Zellic

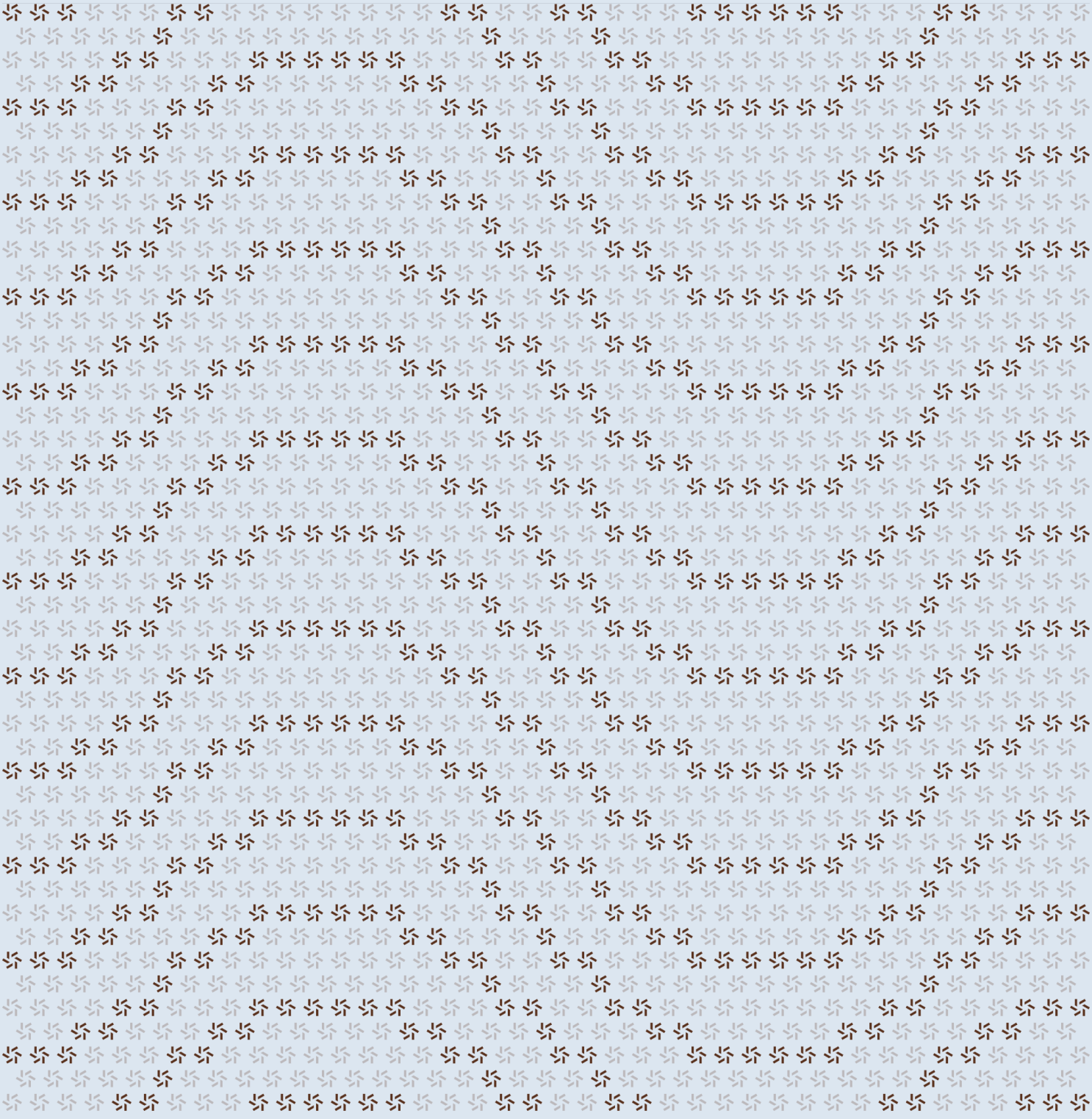
Prepared for
Drake Evans
Mysten Labs, Inc.

Prepared by
Junghoon Cho
Aaron Esau
Zellic

August 5, 2024

AUSD

Smart Contract Security Assessment



Contents

About Zellic **3**

1. Overview **3**

1.1. Executive Summary 4

1.2. Goals of the Assessment 4

1.3. Non-goals and Limitations 4

1.4. Results 4

2. Introduction **5**

2.1. About AUSD 6

2.2. Methodology 6

2.3. Scope 8

2.4. Project Overview 8

2.5. Project Timeline 9

3. Detailed Findings **9**

3.1. No version check for treasury in `roles` 10

4. Threat Model **11**

5. Assessment Results **13**

5.1. Disclaimer 14

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 Mysten Labs, Inc. from August 1st to August 2nd, 2024. During this engagement, Zellic reviewed AUSD'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:

- Are there any bugs that may result in loss of user funds?
- Can unauthorized entities mint or burn?
- Are roles properly respected?

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 AUSD modules, we discovered one finding, which was informational in nature.

Breakdown of Finding Impacts

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

2. Introduction

2.1. About AUSD

Mysten Labs, Inc. contributed the following description of AUSD:

> AUSD is a digital dollar to be used across blockchains, enabling permissionless transfers for users.

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 modules.

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.

2.3. Scope

The engagement involved a review of the following targets:

AUSD Modules

Type	Move
Platform	Sui
Target	ausd
Repository	https://github.com/MystenLabs/ausd ↗
Version	`469648c5fbcaa397e34c0a2410c44a0bef1a44dc`

Programs	`packages/sources/admin/admin.move`
	`packages/sources/admin/proposals.move`
	`packages/sources/constants.move`
	`packages/sources/ausd.move`
	`packages/sources/roles.move`
	`packages/sources/roles/pauser.move`
	`packages/sources/roles/minter.move`
	`packages/sources/roles/freezer.move`
	`packages/sources/roles/burner.move`
	`packages/sources/treasury.move`
	`packages/sources/setup.move`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 0.6 person-weeks. The assessment was conducted by two consultants over the course of two calendar days.

Contact Information

The following project manager was associated with the engagement:

Jacob Goreski
Jr. Engagement Manager
jacob@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Junghoon Cho
Engineer
junghoon@zellic.io ↗

Aaron Esau
Engineer
aaron@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

August 1, 2024	Start of primary review period
August 2, 2024	End of primary review period
August 2, 2024	Kick-off call

3. Detailed Findings

3.1. No version check for treasury in `roles`

Target	treasury.move		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The function `roles` returns a reference to the `Roles` object of the given `treasury`.

```
public(package) fun roles<T>(treasury: &ManagedTreasury<T>): &Roles {
    &treasury.roles
}
```

The returned reference is used to check if the sender of the transaction can use privileged functions such as mint, freeze, and burn. The code snippet below is an example of `freeze_address` using the reference to `roles` to perform this authorization.

```
public fun freeze_address<T>(
    treasury: &mut ManagedTreasury<T>,
    denylist: &mut DenyList,
    addr: address,
    ctx: &mut TxContext,
) {
    treasury.roles().assert_is_authorized<FreezerRole>(ctx.sender());
    treasury.roles().assert_is_not_paused<FreezerRole>();
    coin::deny_list_add(denylist, treasury.denylist_cap_mut(), addr, ctx);
}
```

The `roles` function currently lacks a check that the `treasury` is not outdated.

Impact

The `Roles` object of an outdated version of `treasury` can be borrowed, allowing authentication on capability. However, any operations that require a mutable reference to a capability cannot be done.

Recommendations

Add the following check to `roles`.

```
public(package) fun roles<T>(treasury: &ManagedTreasury<T>): &Roles {
    treasury.assert_is_valid_version();
    &treasury.roles
}
```

Remediation

Mysten Labs, Inc. explained that the decision not to require version checks on immutable access to roles was made to preserve the ability for admins to switch versions across all packages and similar functionalities.

4. Threat Model

The system defines the roles described in this section.

Admin

The role has the following abilities:

- Set a new config version
- Authorize a new admin
- Authorize a new minter
- Authorize a new freezer
- Authorize a new burner
- Authorize a new pauser
- Deauthorize an admin
- Reject proposals

Timelocked admin

The role has the following abilities:

- Create a proposal to authorize an admin
- Create a proposal to authorize a minter
- Create a proposal to authorize a freezer
- Create a proposal to authorize a burner
- Create a proposal to authorize a pauser
- Create a proposal to deauthorize an admin
- Reject proposals created by themselves
- Execute proposals created by themselves that have been queued for the timelock period

Freezer

The role has the following abilities:

- Freeze an address, provided the role is not paused
- Unfreeze an address, provided the role is not paused

Burner

The role has the ability to permanently freeze any number of tokens, provided the role is not paused.

Minter

The role has the ability to mint new tokens, provided the role is not paused, within limits defined by the `MintConfig` (defined by the admin who authorized the minter).

User

The role has the ability to transfer their own tokens, provided their address is not frozen.

5. Assessment Results

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

During our assessment on the scoped AUSD modules, we discovered one finding, which was informational in nature.

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.