Zellic

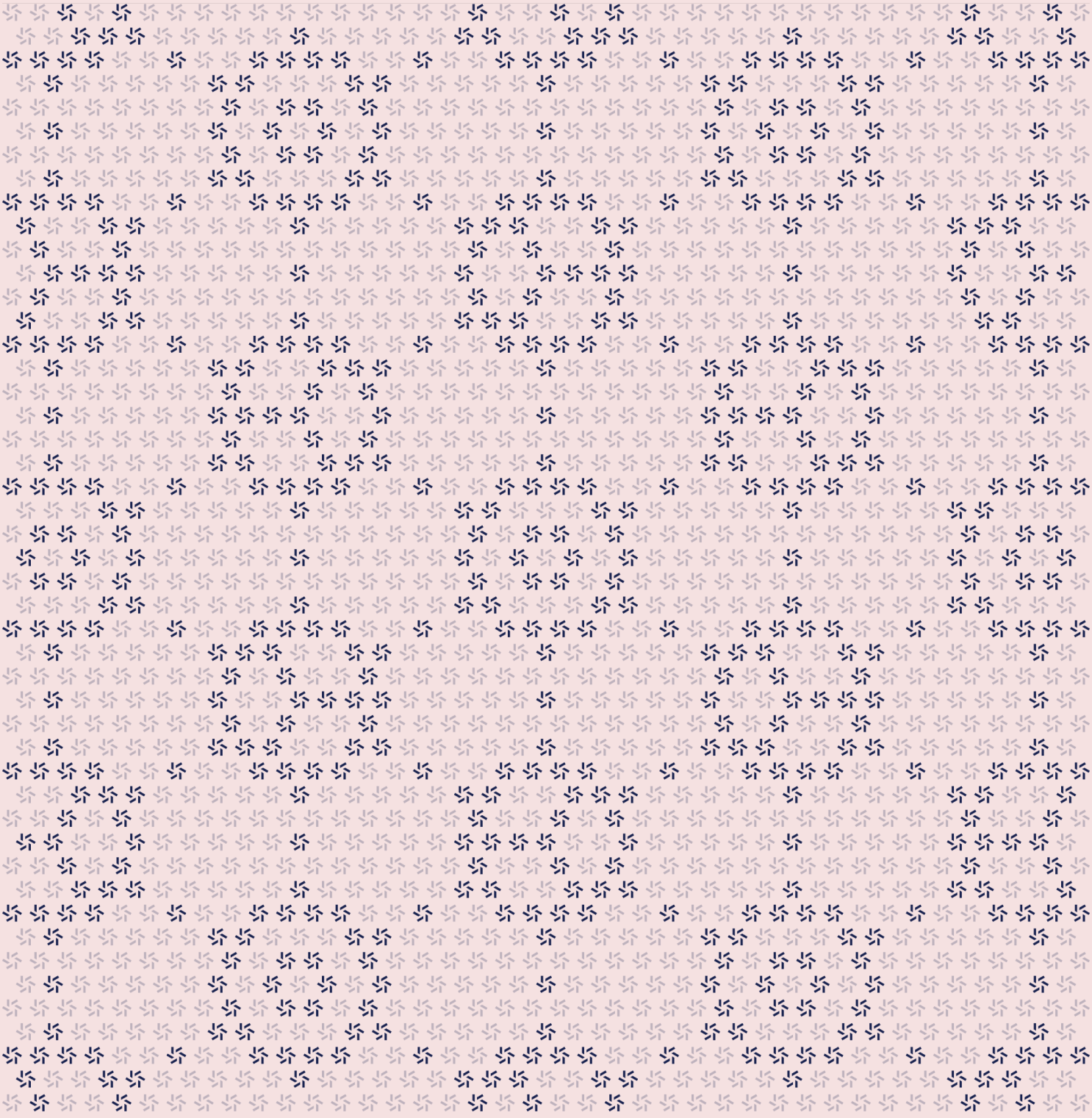
Prepared for
Sofiya Vasylyeva
Stake Technologies Pte. Ltd.

Prepared by
Syed Faraz Abrar
Filippo Cremonese
Zellic

February 12, 2024

Astar Network

Assets Pallet Chain Extension 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 Stake Technologies Pte. Ltd. from February 9th to February 12th, 2024. During this engagement, Zellic reviewed Astar Network'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:

- Can an attacker abuse the chain extension to gain approval to transfer other users' assets?
- Can an attacker abuse the chain extension to mint free tokens to themselves?
- Can an attacker abuse the chain extension to burn other users' assets?
- Can an attacker abuse the chain extension to cause a denial of service?

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 Astar Network contracts, we discovered one finding, which was of low impact.

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

Breakdown of Finding Impacts

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

2. Introduction

2.1. About Astar Network

Astar Network, Japan's leading blockchain, supports EVM, Substrate, WebAssembly (Wasm), and ink! environments to provide a scalable, cross-layer, and cross-machine protocol at the bleeding edge of innovation and interoperability. Their unique Build2Earn mechanism empowers developers, allowing them to earn incentives for building the decentralized future.

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:

Astar Network Contracts

Repository	https://github.com/AstarNetwork/Astar ↗
Version	Astar: `fc14b13401e1fb5e7391715fc76a308204173802`
Program	AssetsExtension in chain-extensions/pallet-assets/
Type	Rust
Platform	Astar

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for two person-days. 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:

Syed Faraz Abrar
Engineer
faith@zellic.io ↗

Filippo Cremonese
Engineer
fcremo@zellic.io ↗

2.5. Project Timeline

February 9, 2024	Start of primary review period
February 12, 2024	End of primary review period

3. Detailed Findings

3.1. Weight calculation

Target	Assets Pallet Chain Extension		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

All the assets pallet functions exposed by the chain extension are weighted with a constant amount. The weight of some operations is charged using the same quantity that the assets pallet benchmarks have computed. However, other operations only charge the weight of one runtime database read operation — `T::DbWeight::get().reads(1_u64)`.

Two functions, `MetadataSymbol` and `MetadataName`, operate on a variable amount of data, but they also only account for one runtime database read operation.

Impact

While we do not believe using constant weight is likely to cause significant issues, it is likely that the currently used weights do not represent fairly the load imposed on the network by the smart contracts invoking the operations exposed by the extension.

Recommendations

Implement more precise accounting for the weights charged for each function.

One possible option for `MetadataSymbol` and `MetadataName` could be to populate the `weight_per_byte` argument to the `env.write` call done at the end of the handlers of the respective function to return the value to the contract.

Remediation

The Astar team acknowledged this issue and stated that the gas weight currently being charged for one read is equivalent to a read of length 80 bytes. They also pointed out that the `MetadataName` and `MetadataSymbol` are bounded by the `AssetsStringLimit` config variable, which is set to 50 bytes. Therefore, charging for one read is fine in this instance.

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. Unexposed functionality

The chain extension does not expose all the functionality available from pallet assets. This is a deliberate and documented ↗ choice, because "some extrinsics are NOT part of the chain extension because they have no or limited usage for smart contracts".

We note that the choice to not expose some functionality has some potential security repercussions. For instance, the extension exposes the `approve_transfer` pallet function (as `ApproveTransfer`), but it does not expose the opposite `cancel_approval` function.

The `ApproveTransfer` function can only increase the allowance, not decrease it; if invoked repeatedly on the same recipient, the approvals are added up — invoking it with an amount of zero does not cancel the current approval. Therefore, smart contracts are able to grant approval over some asset to other addresses, but they are unable to revoke that approval.

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. Functions

Function: `Transfer`

Weight charged: Yes (same weight as assets pallet).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests only test success results. Failure results are untested.

Function: `Mint`

Weight charged: Yes (same weight as assets pallet).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests only test success results. Failure results are untested.

Function: `Burn`

Weight charged: Yes (same weight charged by assets pallet).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests only test success results. Failure results are untested.

Function: `ApproveTransfer`

Weight charged: Yes (same weight charged by assets pallet).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests only test success results. Failure results are untested.

Function: `TransferApproved`

Weight charged: Yes (same weight charged by assets pallet).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests only test success results. Failure results untested.

Function: `BalanceOf`

Weight charged: Yes (as one read from DB).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests.

Function: `TotalSupply`

Weight charged: Yes (as one read from DB).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests.

Function: `Allowance`

Weight charged: Yes (as one read from DB).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests.

Function: `MetadataName`

Weight charged: Yes (as one read from DB). Note: No need to take the length of the `MetadataName` into account for the gas weight as the length is limited to 50 bytes.

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests.

Function: `MetadataSymbol`

Weight charged: Yes (as one read from DB). Note: No need to take the length of the `MetadataSymbol` into account for the gas weight as the length is limited to 50 bytes.

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests.

Function: `MetadataDecimals`

Weight charged: Yes (as one read from DB).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests.

Function: `MinimumBalance`

Weight charged: Yes (as one read from DB).

Inputs checked: Responsibility of assets pallet.

Tested: Integration tests and unit tests.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Astar mainnet.

During our assessment on the scoped Astar Network contracts, we discovered one finding, which was of low impact. Stake Technologies Pte. Ltd. acknowledged the finding and implemented a fix.

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.