FCFS Launchpad

Smart Contract Audit Report
Prepared for Ancient8

Date Issued:	Jun 28, 2022
Project ID:	AUDIT2022041
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022041
Version	v1.0
Client	Ancient8
Project	FCFS Launchpad
Auditor(s)	Peeraphut Punsuwan Fungkiat Phadejtaku
Author(s)	Peeraphut Punsuwan Fungkiat Phadejtaku
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Jun 28, 2022	Full report	Peeraphut Punsuwan Fungkiat Phadejtaku

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 Ancient8, Inspex team conducted an audit to verify the security posture of the FCFS Launchpad smart contracts between Jun 22, 2022 and Jun 24, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of FCFS Launchpad smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract 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 2 high, 1 low, 1 very low, and 1 info-severity issues. With the project team's prompt response, 2 high and 1 info-severity issues were resolved or mitigated in the reassessment, while 1 low and 1 very low-severity issues were acknowledged by the team. Therefore, Inspex trusts that FCFS Launchpad smart contracts have sufficient protections to be safe for public use. However, as the source code is not publicly available, the bytecode of the smart contracts deployed should be compared with the bytecode of the smart contracts audited before interacting with them. In the long run, Inspex suggests resolving all issues found in this report.

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

Ancient8 is building a DAO that develops a community and software platform to enable everyone to play and build the Metaverse while earning rewards. As Vietnam's largest blockchain gaming guild, Ancient8 has helped tens of thousands of blockchain gamers and enthusiasts by providing scholarship and educational opportunities, community, and blockchain and software products. Ancient8's vision is to democratize social and financial access in the Metaverse, and is on a mission to reach, educate, and empower the next 100 million Metaverse citizens through the blockchain. Ancient8 is backed by leading investors including Dragonfly Capital, Pantera Capital, Hashed, Mechanism Capital, Coinbase Ventures, Alameda Research, Animoca Brands, among others.

Launchpad is a program that allows users of the platform to purchase interesting GameFi tokens that Ancient8 has verified. This will allow the potential gem of the GameFi project to raise funds and contribute their part to improving the GameFi experience for the betterment of the GameFi ecosystem.

The `fcfs_join()` function allows users who have the winning tickets to stake the token after the exclusive round to redeem another token for their rewards. And the `set_is_cancelled()` function allows the authority parties to change the pool's `is_cancelled` status, so they can withdraw all stake tokens and redeem tokens from the pool after the status has changed.

Scope Information:

Project Name	FCFS Launchpad
Website	https://ancient8.gg
Smart Contract Type	Solana Program
Chain	Solana
Programming Language	Rust
Category	Launchpad

Audit Information:

Audit Method	Whitebox
Audit Date	Jun 22, 2022 - Jun 24, 2022
Reassessment Date	Jun 27, 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 smart contracts with the following bytecodes were audited and reassessed by Inspex in detail:

Initial Audit:

Contract	Bytecode SHA256 Hash
launchpad	6cfccfe71c86351234653c05140d19f3ce1fcb980782495bc06e10341927fb71

Reassessment Audit:

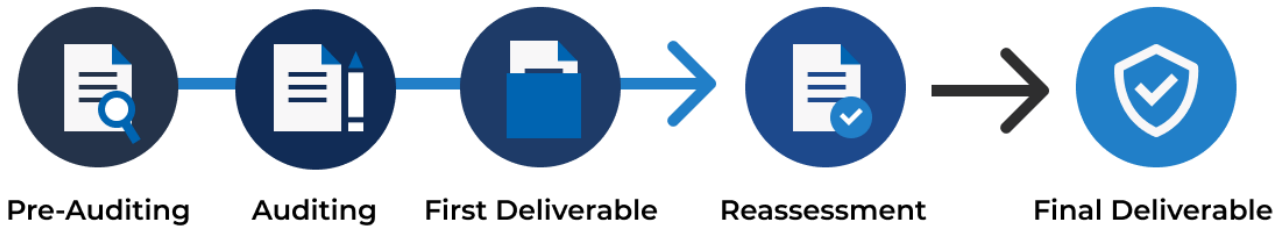
Contract	Bytecode SHA256 Hash
launchpad	a63766a846e1bc0dcec8e10058ff8d8527c66f75060abc56873eac99ebf00425

As the Ancient8 team has decided not to publish the source code to protect their intellectual property, the users should compare the bytecode hashes with the smart contracts deployed before interacting with them to make sure that they are the same with the contracts audited.

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 testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/InspexCo/SCSTG/releases/download/v1.0/SCSTG_v1.0.pdf) which covers most prevalent risks in smart contracts. The latest version of the document can also be found at https://inspex.gitbook.io/testing-guide/.

The following audit items were checked during the auditing activity:

Testing Category	Testing Items
1. Architecture and Design	1.1. Proper measures should be used to control the modifications of smart contract logic 1.2. The latest stable compiler version should be used 1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds 1.4. The smart contract source code should be publicly available 1.5. State variables should not be unfairly controlled by privileged accounts 1.6. Least privilege principle should be used for the rights of each role
2. Access Control	2.1. Contract self-destruct should not be done by unauthorized actors 2.2. Contract ownership should not be modifiable by unauthorized actors 2.3. Access control should be defined and enforced for each actor roles 2.4. Authentication measures must be able to correctly identify the user 2.5. Smart contract initialization should be done only once by an authorized party 2.6. tx.origin should not be used for authorization
3. Error Handling and Logging	3.1. Function return values should be checked to handle different results 3.2. Privileged functions or modifications of critical states should be logged 3.3. Modifier should not skip function execution without reverting
4. Business Logic	4.1. The business logic implementation should correspond to the business design 4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions 4.3. msg.value should not be used in loop iteration
5. Blockchain Data	5.1. Result from random value generation should not be predictable 5.2. Spot price should not be used as a data source for price oracles 5.3. Timestamp should not be used to execute critical functions 5.4. Plain sensitive data should not be stored on-chain 5.5. Modification of array state should not be done by value 5.6. State variable should not be used without being initialized

Testing Category	Testing Items
6. External Components	6.1. Unknown external components should not be invoked 6.2. Funds should not be approved or transferred to unknown accounts 6.3. Reentrant calling should not negatively affect the contract states 6.4. Vulnerable or outdated components should not be used in the smart contract 6.5. Deprecated components that have no longer been supported should not be used in the smart contract 6.6. Delegatecall should not be used on untrusted contracts
7. Arithmetic	7.1. Values should be checked before performing arithmetic operations to prevent overflows and underflows 7.2. Explicit conversion of types should be checked to prevent unexpected results 7.3. Integer division should not be done before multiplication to prevent loss of precision
8. Denial of Services	8.1. State changing functions that loop over unbounded data structures should not be used 8.2. Unexpected revert should not make the whole smart contract unusable 8.3. Strict equalities should not cause the function to be unusable
9. Best Practices	9.1. State and function visibility should be explicitly labeled 9.2. Token implementation should comply with the standard specification 9.3. Floating pragma version should not be used 9.4. Builtin symbols should not be shadowed 9.5. Functions that are never called internally should not have public visibility 9.6. Assert statement should not be used for validating common conditions

3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) 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**.

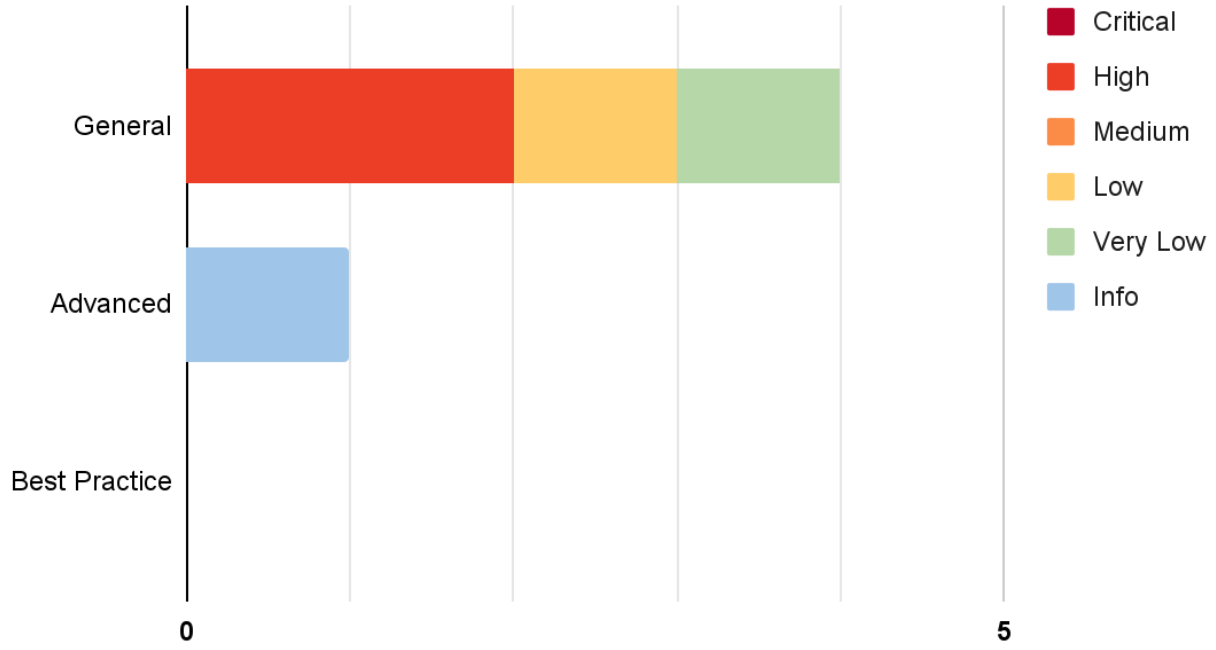
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

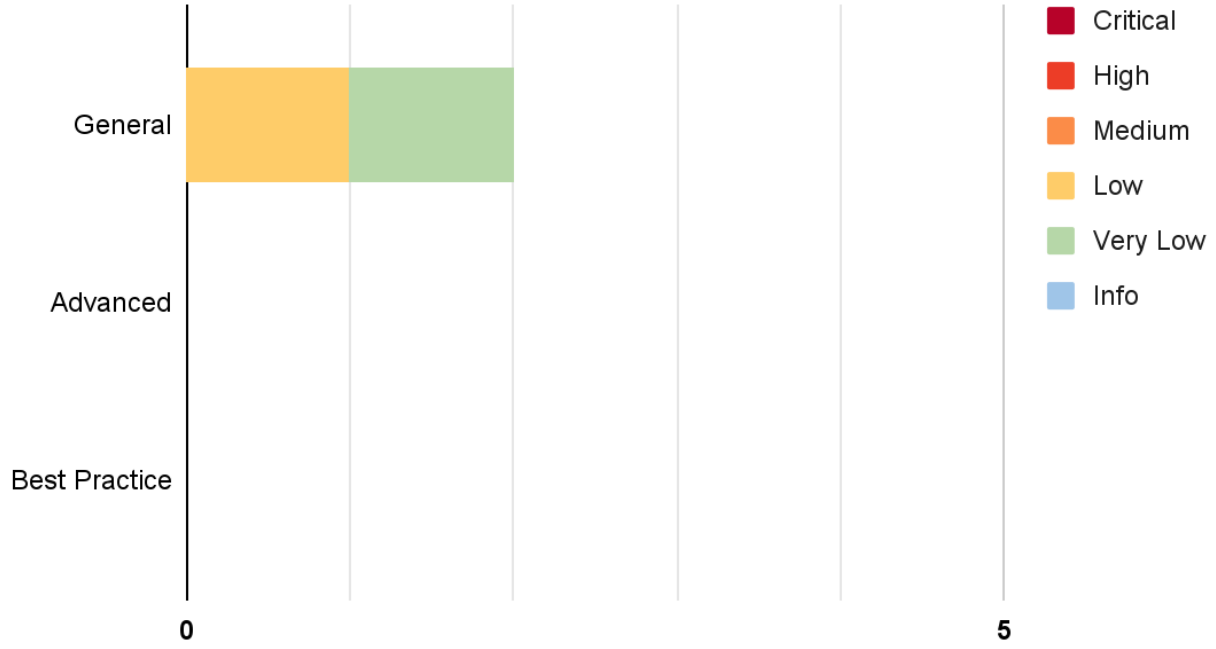
4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

Assessment:



Reassessment:



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	Centralized Control of State Variable	General	High	Resolved *
IDX-003	Smart Contract with Unpublished Source Code	General	Low	Acknowledged
IDX-004	Use of Outdated Dependency	General	Very Low	Acknowledged
IDX-005	Missing Update of no_joined_tickets State Variable	Advanced	Info	Resolved

* The mitigations or clarifications by Ancient8 can be found in Chapter 5.

5. Detailed Findings Information

5.1. Upgradability of Solana Program

ID	IDX-001
Target	launchpad
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The logic of the program 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 * The Ancient8 team has mitigated this issue by using goki.so multi-sig wallet.

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. Centralized Control of State Variable

ID	IDX-002
Target	launchpad
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood: Medium** There is nothing to restrict the changes from being done by the owner. However, only some owner roles can call these functions to change the states.
Status	Resolved * The Ancient8 team has mitigated this issue by using goki.so multi-sig wallet.

5.2.1. Description

Critical state variables can be updated anytime by the controlling authorities. Changes in these variables can impact the users, so the users should accept or be notified before these changes are effective.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

For example, the controlling authorities can use the `set_is_cancelled()` function to change the pool's `is_cancelled` status, which allows them to withdraw all stake tokens and redeem tokens from the pool.

lib.rs

```
91  pub fn set_is_cancelled(
92      ctx: Context<SetIsCancelledAction>,
93      is_cancelled: bool,
94  ) -> ProgramResult {
95      ctx.accounts.process(is_cancelled)?;
96      Ok(())
97  }
```

set_is_cancelled.rs

```
3  #[derive(Accounts)]
4  pub struct SetIsCancelledAction<'info> {
5      #[account(
```

```
6           mut,
7           address = pool_account.admins.root_admin @
ErrorCode::OnlyTicketPoolAdmin
8       )]
9       pub sender: Signer<'info>,
10
11      #[account(mut,
12          seeds = [POOL_SEED, pool_account.id.to_le_bytes().as_ref()],
13          bump = pool_account.bumps.pool_account,
14      )]
15      pub pool_account: Box<Account<'info, TicketPoolAccount>>,
16  }
```

5.2.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the program. However, if modifications are needed, Inspex suggests limiting the use of these functions by the following options:

- Using a multisig account controlled by multiple trusted parties to ensure that the changes of critical states are well prepared.
- Implementing a community-run governance to control the use of these functions.

5.3. Smart Contract with Unpublished Source Code

ID	IDX-003
Target	launchpad
Category	General Smart Contract Vulnerability
CWE	CWE-1006: Bad Coding Practices
Risk	**Severity:** Low **Impact:** Medium The logic of the smart contract may not align with the user's understanding, causing undesired actions to be taken when the user interacts with the smart contract. **Likelihood:** Low The possibility for the users to misunderstand the functionalities of the contract is not very high with the help of the documentation and user interface.
Status	**Acknowledged** The Ancient8 team has clarified that they will be releasing the source code after the product is released.

5.3.1. Description

The smart contract source code is not publicly published, so the users will not be able to easily verify the correctness of the functionalities and the logic of the smart contract by themselves. Therefore, it is possible that the user's understanding of the smart contract does not align with the actual implementation, leading to undesired actions on interacting with the smart contract.

5.3.2. Remediation

Inspex suggests publishing the contract source code through a public code repository or verifying the smart contract source code on the blockchain explorer so that the users can easily read and understand the logic of the smart contract by themselves.

5.4. Use of Outdated Dependency

ID	IDX-004
Target	launchpad
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 the target program.
Status	**Acknowledged** The Ancient8 team has clarified that the breaking changes in the newer anchor-lang version are incompatible with their source code.

5.4.1. Description

The dependencies used by the launchpad are outdated. The version may have publicly known inherent bugs that may potentially be used to cause damage to the program or the users of the program.

Cargo.toml

```
18 [dependencies]
19 anchor-lang = "0.20.1"
20 anchor-spl = "0.20.1"
21 spl-token = { version = "3.1.1", features = ["no-entrypoint"] }
22 vipers = "1.5.7"
```

The outdated dependencies version specified in the program are as the following table:

Dependency	Version
anchor-lang	0.20.1
anchor-spl	0.20.1
spl-token	3.1.1
vipers	1.5.7

5.4.2. Remediation

Inspex suggests upgrading the dependency to the latest stable version. However, the compatibility of the components must be checked to make sure that the program is functional as intended.

The latest stable version of dependencies at this time of the audit are as follows:

Dependency	Version
anchor-lang	0.24.2
anchor-spl	0.24.2
spl-token	3.3.0
vipers	2.0.4

However, there are some breaking changes to the anchor-lang and vipers. Before making a decision to upgrade, please confirm the breaking changes at the following links:

- anchor - https://github.com/coral-xyz/anchor/blob/master/CHANGELOG.md
- vipers - https://github.com/saber-hq/vipers/blob/master/CHANGELOG.md

5.5. Missing Update of no_joined_tickets State Variable

ID	IDX-005
Target	launchpad
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Ancient8 team has resolved this issue by adding the number of joined tickets with the `no_winning_tickets` as suggested.

5.5.1. Description

When the user joins the FCFS event, the `fcfs_join()` function will update the state of the user's and the pool's accounts. The `no_joined_tickets` state of the pool's account is used to track the number of tickets that have joined the event. However, the `no_joined_tickets` state is not updated in this action. It will be incorrect when the platform is required to display the number of the winning tickets that have joined the event. Nevertheless, this issue does not have any security impact on the program.

fcfs_join.rs

```
40  impl<'info> FCFSJoin<'info> {
41      pub fn process(
42          &mut self,
43          incoming_amount: u64,
44          max_staked_amount: u64,
45          no_winning_tickets: u64,
46      ) -> ProgramResult {
47          invariant!(incoming_amount > 0);
48          msg!("incoming_amount = {}", incoming_amount);
49
50          let now = Clock::get()?.unix_timestamp;
51          let pool_account = &mut self.pool_account;
52          invariant!(
53              pool_account.pool_time.close < now,
54              "require pool_account.pool_time.close < now"
55          );
56          msg!("close = {}, now = {}", pool_account.pool_time.close, now);
57
```

```
58          token::transfer(
59              CpiContext::new(
60                  self.token_program.to_account_info(),
61                  Transfer {
62                      from: self.sender_stake_token_account.to_account_info(),
63                      to: self.stake_token_account.to_account_info(),
64                      authority: self.sender.to_account_info(),
65                  },
66              ),
67              incoming_amount,
68          )?;
69
70          let member_account = &mut self.member_account;
71          // sync number of tickets from backend
72          // when user skip exclusive round, then buy in fcfs round
73          if member_account.no_winning_tickets == 0 {
74              member_account.no_winning_tickets = no_winning_tickets;
75          }
76
77          if member_account.staked_token_amount == 0 {
78              pool_account.no_joined_users =
    pool_account.no_joined_users.checked_add(1).unwrap();
79          }
80
81          // fcfs_staked_token_amount += incoming_amount
82          member_account.fcfs_staked_token_amount = member_account
83              .fcfs_staked_token_amount
84              .checked_add(incoming_amount)
85              .unwrap();
86          invariant!(member_account.fcfs_staked_token_amount <=
    max_staked_amount);
87
88          // old_bought_amount = claimable_token_amount + remain_token_amount
89          let old_bought_amount = member_account
90              .claimable_token_amount
91              .checked_add(member_account.remain_token_amount)
92              .unwrap();
93
94          // new_staked_token_amount = staked_token_amount + incoming_amount
95          let new_staked_token_amount = member_account
96              .staked_token_amount
97              .checked_add(incoming_amount)
98              .unwrap();
99          member_account.staked_token_amount = new_staked_token_amount;
100         msg!("new_staked_token_amount = {}", new_staked_token_amount);
101
102         let old_claimable = member_account.claimable_token_amount;
```

```
103         member_account
104             .update_claimable_amount(&pool_account.ticket_info,
    &pool_account.claim_percentage);
105
106         // new_bought_amount = claimable_token_amount + remain_token_amount
107         let new_bought_amount = member_account
108             .claimable_token_amount
109             .checked_add(member_account.remain_token_amount)
110             .unwrap();
111
112         // claimable_amount = claimable_amount - old_claimable + new_claimable
113         pool_account.claimable_amount = pool_account
114             .claimable_amount
115             .checked_sub(old_claimable)
116             .unwrap()
117             .checked_add(member_account.claimable_token_amount)
118             .unwrap();
119
120         // stake_token_total_raised = stake_token_total_raised +
    incoming_amount
121         pool_account.stake_token_total_raised = pool_account
122             .stake_token_total_raised
123             .checked_add(incoming_amount)
124             .unwrap();
125
126         // redeem_token_total_raised = redeem_token_total_raised -
    old_bought_amount + new_bought_amount
127         pool_account.redeem_token_total_raised = pool_account
128             .redeem_token_total_raised
129             .checked_sub(old_bought_amount)
130             .unwrap()
131             .checked_add(new_bought_amount)
132             .unwrap();
133
134         // fcfs_bought_token_amount = fcfs_bought_token_amount -
    old_bought_amount + new_bought_amount
135         member_account.fcfs_bought_token_amount = member_account
136             .fcfs_bought_token_amount
137             .checked_add(new_bought_amount)
138             .unwrap()
139             .checked_sub(old_bought_amount)
140             .unwrap();
141
142         invariant!(pool_account.no_joined_tickets <=
    pool_account.ticket_info.no_tickets);
143         invariant!(
144             pool_account.stake_token_total_raised
```

```
145                 <= pool_account
146                     .ticket_info
147                     .no_tickets
148
    .checked_mul(pool_account.ticket_info.stake_amount_each_ticket)
149                     .unwrap()
150         );
151
152         launchpad_emit!(FCFSJoinEvent {
153             sender: self.sender.key(),
154             pool_account: pool_account.key(),
155             token_id: pool_account.stake_token_id,
156             amount: incoming_amount,
157             timestamp: now
158         });
159
160         Ok(())
161     }
```

5.5.2. Remediation

Inspex suggests adding the number of joined tickets with the `no_winning_tickets` when the users join the FCFS event to count the tickets that have joined the event at line 79.

fcfs_join.rs

```
40  impl<'info> FCFSJoin<'info> {
41      pub fn process(
42          &mut self,
43          incoming_amount: u64,
44          max_staked_amount: u64,
45          no_winning_tickets: u64,
46      ) -> ProgramResult {
47          invariant!(incoming_amount > 0);
48          msg!("incoming_amount = {}", incoming_amount);
49
50          let now = Clock::get()?.unix_timestamp;
51          let pool_account = &mut self.pool_account;
52          invariant!(
53              pool_account.pool_time.close < now,
54              "require pool_account.pool_time.close < now"
55          );
56          msg!("close = {}, now = {}", pool_account.pool_time.close, now);
57
58          token::transfer(
59              CpiContext::new(
60                  self.token_program.to_account_info(),
61                  Transfer {
```

```
62                      from: self.sender_stake_token_account.to_account_info(),
63                      to: self.stake_token_account.to_account_info(),
64                      authority: self.sender.to_account_info(),
65                  },
66              ),
67              incoming_amount,
68          )?;
69
70          let member_account = &mut self.member_account;
71          // sync number of tickets from backend
72          // when user skip exclusive round, then buy in fcfs round
73          if member_account.no_winning_tickets == 0 {
74              member_account.no_winning_tickets = no_winning_tickets;
75          }
76
77          if member_account.staked_token_amount == 0 {
78              pool_account.no_joined_users =
pool_account.no_joined_users.checked_add(1).unwrap();
79              pool_account.no_joined_tickets =
pool_account.no_joined_tickets.checked_add(no_winning_tickets).unwrap();
80          }
81
82          // fcfs_staked_token_amount += incoming_amount
83          member_account.fcfs_staked_token_amount = member_account
84              .fcfs_staked_token_amount
85              .checked_add(incoming_amount)
86              .unwrap();
87          invariant!(member_account.fcfs_staked_token_amount <=
max_staked_amount);
88
89          // old_bought_amount = claimable_token_amount + remain_token_amount
90          let old_bought_amount = member_account
91              .claimable_token_amount
92              .checked_add(member_account.remain_token_amount)
93              .unwrap();
94
95          // new_staked_token_amount = staked_token_amount + incoming_amount
96          let new_staked_token_amount = member_account
97              .staked_token_amount
98              .checked_add(incoming_amount)
99              .unwrap();
100         member_account.staked_token_amount = new_staked_token_amount;
101         msg!("new_staked_token_amount = {}", new_staked_token_amount);
102
103         let old_claimable = member_account.claimable_token_amount;
104         member_account
105             .update_claimable_amount(&pool_account.ticket_info,
```

```
&pool_account.claim_percentage);

        // new_bought_amount = claimable_token_amount + remain_token_amount
        let new_bought_amount = member_account
            .claimable_token_amount
            .checked_add(member_account.remain_token_amount)
            .unwrap();

        // claimable_amount = claimable_amount - old_claimable + new_claimable
        pool_account.claimable_amount = pool_account
            .claimable_amount
            .checked_sub(old_claimable)
            .unwrap()
            .checked_add(member_account.claimable_token_amount)
            .unwrap();

        // stake_token_total_raised = stake_token_total_raised +
incoming_amount
        pool_account.stake_token_total_raised = pool_account
            .stake_token_total_raised
            .checked_add(incoming_amount)
            .unwrap();

        // redeem_token_total_raised = redeem_token_total_raised -
old_bought_amount + new_bought_amount
        pool_account.redeem_token_total_raised = pool_account
            .redeem_token_total_raised
            .checked_sub(old_bought_amount)
            .unwrap()
            .checked_add(new_bought_amount)
            .unwrap();

        // fcfs_bought_token_amount = fcfs_bought_token_amount -
old_bought_amount + new_bought_amount
        member_account.fcfs_bought_token_amount = member_account
            .fcfs_bought_token_amount
            .checked_add(new_bought_amount)
            .unwrap()
            .checked_sub(old_bought_amount)
            .unwrap();

        invariant!(pool_account.no_joined_tickets <=
pool_account.ticket_info.no_tickets);
        invariant!(
            pool_account.stake_token_total_raised
                <= pool_account
                    .ticket_info
```

```
148                     .no_tickets
149
    .checked_mul(pool_account.ticket_info.stake_amount_each_ticket)
150                     .unwrap()
151         );
152
153         launchpad_emit!(FCFSJoinEvent {
154             sender: self.sender.key(),
155             pool_account: pool_account.key(),
156             token_id: pool_account.stake_token_id,
157             amount: incoming_amount,
158             timestamp: now
159         });
160
161         Ok(())
162     }
163 }
```

6. Appendix

6.1. About Inspex

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

inspeX

CYBERSECURITY PROFESSIONAL SERVICE