Zellic

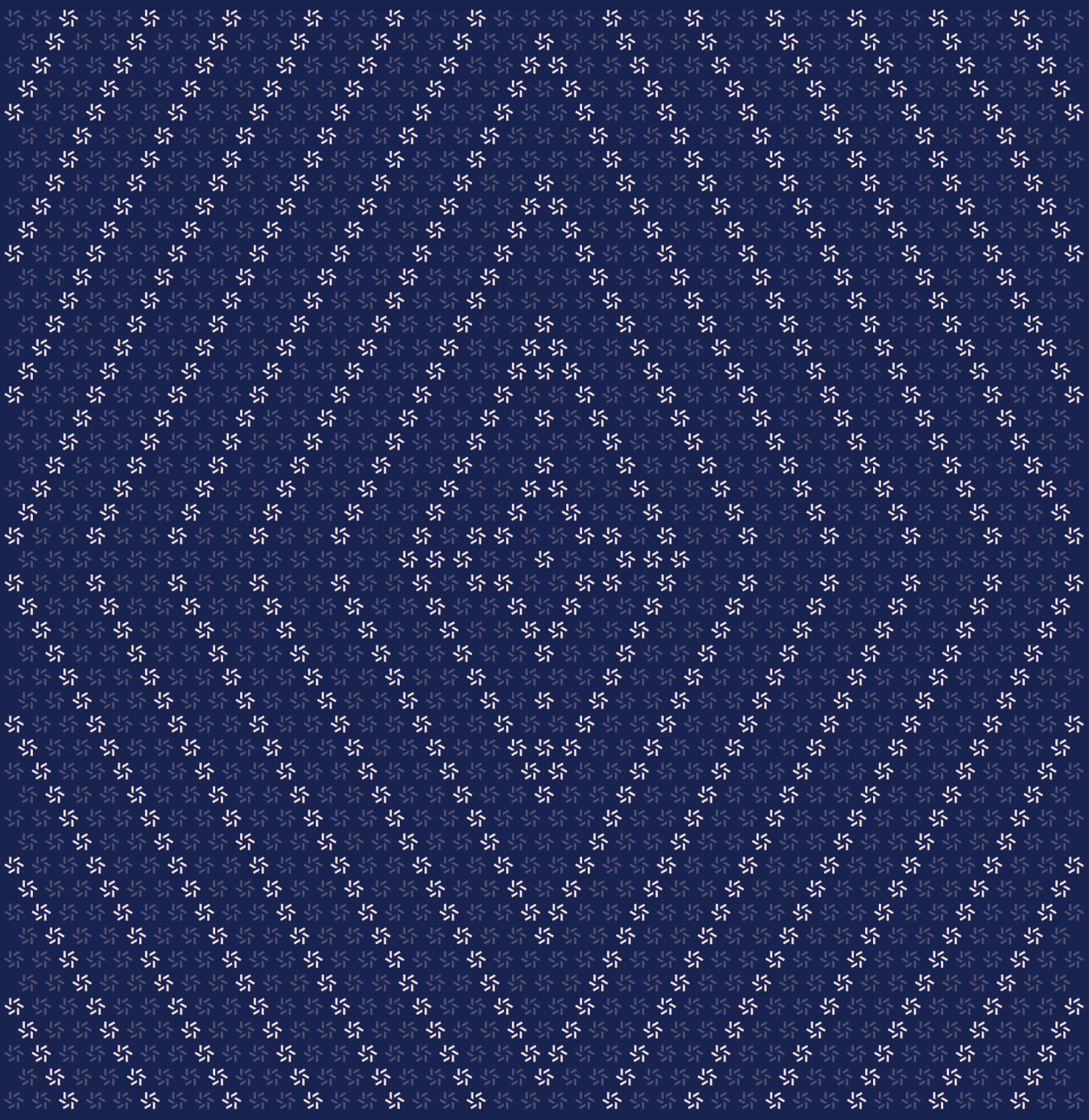
Prepared for
Geng Sng
Trillion Network

Prepared by
Seunghyeon Kim
Jaeeu Kim
Zellic

May 1, 2024

Trillion

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Trillion Network on April 25th, 2024. During this engagement, Zellic reviewed Trillion's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is there a possibility of unauthorized transfers?
- Is there a possibility of unauthorized mints?
- Are the wrapper contracts implemented correctly?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Trillion contracts, we discovered one finding, which was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Trillion Network's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	1

2. Introduction

2.1. About Trillion

Trillion Network contributed the following description of Trillion:

> Trillion is a new stablecoin provider issuing fiat-backed, fully reserved centralized stablecoins.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and

Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Trillion Contracts

Repository	https://github.com/trillion-network/trillion-contracts ↗
Version	trillion-contracts: `aa020241ff64e9808613fae60cf797615c1e597e`
Programs	• BlacklistableV1.sol • FiatTokenV1.sol • RescuableV1.sol
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of one half person-day. The assessment was conducted over the course of one calendar day.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Seunghyeon Kim
Engineer
seunghyeon@zellic.io

Jaeeu Kim
Engineer
jaeeu@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 25, 2024	Kick-off call
April 25, 2024	Start of primary review period
April 25, 2024	End of primary review period

3. Detailed Findings

3.1. Potential front-run on `initialize` of FiatTokenV1

Target	FiatTokenV1		
Category	Protocol Risks	**Severity**	Low
Likelihood	Low	**Impact**	Informational

Description

The FiatTokenV1 contract is designed as an upgradable contract and does not include a constructor. Instead, it features an `initialize` function responsible for configuring token settings and assigning roles. However, the current implementation of the `initialize` function does not include verification checks to ensure that `msg.sender` is a trusted address.

If the contract is deployed but not initialized in the same transaction, this could potentially allow an attacker to front-run the `initialize` function and take control of the contract. We note that in the current deployment script, `initialize` is indeed called during deployment.

```
function initialize(
    address defaultAdmin,
    address pauser,
    address minter,
    address upgrader,
    address rescuer,
    address blacklister,
    string memory tokenName,
    string memory tokenSymbol
) public initializer {
    __ERC20_init(tokenName, tokenSymbol);
    __ERC20Capped_init(MAX_TOKEN_SUPPLY);
    __ERC20Pausable_init();
    __ERC20Burnable_init();
    __AccessControl_init();
    __ERC20Permit_init(tokenName);
    __UUPSUpgradeable_init();
    __ERC20Rescuable_init();
    __ERC20Blacklistable_init();

    _grantRole(DEFAULT_ADMIN_ROLE, defaultAdmin);
    _grantRole(PAUSER_ROLE, pauser);
    _grantRole(MINTER_ROLE, minter);
    _grantRole(UPGRADER_ROLE, upgrader);
    _grantRole(RESCUER_ROLE, rescuer);
```

```
    _grantRole(BLACKLISTER_ROLE, blacklister);
}
```

Impact

The attacker can set token-related settings and `grantRoles` on an arbitrary address.

Recommendations

Consider adding the trusted address-check logic for the `initialize` function.

Remediation

This issue has been acknowledged by Trillion Network, and a fix was implemented in commit 69076f00 ↗.

The team resolved this issue by adding the trusted address for this function.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Centralized risk

We observe that the project is intentionally centralized by design. Trillion retains the authority to perform functions such as minting, pausing, blacklisting, and upgrading, and users should be mindful of this fact. You can refer to the list of roles in Trillion's Token Design documentation here ↗ to understand their respective capabilities.

Trillion maintains control over the addresses associated with all roles, as they clarify:

> Trillion's wallet governance policy follows the principle of least privilege and the roles Minter, Pauser, Blacklister, and Rescuer on the FiatTokenV1 smart contract are each associated with a unique individual FireBlocks wallet, which only Trillion has control over. Each of these wallets implement MPC signing, which ensure that there is no single point of compromise for the private key used to sign these functions.
>
> Trillion has strict wallet governance policies internally that allows us to ensure that even in the event of a compromise of any of the FireBlocks wallets, Trillion is able revoke access to any compromised wallet

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: FiatTokenV1.sol

Function: `blacklist(address account)`

This function blacklists a given address.

Inputs

- `account`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address to blacklist.

Branches and code coverage

Intended branches

- Add a given address to the blacklist.
  - ☑ Test coverage

Negative behavior

- Revert if caller does not have `BLACKLISTER_ROLE`.

Function: `burnFrom(address account, uint256 value)`

This function burns the tokens of a given address.

Inputs

- `account`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address to burn from.
- `value`

- **Control**: Completely controlled by the caller.
- **Constraints**: None.
- **Impact**: Amount to burn.

Branches and code coverage

Intended branches

- Burn the tokens from a given address.
  - ☑ Test coverage

Negative behavior

- Revert if caller does not have `MINTER_ROLE`.
  - ☑ Negative test

Function: `burn(uint256 value)`

This function burns the tokens.

Inputs

- `value`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Amount to burn.

Branches and code coverage

Intended branches

- Burn the token of a given amount.
  - ☑ Test coverage

Negative behavior

- Revert if caller does not have `MINTER_ROLE`.
  - ☑ Negative test

Function: `initialize(address defaultAdmin, address pauser, address minter, address upgrader, address rescuer, address blacklister, string tokenName, string tokenSymbol)`

This function sets ERC-20's default settings and specific addresses for the `defaultAdmin`, `pauser`, `minter`, `upgrader`, `rescuer`, and `blacklister`.

Inputs

- `defaultAdmin`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address of admin to set.
- `pauser`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address of pauser to set.
- `minter`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address of minter to set.
- `upgrader`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address of upgrader to set.
- `rescuer`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address of rescuer to set.
- `blacklister`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address of blacklister to set.
- `tokenName`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Name of token to set.
- `tokenSymbol`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Symbol of token to set.

Branches and code coverage

Intended branches

- Set the ERC-20's settings.
  - ☑ Test coverage
- Grant the right role on given addresses.
  - ☑ Test coverage

Negative behavior

- Called multiple times.
  - ☑ Negative test

Function: `mint(address to, uint256 amount)`

This function mints the specific amount of tokens to a specific address.

Inputs

- `to`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address to mint to.
- `amount`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Amount to mint.

Branches and code coverage

Intended branches

- Mint the given amount of tokens to a specific address.
  - ☑ Test coverage

Negative behavior

- Revert if caller does not have `MINTER_ROLE`.
  - ☑ Negative test

Function: `pause()`

This function pauses the contract.

Branches and code coverage

Intended branches

- Pause the contract.
  - ☑ Test coverage

Negative behavior

- Revert if the caller does not have `PAUSER_ROLE`.
  - ☑ Negative test

Function: `rescue(IERC20 token, address to, uint256 amount)`

This function transfers a given token to a given address with a given amount. This function is meant be called by the `RESCUER_ROLE` when rescuing tokens that were mistakenly sent to the contract.

Inputs

- `token`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: ERC-20 token to rescue.
- `to`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address to rescue.
- `amount`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Amount to rescue.

Branches and code coverage

Intended branches

- Transfer a given token to a given address with a given amount.
  - ☑ Test coverage

Negative behavior

- Revert if caller does not have `RESCUER_ROLE`.
  - ☑ Negative test

Function: `unBlacklist(address account)`

This function removes a given address from the blacklist.

Inputs

- `account`
  - **Control**: Completely controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address to remove from the blacklist.

Branches and code coverage

Intended branches

- Remove a given address from the blacklist.
  - ☑ Test coverage

Negative behavior

- Revert if caller does not have `BLACKLISTER_ROLE`.
  - ☑ Negative test

Function: `unpause()`

This function unpauses the contract.

Branches and code coverage

Intended branches

- Unpause the contract.
  - ☑ Test coverage

Negative behavior

- Revert if the caller does not have `PAUSER_ROLE`.
  - ☑ Negative test

6. Assessment Results

At the time of our assessment, the reviewed code was deployed to the Ethereum Mainnet.

During our assessment on the scoped Trillion contracts, we discovered one finding, which was informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.