Zellic

Prepared for
Scallop Team
Scallop Labs

Prepared by
Junghoon Cho
Daniel Lu
Zellic

March 27, 2024

Scallop

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 Scallop Labs from March 27th to April 19th, 2024. During this engagement, Zellic reviewed Scallop'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 issues in protocol math or logic that lead to loss of funds?
- Do the changes made after previous audits properly address the raised issues?
- Does the project have any centralization risks?

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. In particular, the unfinished nature of the flash-loan discount tickets prevented us from fully assessing its impact on the project.

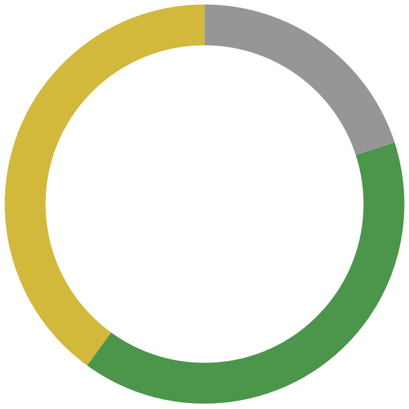
1.4. Results

During our assessment on the scoped Scallop modules, we discovered five findings. No critical issues were found. Two findings were of medium impact, two were of low impact, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Scallop Labs'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	2
🟩 Low	2
⬜ Informational	1



2. Introduction

2.1. About Scallop

Scallop Labs contributed the following description of Scallop:

> Scallop is the pioneering Next Generation peer-to-peer Money Market for the Sui ecosystem.

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.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped modules 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:

Scallop Modules

Repository	https://github.com/scallop-io/sui-lending-protocol ↗
Version	sui-lending-protocol: `a0c2ffc9eadeae086df2da311ac358db92c22c1f`
Programs	• libs/coin_decimals_registry/sources/*.move • libs/whitelist/sources/*.move • libs/x/sources/*.move • protocol/sources/*.move • sui_x_oracle/pyth_rule/*.move • sui_x_oracle/supra_rule/*.move • sui_x_oracle/switchboard_rule/sources/*.move • sui_x_oracle/x_oracle/sources/*.move
Type	Move
Platform	Sui

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of 5.5 person-weeks. The assessment was conducted over the course of 3.5 calendar weeks.

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:

Junghoon Cho
Engineer
junghoon@zellic.io ↗

Daniel Lu
Engineer
daniel@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

March 27, 2024	Start of primary review period
April 19, 2024	End of primary review period

3. Detailed Findings

3.1. Obligation-key management allows admin to block liquidations

Target	protocol::lock_obligation		
Category	Protocol Risks	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The protocol admin has the ability to manage obligation keys. With a key of a valid type, the owner of an obligation can lock specific functionality, including liquidations. When this happens, the key is dropped and its type name is saved in the obligation.

Since locked and undercollateralized obligations would otherwise be an issue, the protocol provides a `force_unlock_unhealthy` function that allows anyone — not just the obligation owner — to unlock obligations when they are not sufficiently collateralized.

```
public fun force_unlock_unhealthy<T: drop>(
    obligation: &mut Obligation,
    market: &mut Market,
    coin_decimals_registry: &CoinDecimalsRegistry,
    x_oracle: &XOracle,
    clock: &Clock,
    key: T
)
```

This function checks that `key` is of the same type as the witness originally used to lock the obligation. It is dropped.

Impact

The locking mechanics pose some centralization risk because the protocol admin is able to create obligations that cannot be unlocked — even when unhealthy. One way to accomplish this would be to add a lock key (an admin ability) that only they can create. Then, if they lock obligations that they create, those obligations have no way of being liquidated.

By creating unhealthy obligations, the admin may be able to drain the protocol.

Recommendations

We recommend ensuring that unhealthy positions can always be liquidated.

Remediation

Scallop Labs acknowledged this risk, explaining that issuing allowlisted witnesses is the responsibility of extensions under the protocol's own control.

3.2. Whitelist functionality creates protocol risk

Target	user/		
Category	Protocol Risks	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The protocol admin has the ability to arbitrarily add and remove addresses from the whitelist. This whitelist controls nearly all user interactions with the market; this includes deposits, withdrawals, flash loans, minting, redeeming, liquidating, and so forth.

This means that users can be blocked from any action, including exiting the protocol.

Impact

First, this allows a malicious admin to prevent liquidations and possibly drain the protocol with their own bad obligations — as described in the previous finding, this poses some centralization risk and requires extra trust from the user.

Additionally, the protocol contains optional functionality for putting important parameter updates behind time locks. This breaks that mitigation because it would allow a malicious admin to prevent users from actually reacting to such changes during the time-lock period.

Recommendations

We recommend only checking the whitelist on select functionality, such as taking flash loans, borrowing, and minting.

Remediation

Scallop Labs acknowledged this risk, explaining that it is no longer in use and should be limited. Whether the feature will be entirely removed or retained for specific actions is under disussion.

3.3. Insufficient controls on parameter changes

Target	protocol::app, protocol::interest_model, protocol::limiter, protocol::risk_model		
Category	Protocol Risks	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The admin has the ability to update a number of protocol parameters. For example, each asset that can be borrowed has an interest model. Each asset that can be used as collateral has a risk model. Other parameters include fees for borrowing, fees for taking flash loans, and so on.

If the admin were able to arbitrarily change many of these parameters, it would both require unnecessary trust from users and create additional risk in the case of key compromise. Hence, the protocol does include some functionality for mitigating this issue; namely, many parameters have a two-step update process that can enforce delays on updates. These delays can be lengthened (but not decreased) by some admin functions, like `extend_interest_model_change_delay`.

One problem is that some essential parameters do not have proper bounds checks. For example, the interest model of assets includes a `revenue_factor` parameter that decides what proportion of interest accrued is taken out of the protocol as fees. But there are no checks that this parameter is below one. Although this parameter could be gated by the time locks mentioned above, the required delay for updates begins at zero.

Impact

Depending on the current protocol state — such as whether the parameter change delay has been set — the admin may be able to drain protocol funds without warning. Even when these delays are in place, the lack of bounds can result in accidental changes to protocol parameters that result in economic issues.

Recommendations

We recommend adding a nonzero lower limit for parameter change delays to make sure users have time to react to changes. Additionally, we recommend adding assertions to constrain parameter ranges when necessary.

Remediation

Scallop Labs acknowledged that assertions should be added, and explained that the time lock delay is currently set to zero in case of urgent situations. They plan to incorporate the delay when they have more confidence in the stability of the protocol.

3.4. Ticket issuing is broken

Target	protocol::ticket_issuer_policy		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

The project allows users to exercise tickets that provide discounts on flash-loan and borrowing fees. However, the functionality in its current state does not work.

For example, tickets issued need to be in accord with the ticket-issuer policy, which dictates permissions for creating these discounts. But the object that holds this data is never created or shared, and the functionality has not been integrated into the protocol.

Additionally, the logic for adding a witness type in `protocol::ticket_issuer_policy` is as follows.

```
public(friend) fun add_witness_type<Ticket, Witness: drop>(
    ticket_issuer_policy: &mut TicketIssuerPolicy,
) {
    let ticket_type = type_name::get<Ticket>();
    let witness_type = type_name::get<Witness>();

    if (table::contains(&ticket_issuer_policy.witness_types, ticket_type)) {
        table::add(&mut ticket_issuer_policy.witness_types, ticket_type,
    vec_set::singleton(witness_type));
    } else {
        let sets = table::borrow_mut(&mut ticket_issuer_policy.witness_types,
    ticket_type);
        vec_set::insert(sets, witness_type);
    };
}
```

The branch on `table::contains` is flipped.

Impact

The ticketing functionality does not work.

Recommendations

We recommend adding the necessary functionality and flipping the branch in `add_witness_type`.

Remediation

Scallop Labs explained that this feature is still under development, and has not been deployed. It has been replaced by a simpler referral system for borrowing fee discounts.

3.5. Missing bounds on parameter-update delays

Target	protocol::app		
Category	Business Logic	**Severity**	Informational
Likelihood	Low	**Impact**	Informational

Description

The protocol includes functionality to require delays for updating some important parameters. For example, the following increases the delay for interest-model updates.

```
public fun extend_interest_model_change_delay(
    admin_cap: &mut AdminCap,
delay: u64,
) {
    admin_cap.interest_model_change_delay =
    admin_cap.interest_model_change_delay + delay;
}
```

However, the protocol does not provide a way to decrease these delays.

Impact

This is risky because there are no upper bounds on these delays. If these are accidentally set too high — or if intentional delay increases turn out to be too high — it could brick this functionality or prevent the protocol admin from reacting to important changes.

Recommendations

We recommend adding an upper bound to these delays or a way to safely decrease them.

Remediation

Scallop Labs acknowledged this risk and plans to add an upper bound of 24 hours.

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. Risk management

The current version of the protocol supports a single shared market, allowing users to deposit collateral and borrow any asset type. This structure allows a problem with one asset to jeopardize all the funds in the protocol. To mitigate this risk, we would recommend supporting multiple pools that accept different collections of assets to be borrowed and used as collateral.

Scallop Labs explained that this change is already planned. We commend Scallop Labs for their proactive approach to risk management.

4.2. Prevention of depositing borrowed coins is fragile

In `user/deposit_collateral.move`, the `deposit_collateral` function includes a check that the obligation does not have the coin borrowed. Symmetrically, it includes a check while borrowing that the coin is not deposited as collateral.

It is worth noting that this check is somewhat weak because it does not prevent users from effectively taking the same position across different obligations.

4.3. Limitations of the limiter

The protocol::limiter module functions to mitigate security or economic issues by limiting the amount of funds that can exit the protocol in a given period. The limiter for each asset type is parameterized by

1. the total duration of an outflow cycle (currently documented as 24 hours);
2. the duration of a *segment* within that cycle; and
3. the amount of the asset that can be borrowed within the cycle.

Then, the protocol maintains a rolling window of segments. Borrowing increases the outflow within a segment, while liquidations and repayments decrease the outflow within a segment. If the previous cycle contains more than the permitted outflow, then borrowing is blocked.

Although this system may mitigate some attacks, there are ways to reduce its effectiveness. For example, an attacker could gradually borrow funds from the protocol *ahead of time*. Then, if they have a bug to exploit, they could perform the bad borrows in a single segment while gradually repaying the previously borrowed funds to keep their outflow low. This brings the debt in their previously opened positions back down, so they can exit them normally at a later date.

Scallop Labs explained that deducting outflow after repayment is somewhat necessary to prevent denial-of-service attacks by repeated borrowing and repaying. They acknowledged that the above approach could circumvent the limiter to some degree.

4.4. Code quality and module design

Although the protocol code is good quality, there are some modules written in a way that introduces more maintenance risk. For example, the `ac_table` and `wit_table` libraries are used in a number of modules to handle access control for some stored data. These include optional functionality for maintaining a set of keys. In the case where the consumer opts out of that functionality, the `keys` API (that no longer makes sense) fails silently.

```
public fun keys<T: drop, K: copy + drop + store, V: store>(
  self: &AcTable<T, K, V>,
): vector<K> {
  if (self.with_keys) {
    let keys = option::borrow(&self.keys);
    vec_set::into_keys(*keys)
  } else {
    vector::empty()
  }
}
```

We recommend structuring these modules in a way that makes failures more apparent. This can be done by wrapping the modules with a new one that manages keys, which would let incorrect usage be caught statically. Alternatively, this `keys` function could revert when called on a table that does not track keys. It is not unreasonable to expect some mistakes with the `with_keys` parameter; for example, the asset_active_state module enables the functionality but never uses it.

Additionally, some modules in `evaluator/` have quite a bit of duplication. Namely, protocol::collateral_value and protocol::debt_value each have two functions that at a high level reuse the same overall logic: iterating over collateral types and performing some accounting on their values. Abstracting this behavior out could make the protocol easier to maintain.

4.5. Test coverage

Overall, the quality of the code is good, but the codebase would benefit from increased test coverage. Currently, only a few of the libraries have unit tests. There are also some test cases for specific protocol use cases and scenarios.

These end-to-end tests are good to see, and we would strongly encourage Scallop to include more unit tests for important protocol mechanics. Another good target for improving test coverage would be the utility modules used throughout different protocol components, such as `x::supply_bag`.

Scallop Labs explained that they are in the process of adding more unit tests and end-to-end tests, with the goal of covering every part of the protocol mechanics.

5. Assessment Results

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

During our assessment on the scoped Scallop modules, we discovered five findings. No critical issues were found. Two findings were of medium impact, two were of low impact, and the remaining finding 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.