Zellic

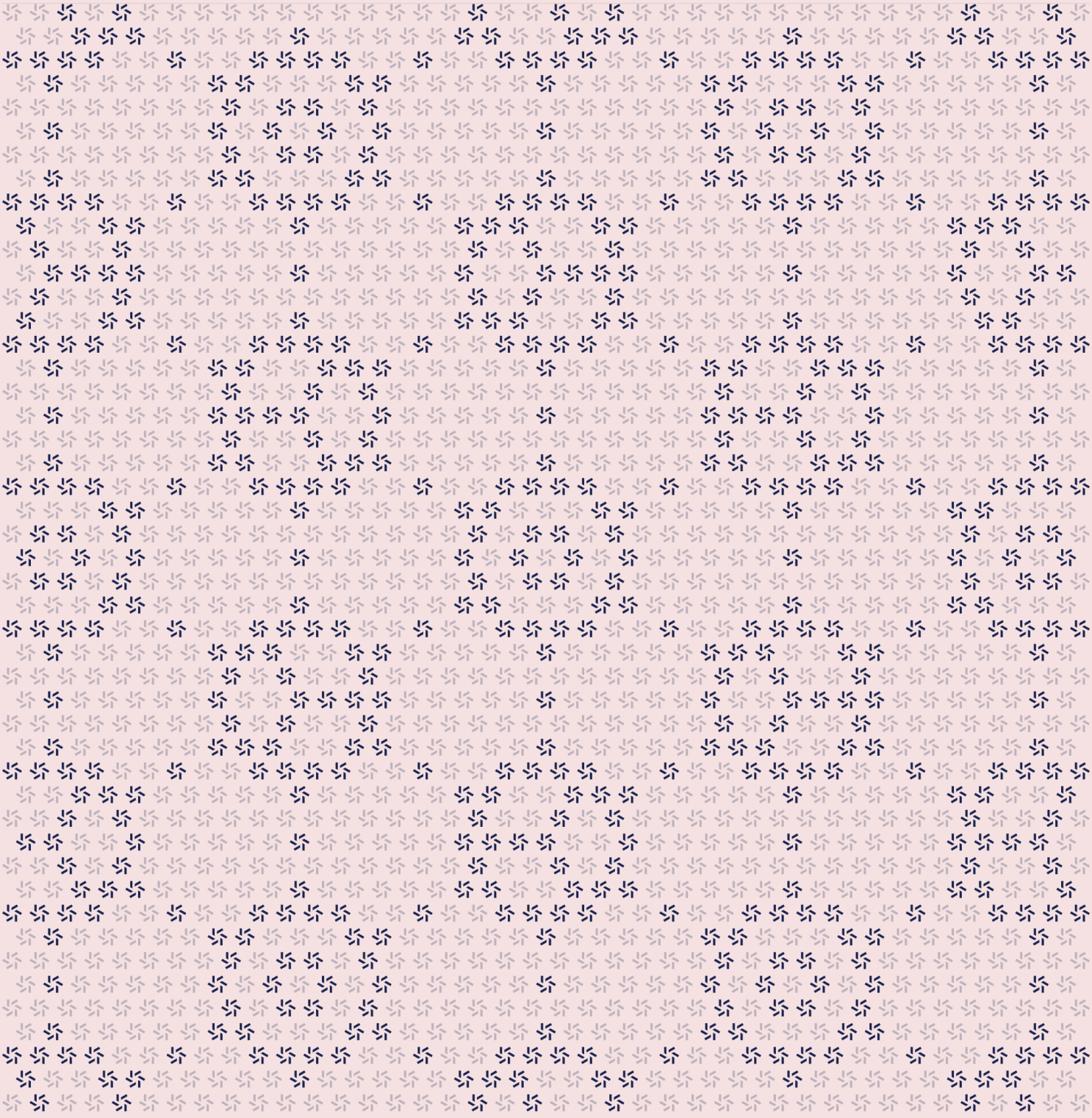
Prepared for
Dr. Elias
The Open Network

Prepared by
Filippo Cremonese
Daniel Lu
Zellic

November 16, 2023

Vesting Wallet

Smart Contract Security Assessment



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 The Open Network from November 10th to November 13th, 2023. During this engagement, Zellic reviewed Vesting Wallet'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:

- Does the protocol allow users ways to misuse locked funds?
- Are there issues in how funds are locked?
- Are there logic bugs related to whitelisting?

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

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

During this assessment, the limited scope prevented us from verifying the contract's precise interactions with others in the protocol. For instance, an improperly designed, whitelisted component could create a mechanism for prematurely withdrawing funds.

1.3. Results

During our assessment on the scoped Vesting Wallet contracts, there were no security vulnerabilities discovered.

Zellic recorded its notes and observations from the assessment for The Open Network's benefit in the Discussion section (3. ↗).

Breakdown of Finding Impacts

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

2. Introduction

2.1. About Vesting Wallet

Vesting Wallet is a lockup and vesting smart contract with an ability to use locked funds as a stake for validation in TON blockchain.

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 (3. ↗) 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:

Vesting Wallet Contracts

Repository	https://github.com/ton-blockchain/vesting-contract ↗
Version	vesting-contract: `f486303a899365a4882f4f826e72536d6872b44a`
Program	vesting_wallet.fc
Type	FunC
Platform	TVM

2.4. Project Overview

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

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:

Filippo Cremonese
Engineer
fcremo@zellic.io

Daniel Lu
Engineer
daniel@zellic.io

2.5. Project Timeline

November 10, 2023	Start of primary review period
November 13, 2023	End of primary review period

3. 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.

3.1. Contract description

This is a wallet contract that has additional functionality for gradually unlocking deposited Toncoin. The vesting is parameterized by a start time, a duration, and an unlock period for unlocking funds in discrete intervals. It also allows a cliff period to be set during deployment, which determines when funds can first be withdrawn.

Funds are gradually unlocked during vesting. During that period, the owner of the wallet cannot access the remaining locked funds, except when sending them to a set of whitelisted addresses that the vesting sender determines or to the vesting sender themself. The contract also implements additional controls for what kinds of transactions can be sent to these whitelisted addresses.

3.2. Global variables

The contract global data stores the following data:

- `int stored_seqno`: This is the sequence number; it is used to reject replayed signatures when processing external messages.
- `int stored_subwallet`: This is the subwallet ID; it allows the creation of multiple wallets using the same private key and prevents signature replays across wallets with the same public key.
- `int public_key`: This is the public key of the wallet owner; it is used to sign external messages.
- `cell whitelist`: This is the dictionary containing whitelisted addresses.
- Vesting parameters cell ref:
    - `int vesting_start_time`: This is the timestamp at which the vesting period begins.
    - `int vesting_total_duration`: This is the duration of the vesting period in seconds.
    - `int unlock_period`: This is the duration of the unlock period in seconds.
    - `int cliff_duration`: This is the duration of the cliff period in seconds.
    - `int vesting_total_amount`: This is the amount of locked Toncoins.
    - `slice vesting_sender_address`: This address is inherently whitelisted, and can add whitelisted addresses by sending an `op::add_whitelist` internal message; whitelisted addresses can be sent locked funds at any time using an external message signed by `owner_address`.
    - `slice owner_address`: This is the address of the owner of the contract;

this address can send `op::send` internal messages.

Note 1: The `vesting_total_amount` is uncorrelated with the actual contract balance.

Note 2: The `owner_address` does not technically have to be correlated with `public_key`.

3.3. Internal messages processing

The contract explicitly processes two kinds of internal messages. Other unhandled messages are not rejected and are simply ignored.

`op::send`

This operation can only be requested by a sender matching `owner_address`. If the sender matches, the function invokes `send_message`, described below in 3.5. ↗

`op::add_whitelist`

This operation can be requested by a sender that matches `vesting_sender_address`.

It can be used to add one or more addresses to the whitelisted set. Whitelisted addresses can be sent locked funds at any time via an external message signed by `owner_address`.

3.4. External messages processing

External messages can be used to send a message from the vesting contract and move assets from it. The outgoing message has to be signed by the `public_key` stored in the contract data. In addition, the incoming message sequence number and subwallet ID must match the one stored in the contract storage.

3.5. Function `send_message`

This function is called when processing internal or external messages to in turn send a message from the Vesting Wallet contract. The amount attached to the message is limited by the logic of the function. The amount that has yet to be vested is computed using `_get_locked_amount`, and it is used to limit the outgoing amount using `raw_reserve`.

The attached amount and other message details are not limited if the vesting period has ended.

If the vesting period has not ended, the outgoing message must use mode `SEND_MODE_IGNORE_ERRORS | SEND_MODE_PAY_FEES_SEPARETELY`. The amount is also not limited if the destination is `vesting_sender_address` or another whitelisted address.

In case the destination is whitelisted, further restrictions are applied on the body of the outgoing

message. The outgoing message must be bounceable and must not have the state init flag set. Furthermore, the following requirements are enforced:

- Destination address matches configuration `elector_id`: operation must be `op::elector_new_stake`, `op::elector_recover_stake`, `op::vote_for_complaint`, or `op::vote_for_proposal`.
- destination address matches configuration `config_id`: operation must be `op::vote_for_proposal`.

If the destination address does not match the two above and the message body is not empty, the operation must pass the following check:

```
(op == 0) |
(op == op::single_nominator_pool_withdraw) | (op ==
    op::single_nominator_pool_change_validator) |
(op == op::ton_stakers_deposit) | (op == op::jetton_burn) | (op ==
    op::ton_stakers_vote) |
(op == op::vote_for_proposal) | (op == op::vote_for_complaint)
```

The operations match some operations supported by contracts not covered by the scope of this audit.

If `op` is zero, a further eight bits are loaded from the outgoing message body, which are required to match any of the ASCII values `dwDW`.

3.6. Test coverage

The project includes an extensive test suite with thorough coverage of important branches. These tests include both unit tests and integration tests, and they cover both positive and negative behavior.

In the current state of the project, a couple of tests fail because they are dependent on external factors (specifically, the *current* time when the tests are run). We recommend restructuring the tests to remove this dependence on the real-world timestamp because the broken tests are critical — they are exactly the ones checking that funds are locked properly.

It is important to note that code coverage is not a security guarantee: the state space of any contract is almost certainly much larger than the number of lines of code. For this reason, we recommend adopting fuzz testing or formal verification to further improve confidence in the protocol's correctness.

4. Assessment Results

At the time of our assessment, the reviewed code was not deployed to The Open Network mainnet.

During our assessment on the scoped Vesting Wallet contracts, there were no security vulnerabilities discovered.

4.1. 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.