Cykura V.2

Smart Contract Audit Report
Prepared for Cykura

Date Issued: Feb 28, 2022
Project ID: AUDIT2022001
Version: v2.0
Confidentiality Level: Public

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022001
Version	v2.0
Client	Cykura
Project	Cykura V.2
Auditor(s)	Patipon Suwanbol Ronnachai Chaipha
Author(s)	Ronnachai Chaipha
Reviewer	Weerawat Pawanawiwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Feb 25, 2022	Full report	Ronnachai Chaipha

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Cykura, Inspex team conducted an audit to verify the security posture of the Cykura V.2 program between Feb 8, 2022 and Feb 15, 2022. During the audit, Inspex team examined the Solana program and the overall operation within the scope to understand the overview of Cykura V.2 Solana program. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify Solana program vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 3 high, 1 medium, and 1 very low-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved or mitigated in the reassessment. Therefore, Inspex trusts that Cykura V.2 program has high-level protections in place to be safe from most attacks.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Cykura is the first concentrated liquidity market making platform built on the Solana blockchain. Cykura combines the capital efficiency of concentrated liquidity with the low-gas cost environment of Solana to offer a fully-optimized trading and liquidity bootstrapping environment.

Scope Information:

Project Name	Cykura V.2
Website	https://app.cykura.io/
Smart Contract Type	Solana Program
Chain	Solana
Programming Language	Rust

Audit Information:

Audit Method	Whitebox
Audit Date	Feb 8, 2022 - Feb 15, 2022
Reassessment Date	Feb 24, 2022

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 6879092f11a962f94a339c6549f6ee33b292e197)

Contract	Location (URL)
cyclos_core	https://github.com/cykura/protocol-v2/tree/6879092f11/programs/core/src

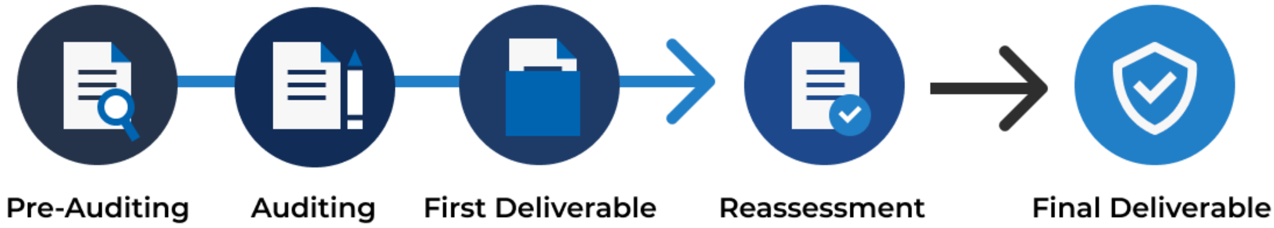
Reassessment: (Commit: 2da02766898f6be4ce72a7a726199331586adc9c)

Contract	Location (URL)
cyclos_core	https://github.com/cykura/protocol-v2/tree/2da0276689/programs/core/src

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Integer Overflows and Underflows
Bad Randomness
Use of Known Vulnerable Component
Use of Deprecated Component
Solana Account Confusions
Missing Rent Exemption Checking
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Implicit Type Inference
Function Declaration Inconsistency
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

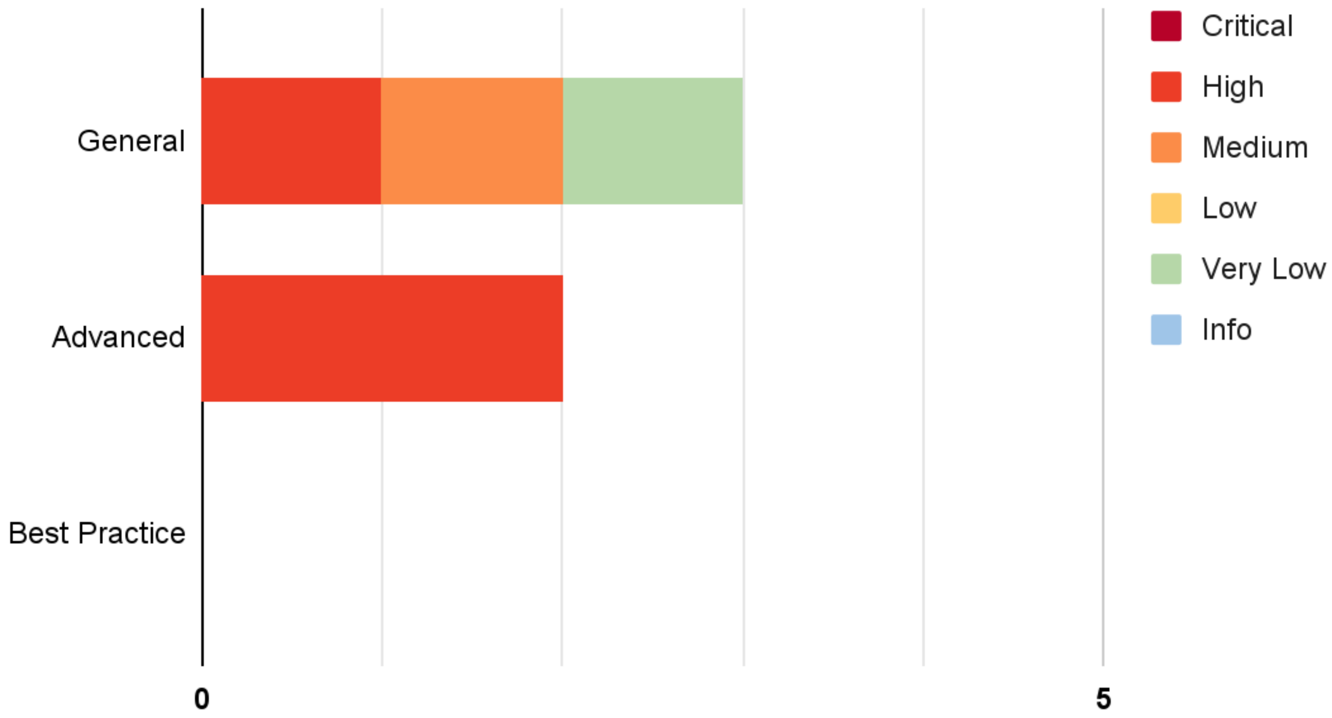
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Likelihood / Impact	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

4. Summary of Findings

From the assessments, Inspex has found 5 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Upgradability of Solana Program	General	High	Resolved *
IDX-002	Design Flaw in collect_protocol() Function	Advanced	High	Resolved
IDX-003	Design Flaw in mint() Function	Advanced	High	Resolved
IDX-004	Unusable collect_protocol() Function	General	Medium	Resolved
IDX-005	Use of Outdated Dependency	General	Low	Resolved

* The mitigations or clarifications by Cykura can be found in Chapter 5.

5. Detailed Findings Information

5.1. Upgradability of Solana Program

ID	IDX-001
Target	cyclos_core
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The logic of the affected programs can be arbitrarily changed. This allows the upgrade authority to change the logic of the program in favor of the platform, e.g., transferring the users' funds to the platform owner's account. **Likelihood: Medium** Only the program upgrade authority can redeploy the program to the same program address; however, there is no restriction to prevent the authority from inserting malicious logic.
Status	**Resolved *** Cykura team has confirmed to mitigate this issue by transferring the governance of the program to Tribeca DAO after it is fully released to the public. This includes the voting proposal mechanism with changes delayed by timelock mechanism.

5.1.1. Description

Programs on Solana can be deployed through the upgradable BPF loader to make them upgradable, allowing the program's upgrade authority to redeploy the program with the new logic, bug fixes, or upgrades to the same program address.

However, there is no restriction on how and when the program will be upgraded. This opens up an attack surface on the program, allowing the upgrade authority to redeploy the program with malicious logic and gain unfair benefits from the users, for example, transferring funds out from the users' accounts.

5.1.2. Remediation

Inspex suggests deploying the program as an immutable program to prevent the program logic from being modified. However, if the upgradability is needed, Inspex suggests mitigating this issue by the following options:

- Using a multisig account controlled by multiple trusted parties as the upgrade authority
- Implementing a community-run governance to control the redeployment of the program

5.2. Design Flaw in collect_protocol() Function

ID	IDX-002
Target	cyclos_core
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** High **Impact:** Medium According to the business design, the platform's owner will be the one who can collect the protocol fee. However, the protocol fee can be claimed by anyone instead, resulting in monetary loss for the platform. **Likelihood:** High It is likely that the attacker will perform this attack since there is no restriction to prevent it and the cost is very low compared to the profit.
Status	Resolved Cykura team has resolved this issue by adding a derive macro to verify the authorized caller as in commit `be249706d226968400ad579fff30a853be59e4c3`.

5.2.1 Description

In the `cyclos_core` program, according to the business design, the `collect_protocol()` function allows the platform's owner to collect the protocol accrued fee from the pools as represented in the code below.

protocol-v2/programs/core/src/lib.rs

```
/// Collect the protocol fee accrued to the pool
///
///
/// * ctx - Checks for valid owner by looking at signer and factory owner addresses.
/// Holds the Pool State account where accrued protocol fee is saved, and token accounts to perform
/// transfer.
/// * amount_0_requested - The maximum amount of token_0 to send, can be 0 to collect fees in only token_1
/// * amount_1_requested - The maximum amount of token_1 to send, can be 0 to collect fees in only token_0
///
pub fn collect_protocol(
    ctx: Context<CollectProtocol>,
    amount_0_requested: u64,
    amount_1_requested: u64,
```

```
) -> ProgramResult {
    let mut pool_state = ctx.accounts.pool_state.load_mut()?;
    require!(pool_state.unlocked, ErrorCode::LOK);
    pool_state.unlocked = false;

    let amount_0 = amount_0_requested.min(pool_state.protocol_fees_token_0);
    let amount_1 = amount_1_requested.min(pool_state.protocol_fees_token_1);

    let pool_state_seeds = [
        &POOL_SEED.as_bytes(),
        &pool_state.token_0.to_bytes() as &[u8],
        &pool_state.token_1.to_bytes() as &[u8],
        &pool_state.fee.to_be_bytes(),
        &[pool_state.bump],
    ];

    pool_state.protocol_fees_token_0 -= amount_0;
    pool_state.protocol_fees_token_1 -= amount_1;
    drop(pool_state);

    if amount_0 > 0 {
        token::transfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info().clone(),
                token::Transfer {
                    from: ctx.accounts.vault_0.to_account_info().clone(),
                    to:
ctx.accounts.recipient_wallet_0.to_account_info().clone(),
                    authority:
ctx.accounts.pool_state.to_account_info().clone(),
                },
                &[&pool_state_seeds[..]],
            ),
            amount_0,
        )?;
    }
    if amount_1 > 0 {
        token::transfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info().clone(),
                token::Transfer {
                    from: ctx.accounts.vault_1.to_account_info().clone(),
                    to:
ctx.accounts.recipient_wallet_1.to_account_info().clone(),
                    authority:
ctx.accounts.pool_state.to_account_info().clone(),
                },
```

```
                &[&pool_state_seeds[..]],
            ),
            amount_1,
        )?;
    }

    emit!(CollectProtocolEvent {
        pool_state: ctx.accounts.pool_state.key(),
        sender: ctx.accounts.owner.key(),
        recipient_wallet_0: ctx.accounts.recipient_wallet_0.key(),
        recipient_wallet_1: ctx.accounts.recipient_wallet_1.key(),
        amount_0,
        amount_1,
    });

    pool_state.unlocked = true;
    Ok(())
}
```

However, this instruction can actually be called by anyone as there is no constraint for the caller as in line 178.

protocol-v2/programs/core/src/context.rs

```
175 #[derive(Accounts)]
176 pub struct CollectProtocol<'info> {
177     /// Valid protocol owner
178     pub owner: Signer<'info>,
179 
180     /// Factory state stores the protocol owner address
181     #[account(mut)]
182     pub factory_state: Loader<'info, FactoryState>,
183 
184     /// Pool state stores accumulated protocol fee amount
185     #[account(mut)]
186     pub pool_state: Loader<'info, PoolState>,
187 
188     /// The address that holds pool tokens for token_0
189     #[account(
190         mut,
191         associated_token::mint = pool_state.load()?.token_0.key(),
192         associated_token::authority = pool_state,
193     )]
194     pub vault_0: Box<Account<'info, TokenAccount>>,
195 
196     /// The address that holds pool tokens for token_1
197     #[account(
198         mut,
```

```
        associated_token::mint = pool_state.load()?.token_1.key(),
        associated_token::authority = pool_state,
    )]
    pub vault_1: Box<Account<'info, TokenAccount>>,

    /// The address that receives the collected token_0 protocol fees
    #[account(mut)]
    pub recipient_wallet_0: Box<Account<'info, TokenAccount>>,

    /// The address that receives the collected token_1 protocol fees
    #[account(mut)]
    pub recipient_wallet_1: Box<Account<'info, TokenAccount>>,

    /// The SPL program to perform token transfers
    pub token_program: Program<'info, Token>,
}
```

As a result, anyone can call the `collect_protocol()` instruction to claim the protocol accrued fee.

5.2.2. Remediation

Inspex suggest adding the constraint to the function caller to fit the business design, for example as in line 178:

protocol-v2/programs/core/src/context.rs

```
175 #[derive(Accounts)]
176 pub struct CollectProtocol<'info> {
177     /// Valid protocol owner
178     #[account(address = factory_state.load()?.owner)]
179     pub owner: Signer<'info>,
```

5.3. Design Flaw in mint() Function

ID	IDX-003
Target	cyclos_core
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity: High** **Impact: Medium** The position value of the liquidity provided by the users will be incorrect, resulting in either the user capital will be partially lost, or the transaction will be failed. **Likelihood: High** It is likely that this issue will occur since it is a programming flow that is required to be executed every time when the `mint()` function is executed.
Status	**Resolved** Cykura team has resolved this issue by changing the code logic as suggested to avoid incorrect calculation as in commit `8e2d44d196386e47a0c26a23023f6e37afd9e5e5`.

5.3.1. Description

In `cyclos_core` program, when the user provides the liquidity to the platform, the `mint()` function is executed in order to mint the NFT (non fungible token) for the users as the representative of that provided position.

The value of the position is validated and modified in the `mint()` function as shown in the code below. In line 717 and 722, the program finds the value of `balance_0_before` and `balance_1_before` which both will be used as the criteria before allowing this transaction to go further as in line 744 and 750.

protocol-v2/programs/core/src/lib.rs

```
702     let (amount_0_int, amount_1_int) = _modify_position(
703         pool.deref_mut(),
704         &position_state,
705         &ctx.accounts.tick_lower_state,
706         &ctx.accounts.tick_upper_state,
707         &bitmap_lower_state,
708         &bitmap_upper_state,
709         &last_observation_state,
710         &next_observation_state,
711         i64::try_from(amount).unwrap(),
712     )?;
713
714     let amount_0 = amount_0_int as u64;
```

```
    let amount_1 = amount_1_int as u64;

    let balance_0_before = if amount_0 > 0 {
        ctx.accounts.vault_0.amount
    } else {
        0
    };
    let balance_1_before = if amount_0 > 0 {
        ctx.accounts.vault_1.amount
    } else {
        0
    };

    drop(pool);

    let mint_callback_ix = cyclos_core::instruction::MintCallback {
        amount_0_owed: amount_0,
        amount_1_owed: amount_1,
    };
    let ix = Instruction::new_with_bytes(
        ctx.accounts.callback_handler.key(),
        &mint_callback_ix.data(),
        ctx.accounts.to_account_metas(None),
    );
    solana_program::program::invoke(&ix, &ctx.accounts.to_account_infos())?;

    ctx.accounts.vault_0.reload()?;
    ctx.accounts.vault_1.reload()?;

    if amount_0 > 0 {
        require!(
            balance_0_before + amount_0 <= ctx.accounts.vault_0.amount,
            ErrorCode::M0
        );
    }
    if amount_1 > 0 {
        require!(
            balance_1_before + amount_1 <= ctx.accounts.vault_1.amount,
            ErrorCode::M1
        );
    }

    emit!(MintEvent {
        pool_state: ctx.accounts.pool_state.key(),
        sender: ctx.accounts.minter.key(),
        owner: ctx.accounts.recipient.key(),
        tick_lower: tick_lower.tick,
```

```
            tick_upper: tick_upper.tick,
            amount,
            amount_0,
            amount_1
        });

        ctx.accounts.pool_state.load_mut()?.unlocked = true;
        Ok(())
```

However, for the `balance_1_before` value as shown in line 722, it checks with the `amount_0` instead, which is incorrect.

5.3.2. Remediation

Inspex suggests fixing the validating condition in line 722 as follows:

protocol-v2/programs/core/src/lib.rs

```
702         let (amount_0_int, amount_1_int) = _modify_position(
703             pool.deref_mut(),
704             &position_state,
705             &ctx.accounts.tick_lower_state,
706             &ctx.accounts.tick_upper_state,
707             &bitmap_lower_state,
708             &bitmap_upper_state,
709             &last_observation_state,
710             &next_observation_state,
711             i64::try_from(amount).unwrap(),
712         )?;
713 
714         let amount_0 = amount_0_int as u64;
715         let amount_1 = amount_1_int as u64;
716 
717         let balance_0_before = if amount_0 > 0 {
718             ctx.accounts.vault_0.amount
719         } else {
720             0
721         };
722         let balance_1_before = if amount_1 > 0 {
723             ctx.accounts.vault_1.amount
724         } else {
725             0
726         };
```

5.4. Unusable collect_protocol() Function

ID	IDX-004
Target	cyclos_core
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Medium **Impact:** Medium The platform's owner will not be able to claim the protocol accrued fee from all liquidity pools. **Likelihood:** Medium This issue will happen when the platform's owner executes the `collect_protocol()` function to acquire the protocol accrued fee from the pools.
Status	Resolved Cykura team has resolved this issue by adjusting the code flow as suggested to avoid the incorrect execution flow in commit `3134de2bfc2c9a0a613a1ee53d96ae1e3cbe0883`.

5.4.1. Description

In `cyclos_core` program, according to the business design, the `collect_protocol()` function allows the platform's owner to collect the protocol accrued fee from the pools as represented in the code below.

The `collect_protocol()` function firstly takes the mutable `pool_state` state as in line 318 and performs the state changing operation as in line 334 and 349.

protocol-v2/programs/core/lib.rs

```
303 /// Collect the protocol fee accrued to the pool
304 ///
305 ///
306 /// * ctx - Checks for valid owner by looking at signer and factory owner
    addresses.
307 /// Holds the Pool State account where accrued protocol fee is saved, and token
    accounts to perform
308 /// transfer.
309 /// * amount_0_requested - The maximum amount of token_0 to send, can be 0 to
    collect fees in only token_1
310 /// * amount_1_requested - The maximum amount of token_1 to send, can be 0 to
    collect fees in only token_0
311 ///
312 pub fn collect_protocol(
313     ctx: Context<CollectProtocol>,
```

```
    amount_0_requested: u64,
    amount_1_requested: u64,
) -> ProgramResult {
    let mut pool_state = ctx.accounts.pool_state.load_mut()?;
    require!(pool_state.unlocked, ErrorCode::LOK);
    pool_state.unlocked = false;

    let amount_0 = amount_0_requested.min(pool_state.protocol_fees_token_0);
    let amount_1 = amount_1_requested.min(pool_state.protocol_fees_token_1);

    let pool_state_seeds = [
        &POOL_SEED.as_bytes(),
        &pool_state.token_0.to_bytes() as &[u8],
        &pool_state.token_1.to_bytes() as &[u8],
        &pool_state.fee.to_be_bytes(),
        &[pool_state.bump],
    ];

    if amount_0 > 0 {
        pool_state.protocol_fees_token_0 -= amount_0;
        token::transfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info().clone(),
                token::Transfer {
                    from: ctx.accounts.vault_0.to_account_info().clone(),
                    to:
ctx.accounts.recipient_wallet_0.to_account_info().clone(),
                    authority:
ctx.accounts.pool_state.to_account_info().clone(),
                },
                &[&pool_state_seeds[..]],
            ),
            amount_0,
        )?;
    }
    if amount_1 > 0 {
        pool_state.protocol_fees_token_1 -= amount_1;
        token::transfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info().clone(),
                token::Transfer {
                    from: ctx.accounts.vault_1.to_account_info().clone(),
                    to:
ctx.accounts.recipient_wallet_1.to_account_info().clone(),
                    authority:
ctx.accounts.pool_state.to_account_info().clone(),
                },
```

```
                &[&pool_state_seeds[..]],
            ),
            amount_1,
        )?;
    }

    emit!(CollectProtocolEvent {
        pool_state: ctx.accounts.pool_state.key(),
        sender: ctx.accounts.owner.key(),
        recipient_wallet_0: ctx.accounts.recipient_wallet_0.key(),
        recipient_wallet_1: ctx.accounts.recipient_wallet_1.key(),
        amount_0,
        amount_1,
    });

    pool_state = ctx.accounts.pool_state.load_mut()?;
    pool_state.unlocked = true;
    Ok(())
}
```

After that, it invokes the transfer operation on the token program which takes the reference (also known as borrow) from mutable `pool_state` which is currently borrowed by the `cyclos_core` program. Hence, the execution will fail.

5.4.2. Remediation

Inspex suggests releasing the borrowed references before calling the token program, which will try to borrow the `pool_state` state, for example as shown in line 333 - 335:

protocol-v2/programs/core/lib.rs

```
303 /// Collect the protocol fee accrued to the pool
304 ///
305 ///
306 /// * ctx - Checks for valid owner by looking at signer and factory owner addresses.
307 /// Holds the Pool State account where accrued protocol fee is saved, and token accounts to perform
308 /// transfer.
309 /// * amount_0_requested - The maximum amount of token_0 to send, can be 0 to collect fees in only token_1
310 /// * amount_1_requested - The maximum amount of token_1 to send, can be 0 to collect fees in only token_0
311 ///
312 pub fn collect_protocol(
313     ctx: Context<CollectProtocol>,
314     amount_0_requested: u64,
315     amount_1_requested: u64,
```

```
) -> ProgramResult {
    let mut pool_state = ctx.accounts.pool_state.load_mut()?;
    require!(pool_state.unlocked, ErrorCode::LOK);
    pool_state.unlocked = false;

    let amount_0 = amount_0_requested.min(pool_state.protocol_fees_token_0);
    let amount_1 = amount_1_requested.min(pool_state.protocol_fees_token_1);

    let pool_state_seeds = [
        &POOL_SEED.as_bytes(),
        &pool_state.token_0.to_bytes() as &[u8],
        &pool_state.token_1.to_bytes() as &[u8],
        &pool_state.fee.to_be_bytes(),
        &[pool_state.bump],
    ];

    pool_state.protocol_fees_token_0 -= amount_0;
    pool_state.protocol_fees_token_1 -= amount_1;
    drop(pool_state);

    if amount_0 > 0 {
        token::transfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info().clone(),
                token::Transfer {
                    from: ctx.accounts.vault_0.to_account_info().clone(),
                    to:
ctx.accounts.recipient_wallet_0.to_account_info().clone(),
                    authority:
ctx.accounts.pool_state.to_account_info().clone(),
                },
                &[&pool_state_seeds[..]],
            ),
            amount_0,
        )?;
    }
    if amount_1 > 0 {
        token::transfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info().clone(),
                token::Transfer {
                    from: ctx.accounts.vault_1.to_account_info().clone(),
                    to:
ctx.accounts.recipient_wallet_1.to_account_info().clone(),
                    authority:
ctx.accounts.pool_state.to_account_info().clone(),
                },
```

```
                &[&pool_state_seeds[..]],
            ),
            amount_1,
        )?;
    }

    emit!(CollectProtocolEvent {
        pool_state: ctx.accounts.pool_state.key(),
        sender: ctx.accounts.owner.key(),
        recipient_wallet_0: ctx.accounts.recipient_wallet_0.key(),
        recipient_wallet_1: ctx.accounts.recipient_wallet_1.key(),
        amount_0,
        amount_1,
    });

    pool_state = ctx.accounts.pool_state.load_mut()?
    pool_state.unlocked = true;
    Ok(())
}
```

5.5. Use of Outdated Dependency

ID	IDX-005
Target	cyclos_core
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Very Low **Impact:** Low Outdated dependencies have publicly known issues and bugs. It is possible that attackers can use those flaws to attack the program and cause monetary loss or business impact to the platform and its users. **Likelihood:** Low With the current dependency version, it is very unlikely that the publicly known bugs and issues will affect these programs.
Status	**Resolved** Cykura team has resolved this issue by upgrading all dependencies to the latest version as suggested in commit `68fcb30c7589190c6128b38f758afb23a6456518`.

5.5.1. Description

The dependencies specified in the programs were outdated. These versions have publicly known inherent bugs, for example, `anchor-lang`[2] that may potentially be used to cause damage to the program or the users of the program.

protocol-v2/programs/core/Cargo.toml

```
[package]
name = "cyclos-core"
version = "0.1.0"
description = "Core program for pool management"
edition = "2018"

[lib]
crate-type = ["cdylib", "lib"]
name = "cyclos_core"
doctest = false

[features]
no-entrypoint = []
no-idl = []
cpi = ["no-entrypoint"]
default = []
```

```

[dependencies]
anchor-lang = "0.18.0"
anchor-spl = "0.18.0"
uint = "0.9.1"
metaplex-token-metadata = { version = "0.0.1", features = ["no-entrypoint"] }
spl-token = { version = "3.2.0", features = ["no-entrypoint"] }

[dev-dependencies]
quickcheck = "0.9"
```

The applied dependencies outdated are as follows:

Dependency	Latest Stable Major Version
anchor-lang	0.22.0
anchor-spl	0.22.0
spl-token	3.3.0

5.5.2. Remediation

Inspex suggests updating the outdated dependencies to the latest stable version. For example, at the time of the audit, the latest stable version of `anchor-lang` is 0.22.0[2].

However, as there are multiple breaking changes, testing should be comprehensively done to ensure that the program is working as intended.

6. Appendix

6.1. About Inspex

CYBERSECURITY
PROFESSIONAL
SERVICE

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] "OWASP Risk Rating Methodology." [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

[2] "anchor/CHANGELOG.md at master · project-serum/anchor" [Online]. Available: https://github.com/project-serum/anchor/blob/master/CHANGELOG.md. [Accessed: 17-Feb-2022]

inspex

CYBERSECURITY PROFESSIONAL SERVICE