Zellic

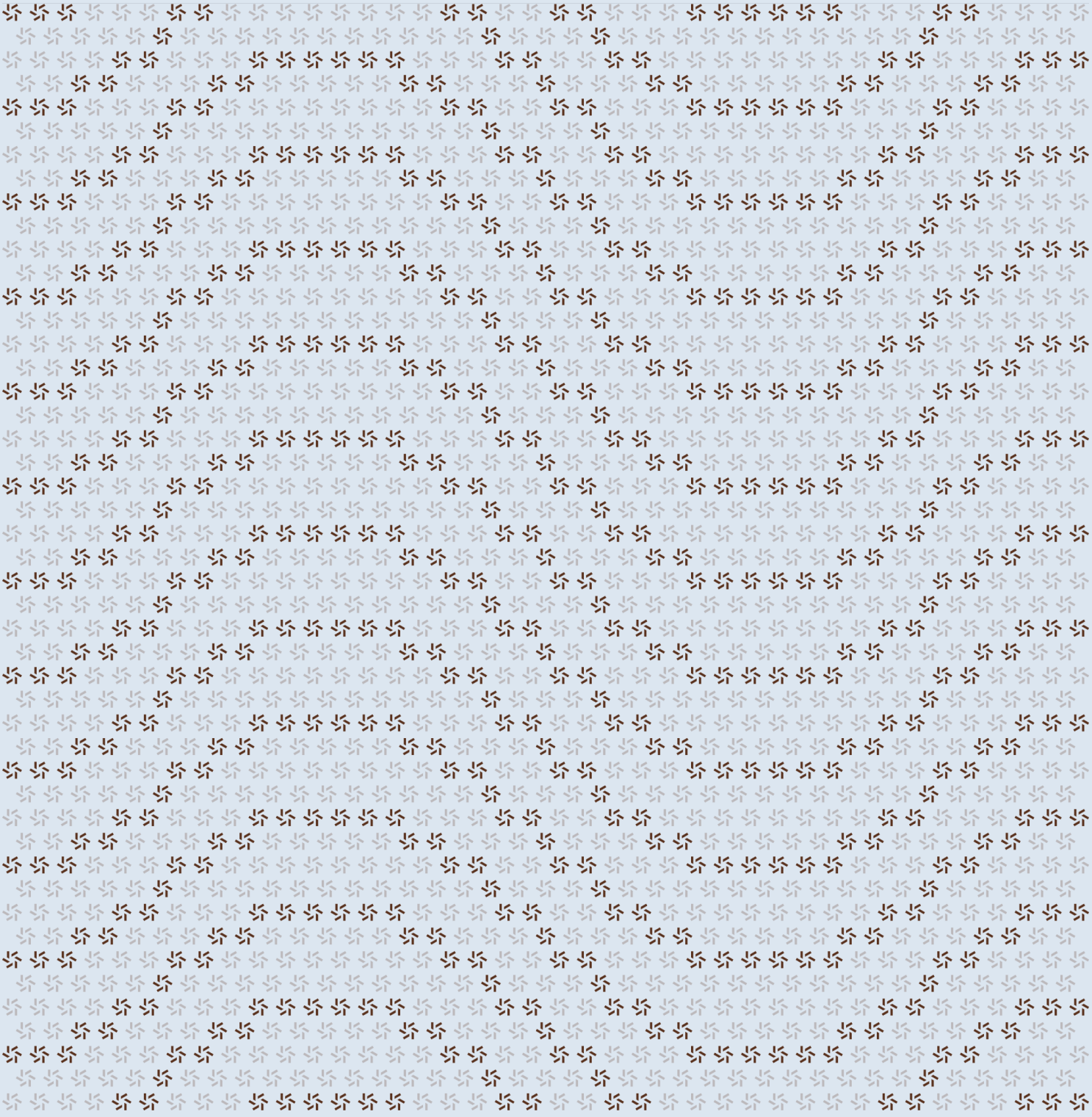
Prepared for
Echelon

Prepared by
Junghoon Cho
Aaron Esau
Zellic

July 1, 2024

Echelon

Smart Contract Security Assessment



Contents

About Zellic **4**

1. Overview **4**

1.1. Executive Summary 5

1.2. Goals of the Assessment 5

1.3. Non-goals and Limitations 5

1.4. Results 5

2. Introduction **6**

2.1. About Echelon 7

2.2. Methodology 7

2.3. Scope 9

2.4. Project Overview 9

2.5. Project Timeline 10

3. Detailed Findings **10**

3.1. Improper bounds on collateral and liquidation base points 11

3.2. Zero close factor allows admin to block liquidations 13

3.3. Position can be in shortfall after removing collateral 14

4. Threat Model **14**

4.1. Supply 15

4.2. Withdraw 15

4.3. Borrow 15

4.4. Repay 15

4.5. Liquidate 15

4.6. Adding collateral 16

4.7. Removing collateral 16

5. Assessment Results 16

5.1. Disclaimer 17

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 Echelon from June 24th to July 1st, 2024. During this engagement, Zellic reviewed Echelon'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:

- Does Echelon have any centralization risks?
- Are there issues in protocol math or logic that lead to loss of funds?
- Are there any concerns relating to rounding? Do all math operations that lose precision round in the direction to benefit the protocol instead of the user?
- Is Echelon secure against common smart contract vulnerabilities?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Thala oracle
- Front-end components
- Infrastructure relating to the project
- Key custody

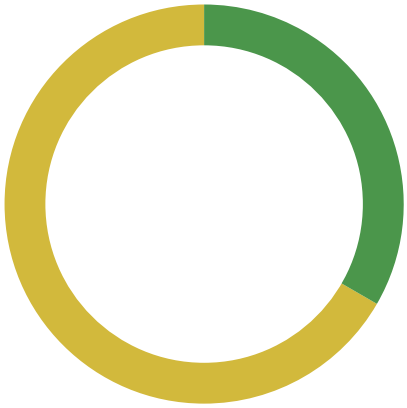
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Echelon modules, we discovered three findings. No critical issues were found. Two findings were of medium impact and one was of low impact.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	2
Low	1
Informational	0



2. Introduction

2.1. About Echelon

Echelon contributed the following description of Echelon:

> Echelon is a high-efficiency move money market for the permissionless borrowing and lending of various crypto and real world assets, allowing users to gain interest or leverage.

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:

Echelon Modules

Type	Move
Platform	Aptos
Target	echelon-modules
Repository	https://github.com/EchelonMarket/echelon-modules ↗
Version	`80e49b116bd9b500de118c12416e0a885b971067`
Programs	`lending_core/sources/isolated_lending.move`

2.4. Project Overview

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

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.

June 24, 2024	Kick-off call
June 24, 2024	Start of primary review period
July 1, 2024	End of primary review period

3. Detailed Findings

3.1. Improper bounds on collateral and liquidation base points

Target	isolated_lending		
Category	Protocol Risks	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The protocol admin has the ability to update a number of protocol parameters. However, some essential parameters do not have proper bounds checks, which leads to a centralization risk making the protocol require unnecessary trust from users.

- The `liquidation_incentive_bps` determines the multiplier applied to the repaid liability to calculate the collateral amount seized. The admin can modify this parameter via the `set_liquidation_incentive_bps` function.

```
public entry fun set_liquidation_incentive_bps(manager: &signer,
    liquidation_incentive_bps: u64) acquires IsolatedLending, Pair {
    assert!(manager::is_manager(manager),
    ERR_ISOLATED_LENDING_UNAUTHORIZED);
    assert!(liquidation_incentive_bps >= BPS_BASE,
    ERR_ISOLATED_LENDING_INVALID_LIQUIDATION_INCENTIVE_BPS);
```

- This function does not set an upper bound on `liquidation_incentive_bps`, allowing it to be set to a very high value up to the maximum value of u64 (18446744073709551615).

- The `collateral_factor_bps` represents the percentage of the collateral value that can be borrowed against per asset pair. The admin can modify this parameter of the specific asset pair via `set_pair_collateral_factor_bps`.

```
public entry fun set_pair_collateral_factor_bps(manager: &signer,
    pair_obj: Object<Pair>, collateral_factor_bps: u64)
    acquires IsolatedLending, Pair {
    assert!(manager::is_manager(manager),
    ERR_ISOLATED_LENDING_UNAUTHORIZED);
    assert!(collateral_factor_bps <= BPS_BASE,
    ERR_ISOLATED_LENDING_INVALID_COLLATERAL_FACTOR_BPS);
```

- This function does not set a lower bound on `collateral_factor_bps`, allowing it to be set to a very low value, even zero.

- Both `set_liquidation_incentive_bps` and `set_pair_collateral_factor_bps` check whether it is guaranteed that there is enough collateral available to cover the liquidation bonus at the moment of liquidation, ensuring that the newly set `liquida-`

`tion_incentive_bps` / `collateral_factor_bps` is valid.

```
inline fun
    validate_liquidation_collateral_coverage(collateral_factor_bps:
    u64, liquidation_incentive_bps: u64) {
    assert!(collateral_factor_bps *
    liquidation_incentive_bps <= BPS_BASE * BPS_BASE,
    ERR_ISOLATED_LENDING_INSUFFICIENT_COLLATERAL_COVERAGE
    _FOR_LIQUIDATION);
}
```

This check is performed by calling `validate_liquidation_collateral_coverage`, which can be bypassed by setting `collateral_factor_bps` to zero.

Impact

The protocol admin may take the entire collateral from the borrower by setting `collateral_factor_bps` to zero and giving an extremely high value to `liquidation_incentive_bps`, then liquidating the positions but repaying only a very small portion of the debt.

Recommendations

Add proper upper/lower bounds to `liquidation_incentive_bps` and `collateral_factor_bps`. Additionally, consider using a time lock or a governance process to update these parameters.

Remediation

This issue has been acknowledged by Echelon, and a fix was implemented in commit `f7ff1ced`↗.

3.2. Zero close factor allows admin to block liquidations

Target	isolated_lending		
Category	Protocol Risks	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The `close_factor_bps` represents the percentage of a liquidatable account's liability that can be repaid in a single liquidation transaction, which is applied globally to every asset pair.

```
public entry fun set_close_factor_bps(manager: &signer, close_factor_bps: u64)
    acquires IsolatedLending {
    assert!(manager::is_manager(manager), ERR_ISOLATED_LENDING_UNAUTHORIZED);
    assert!(close_factor_bps <= BPS_BASE,
    ERR_ISOLATED_LENDING_INVALID_CLOSE_FACTOR_BPS);

    // update close_factor_bps
    let isolated_lending =
    borrow_global_mut<IsolatedLending>(package::package_address());
    isolated_lending.close_factor_bps = close_factor_bps;
}
```

The protocol admin can modify this parameter via the `set_close_factor_bps` function, but the function does not check if the value of `close_factor_bps` is nonzero.

Impact

The protocol admin can prevent all positions from being liquidated.

Recommendations

Add a check to ensure the value of `close_factor_bps` is nonzero.

Remediation

Echelon acknowledged that the mechanism can be abused to block liquidation, and explained the current bound of the close factor will be kept to allow temporary blocking of liquidation in case a technical issue is discovered.

3.3. Position can be in shortfall after removing collateral

Target	isolated_lending		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The `remove_collateral_internal` function decides to check if the position is in shortfall after removing collateral depending on whether the `checkShortfall` argument is true or false:

```
else if (checkShortfall) {
    // check account position not shortfall
    // if caller is `liquidate_internal`, we don't check liquidatee's status
    assert!(!is_shortfall_internal(account_addr, pair_obj),
    ERR_ISOLATED_LENDING_SHORTFALL);
};
```

This option exists to skip the shortfall check when calling `remove_collateral_internal` to seize collateral from a position being liquidated in the `liquidate_internal` function. However, the `remove_collateral` and `remove_collateral_fa` functions also call `remove_collateral_internal` with the `checkShortfall` argument set to false, which is not in line with this intention.

Impact

After a user removes collateral, their position can be liquidated. This allows the user to intentionally create a shortfall position, leading to unintended situations.

Recommendations

Pass `checkShortfall` as true when calling `remove_collateral_internal` from `remove_collateral` or `remove_collateral_fa`.

Remediation

Echelon explained that this issue was intentionally included in the codebase; however, we have no way of verifying this.

4. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the modules and created a written threat model for some critical functions. A threat model documents ways an attacker may approach breaking a given function.

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. Supply

- ☑ It must assert that the pair exists.
- ☑ It must ensure the types passed in match the specified pair.
- ☑ Pair must not be paused.
- ☑ It must not exceed the supply cap when supplying `amount`.

4.2. Withdraw

- ☑ It must assert that the pair exists.
- ☑ It must ensure the types passed in match the specified pair.
- ☑ Pair must not be paused.
- ☑ Borrower must have enough borrowing shares to support the withdrawal amount.

4.3. Borrow

- ☑ It must assert that the pair exists.
- ☑ It must ensure the types passed in match the specified pair.
- ☑ It must not exceed the borrow cap when borrowing `amount`.
- ☑ Pair must have enough liquidity to support the borrowed amount.
- ☑ Borrower account must be healthy after the borrow operation (not in shortfall).

4.4. Repay

- ☑ It must assert that the pair exists.
- ☑ It must ensure the types passed in match the specified pair.
- ☑ Caller-specified amount of `shares` must be less than or equal to the borrowed number of shares.

4.5. Liquidate

- ☑ It must assert that the pair exists.
- ☑ It must ensure the types passed in match the specified pair.

- ☑ Pair must not be paused.
- ☑ Liquidator signer must not also be the borrower being liquidated.
- ☑ Borrower account must be unhealthy (in shortfall).
- ☑ Caller-specified `repay_shares` must be less than or equal to the closing shares.
- ☑ Seized amount must be greater than the `min_amount_out`.

4.6. Adding collateral

- ☑ It must assert that the pair exists.
- ☑ It must ensure the types passed in match the specified pair.
- ☑ Pair must not be paused.

4.7. Removing collateral

- ☑ It must assert that the pair exists.
- ☑ It must ensure the types passed in match the specified pair.
- ☑ Pair must not be paused.
- ☑ Amount requested must be less than the signer's deposited collateral amount.
- ☐ After removing collateral, the account must remain healthy (not in shortfall). See Finding 3.3. ↗ for details about this missing check.

5. Assessment Results

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

During our assessment on the scoped Echelon modules, we discovered three findings. No critical issues were found. Two findings were of medium impact and one was of low impact.

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.