Zellic

WSTBT

Smart Contract Security Assessment

June 29, 2023

Prepared for:

Ding Yao

Matrixdock

Prepared by:

Aaron Esau

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Matrixdock from June 27th to June 29th, 2023. During this engagement, Zellic reviewed WSTBT's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 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:

- Are there any logic flaws in the contracts?
- What are the centralization risks?

1.2 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
- Plans for deployment

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

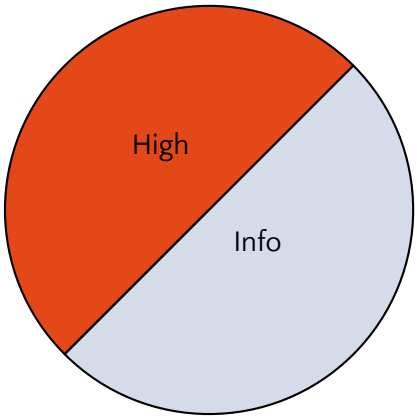
1.3 Results

During our assessment on the scoped WSTBT contract, we discovered two findings. No critical issues were found. One was of high impact and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Matrixdock's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

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



2 Introduction

2.1 About WSTBT

WSTBT is a contract specifically designed to wrap or unwrap STBT tokens.

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 contract.

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 code base 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, upgradeability 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.

2.3 Scope

The engagement involved a review of the following targets:

WSTBT Contract

Repository	https://github.com/Matrixdock-STBT/STBT-contracts/
Version	STBT-contracts: `0f5d608eccf8a72c83b9e7e4682fae0cb7f5f828`
Program	WSTBT
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with one consultant for a total of twelve person-hours. The assessment was conducted over the course of three calendar days.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultant was engaged to conduct the assessment:

Aaron Esau, Engineer
aaron@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

June 27, 2023	Start of primary review period
June 29, 2023	End of primary review period

3 Detailed Findings

3.1 Arbitrary funds transfer function

- **Target**: WSTBT
- **Category**: Business Logic
- **Likelihood**: Low
- **Severity**: Critical
- **Impact**: High

Description

The `controllerTransfer` function allows any address returned by `stbtAddress.controller()` to transfer funds from and to arbitrary accounts:

```
function controllerTransfer(address _from, address _to, uint256 _value,
    bytes calldata _data, bytes calldata _operatorData)
    external onlyController {
    _transfer(_from, _to, _value);
    emit ControllerTransfer(msg.sender, _from, _to, _value, _data,
    _operatorData);
}
```

Impact

If the contract whose address is stored in `stbtAddress` is malicious or maliciously upgraded, or if the wallet whose address is returned by `stbtAddress.controller()` is compromised, a malicious actor can transfer funds arbitrarily between users.

Recommendations

Ensure the contract configured in `stbtAddress` on deployment is trusted. Also, regardless of what the contract configured is, ensure `stbtAddress.controller()` returns an immutably configured address of a multi-sig or governance contract to minimize centralization risk.

Remediation

Matrixdock noted that this is by design, and that they do not currently have plans to make changes in response to this finding.

3.2 Missing permission checks when wrapping and unwrapping

- **Target**: WSTBT
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: N/A

Description

The `transfer` and `transferFrom` functions call `_checkSendPermission` and `_checkReceivePermission` on the sender and receiver to ensure they are permitted to send or receive.

However, the `wrap` and `unwrap` functions mint and burn using STBT as an underlying token, which may be transferred independently of the wrapped WSTBT token.

Impact

The implication is that the `_checkSendPermission` and `_checkReceivePermission` checks may be bypassed using the ERC-20 token contract at `stbtAddress`; however, in practice, the `stbtAddress` address will be the STBT contract that implements the same permissions checks, causing this to be unexploitable.

Recommendations

We still recommend using `_checkReceivePermission` before minting and `_checkSendPermission` before burning to keep the contract logically consistent:

```
function wrap(uint256 stbtAmount) public returns (uint wrappedShares) {
    require(stbtAmount ≠ 0, "WSTBT: ZERO_AMOUNT");
    _checkReceivePermission(msg.sender);
    wrappedShares = ISTBT(stbtAddress).getSharesByAmount(stbtAmount);
    ISTBT(stbtAddress).transferFrom(msg.sender, address(this),
    stbtAmount);
    _mint(msg.sender, wrappedShares);
    emit Wrap(msg.sender, stbtAmount, wrappedShares);
}

function unwrap(uint256 unwrappedShares) public returns (uint stbtAmount)
    {
    require(unwrappedShares ≠ 0, "WSTBT: ZERO_AMOUNT");
```

```
    _checkSendPermission(msg.sender);
    stbtAmount = ISTBT(stbtAddress).getAmountByShares(unwrappedShares);
    ISTBT(stbtAddress).transfer(msg.sender, stbtAmount);
    _burn(msg.sender, unwrappedShares);
    emit Unwrap(msg.sender, stbtAmount, unwrappedShares);
}
```

Remediation

This issue has been acknowledged by Matrixdock, and a fix was implemented in commit 205050b3.

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 STBT moderator can disable wrapping and unwrapping

Note that the STBT moderator can disable or enable wrapping and unwrapping at any time by revoking receiving permissions from the WSTBT contract or sending permissions, respectively.

4.2 Functions rounding down cause loss to user

The `wrap` and `unwrap` functions use the `getSharesByAmount` and `getAmountByShares` functions to calculate the number of shares to mint to the user and the amount of unwrapped shares to pay the user, respectively.

```
function getSharesByAmount(uint256 _amount)
    public view returns (uint256 result) {
    // unchecked {
    // result = _amount * totalShares / totalSupply; // divide-by-zero
    will return zero
    // }
    return totalSupply == 0 ? 0 : _amount * totalShares / totalSupply;
}

function getAmountByShares(uint256 _shares)
    public view returns (uint256 result) {
    // unchecked {
    // result = _shares * totalSupply / totalShares; // divide-by-zero
    will return zero
    // }
    return totalShares == 0 ? 0 : _shares * totalSupply / totalShares;
}
```

Note that these functions round down and may cause a negligible loss to the user, though this is preferable to loss to the contracts.

5 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart 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: WSTBT.sol

Function: `controllerTransfer(address _from, address _to, uint256 _value, byte[] _data, byte[] _operatorData)`

Allows the user returned by `stbtAddress.controller()` to transfer arbitrary funds from any address to any address.

Inputs

- `_from`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The address to transfer funds from.
- `_to`
  - **Control**: Full.
  - **Constraints**: Must be a nonzero address.
  - **Impact**: The address to transfer funds to.
- `_value`
  - **Control**: Full.
  - **Constraints**: Must be less than or equal to the balance of the `_from` address.
  - **Impact**: The amount of funds to transfer.
- `_data`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Emitted in the `ControllerTransfer` event.
- `_operatorData`
  - **Control**: Full.
  - **Constraints**: None.

- **Impact**: Emitted in the `ControllerTransfer` event.

Branches and code coverage (including function calls)

Intended branches

- Transfers funds.
  - ☐ Test coverage

Negative behavior

- Cannot transfer to a zero address.
  - ☐ Negative test
- Cannot transfer an amount greater than the balance of the `_from` address.
  - ☐ Negative test
- Caller must be the address returned by `stbtAddress.controller()`.
  - ☐ Negative test

Function: `transferFrom(address _sender, address _recipient, uint256 _amount)`

Transfers WSTBT from the `_sender` to the `_recipient` address and updates the approval amount.

Inputs

- `_sender`
  - **Control**: Full.
  - **Constraints**: Must have sending permissions.
  - **Impact**: The address to transfer tokens from.
- `_recipient`
  - **Control**: Full.
  - **Constraints**: Address must be nonzero and must have receiving permissions.
  - **Impact**: The address to transfer tokens to.
- `_amount`
  - **Control**: Full.
  - **Constraints**: Must be less than or equal to the approved amount from `msg.sender` of `_sender` and must be less than or equal to the balance of `_sender`.
  - **Impact**: The amount of tokens to send and the amount to lower `msg.sender`'s approval by on `_sender`.

Branches and code coverage (including function calls)

Intended branches

- Transfers tokens to the receiver and lowers approval amount.
  - ☐ Test coverage

Negative behavior

- Cannot transfer to a zero address.
  - ☐ Negative test
- Sender cannot have sending permissions.
  - ☐ Negative test
- Receiver cannot have receiving permissions.
  - ☐ Negative test
- Amount cannot be greater than the approved amount.
  - ☐ Negative test
- Amount cannot be greater than the sender's balance.
  - ☐ Negative test

Function: `transfer(address _recipient, uint256 _amount)`

Transfers WSTBT from the `msg.sender` to the `_recipient` address.

Inputs

- `_recipient`
  - **Control**: Full.
  - **Constraints**: Must have receive permissions and be a nonzero address.
  - **Impact**: The recipient of tokens.
- `_amount`
  - **Control**: Full.
  - **Constraints**: Must be less than or equal to the sender's balance.
  - **Impact**: The amount of tokens to send.

Branches and code coverage (including function calls)

Intended branches

- Transfers tokens to the receiver.
  - ☐ Test coverage

Negative behavior

- Cannot transfer to a zero address.
  - ☐ Negative test
- Sender does not have sending permissions.
  - ☐ Negative test
- Receiver does not have receiving permissions.
  - ☐ Negative test
- Cannot send more than the balance of the sender.
  - ☐ Negative test

Function: `unwrap(uint256 unwrappedShares)`

Allows a user to exchange WSTBT for STBT.

Inputs

- `unwrappedShares`
  - **Control**: Full.
  - **Constraints**: Caller's WSTBT balance must be greater than or equal to this value.
  - **Impact**: The amount of WSTBT to unwrap as STBT.

Branches and code coverage (including function calls)

Intended branches

- Successfully unwraps WSTBT as STBT.
  - ☐ Test coverage

Negative behavior

- Caller must have enough WSTBT to burn.
  - ☐ Negative test
- Caller must have receiving permissions for STBT.
  - ☐ Negative test
- The WSTBT contract must have sending permissions for STBT.
  - ☐ Negative test
- Caller must have receiving permissions for WSTBT.
  - ☐ Negative test

Function: `wrap(uint256 stbtAmount)`

Allows a user to exchange STBT for WSTBT.

Inputs

- `stbtAmount`
  - **Control**: Full.
  - **Constraints**: Caller's STBT balance must be greater than or equal to this value.
  - **Impact**: The amount of STBT to wrap as WSTBT.

Branches and code coverage (including function calls)

Intended branches

- Successfully wraps STBT to WSTBT.
  - ☐ Test coverage

Negative behavior

- Caller must have enough STBT to transfer.
  - ☐ Negative test
- Caller must have sending permissions for STBT.
  - ☐ Negative test
- The WSTBT contract must have receiving permissions for STBT.
  - ☐ Negative test
- Caller must have receiving permissions for WSTBT.
  - ☐ Negative test

6 Audit Results

At the time of our audit, the audited code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped WSTBT contract, we discovered two findings. No critical issues were found. One was of high impact and the remaining finding was informational in nature. Matrixdock acknowledged all findings and implemented fixes.

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 audit version of 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.

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.