Zellic

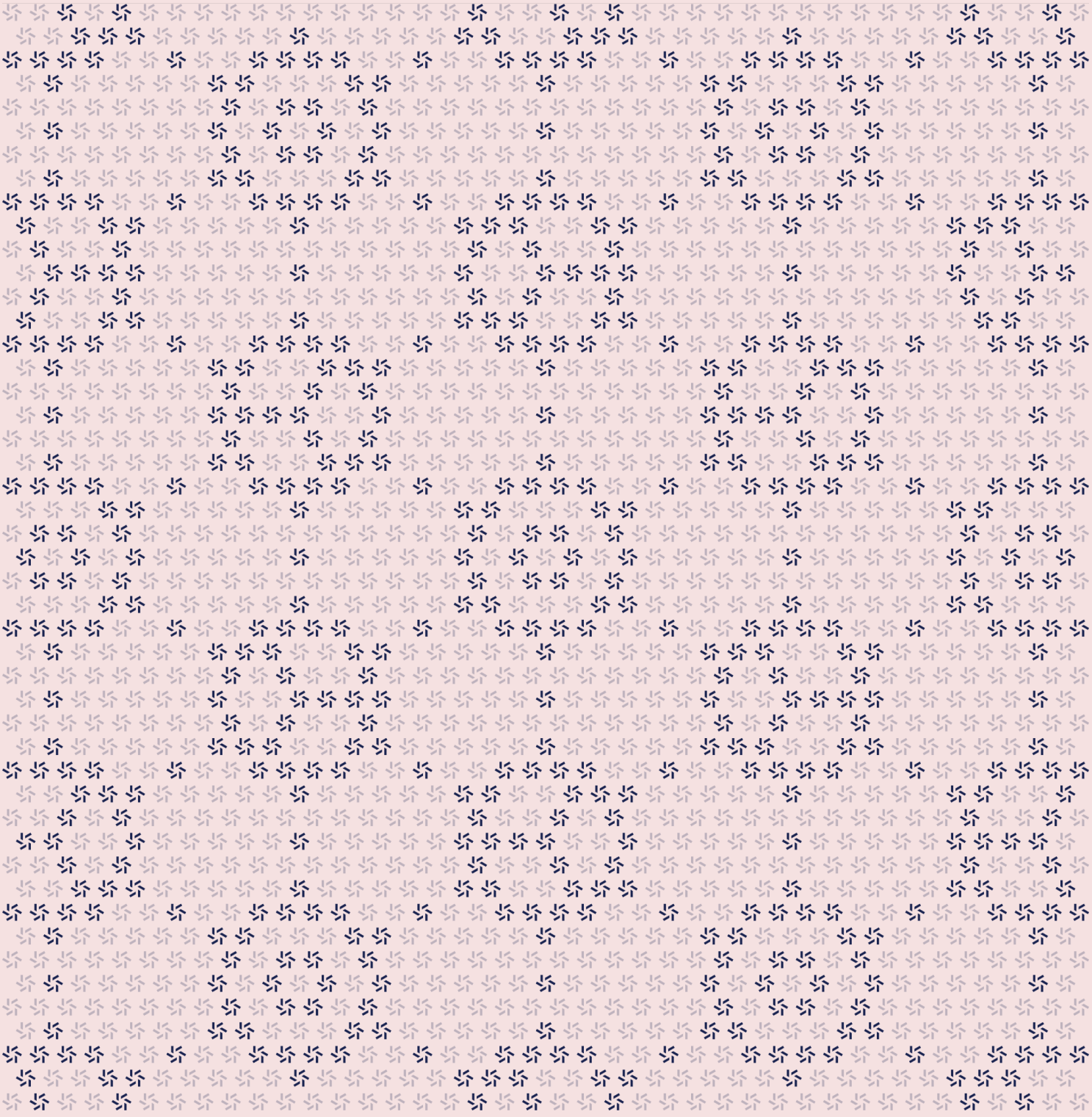
Prepared for
badconfig.eth
Cytonic Network

Prepared by
Frank Bachman
Aaron Esau
Zellic

July 31, 2024

Solana & EVM Depositors

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 Cytonic Network from July 22nd to July 24th, 2024. During this engagement, Zellic reviewed Solana & EVM Depositors'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:

- Is it possible for an attacker to drain the contract of funds?
- Can funds be locked in the contract?
- Is the accounting sound?

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
- Backend functionality for airdrops and user rewards

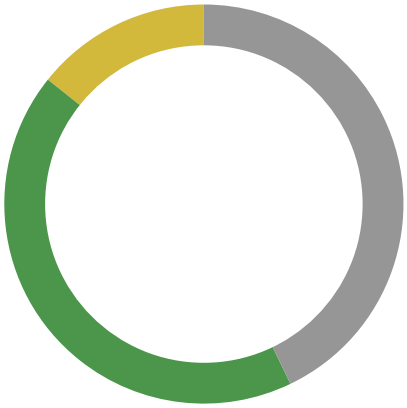
1.4. Results

During our assessment on the scoped Solana & EVM Depositors contracts, we discovered seven findings. No critical issues were found. One finding was of medium impact, three were of low impact, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Cytonic Network's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	1
Low	3
Informational	3



2. Introduction

2.1. About Solana & EVM Depositors

Cytonic Network contributed the following description of Solana & EVM Depositors:

> Cytonic Bridge V1 bootstraps liquidity for the cytonic ecosystem. The bridge will eventually evolve to allow freely bridging assets between Cytonic and other blockchains.

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:

Solana & EVM Depositors Contracts

Types	Solidity, Solana
Platforms	EVM-compatible, Solana
Target	cytonic-bridge-evm
Repository	https://github.com/cytonic-network/cytonic-bridge-evm ↗
Version	`5a45f51b09f91444d01b0e1c23fd3b9016170702`
Programs	`src/Depositor.sol`
Target	cytonic-bridge-solana
Repository	https://github.com/cytonic-network/cytonic-bridge-solana ↗
Version	`3553735e96ae29528870f1c0fc4c3354b80f8a63`
Programs	`programs/depositor/**`

2.4. Project Overview

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

Frank Bachman
Engineer
frank@zellic.io ↗

Aaron Esau
Engineer
aaron@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

July 22, 2024	Kick-off call
July 22, 2024	Start of primary review period
July 24, 2024	End of primary review period

3. Detailed Findings

3.1. DOS in claim/migrate due to unenforced ATA usage in deposit

Target	Depositor		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

It was found that in `Deposit` and moreover throughout the Depositor program, the usage of Associated Token Accounts (ATAs) is not enforced for the vault token account.

```
#[account(mut,
  constraint =
     receiver_token_account.owner == vault_data.key() &&
     receiver_token_account.mint == vault_data.mint,
 )]
pub receiver_token_account: Box<Account<'info, TokenAccount>>,

#[account(mut,
    seeds = [ b"vault-data".as_ref(), &vault_data.owner.to_bytes(),
    &vault_data.mint.to_bytes() ],
    bump = vault_data.bump,
)]
pub vault_data: Box<Account<'info, VaultData>>,
```

The Solana token program allows users to arbitrarily create many token accounts belonging to the same mint. Note that anyone can create these accounts for a specified owner.

It is therefore possible for a caller to create a separate vault token account to deposit the funds.

Impact

The amount deposited by the user is transferred to the newly created token account, not the vault ATA. While the ATA is in fact still owned by `vault_data`, the claims/withdrawals will be processed through the ATA. This could result in withdrawals failing and manual intervention being required to process the withdrawals.

Moreover, this completely breaks the migrate function, which uses the total amount from `vault_data`.

```
impl Migrate<'_> {
    pub fn actuate(ctx: &mut Context<Self>, _params: &MigrateParams) ->
    Result<()> {
        spl_transfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                SplTransfer {
                    authority: ctx.accounts.vault_data.to_account_info(),
                    from: ctx.accounts.vault_token_account.to_account_info(),
                    to: ctx.accounts.sender_token_account.to_account_info
[...]
            ctx.accounts.vault_data.total_deposited,
```

The SPL transfer will fail here since the amount is divided into multiple token accounts. However, the total deposited amount is used for transfer.

Recommendations

It is recommended to enforce the usage of ATAs for the vault accounts throughout the Depositor program.

Moreover, in the migrate function, the token-account balance should be used instead of the vault-deposited amount. This helps avoid failures in transfer in case of any inconsistencies between `vault_data` and the vault token account.

Remediation

This issue has been acknowledged by Cytonic Network, and fixes were implemented in the following commits:

- 636adc24 ↗
- 818a3e5c ↗

3.2. Edge case when contract is paused after a withdrawal

Target	Depositor		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

When a user withdraws, the code immediately removes funds from the mapping (i.e., it unaccounts for the funds immediately):

```
function _withdraw(address asset, uint256 amount, address user) internal {
    if (amount == 0) revert ZeroWithdraw();
    totalDeposited[address(asset)] -= amount;
    usersBalances[address(asset)][user] -= amount;
}
```

However, if the owner then pauses the contract before the user has the opportunity to claim the withdrawal, the funds remain unaccounted for in the mappings.

The user can also no longer claim their withdrawal because the `claim` function has the `whenNotPaused` modifier.

Impact

The impact depends on how the off-chain software interacts with the contract. If the off-chain software is not aware of pending withdrawals when the contract is paused, the off-chain software may not account for the funds that are still in the contract that are unable to be claimed; that is, at worst, the user will be at a loss, and funds will remain in the contract unless migrated.

Recommendations

Ensure the off-chain software tracks `Claim` events rather than `Withdraw` events, or allow the `claim` function to be called even when the contract is paused.

Remediation

Cytonic Network provided the following explanation in relation to this issue:

As with the recommendation, backend also tracks pending withdrawals and will determine their migration logic if they won't be claimed from the contract in due time.

3.3. Unpausing after migration causes insolvency

Target	Depositor		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `migrate` function — which may only be called when the contract is paused — transfers funds without updating the `totalDeposited` and `userBalances` mappings:

```
function migrate(address asset, address to) external onlyOwner whenPaused {
    if (asset == address(0)) {
        payable(to).transfer(address(this).balance);
    } else {
        IERC20(asset).safeTransfer(msg.sender,
    IERC20(asset).balanceOf(address(this)));
    }
}
```

Impact

Note that unpausing after migrating funds will cause the contract to be insolvent, as users may still attempt to withdraw their funds.

Recommendations

Though it is likely not worth the added complexity, it would be ideal to have a separate pausing mechanism for migration that permanently prevents depositing and withdrawing.

Remediation

Cytonic Network provided the following explanation in relation to this issue:

> Creating extra pausing mechanics truthfully don't worth it as there are still may ways of bringing contract itself into inconsistency e.g point 3.2, however after migration process starts, it won't be an issue as the contract won't be more responsible for fund keeping at all.

3.4. Payable `receive` function missing accounting

Target	Depositor		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `receive` function is marked as `payable`, which allows Ether to be transferred into the contract using external calls with empty calldata:

```
receive() external payable whenNotPaused {}
```

However, the function lacks accounting, leaving Ether in the contract without emitting the `Deposit` event and recording the deposit in the mapping.

Impact

The contract will accept Ether without recording the deposit, which may lead to confusion and loss of funds.

Recommendations

Call `_deposit` and emit a `Deposit` event in the `receive` function using `msg.sender` as the depositor.

Remediation

Cytonic Network provided the following explanation in relation to this issue:

> Due to the fact that it's prototyped in a way users need to use `depositETH` and specify referral code, we've decided to simply add revert functionality to ban the ability of row funds receiving from the contract. If a user will not use our frontend (with depositETH) and manually will transfer eth on contract, he will receive failure and finally will deposit through our app what will allow him to utilise airdrop mechanics

A change mitigating this finding was made in commit `d8b9143b` ↗.

3.5. The `_deposit` function emits the wrong event

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

Description

The `_deposit` function emits the `ZeroWithdrawal()` event instead of `ZeroDeposit()`:

```
function _deposit(address asset, uint256 amount, address user) internal {
    if (amount == 0) revert ZeroWithdraw();
    if (!allowedAssets[asset]) revert AssetNotAllowed();
    totalDeposited[address(asset)] += amount;
    usersBalances[address(asset)][user] += amount;
}
```

Impact

The failure event is misleading and does not reflect the actual operation that occurred.

Recommendations

Replace the `ZeroWithdrawal()` event with `ZeroDeposit()`.

Remediation

This issue has been acknowledged by Cytonic Network, and a fix was implemented in commit a3acea04 ↗.

3.6. The `cancelWithdrawal` function emits an event with incorrect value

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

Description

The `cancelWithdrawal` function emits a `Claim` event:

```
function cancelWithdraw(uint128 id) external whenNotPaused {
    ClaimData memory claimOrder = usersClaims[id];
    if (claimOrder.user != msg.sender) revert InvalidClaimAuthority();

    _deposit(claimOrder.asset, claimOrder.amount, claimOrder.user);

    delete usersClaims[id];
    emit Claim(id, false, block.timestamp);
    emit Deposit(claimOrder.asset, claimOrder.user, "", claimOrder.amount,
    block.timestamp);
}
```

The `Claim` event is used to notify off-chain systems that a withdrawal ticket has been used — whether claimed or canceled — where the `canceled` boolean indicates how the ticket was used:

```
/// @notice Claim event
/// @param id unique id for a claim
/// @param canceled if true - this claim is a cancelation and if false - user
    retreived funds
/// @param ts block timestamp
event Claim(uint256 id, bool canceled, uint256 ts);
```

The `cancelWithdrawal` function mistakenly passes `false` as the `canceled` parameter.

Impact

The `Claim` event emitted may be misleading to off-chain systems that rely on it to track the status of withdrawal tickets.

Recommendations

Replace the `false` value with `true` in the `Claim` event emitted by the `cancelWithdrawal` function.

Remediation

This issue has been acknowledged by Cytonic Network, and a fix was implemented in commit `b2a5e5f5` ↗.

3.7. Integer underflow in `_withdraw`

Target	Depositor		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `_withdraw` function does not ensure that the `userBalances` entry for the user and asset is sufficient to cover the withdrawal amount:

```
function _withdraw(address asset, uint256 amount, address user) internal {
    if (amount == 0) revert ZeroWithdraw();
    totalDeposited[address(asset)] -= amount;
    usersBalances[address(asset)][user] -= amount;
}
```

Impact

A user requesting a withdrawal that is too large will be confronted with an underflow reversion as opposed to an intuitive reversion reason.

Recommendations

Consider requiring that the withdrawal amount is less than or equal to the user's balance before attempting the subtraction.

Remediation

This issue has been acknowledged by Cytonic Network, and a fix was implemented in commit `3e9d24e5` ↗.

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. Warning regarding the contract owner's abilities

Note that the contract owner has the ability to pause the contract and withdraw funds at any time without any obligations to users enforced in the contract.

Cytonic Network stated that this functionality is only meant to be used in adverse scenarios (e.g., smart contract hacks) and the owner will be a multi-sig. While this does not fully mitigate the centralization risks, it reduces the impact of a single key being compromised.

We recommended that these centralization risks be clearly documented for users so that they are aware of the extent of the owner's control over the platform. This can help users make informed decisions about their participation in the project. Additionally, clear communication about the circumstances in which the owner may exercise these powers can help build trust and transparency with users.

4.2. Anti-patterns identified in the Solana program

The following are certain anti-patterns that were identified within the Solana program during the audit. While these presently do not pose risk, future refactoring of the codebase may change this. It is therefore recommended to fix these anti-patterns to mitigate any future risk.

Size calculation for program accounts

The Depositor program uses the Anchor framework, which uses Borsh serialization for program accounts. It was found that `core::mem::size_of` was used to compute the size to be reserved for multiple program accounts.

```
#[account(
    init_if_needed,
    payer = sender,
    space = 8 + core::mem::size_of::<UserData>(),
    seeds = [ b"vault-user".as_ref(), &vault_data.key().to_bytes(),
    &sender.key().to_bytes() ],
    bump,
)]
pub user_data: Box<Account<'info, UserData>>,
```

This is currently not an issue because the types being used for the current program structures just happen to have the same sizes with Borsh and native Rust types. However, there are types for which Borsh sizing varies and therefore it is recommended to hardcode structure sizes accordingly. Please refer to *The Anchor Book* for more information.

Use of checked math

It was noticed that there were some inconsistencies in regards to the usage of checked math in the Depositor program.

```
ctx.accounts.vault_data.total_deposited += params.amount;
ctx.accounts.user_data.deposit_amount += params.amount;
```

While this is not an issue since the `overflow-checks` config is enabled, it is recommended to use checked math where integer overflows are security critical for clarity.

Unused accounts present in `Withdraw`

The `Withdraw` instruction requires the `sender_token_account` and `vault_token_account` accounts. However, these accounts are unused and can be removed.

Remediation

These issues have been acknowledged by Cytonic Network, and fixes were implemented in commits b6b5fb, 170db5 and 9d2ba1.

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet or Solana.

During our assessment on the scoped Solana & EVM Depositors contracts, we discovered seven findings. No critical issues were found. One finding was of medium impact, three were of low impact, and the remaining findings were 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.