Zellic

Prepared for
Haim Krasniker
StarkWare

Prepared by
Jinseo Kim
Ulrich Myhre
Zellic

October 3, 2024

StarkGate

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 StarkWare from September 26th to September 27th, 2024. During this engagement, Zellic reviewed StarkGate's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

For this audit, we compared the scope of this audit to the snapshot of the commit hash (eedee830 ↗).

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:

- Could an accidental or malicious behavior improperly increase the cost of execution?
- Could a malicious attacker steal or lock the assets of users?
- Are the features correctly implemented?

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 StarkGate contracts, we discovered one finding, which was of medium impact.

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	1
🟩 Low	0
⬜ Informational	0

2. Introduction

2.1. About StarkGate

StarkWare contributed the following description of StarkGate:

> StarkGate, developed by StarkWare, bridges ETH and ERC-20 tokens between Ethereum and Starknet. Each supported token is associated with an L1 and L2 bridge contract that communicates via Starknet's messaging mechanism.

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.

Architecture risks. This encompasses potential hazards originating from the blueprint of a system, which involves its core validation mechanism and other architecturally significant constituents influencing the system's fundamental security attributes, presumptions, trust mode, and design.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Implementation risks. This encompasses risks linked to translating a system's specification into practical code. Constructing a custom system involves developing intricate on-chain and off-chain elements while accommodating the idiosyncrasies and challenges presented by distinct programming languages, frameworks, and execution environments.

Availability. Denial-of-service attacks are another leading issue in custom systems. Issues including but not limited to unhandled panics, unbounded computations, and incorrect error handling can potentially lead to consensus failures.

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:

StarkGate Contracts

Types	Solidity, Cairo
Platforms	EVM-compatible, Starknet

Target	starkgate-contracts
Repository	https://github.com/starknet-io/starkgate-contracts ↗
Version	`5a10fd263d29cd032b7229691d043520edae0737`
Programs	`src/**/*.cairo` `src/**/*.sol`

2.4. Project Overview

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

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Jinseo Kim
Engineer
jinseo@zellic.io

Ulrich Myhre
Engineer
unblvr@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 26, 2024	Start of primary review period
September 27, 2024	End of primary review period

3. Detailed Findings

3.1. LegacyBridge does not check the token of messages in the old format

Target	LegacyBridge		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

LegacyBridge is the abstract bridge contract that also tries to handle the messages in the old format. Specifically, when this contract consumes the message, it tries to consume the message in the old format:

```
function consumeMessage(
    address token,
    uint256 amount,
    address recipient
) internal virtual override {
    require(l2TokenBridge() != 0, "L2_BRIDGE_NOT_SET");
    uint256 u_recipient = uint256(uint160(recipient));
    uint256 amount_low = amount & (UINT256_PART_SIZE - 1);
    uint256 amount_high = amount >> UINT256_PART_SIZE_BITS;

    // Compose the new format of L2-L1 consumption.
    uint256[] memory payload = new uint256[](5);
    payload[0] = TRANSFER_FROM_STARKNET;
    payload[1] = u_recipient;
    payload[2] = uint256(uint160(token));
    payload[3] = amount_low;
    payload[4] = amount_high;
    // Contain failure of comsumption (e.g. no message to consume).
    try messagingContract().consumeMessageFromL2(l2TokenBridge(), payload) {}
    catch Error(
        string memory
    ) {
        // Upon failure with the new format,
        // compose the old format,
        // in case the withdrawal was initiated on a bridge with an older
    version.
        payload = new uint256[](4);
        payload[0] = TRANSFER_FROM_STARKNET;
        payload[1] = u_recipient;
        payload[2] = amount_low;
```

```
            payload[3] = amount_high;
            messagingContract().consumeMessageFromL2(l2TokenBridge(), payload);
        }
    }
```

The difference between the old message format and the new format is the token parameter; a message can specify the token to be sent with the new message format, unlike the old message format. This implies that the messages with the old format assume the specific hardcoded token, which is not visible in the message payload.

It is important to note that the above function does not fail when it consumes the messages in the old format with the incorrect token parameter, compared to the assumed hardcoded token.

Impact

An attacker can exploit this issue to lock the funds sent with the old message format.

Let's assume there is one bridge which is upgraded to the LegacyBridge contract. This bridge has some pending messages in the old format, and its `consumeMessage` function can handle both the new and the old message formats. Here, an attacker can invoke the `withdraw` function for this message, setting the token parameter to the no-op contract which implements ERC-20 interface. This will successfully consume the message transfer of the token of the no-op ERC-20 contract, not the expected one.

In addition, if an upgraded bridge handles multiple tokens, an attacker can leverage this issue to withdraw the same amount of the different token.

Recommendations

Consider checking the token address for the old message format.

Remediation

This issue has been acknowledged by StarkWare and reportedly, a fix was implemented in Commit 45941888 ↗ and that fix was applied on all applicable deployed contracts.

4. 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.

4.1. Module: erc20_lockable.cairo

Function: `lock_and_delegate()`

This locks the given amount of tokens of the caller and delegates the vote to the specified delegatee.

Inputs

- `delegatee`
    - **Validation**: None.
    - **Impact**: An address that the vote is delegated to.
- `amount`
    - **Validation**: Must be less than or equal to the amount of the token the caller holds.
    - **Impact**: An amount of the token that is locked and delegated.

Branches and code coverage (including function calls)

Intended branches

- Lock the balance.
    - [x] Test coverage
- Delegate the vote to the delegatee.
    - [x] Test coverage

Negative behavior

- Locking contract is unspecified.
    - [x] Negative test
- Specified amount is greater than the balance of the caller.
    - [ ] Negative test

Function: `lock_and_delegate_by_sig()`

This locks the given amount of tokens of the signer and delegates the vote to the specified delegatee.

Inputs

- `account`
    - **Validation**: Must be validated with the given signature.
    - **Impact**: An address that the balance is to be locked from.
- `delegatee`
    - **Validation**: None.
    - **Impact**: An address that the vote is delegated to.
- `amount`
    - **Validation**: Must be less than or equal to the amount of the token the caller holds.
    - **Impact**: An amount of the token that is locked and delegated.
- `nonce`
    - **Validation**: Must be different in the case all of the other parameters are equal to the previous request.
    - **Impact**: An address that the vote is delegated to.
- `expiry`
    - **Validation**: Must be greater than or equal to the current timestamp.
    - **Impact**: An address that the vote is delegated to.
- `signature`
    - **Validation**: None.
    - **Impact**: A signature from the given `account`.

Branches and code coverage (including function calls)

Intended branches

- Validate the signature.
    - ☑ Test coverage
- Lock the balance.
    - ☑ Test coverage
- Delegate the vote to the delegatee.
    - ☑ Test coverage

Negative behavior

- Request is expired.
    - ☑ Negative test
- Request is replayed.
    - ☑ Negative test
- Signature is invalid.
    - ☑ Negative test
- Locking contract is unspecified.
    - ☑ Negative test
- Specified amount is greater than the balance of the caller.

- ☐ Negative test

Function: `permissioned_burn()`

This burns the given amount of tokens of the specified address.

Inputs

- `account`
  - **Validation**: None.
  - **Impact**: An address whose balance is to be decreased.
- `amount`
  - **Validation**: Must be less than or equal to the amount of tokens the account holds.
  - **Impact**: An amount of the token that is to be burnt.

Branches and code coverage (including function calls)

Intended branches

- Burn the token.
  - ☐ Test coverage

Negative behavior

- Invoked by a caller that does not have the required permission.
  - ☐ Negative test

Function: `permissioned_mint()`

This mints the given amount of tokens at the specified address.

Inputs

- `account`
  - **Validation**: None.
  - **Impact**: An address whose balance is to be increased.
- `amount`
  - **Validation**: Must be less than or equal to the amount of tokens the account holds.
  - **Impact**: An amount of the token that is to be minted.

Branches and code coverage (including function calls)

Intended branches

- Mint the token.
    - ☐ Test coverage

Negative behavior

- Invoked by a caller that does not have the required permission.
    - ☐ Negative test

Function: `set_locking_contract()`

This sets the locking contract to the specified address.

Inputs

- `locking_contract`
    - **Validation**: None.
    - **Impact**: Address of the locking contract.

Branches and code coverage (including function calls)

Intended branches

- Set the locking contract to the given address.
    - ☑ Test coverage

Negative behavior

- Invoked by a caller that is not the upgrade governor.
    - ☑ Negative test
- Address is the zero address.
    - ☑ Negative test
- Locking contract is already set.
    - ☑ Negative test

5. Assessment Results

At the time of our assessment, the reviewed code was deployed to the Starknet mainnet.

During our assessment on the scoped StarkGate contracts, we discovered one finding, which was of medium impact. The discovered issue has been acknowledged by StarkWare and reportedly, a fix was implemented in Commit 45941888 ↗ and that fix was applied on all applicable deployed contracts.

5.1. 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.