owner sets a new `merkleRoot`). The contract may block user funds, or the user may receive less funds than expected.

Consider this leaf's entity structure:

```
[
  [
    "0x0000000000000000000000000000000000000001",
    "0",
    "10000000000000000000"
  ],
  [
    "0x0000000000000000000000000000000000000001",
    "1",
    "20000000000000000000"
  ]
]
```

In the leaf structure above, the user with the address is eligible to claim:

1. 10 TRUF tokens (leaf with index 0)
2. 20 TRUF tokens (leaf with index 1)

If the user claims 10 tokens first and then 20 tokens, he'll receive only 20 tokens instead of 30 because the amount of already claimed tokens is subtracted.

On the contrary, if the user claims 20 tokens first and then attempts to claim 10 more tokens, only the initial 20 tokens will be received, as the second 10 token claim will be reverted.

Recommendation

We recommend using the leaf index (along with `msg.sender`) in the `migratedAmount` mapping.

Client's commentary

Fixed in PR-42

M-2 ● Medium No limits for variables Fixed

Description

There are no limits for the following variables:

`_rewardsDuration`:

StakingRewards.sol#L143

`_rewardsDuration`:

VirtualStakingRewards.sol#L160

`info`:

TrufVesting.sol#L494

Recommendation

We recommend adding limits.

Client's commentary

Fixed in PR-43

M-3 ● Medium Reentrancy risk Fixed

Description

TrufVesting.sol#L405-L408

It is not safe to change the state after an external call.

Recommendation

We recommended using the `Checks Effects Interactions` pattern.

Client's commentary

Fixed in PR-57

M-4 ● Medium **Unexpected behavior of the `TrufPartner` contract with some tokens** Fixed

Description

TrufPartner.sol#L138-L147

The current implementation of the `TrufPartner` contract is incompatible with the `USDT` token.

Let's delve deeper into the `buy` function:

- At the first approval to `UniV2Router` from `TrufPartner`, we execute: `pairToken.safeApprove(address(uniV2Router), pairTokenMaxIn)`;
- Next, a call to `UniV2Router` is made: `uniV2Router.addLiquidity`;
- But `uniV2Router` may use only part of the approved assets;
- So, for the USDT token, the current implementation won't work because:
- Call `buy(Alice)`, where approval for 100 USDT was given;
- `uniV2Router` used only 90 USDT;
- Now we have 10 USDT approvals remaining;
- When `buy(Bob)` is called, the transaction will fail due to the USDT specificity;

Recommendation

We recommend adding:

```
    pairToken.safeApprove(address(uniV2Router), pairTokenMaxIn);
    trufToken.safeApprove(address(uniV2Router), subscription.trufAmount);
    (, uint256 pairTokenIn, uint256 lpAmount) = uniV2Router.addLiquidity(
        address(trufToken),
        address(pairToken),
        subscription.trufAmount,
        pairTokenMaxIn,
        subscription.trufAmount,
        0,
        address(this),
        deadline
    );
++      pairToken.safeApprove(address(uniV2Router), 0);
```

Client's commentary

Fixed in PR-44

M-5 ● Medium Possible stake zero points Fixed

Description

VotingEscrowTruf.sol#L158-L159

During stake, point amounts are calculated as: `amount * duration / MAX_DURATION`. This might lead to a loss of precision, and it's possible to stake and mint zero tokens.

Recommendation

We recommend adding zero check for points.

Client's commentary

Fixed in PR-45

M-6 ● Medium `RewardsDistribution` can stake less than needed Fixed

Description

StakingRewards.sol#L115-L134

It's possible for the owner of the staking contract to deposit fewer reward tokens than needed because they rely solely on the actual balance: `uint256 balance = IERC20(rewardsToken).balanceOf(address(this));` without any assumptions about user's withdrawals.

For example:

1. One user stakes a few base tokens;
2. The user doesn't withdraw rewards for the entire period;
3. Next, the `RewardsDistribution` calls `notifyRewardAmount` again, but this function can be called without any additional transfer.

Recommendation

We recommend tracking users' withdrawals and how many tokens must be distributed.

Client's commentary

Fixed in PR-46

M-7 ● Medium A two-step ownership transfer Fixed

Description

TrufVesting.sol#L4

VirtualStakingRewards.sol#L4

StakingRewards.sol#L5

TrufPartner.sol#L4

The contract owner can call the `transferOwnership` function with an inactive address, leading to loss of access to the contract. `Ownable` also has a one-step transfer of ownership.

Recommendation

We recommend using the `Ownable2Step` contract.

Client's commentary

Fixed in PR-47

M-8 ● Medium The `owner` can't cancel a subscription Fixed

Description

TrufPartner.sol#L212

Let's consider the following case:

1. The owner initiates a `Subscription` with a big `startTime` value (for example, 1000 years).
2. The owner calls the `cancel` function. Since the `Subscription.status == Initiated` and `Subscription.startTime >= block.timestamp`, the function reverts.
3. As a result, the `trufToken` sent by the `owner` in the `initiate` function will get stuck on the contract.

Recommendation

We recommend adding limits for the `startTime` value.

Client's commentary

Fixed in PR-61.

Low

L-1 ● Low **Additional checks** Fixed

Description

No state checks:

`merkleRoot`:

TrufMigrator.sol#L51

(`merkleRoot` != bytes32(0x00));

No limits checks:

`_minStakeDuration`:

VotingEscrowTruf.sol#L74

(`_minStakeDuration` < `MAX_DURATION`);

Non-zero transfer value check:

`pairTokenMaxIn`

TrufPartner.sol#L135

No null address checks:

`user`:

TrufVesting.sol#L515

`newUser`:

TrufVesting.sol#L337

`_rewardsDistribution`:

StakingRewards.sol#L153

Recommendation

We recommend adding the checks.

Client's commentary

Fixed in PR-48

L-2 ● Low Variables can be declared as `immutable` Fixed

Description

`rewardsToken`:

[StakingRewards.sol#L21]

`stakingToken`:

[StakingRewards.sol#L22]

Recommendation

We recommend declaring variables as `immutable`.

Client's commentary

Fixed in PR-50

L-3 ● Low Unused imports Fixed

Description

`RewardsSource`:

VotingEscrowTruf.sol#L8

Recommendation

We recommend removing this import.

Client's commentary

Fixed in PR-49

L-4 ● Low Pragma solidity version Fixed

Description

The Solc version specified in contracts is 0.8.19, which is outdated.

Recommendation

We recommend setting the latest stable version of the Solidity compiler.

Client's commentary

Fixed in PR-54

L-5 ● Low The NatSpec is missing Fixed

Description

TrufPartner.sol

StakingRewards.sol

VirtualStakingRewards.sol

The NatSpec is missing for these contracts.

Recommendation

We recommend adding the NatSpec for these contracts.

Client's commentary

Fixed in PR-55

L-6 ● Low The `SafeApprove` function is deprecated Fixed

Description

TrufPartner.sol#L136

TrufPartner.sol#L137

TrufPartner.sol#L157

TrufPartner.sol#L189

TrufPartner.sol#L238

The `SafeApprove` is deprecated in favour of the `safeIncreaseAllowance` and `safeDecreaseAllowance` functions.

Recommendation

We recommended using `safeIncreaseAllowance` and `safeDecreaseAllowance`.

Client's commentary

Fixed in PR-53

L-7 ● Low **Unsafe cast** Fixed

Description

Unsafe cast:

TrufVesting.sol#L444

TrufVesting.sol#L449

Recommendation

We recommend using a safe cast from `@openzeppelin`.

Client's commentary

Fixed in PR-52

Conclusion

Altogether, the audit process has revealed 1 HIGH, 8 MEDIUM, and 7 LOW severity findings.

Disclaimer

The Stronghold audit makes no statements or warranties about the utility of the code, the safety of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug-free status. The audit documentation is for discussion purposes only.