Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Audit-Report Fractal Sale, Citizend & Rising Tide SCs 05.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi

Index

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

> *"Building decentralized identity infrastructure for the leading blockchain ecosystems - As blockchain ecosystems continue to evolve, there is a growing realization that identity has a far more vital role to play. Decentralized identity is being integrated as core blockchain infrastructure component - moving up from the dApp layer to its fundamental infrastructure layer just like explorers, wallets, and oracles."*
>
> From https://web.fractal.id/solutions-for-blockchain/

This report documents the scope, test methodology, findings, and conclusions of a crypto review and source code audit against the Fractal Sale, Citizend, and Rising Tide smart contracts.

The assessment was requested by Trust Fractal GmbH in April 2024 and subsequently performed by Cure53 in May 2024, specifically during CW19. Five work days were invested and allocated to an experienced two-person technical team to reach the coverage expected for this project.

Only a single Work Package (WP) was required for this round of testing, entitled *WP1: Crypto reviews & code audits against Sale, Citizend & Rising Tide SCs*. Notably, Cure53 has reviewed the Fractal smart contracts on one previous occasion back in July 2018 (see report *FRC-01*), though the herein prioritized smart contracts had not yet been included in the scope of said project.

The customer provided sources and other miscellaneous facets to aid Cure53's white-box procedures. All necessary preparations were completed in late April and early May, namely CW18 2024, to maximize the productivity of the active evaluation phase.

The Fractal and Cure53 teams conversed using an open forum (Slack) throughout the engagement. All participating personnel from both parties were invited to join the private and shared channel, which was highly conducive to a successful collaboration. The scope was ideally prepared, meaning that cross-team queries and interactions were minimal. No notable blockers or delays were experienced at any point. Live reporting was offered as part of the initial test package and subsequently implemented via the aforementioned Slack channel.

The Cure53 team achieved satisfactory breadth and depth of coverage over the scope's focus aspects, encountering a total of two findings affecting the security premise. One of those constitutes a *High* risk security vulnerability and the other is an *Informational* hardening recommendation.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To summarize the overarching verdict, Cure53 is pleased to confirm that the inspected Fractal smart contracts are functionally secure and adhere to best practices, while the Fractal Sale management logic was soundly integrated. However, the implementation presents a complex architecture; simplifying this construct could reduce the risk of errors during future modifications and facilitate easier codebase comprehension for new developers. The audit identified potential security issues, though the proposed mitigation strategies will comprehensively address them if followed.

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. This will be followed by a chapter outlining the test methodology, which serves to provide greater clarity on the techniques applied and coverage achieved throughout this audit. Subsequently, the report will list all findings identified in chronological order, starting with the detected vulnerabilities and followed by the general weaknesses unearthed. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summary, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the Fractal Sale, Citizend, and Rising Tide smart contracts, giving high-level hardening advice where applicable.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Crypto Reviews & Source Code Audits against Sale, Citizend & Rising Tide Smart Contracts**
  - **WP1:** Crypto reviews & code audits against Sale, Citizend & Rising Tide SCs
    - **Source code:**
      - https://github.com/subvisual/citizend
    - **Smart contracts in scope:**
      - *contracts/token/Sale.sol*
      - *contracts/token/ISale.sol*
      - *contracts/RisingTide/RisingTide.sol*
      - *contracts/RisingTide/TestRisingTideWithStaticAmounts.sol*
      - *contracts/RisingTide/TestRisingTideWithCustomAmounts.sol*
      - *contracts/token/Citizend.sol*
    - **Commit:**
      - *b77c5e78621812e98a8cc4f7ad6e6a170f3a7a65 on main*
    - **Pull request:**
      - https://github.com/subvisual/citizend/pull/245/files
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Test Methodology

The Citizend smart contract stack is a system designed to facilitate token sales and distribution in a secure and equitable manner. This security audit primarily focused on two pivotal components of the Citizend framework: the Rising Tide logic and Sale contract. These elements were identified as critical due to their central role in managing and executing token distributions and sales, which are core functionalities of the Citizend platform.

Audit Scope & Rationale

While the review procedures sufficiently covered all in-scope smart contracts, the test team prioritized the following components for heightened scrutiny:

- **Rising Tide logic:** This feature implements a dynamic cap system to ensure fair token distribution among investors based on predefined criteria. The logic handles sensitive calculations and state transitions that could potentially impact the integrity of the token distribution process.
- **Sale contract:** The Sale contract interfaces directly with users, handling the receipt of funds, token allocation, and adherence to the sale timeline. Moreover, mechanisms for enforcing contribution limits, managing sale states, and ensuring that transactions adhere to the rules defined by the Rising Tide logic are provided.

This assignment's core intention was to ensure that both the Rising Tide logic and Sale contract are secure and capable of handling edge cases without compromising the platform's integrity or user funds. Cure53 specifically honed in on:

- **Security:** Identifying vulnerabilities such as reentrancy, overflow/underflow, and unauthorized access.
- **Functionality:** Ensuring that all contract functions behave as expected under various conditions.
- **Performance:** Optimizing gas usage and ensuring the contracts can efficiently handle high transaction volumes.

Sale Logic

The key driving factor behind this proportion of the project was to ascertain whether the Sale contract operates securely and efficiently, as well as confirm whether the implementation conformed with established smart contract best practices. The testing methodology focused on validating contract logic, transaction safety, and sound execution of the expected functionality under various scenarios.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The testing process for the Sale contract was divided into several phases to cover both functional behavior and security aspects:

- **Manual review:**
  - **Code review:** Manually inspect the contract code to ensure it conforms with Solidity best practices and the latest Ethereum development standards. Key focus areas include function visibility, modifier usage, state variable handling, and contract specification compliance.
  - **Logic verification:** Verify the contract's logical flow, particularly the handling of the registration period, sale period, and token allocation logic, including interactions with the Rising Tide contract for investment cap management.
- **Scenario-based testing:**
  - **Integration tests:** Conduct evaluations that simulate real-world operational scenarios to verify the contract's interaction with external contracts, e.g., ERC20 tokens, and other components such as the Rising Tide logic.
- **Security-specific tests:**
  - **Reentrancy attacks:** Specifically test all public and external functions for reentrancy vulnerabilities, particularly those that transfer funds.
  - **Access control checks:** Ensure that only authorized roles can execute sensitive functions such as withdrawing funds, setting the sale period, or updating the Merkle root.
  - **Merkle proof verification:** Test the verification process of Merkle proofs for buyer eligibility, ensuring that only valid participants can purchase tokens during the sale.
  - **Gas usage optimization:** Evaluate functions for excessive gas usage, identifying potential areas for optimization to ensure transactions remain cost-effective for users.
- **Audit reporting:**
  - Document all findings from the audit process, categorizing them by severity (i.e., *Critical*, *High*, *Medium*, *Low*, or *Info*). Provide detailed explanations of each detected issue and suggested recommendations for mitigating the corresponding risks.

Rising Tide Logic

This section outlines the audit conducted for the Rising Tide smart contract logic implemented in Solidity for the Citizend smart contract stack. The Rising Tide logic represents an implementation of a token sale management mechanism. As stated in the Rising Tide documentation:

> *The Rising Tide Mechanism governs the sale of the $CTND token and optimizes for maximum broad participation and is a fixed-price sale. The Rising Tide Mechanism is a fixed price sale that optimizes for community participation. Every participant that for the sale has a chance to get an equal allocation like everyone else. During the*

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

collection phase, all qualifying participants can commit their preferred allocation amount.

The Rising Tide contract is an abstract implementation designed to handle investments with a dynamic cap, ensuring equitable distribution based on predefined conditions. It employs the Math library for safe mathematical operations and introduces complex state and validation management via multiple functions and state variables.

Cure53's security analysis covered the following key components of the smart contract:

- **State variables & enums:**
  - *RisingTideState*: Enum detailing the contract state concerning the investment cap's validation status.
  - *RisingTideCache*: Struct for caching values during the cap validation process.
  - *individualCap*: Stores the maximum amount of tokens an individual investor can receive.
- **Functions & logic validation:**
  - *_risingTide_setCap*: Sets a new investment cap and initializes the validation process.
  - *risingTide_validate*: Continues the cap validation process, ensuring it adheres to gas constraints.
  - *_risingTide_validCap*: Internal function determining the validity of the set cap based on the total allocated and distribution rules.
- **Security mechanisms:**
  - Use of *require* statements to enforce state prerequisites and logical consistency.
  - Integration of gas management techniques to prevent out-of-gas errors during iterative validation processes.
- **Mathematical calculations:**
  - Use of the imported Math library for safe arithmetic operations to avoid overflows and underflows.
  - Calculation logic within *_risingTide_validCap* ensures that the investment cap aligns with the overall allocated amount and individual investments.
- **State transition management:**
  - Diligent management of state transitions between different phases of cap validation and finalization.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *FRC-03-001*) to facilitate any future follow-up correspondence.

FRC-03-002 WP1: Absent validation checks on *minContribution* *(High)*

While reviewing the Sale.sol smart contract, Cure53 noted that the *buy* function contains a condition to check if the *_amount* specified by the user meets or exceeds the *minContribution*. However, the implementation neglects to include an explicit safeguard against configuring *minContribution* as zero. This could lead to unintended behaviors and vulnerabilities in the contract's logic, particularly with regards to fundraising and spam prevention.

Similarly, the configuration omits a check to ensure that *minContribution* is less than *minTarget*; this should also be incorporated in accordance with best practices.

Affected file:
Sale.sol

Affected code:

```
/// Sets the minimum contribution
/// @param _minContribution new minimum contribution
function setMinContribution(
	uint256 _minContribution
) external onlyRole(DEFAULT_ADMIN_ROLE) nonReentrant {
	minContribution = _minContribution;
}

[...]

function buy(
	uint256 _amount,
	bytes32[] calldata _merkleProof
) external override(ISale) inSale nonReentrant {
	if (_investorCount >= minTarget / minContribution)
		revert MaxTargetReached();

	bytes32 leaf = keccak256(abi.encodePacked(msg.sender));
	bool isValidLeaf = MerkleProof.verify(_merkleProof, merkleRoot,
leaf);
	if (!isValidLeaf) revert InvalidLeaf();
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
        require(_amount >= minContribution, "can't be below minimum");

        uint256 paymentAmount = tokenToPaymentToken(_amount);
        require(paymentAmount > 0, "can't be zero");

        uint256 currentAllocation = accounts[msg.sender].uncappedAllocation;

        if (currentAllocation == 0) {
              investorByIndex[_investorCount] = msg.sender;
              _investorCount++;
        }

        accounts[msg.sender].uncappedAllocation += _amount;
        totalUncappedAllocations += _amount;

        emit Purchase(msg.sender, paymentAmount, _amount);

        IERC20(paymentToken).safeTransferFrom(
              msg.sender,
              address(this),
              paymentAmount
        );
}
```

This suboptimal setup can introduce a number of risks, including (but not limited to):

- **Unrestricted participation**: If *minContribution* is set to zero, any amount including zero can be considered valid for participation in the sale. This can lead to a scenario whereby an unlimited number of participants can join the sale without actually contributing meaningful (or indeed any) amounts, which defeats the purpose of enforcing a minimum contribution threshold.

- **Denial of Service risk**: With no minimum contribution, attackers could flood the contract with a vast volume of zero contribution transactions. This could congest the network and potentially lead to increased transaction fees, affecting typical contract operations and degrading service for legitimate users.

- **Manipulation of investor count**: The contract utilizes *_investorCount* to limit the number of participants based on the ratio of *minTarget* to *minContribution*. If *minContribution* is zero, this could potentially lead to a division by zero error or result in an impractically high limit, effectively removing any cap on the number of investors.

- **Economic impact on token valuation:** The acceptance of minimal or zero contributions can lead to an excessive increase in token supply without

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

corresponding economic inflow, potentially diluting the token value and negatively impacting all token holders.

To mitigate this issue, Cure53 recommends adopting a three-fold strategy, as follows:

- **Ensure non-zero minimum**: Implement a safeguard that ensures *minContribution* is always set to a sensible, non-zero minimum value during the initialization or update processes. This should be enforced either via a smart contract function modifier or by requiring conditions within the function setting this value.

- **Validate parameters thoroughly**: Enhance parameter validation in functions that interact with critical variables such as *minContribution* to prevent misconfiguration or manipulation.

- **Rate limiting and anti-spam measures**: Consider implementing rate-limiting mechanisms or additional checks to prevent sale mechanism abuse, particularly if low contribution thresholds are enforced.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

FRC-03-001 WP1: Time measured by dependency on block timestamps *(Info)*

The test team confirmed that multiple smart contracts in Citizend employ *block.timestamp* to verify the timing constraints on certain operations (e.g., sale periods). The *block.timestamp* represents the timestamp when a block is proposed, which can be manipulated by miners within reasonable limits. This poses a potential risk to functions that rely on precise timing conditions.

Affected files:

- *Citizend.sol*
- *Sale.sol*

The contracts use *block.timestamp* in several modifiers (*beforeSale*, *inSale*, *afterSale*, and *onlyAfter*) to enforce time-based constraints. While generally reliable, *block.timestamp* can be influenced by miners, who are able to manipulate the timestamp to a certain degree (for instance, they are typically permitted to deviate by up to 15 seconds from the real time).

This situation incurs multiple security threats, including (but not limited to):

- **Miner manipulation**: Miners can set the block timestamp to a future time within the allocated range (up to approximately 900 seconds ahead of the last block's timestamp), potentially initiating or extending sale periods in a malicious manner.

- **Reliance on precise timing**: Financial mechanisms that depend on precise timing (e.g., sales initiating or ending at exact moments) are vulnerable to minor timestamp manipulation. This could lead to scenarios whereby users either miss out on participating in a sale due to an artificially advanced *block.timestamp*, or are granted the ability to participate in a sale that should have already concluded.

- **Transaction ordering**: Transactions within the same block can be manipulated with regards to miner ordering, leading to the potential prioritization of transactions that benefit them, particularly near the boundaries of a timed sale period.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 advises incorporating multiple recommended solutions, as proposed below:

- **Alternative time sources**: Consider employing a time source with greater reliability such as an external oracle, which provides a secured and verified timestamp. This approach reduces reliance on the block miner's timestamp.

- **Guard checks**: Implement additional checks and mechanisms to ensure that the difference between *block.timestamp* and the actual real time remains within acceptable bounds, potentially issuing alerts if discrepancies are detected.

- **Event-based triggers**: Rather than use time-based triggers for sales, the Fractal team could utilize event-based triggers where possible. For example, transitions could be based on the completion of certain actions or the occurrence of specific events, which are less vulnerable to manipulation.

- **Safety margins**: Design time-sensitive functionality with safety margins that account for plausible *block.timestamp* variance. For example, one could avoid scheduling critical actions to occur exactly at the start or end of a timed window.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

This final section of the report collates Cure53's various opinions of the scope's security performance, based on the evidence accrued during this Q2 2024 assignment targeting the Fractal Sale, Citizend, and Rising Tide smart contracts. In short, the scrutinized mechanisms exhibit a robust security posture at present, though some opportunities for enhancements were observed.

The two-person audit team sought to verify the efficacy and operational integrity of the platform's security measures prior to its scheduled deployment. After the completion of extensive evaluations across the five-day review window, the Cure53 consultants detected two noteworthy risks.

The first of those relates to the dependence on *block.timestamp* for time-sensitive operations. This poses a risk of manipulation by miners and could affect the integrity of sale periods and transaction ordering, potentially impacting legitimate participants (see FRC-03-001).

The second flaw represents a significant potential vulnerability related to *minContribution* settings in the Sale.sol contract. Specifically, the absence of minimum contribution checks could allow for zero contributions, potentially leading to spam attacks and economic imbalance within the token system. For supporting guidance, please refer to ticket FRC-03-002.

A number of recommendations have been provided to address the identified pitfalls, including implementing non-zero minimum contribution settings, utilizing external time sources, and employing additional safeguards to enhance the robustness of transaction and time handling within the contracts.

Cure53 must emphasize that the Rising Tide logic, while functionally sound, is a compositionally intricate piece of architecture that could benefit from simplification. As such, reducing its complexity could decrease the likelihood of errors in future modifications and ease the onboarding process for new developers.

In conclusion, despite the detection of certain detrimental behaviors, the implementation of the Rising Tide Sale management logic was deemed sound and the smart contracts generally adhered to industry best practices. The Fractal developers should encounter little difficulty improving the security posture of the audited contracts by following the mitigation strategies outlined in the tickets provided.

Cure53 would like to thank Davide Silva and Anna Bikmetova from the Trust Fractal GmbH team for their excellent project coordination, support, and assistance, both before and during this assignment.