verichains

SECURITY AUDIT OF

PROPEASY PROGRAM

Propeasy

Public Report

Jan 12, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Solana	A decentralized blockchain built to enable scalable, user-friendly apps for the world.
SOL	A cryptocurrency whose blockchain is generated by the Solana platform.
Lamport	A fractional native token with the value of 0.000000001 SOL.
Program	An app interacts with a Solana cluster by sending it transactions with one or more instructions. The Solana runtime passes those instructions to program.
Instruction	The smallest contiguous unit of execution logic in a program.
Cross-program invocation (CPI)	A call from one smart contract program to another.
Anchor	A framework for Solana's Sealevel runtime providing several convenient developer tools for writing smart contracts.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Jan 12, 2024. We would like to thank the Propeasy Labs for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Propeasy program. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some minor issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY 5

1.1. About Propeasy program 5

1.2. Audit scope 5

1.3. Audit methodology 6

1.4. Disclaimer 7

1.5. Acceptance Minute 7

2. AUDIT RESULT 8

2.1. Overview 8

2.2. Findings 8

2.2.1. Integer overflow if `property_decimals` higher than 9 LOW 8

2.2.2. Rounding issue in `commission_amount` LOW 10

2.2.3. Redundant `DISCRIMINATOR_SIZE` in `PrivateSaleInfo` and `PublicSaleInfo` LOW 12

3. VERSION HISTORY 14

1. MANAGEMENT SUMMARY

1.1. About Propeasy program

Propeasy is a real estate technology company that is using blockchain technology to make real estate more accessible to investors. It converts estates into tokens on the blockchain helping investors own shares of assets with small capital and trade easily.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Propeasy program. It was conducted on commit `d6c8f398e193c7311baebbabf4bbcbf4a61ccbef` from git repository link: *https://github.com/renec-chain/propeasy-program*.

The latest version of the following file was made available in the course of the review:

SHA256 Sum	File
9ef5cf087973e5d3f0ba0a10da76e042cc91b9b886f35dafc6fd06f918922963	constants.rs
17cc0f08cdd0b1918bca4c5cbd1383e54c373b86f02a446245a563722cd7426f	errors.rs
c148301ea99b108d196b36d46d32a3fad24c0892bd02acf6450a7b9035528058	events.rs
3d60abae704cc0e4c5a5e27b2253d8eae120f12c337d11ad88cce56240f6882b	instructions/change_mint_token_platform.rs
1c3ce1df331c4c9171ece49b7e680147e1fedc9980412e85a8acdef045450f96	instructions/claim_property_token.rs
da7b7919752c03fc8dcc8ded37d742b462b0694e46dac0c8f6e225def064dd9c	instructions/create_property.rs
4b6daf155c6db7e815317f6eb9e105f87cfa162f4b30f5514688fec14aa94204	instructions/initialize_platform.rs
671cbd788d038e59ad02456dbd0c49fd9e933e27153c5d8b90c804df0350bdf3	instructions/mod.rs
d24423e0574f3aadf0602f99aff88e84efe103732489aa8f8835aada6ae65e8c	instructions/purchase_property_token.rs
8d8b3610b3202de9d8a4da987eb9e99cd12d7878e707a9d685b8432bc1b21dcb	instructions/update_property.rs

e3ce75dff990f7acca7b8d91c993d92d5ef8e8cb7e4eb11da578790fa84f6024	instructions/withdraw_token.rs
5fa6dacf31fde80ee9e0027215e32c28f9f662f6dd4ba067f3623b5c957995b6	lib.rs
e28a15d34981dfa35d1b91a9f2fb3be960f3bf635d84ed519922e7c220b463e8	states/locker.rs
ac7f72cab89a51dea436eb3fe7d576d928230e6af2b05ab3c945e0e2f0f9aab9	states/mod.rs
a6772abc955742bf5464550e42481e3eaadf1020f135a68322a66c64d89f8fb0	states/platform.rs
c88c8db73f1c6a1b855de8951947c5f8f5a71921ceb21b3ab32aad1f714bc5b7	states/property.rs
de31087ae4555216a006edcd14785c071c642ce00b21a794e40067f78555e6c3	util/mod.rs
4e602a53227971d757a7d010cbc2342503eea57b3442311937635c5628362f97	util/token.rs
47cb89bbee53dc24da68a2350557b859e9cc22fb95100d158c05edf429a0fbf8	util/util.rs

1.3. Audit methodology

Our security audit process for Solana smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the Solana smart contract:

- Arithmetic Overflow and Underflow
- Signer checks
- Ownership checks
- Rent exemption checks
- Account confusions
- Bump seed canonicalization
- Closing account
- Signed invocation of unverified programs

- Numerical precision errors
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Propeasy Labs acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Propeasy Labs understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Propeasy Labs agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Propeasy Labs will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Propeasy Labs, the final report will be considered fully accepted by the Propeasy Labs without the signature.

2. AUDIT RESULT

2.1. Overview

The Propeasy program was written in `Rust` programming language and `Anchor` framework.

The Propeasy program allows `Propeasy` to create and sell properties in 2 round sales: private and public.

In order to participate in a private sale, users need to hold an enough amount of `platform` tokens. They can then purchase the `property` tokens using `purchase` tokens, with a pre-determined portion of the purchased `property` tokens being immediately allocated (`TGE` percentage). The remaining purchased `property` tokens will be subject to a vesting schedule, releasing them sequentially over a predetermined period.

Public sale is open to everyone after the private sale. No special requirements to the purchaser and the `property` tokens will be allocated immediately.

`Propeasy` also provides bonus commissions (`platform` tokens) for both buyers and their referrers.

2.2. Findings

During the audit process, the audit team had identified some minor issues in the given version of Propeasy program.

#	Issue	Severity	Status
1	Integer overflow if `property_decimals` higher than 9	LOW	Acknowledged
2	Rounding issue in `commission_amount`	LOW	Acknowledged
3	Redundant `DISCRIMINATOR_SIZE` in `PrivateSaleInfo` and `PublicSaleInfo`	LOW	Acknowledged

2.2.1. Integer overflow if `property_decimals` higher than 9 LOW

Affected files:

- `property.rs`

When calculating `commission_amount`, the `purchase_amount` is divided by `10_i32.pow(purchase_decimals as u32)`. If `purchase_decimals` is higher than 9, the `pow` function will be overflowed cause unexpected result for `commission_amount`.

```
// PrivateSaleInfo
pub fn calculate_commission_amount(
    &self,
    purchase_decimals: u8,
    purchase_amount: u64,
    has_referral: bool,
) -> Result<u64, ProgramError> {
    let commission_factor = if has_referral {
        self.referral_commission_amount
    } else {
        self.commission_amount
    };
    let commission_amount = (purchase_amount as u128)
        .checked_div(10_i32.pow(purchase_decimals as u32) as u128)
        .unwrap()
        .checked_mul(commission_factor as u128)
        .unwrap() as u64;

    Ok(commission_amount)
}

// PublicSaleInfo
pub fn calculate_commission_amount(
    &self,
    purchase_decimals: u8,
    purchase_amount: u64,
    has_referral: bool,
) -> Result<u64, ProgramError> {
    let commission_factor = if purchase_amount >= self.referral_commission_boost_threshold
{
        if has_referral {
            self.referral_commission_boost_amount
        } else {
            self.commission_boost_amount
        }
    } else {
        if has_referral {
            self.referral_commission_amount
        } else {
            self.commission_amount
        }
    };

    let commission_amount = (purchase_amount as u128)
        .checked_div(10_i32.pow(purchase_decimals as u32) as u128)
        .unwrap()
        .checked_mul(commission_factor as u128)
        .unwrap() as u64;

    Ok(commission_amount)
}
```

```
fn validate_purchase_amount(
    property_decimals: u8,
    purchase_amount: u64,
    minimum_purchase_amount: u64,
    token_price: u64,
    total_purchased_amount: u64,
    supply_amount: u64,
) -> Result<(u64, u64), ProgramError> {
    require!(
        purchase_amount >= minimum_purchase_amount && purchase_amount > 0,
        ErrorCode::NotReachMinimumAmount
    );

    let property_amount = (purchase_amount as u128)
        .checked_mul(10_i32.pow(property_decimals.into()) as u128)
        .unwrap()
        .checked_div(token_price as u128)
        .unwrap() as u64;

    require!(
        total_purchased_amount.checked_add(property_amount).unwrap() <= supply_amount,
        ErrorCode::ExceedSupplyAmount
    );
    Ok((token_price, property_amount))
}
```

RECOMMENDATION

- Use `checked_pow` instead of `pow`.
- Use `10_u128` instead of `10_i32`.

UPDATES

- *Jan 12, 2024*: This issue has been acknowledged by the Propeasy Labs team.

2.2.2. Rounding issue in `commission_amount` LOW

Affected files:

- `property.rs`

When calculating `commission_amount`, the `purchase_amount` is divided by `purchase_decimals` before multiplying `commission_factor`. This may cause rounding issue when `purchase_amount` is not divisible by `purchase_decimals`.

```
// PrivateSaleInfo
pub fn calculate_commission_amount(
    &self,
    purchase_decimals: u8,
    purchase_amount: u64,
    has_referral: bool,
) -> Result<u64, ProgramError> {
    let commission_factor = if has_referral {
        self.referral_commission_amount
    } else {
        self.commission_amount
    };
    let commission_amount = (purchase_amount as u128)
        .checked_div(10_i32.pow(purchase_decimals as u32) as u128)
        .unwrap()
        .checked_mul(commission_factor as u128)
        .unwrap() as u64;

    Ok(commission_amount)
}

// PublicSaleInfo
pub fn calculate_commission_amount(
    &self,
    purchase_decimals: u8,
    purchase_amount: u64,
    has_referral: bool,
) -> Result<u64, ProgramError> {
    let commission_factor = if purchase_amount >= self.referral_commission_boost_threshold
{
        if has_referral {
            self.referral_commission_boost_amount
        } else {
            self.commission_boost_amount
        }
    } else {
        if has_referral {
            self.referral_commission_amount
        } else {
            self.commission_amount
        }
    };

    let commission_amount = (purchase_amount as u128)
        .checked_div(10_i32.pow(purchase_decimals as u32) as u128)
        .unwrap()
        .checked_mul(commission_factor as u128)
        .unwrap() as u64;

    Ok(commission_amount)
}
```

RECOMMENDATION

We should multiply `purchase_amount` by `commission_factor` before dividing it by `purchase_decimals` to avoid rounding issue.

UPDATES

- *Jan 12, 2024*: This issue has been acknowledged by the Propeasy Labs team.

2.2.3. Redundant `DISCRIMINATOR_SIZE` in `PrivateSaleInfo` and `PublicSaleInfo` LOW

Affected files:

- `property.rs`

`PrivateSaleInfo` and `PublicSaleInfo` is not seperated account but only a struct inside `PropertyState` account, so we don't need to add `DISCRIMINATOR_SIZE` in each of them.

```
impl PrivateSaleInfo {
    pub const LEN: usize = DISCRIMINATOR_SIZE
        + I64_SIZE
        + I64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U32_SIZE
        + U32_SIZE
        + I64_SIZE
        + I64_SIZE
        + U64_SIZE;
    ...
}

impl PublicSaleInfo {
    pub const LEN: usize = DISCRIMINATOR_SIZE
        + I64_SIZE
        + I64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE
        + U64_SIZE;
    ...
}
```

```
pub struct PropertyState {
    pub private_sale_info: PrivateSaleInfo,
    pub public_sale_info: PublicSaleInfo,
    pub property_mint_account: Pubkey,
    pub bump: [u8; 1],
}
```

RECOMMENDATION

Remove `DISCRIMINATOR_SIZE` from `PrivateSaleInfo` and `PublicSaleInfo` to reduce rent cost and redundant storage.

UPDATES

- *Jan 12, 2024*: This issue has been acknowledged by the Propeasy Labs team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 12, 2024*	Public Report	Verichains Lab

Table 2. Report versions history