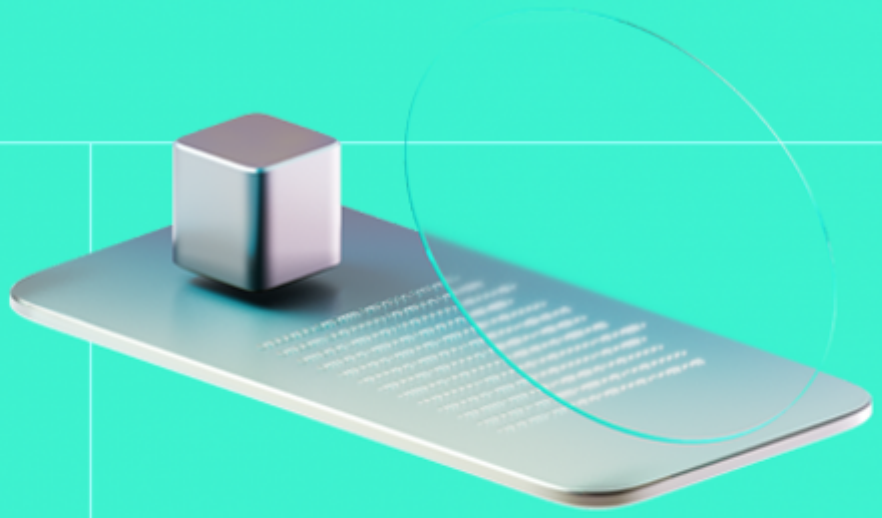
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Propbase

Date: 06/03/2024

We express our gratitude to the Propbase team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The PROPS Staking App facilitates the staking of $PROPS, providing users with additional $PROPS as rewards. Developed using the Move language and Aptos standard libraries, the application operates on the Aptos blockchain.

Platform: Aptos

Language: Move

Tags: Staking, Incentives

Timeline: 09.02.2024 - 20.02.2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/Propbase-Application/propbase_staking_blockchain
Commit	52f4bf6

Audit Summary

10/10	8/10	96%	7/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.1/10

The system users should acknowledge all the risks summed up in the risks section of the report

5	5	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	1
Low	4

Vulnerability	Status
F-2024-0826 - Early withdrawal penalty bypass by chunking withdrawal amount	Fixed
F-2024-0915 - Unnecessary calculations for old stakeholders	Fixed
F-2024-0916 - Users with low stake amounts cannot get any rewards due to precision loss	Fixed
F-2024-0917 - Rewarding formula inconsistency between the documentation and the code	Fixed
F-2024-0934 - Underconfigured state is possible	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Propbase
Audited By	Ataberk Yavuzer, Vladyslav Khomenko
Approved By	Przemyslaw Swiatowiec
Website	https://www.propbase.app/
Changelog	20.02.2024 - Preliminary Report, 06.03.2024 - Final Report

Table of Contents

System Overview

PROPS is a staking protocol with the following contracts:

propbase_staking.move — a smart contract that rewards users for staking their tokens. Reward distribution depends on the max pool cap provided by the owner and could not be re-calculated after staking epoch is started. It was planned that this contract is going to use the $PROPS token (8 decimals) as its main crypto asset. Users can stake their $PROPS token right after the staking epoch is getting started. They can stake until 1 day before epoch end time to benefit from protocol rewards. There are multiple feature of this contract. There will be a total reward amount in the protocol. If the entire reward amount cannot distributed to protocol users, the excessive amount will be stored in the pool until default expiry time passes which is 2 years by default. After that time, the treasury role of the protocol can withdraw the excessive rewards. Also, the owner can freeze the contract in case of emergency by calling the emergency stop functionality. Protocol rewards and stakes can be directly transferred to protocol users by owner if the protocol goes into the emergency stage.

Privileged roles

- Admin:
  - stops staking operations in case of emergency
  - updates admin, reward treasurer and treasury addresses
  - creates or updates a staking pool
  - transfer stakes and rewards back in case of emergency
- Treasury:
  - withdraws excess rewards
  - calculates required rewards
  - receives early withdrawal penalties
- Reward treasurer:
  - adds reward funds to the protocol

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- Documentation contains all information about the protocol itself.
- Functional requirements are provided.
- Technical description are provided.
- There is a minor discrepancy between the documentation and code.

Code quality

The total Code Quality score is **8** out of **10**.

- Event emitting method in the code is deprecated.
- Check-effect-interaction pattern is correctly covered.
- The state management is implemented correctly.

Test coverage

Code coverage of the project is **96%**.

- Deployment and basic user interactions are covered with tests.
- Test cases covers most of possible scenarios.
- Interactions by several users are tested thoroughly.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **1** medium, and **4** low severity issues, leading to a security score of **9** out of **10**.

After fixing all findings, the security score is reached to **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.1**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- Admin can freeze the protocol at any time.
- There are multiple instances of precision losses.
- Early withdrawal penalties can be bypassed.
- The `emergency_asset_distribution()` function is only callable by admin. This function is designed to send tokens back to users during an emergency situation. Therefore, other users should be able to call this function too.
- PROPS token was not audited since it was not included in the audit scope. Therefore, it is impossible to ensure the PROPS token is secure.

Findings

Vulnerability Details

F-2024-0826 - Early withdrawal penalty bypass by chunking withdrawal amount - Medium

Description:

The Propbase protocol uses $PROPS token for staking operations. The protocol distributes staking rewards as $PROPS to stakeholders. In addition, admin of the protocol sets a `penalty_rate` during the pool creation. The penalty rate amount can be set between **1** and **50**. The purpose of this variable is to penalize early withdrawals.

The penalty calculation can be seen at below:

```
let penalty = amount / 100 * stake_pool_config.penalty_rate;
```

Currently, there is no lower bound for withdraw amount in the code. Therefore, it is possible to chunk the total withdraw amount by 99 in order to bypass this penalty according to the formula above.

```
let penalty = 99 / 100 * stake_pool_config.penalty_rate (0-50);
penalty = 0;
```

As a result, it is possible to bypass the early withdrawal penalty due to this precision loss.

Path:

- propbase_staking.move#L562
- propbase_staking.move#L596

Assets:

- propbase_staking.move [https://github.com/Propbase-Application/propbase_staking_blockchain]

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 3
Impact [1-5]: 3
Exploitability [0-2]: 0
Complexity [0-2]: 1

Final Score: 2.8 (Medium)
Hacken Calculator Version: 0.6

Recommendations

Recommendation: Consider adding a lower bound for `withdraw_stake()` function, so it is possible to withdraw only amounts higher than 100. This mitigation will eliminate the precision loss and there will be penalty for early withdrawals.

Remediation (revised commit: **54baa9)**: The minimum stake amount was implemented with the latest code fix. Therefore, chunking the withdraw amount to bypass penalties is not an option anymore.

F-2024-0915 - Unnecessary calculations for old stakeholders - Low

Description:

The `implement_unstake` function currently lacks proper handling of important state variables when a user withdraws their entire stake. Specifically, the **stake_pool_config.staked_addresses** variable is not appropriately decreased in such instances. This oversight has implications for functions like `calculate_required_rewards()` and `emergency_asset_distribution()`, as they currently consider users with zero stakes in their calculations.

```
let length = vector::length(&stake_pool_config.staked_addressess);
```

Additionally, the functions `transfer_principal_and_rewards()` and `get_total_rewards_so_far()` are called for users with zero stakes, resulting in unnecessary gas consumption. To optimize gas usage, the contract should be modified to skip these functions for users with zero stakes, ensuring more efficient and cost-effective execution.

Path:

- propbase_staking.move#L559
- propbase_staking.move#L808
- propbase_staking.move#L1050

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 2
Exploitability [0-2]: 0
Complexity [0-2]: 0
Final Score: 2.0 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation: It is recommended to clear the **stake_pool_config.staked_addresses** state variable in case stakers withdraw their entire stakes from the pool.

Remediation (revised commit: **54baa9)**: The `exited_addressess` variable was implemented along with `update_addresses_on_exit()` function to track completely withdrawn funds.

F-2024-0916 - Users with low stake amounts cannot get any rewards due to precision loss - Low

Description:

The current implementation lacks a lower bound for the `min_stake_amount`, leading to potential issues for users with stakes lower than `seconds_in_year`. For instance, if a pool is created with a `min_stake_amount` of **1000000**, there is a risk that the `interest_per_rate` calculation may consistently return zero due to precision loss.

```
inline fun apply_reward_formula(
principal: u64,
period: u64,
interest_rate: u64,
seconds_in_year: u64
): u128 acquires StakePool {
let interest = ((principal as u128) * (interest_rate as u128));
let interest_per_sec = interest / (seconds_in_year as u128);
let remainder = interest % (seconds_in_year as u128);
let total_interest = (interest_per_sec * (period as u128)) + ((remai
nder * (period as u128)) / (seconds_in_year as u128));
total_interest / 100
}
```

Another scenario where this problem may arise is through the `withdraw_stake` function. In such cases, if the remaining stake (calculated as principal multiplied by interest rate) after a `withdraw_stake` operation falls below `seconds_in_year`, the total reward for these users will be inaccurately calculated as zero. To address this, it is advisable to introduce a lower bound for the `min_stake_amount` to ensure that users with stakes below `seconds_in_year` can still receive benefits from the reward distribution.

Path:

- propbase_staking.move#L658
- propbase_staking.move#L659

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 3
Exploitability [0-2]: 2
Complexity [0-2]: 0
Final Score: 1.8 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation: Consider adding a lower bound for the minimum withdrawal amount. In addition, the `min_stake` amount on the configuration should not be also lower than `seconds_in_year` variable in order to prevent the precision loss.

Remediation (revised commit: **54baa9)**: The minimum stake amount was implemented with the latest code fix. Therefore, precision loss is unlikely at the function specified above.

F-2024-0917 - Rewarding formula inconsistency between the documentation and the code - Low

Description:

During the security audit, a discrepancy was identified between the reward equation depicted in the staking diagram and the implementation in the `apply_reward_formula` function. To maintain a robust and consistent system, it is imperative that the specifications outlined in the whitepaper documentation precisely match the code. We recommend updating the staking diagram to accurately reflect the calculation performed by the code.

The rewarding formula on the staking diagram shown below:

```
(((period) / 100) * (principal / seconds_in_year)) * interest_rate
```

The actual reward calculation in the code:

```
let interest = ((principal as u128) * (interest_rate as u128));
let interest_per_sec = interest / (seconds_in_year as u128);
let remainder = interest % (seconds_in_year as u128);
let total_interest = (interest_per_sec * (period as u128)) + ((remai
nder * (period as u128)) / (seconds_in_year as u128));
total_interest / 100
```

The equation presented in the staking diagram results in higher rewards than the accurate calculation. Should this incorrect equation be utilized during pool creation for the required_reward calculation, it could lead to the distribution of more rewards than initially anticipated. Therefore, it is crucial to rectify the diagram's equation to ensure that the reward distribution aligns with the intended calculations and adheres to the expected security standards.

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 2
Exploitability [0-2]: 0
Complexity [0-2]: 0
Final Score: 2.0 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation: It is recommended to update the official documentation with the correct formula to prevent any possible inconsistencies.

Remediation (revised commit: **54baa9)**: The outdated graph was replaced with the more prior version. The new graph eliminated this issue.

F-2024-0934 - Underconfigured state is possible - Low

Description:

State resources such as `StakePool`, `StakeApp`, `RewardPool` are not initialized atomically. This means there is a possibility for the state to be underconfigured when it is being used. Setting/updating some of the fundamental fields is optional but the validity of the state is mandatory.

```
if(set_interest_rate) {
assert!(interest_rate > 0 && interest_rate <= 100, error::invalid_ar
gument(E_STAKE_POOL_INTEREST_OUT_OF_RANGE));
stake_pool_config.interest_rate = interest_rate;
};
```

In the example above, it is possible for the owner to unintentionally underconfigure a fundamental system variable. After staking has started, the owner would have to deprecate this contract, distribute stakes (the contract has this functionality), redeploy another contract, and configure it from scratch.
What is more, under configured state can produce generic errors that do not give a clue about the actual issue.

Assets:

- propbase_staking.move [https://github.com/Propbase-Application/propbase_staking_blockchain]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 4
Exploitability [0-2]: 1
Complexity [0-2]: 1
Final Score: 2.3 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation:

The state should be initialized and valid at the moment it is being used. It is recommended to implement one from the following options:

- Add `is_valid_state` flags for state resources. It must be flipped to `false` if any configuration is being performed.
- Create `validate_state()` function.

The `validate_state()` would check `is_valid_state` flags in resources, and if it is `false` - check the validity of the state and set a flag to `true` or assert an error if one is encountered. This function must be used in user-accessed functions. Also, access to state configuration functions should depend both on the `epoch_start_time` and `is_valid_state`.

Remediation (revised commit: **54baa9)**: The PROPBASE team solved this issue by introducing `is_valid_state` variable with the latest update. The protocol now checks this new state variable to prevent underconfigured state.

Observation Details

F-2024-0920 - Too generic assertion errors - Info

Description: The presence of assertion errors in the code raises concerns, particularly those deemed too generic. These ambiguous assertion errors could potentially making it challenging to identify and address security issues promptly. When assertions lack specificity, it hinders effective debugging and troubleshooting, potentially leading to overlooked vulnerabilities that might pose security risks. It is crucial to enhance the precision of assertion errors, providing more informative and detailed messages to facilitate the identification and resolution of potential security issues in the smart contract.

```
assert!(now <= stake_pool_config.epoch_end_time, error::out_of_range
(0));
```

Path:

- propbase_staking.move#L574
- propbase_staking.move#L576
- propbase_staking.move#L867
- propbase_staking.move#L1021
- propbase_staking.move#L1115

Remediation (revised commit: 54baa9): All asserts with `error::out_of_range(0)` message were replaced with more specific ones.

Status: Fixed

Recommendations

Recommendation: It is recommended to replace the zero error code with another variable-mapped error code in order to specify the exact error code.

F-2024-0921 - Unknown upgrade policy - Info

Description:

The **Move.toml** configuration file lacks an explicit identification of the upgrade policy. Within the Aptos blockchain, the capability for upgrading Move code, including Move modules, exists. This functionality empowers code owners and module developers to seamlessly update and enhance their contracts. The upgrade process occurs under a single, stable, and well-known account address, ensuring consistency.

In the absence of a specified upgrade policy in the configuration file, there is a risk of ambiguity regarding how and when upgrades may take place. It is essential to clearly define the upgrade policy to provide transparency and ensure that code owners and module developers can effectively manage and evolve their contracts.

Furthermore, if the current version of the code is considered final and should not undergo further modifications, it is strongly recommended to employ the `immutable` keyword for the `upgrade_policy` field. This designation ensures that the code remains immutable, providing clarity and preventing unintentional modifications after reaching the desired final version.

References:

- https://aptos.dev/move/book/package-upgrades/#upgrade-policies

Remediation (revised commit: **54baa9)**: This finding is resolved by the PROPBASE team after they added `immutable` keyword as their `upgrade_policy`.

Status: Fixed

Recommendations

Recommendation: Consider using `immutable` keyword for the **upgrade_policy** field if the code will not take any further updates.

F-2024-0933 - Using deprecated events type - Info

Description:

Event-handle events are deprecated. Module events should be used instead. While event-handle events might still work, the Aptos team encourages projects to move to the module events.

Example of use:

```
#[event]
struct SetAdminEvent has drop, store {
old_admin: address,
new_admin: address,
}
...
event::emit(SetAdminEvent {
old_admin: old_admin,
new_admin: new_admin_address
});
```

Assets:

- propbase_staking.move [https://github.com/Propbase-Application/propbase_staking_blockchain]

Status: Fixed

Recommendations

Recommendation:

Consider migrating to module events. Event structs will need an `#[event]` annotation. Event handles are no longer needed. Emitting is done through `event::emit()` function.

Remediation (revised commit: 54baa9): All events in the code were refactored into their more prior version.

External References:

- Aptos team encourages migration to new events
- Proposal description

F-2024-0935 - Literal value is used - Info

Description: In `create_or_update_stake_pool()`, a literal number `20000000000` is used. It is a best practice to use named constants whenever a known value is used.

Assets:

- propbase_staking.move [https://github.com/Propbase-Application/propbase_staking_blockchain]

Status: Fixed

Recommendations

Recommendation: Consider extracting the aforementioned number into a constant.

Remediation (revised commit: 54baa9): The hardcoded max stake field was removed from the code.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/Propbase-Application/propbase_staking_blockchain
Commit	a81b2c998a43b164083917ff9c08073fc4372acc
Remediation Commit	9416b9a04f7109cc40ef9ad586dfc385d8932f08
Whitepaper	README.md
Requirements	README.md
Technical Requirements	README.md

Contracts in Scope

./sources/propbase_staking.move