// Solana Program Security Assessment

12.05.2024 - 12.19.2024

Program
Proov Protocol

HALBORN

Program - Proov Protocol

Prepared by: HALBORN

Last Updated 03/05/2025

Date of Engagement by: December 5th, 2024 - December 19th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
8	1	0	0	4	3

TABLE OF CONTENTS

1. Introduction

`Proov Protocol` team engaged `Halborn` to conduct a security assessment on their **Slot and Vault Solana programs** beginning on December 5, 2024, and ending on December 19, 2024. The security assessment was scoped to the Solana Programs, provided in a zip file via e-mail. Further details and SHA256 checksum can be found in the Scope section of this report.

`Proov Protocol` is a decentralized platform for fair and efficient casino games that consists of two Solana on-chain `Slot` and `Vault` programs and additional game programs that are out-of-scope of this assessment.

- The `Vault` program is designed to securely store user funds, lock them when necessary, and release them under specific conditions, such as when a user loses a game to an approved game contract. It supports any SPL tokens and interacts with pre-approved game contracts to manage user funds according to game outcomes.
- The `Slot` program is designed to manage game logic and track user metrics such as total wagered and total won amounts. It interacts closely with the Vault Contract to handle fund settlements based on game outcomes.

2. Assessment Summary

`Halborn` was provided 2 weeks for the engagement and assigned one full-time security engineer to review the security of the Solana Programs in scope. The engineer is a blockchain and smart contract security expert with advanced smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the Solana Programs.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly solved by the `Proov Protocol team`. The main ones were the following:

- Make sure that the Token2022 extensions are correctly taken into account.
- Introduce a secure authority transfer where the new authority is required to provide signature.
- Ensure that the correct token program is passed during token allowance initialization.
- Pass accounts that are not read from or written to only as instruction parameters.

3. Test Approach And Methodology

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Scanning dependencies for known vulnerabilities (`cargo audit`).
- Local runtime testing (`anchor test`)

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker’s control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9

SEVERITY	SCORE VALUE RANGE
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: Code provided via email

(b) Assessed Commit ID: 0efb1a0

(c) Items in scope:

- ./programs/slot/Cargo.toml
- ./programs/slot/Xargo.toml
- ./programs/slot/src/instructions/initialize.rs
- ./programs/slot/src/instructions/settle_user_gain.rs
- ./programs/slot/src/instructions/transfer_authority.rs
- ./programs/slot/src/instructions/mod.rs
- ./programs/slot/src/instructions/initialize_token_allowance.rs
- ./programs/slot/src/instructions/withdraw_from_bankroll.rs
- ./programs/slot/src/instructions/sync_config.rs
- ./programs/slot/src/instructions/settle_user_charge.rs
- ./programs/slot/src/instructions/settle_user_payout.rs
- ./programs/slot/src/instructions/settle_user_loss.rs
- ./programs/slot/src/error.rs
- ./programs/slot/src/lib.rs
- ./programs/slot/src/configuration.rs
- ./programs/slot/src/state/mod.rs
- ./programs/slot/src/state/slot_config.rs
- ./programs/slot/src/state/user_token_allowance.rs
- ./programs/slot/src/state/slot_state.rs
- ./programs/slot/src/model.rs
- ./programs/slot/src/utils.rs
- ./programs/vault/Cargo.toml
- ./programs/vault/Xargo.toml
- ./programs/vault/src/instructions/set_lock.rs

- ./programs/vault/src/instructions/settle_loss_spl.rs
- ./programs/vault/src/instructions/recover_spl_by_authority.rs
- ./programs/vault/src/instructions/add_game_contract.rs
- ./programs/vault/src/instructions/approve_address.rs
- ./programs/vault/src/instructions/withdraw_spl.rs
- ./programs/vault/src/instructions/withdraw.rs
- ./programs/vault/src/instructions/remove_authority.rs
- ./programs/vault/src/instructions/initialize.rs
- ./programs/vault/src/instructions/add_authority.rs
- ./programs/vault/src/instructions/mod.rs
- ./programs/vault/src/instructions/remove_approval.rs
- ./programs/vault/src/instructions/initialize_user_deposit.rs
- ./programs/vault/src/instructions/close_spl_token_account.rs
- ./programs/vault/src/instructions/initialize_user_by_authority.rs
- ./programs/vault/src/instructions/remove_game_contract.rs
- ./programs/vault/src/instructions/initialize_user.rs
- ./programs/vault/src/instructions/withdraw_spl_by_authority.rs
- ./programs/vault/src/error.rs
- ./programs/vault/src/lib.rs
- ./programs/vault/src/state/vault_state.rs
- ./programs/vault/src/state/mod.rs
- ./programs/vault/src/state/user_deposit.rs
- ./programs/vault/src/model.rs
- ./common/Cargo.toml
- ./common/src/lib.rs
- ./Cargo.toml
- ./Anchor.toml

Out-of-Scope: Third party dependencies and economic attacks.

FILES AND REPOSITORY

(a) Repository: Code provided via email

(b) Assessed Commit ID: 3333f1c

(c) Items in scope:

- ./programs/slot/Cargo.toml
- ./programs/slot/Xargo.toml
- ./programs/slot/src/instructions/initialize.rs
- ./programs/slot/src/instructions/settle_user_gain.rs
- ./programs/slot/src/instructions/transfer_authority.rs
- ./programs/slot/src/instructions/mod.rs
- ./programs/slot/src/instructions/initialize_token_allowance.rs
- ./programs/slot/src/instructions/withdraw_from_bankroll.rs
- ./programs/slot/src/instructions/sync_config.rs
- ./programs/slot/src/instructions/settle_user_charge.rs
- ./programs/slot/src/instructions/settle_user_payout.rs
- ./programs/slot/src/instructions/settle_user_loss.rs
- ./programs/slot/src/error.rs
- ./programs/slot/src/lib.rs
- ./programs/slot/src/configuration.rs
- ./programs/slot/src/state/mod.rs
- ./programs/slot/src/state/slot_config.rs
- ./programs/slot/src/state/user_token_allowance.rs
- ./programs/slot/src/state/slot_state.rs
- ./programs/slot/src/model.rs
- ./programs/slot/src/utils.rs
- ./programs/vault/Cargo.toml
- ./programs/vault/Xargo.toml
- ./programs/vault/src/instructions/set_lock.rs
- ./programs/vault/src/instructions/settle_loss_spl.rs
- ./programs/vault/src/instructions/recover_spl_by_authority.rs

- ./programs/vault/src/instructions/add_game_contract.rs
- ./programs/vault/src/instructions/approve_address.rs
- ./programs/vault/src/instructions/withdraw_spl.rs
- ./programs/vault/src/instructions/withdraw.rs
- ./programs/vault/src/instructions/remove_authority.rs
- ./programs/vault/src/instructions/initialize.rs
- ./programs/vault/src/instructions/add_authority.rs
- ./programs/vault/src/instructions/mod.rs
- ./programs/vault/src/instructions/remove_approval.rs
- ./programs/vault/src/instructions/initialize_user_deposit.rs
- ./programs/vault/src/instructions/close_spl_token_account.rs
- ./programs/vault/src/instructions/initialize_user_by_authority.rs
- ./programs/vault/src/instructions/remove_game_contract.rs
- ./programs/vault/src/instructions/initialize_user.rs
- ./programs/vault/src/instructions/withdraw_spl_by_authority.rs
- ./programs/vault/src/error.rs
- ./programs/vault/src/lib.rs
- ./programs/vault/src/state/vault_state.rs
- ./programs/vault/src/state/mod.rs
- ./programs/vault/src/state/user_deposit.rs
- ./programs/vault/src/model.rs
- ./common/Cargo.toml
- ./common/src/lib.rs
- ./Cargo.toml
- ./Anchor.toml

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- 3333f1c

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	4	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
TOKEN EXTENSIONS MAY DISABLE VAULT LOCKING MECHANISM	CRITICAL	SOLVED - 01/06/2025
RISK OF FRONT-RUNNING THE INITIALIZATION	LOW	RISK ACCEPTED - 01/06/2025
MULTI-STEP AUTHORITY TRANSFER NOT ENFORCED	LOW	SOLVED - 01/06/2025
UNNECESSARY PASSING OF FULL ACCOUNTINFO STRUCTURES	LOW	SOLVED - 01/06/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
RISK OF INCORRECT USER VAULT ATA DETERMINATION	LOW	SOLVED - 01/06/2025
RELIANCE ON OFF-CHAIN CRITICAL LOGIC	INFORMATIONAL	ACKNOWLEDGED - 01/06/2025
PASSING BUMPS AS INSTRUCTION PARAMETERS	INFORMATIONAL	SOLVED - 01/06/2025
LACK OF SPECIFIC ERROR CODES	INFORMATIONAL	SOLVED - 01/06/2025

7. FINDINGS & TECH DETAILS

7.1 TOKEN EXTENSIONS MAY DISABLE VAULT LOCKING MECHANISM

// CRITICAL

Description

The instruction `initialize_token_allowance` allows anyone to initialize the `UserTokenAllowance` account on behalf of any other user. The program is designed to support arbitrary SPL tokens and the `initialize_token_allowance` instruction can be used to allow any token mint to be provided including tokens that might have various Token2022 extensions enabled.

Both programs however fail to account for potential token extensions, which can lead to operational inconsistencies when such tokens are used. For instance, tokens with the `TransferFeeConfig` extension impose fees on transfers, potentially altering the final transferred amount and causing discrepancies.

Additionally, the `PermanentDelegate` extension grants unrestricted permissions to transfer and burn tokens from any account associated with the given mint. This undermines the vault locking mechanism, effectively disabling it and potentially compromising the program's security guarantees.

user_token_allowance.rs:

```rust
pub fn init(&mut self, user: Pubkey, token_mint: Pubkey) {
    self.user = user;
    self.token_mint = token_mint;
    self.total_wagered = 0;
    self.total_won = 0;
    self.nonce = 0;
    self.total_payout = 0;
    self.total_charge = 0;
    self.extra_space = [0u8; 32];
}
```

initialize_token_allowance.rs:

```rust
#[derive(Accounts)]
pub struct InitializeTokenAllowance<'info> {
    #[account(
        init,
        seeds = [common::USER_TOKEN_ALLOWANCE_SEED.as_bytes(), user.key().as_ref(), token_mint.key().as_ref
        bump,
        payer = signer,
        space = 8 + UserTokenAllowance::INIT_SPACE,
    )]
    pub user_token_allowance: Account<'info, UserTokenAllowance>,

    #[account()]
    /// CHECK: reference to the user account
    pub user: AccountInfo<'info>,

    #[account(mut)]
    pub signer: Signer<'info>,

    pub token_mint: InterfaceAccount<'info, Mint>,

    /// CHECK: reference to the token program
    pub token_program: AccountInfo<'info>,

    pub system_program: Program<'info, System>,
}
```

Proof of Concept

1. Initialize the vault program.
2. Add game contract.
3. Initialize user.

4. Create token mint with PermanentDelegate extension enabled.
5. Deposit funds to the user vault.
6. Set vault lock.
7. Invoke the withdraw_spl instruction before the lock expires. (this will fail as expected)
8. Invoke the transferChecked instruction with delegate before the lock expires -> this will pass and withdraw locked funds.

```
it.only("Should fail to withdraw locked spl 2022 tokens after the second lock with delegate", async () => {
    await setLock(user1, 10000);
    const [userDepositPda, bump] = getUserDepositPDA(user1.publicKey);
    const depositAta = await getAssociatedTokenAddress(
      SPL_TOKEN_2022_MINT.publicKey,
      userDepositPda,
      true,
      TOKEN_2022_PROGRAM_ID
    );

    let balance = await PROVIDER.connection.getTokenAccountBalance(depositAta);
    console.log("user vault balance before: " + balance.value.amount);
    console.log(">>> withdraw_spl instruction");
    await assertPromiseThrows(withdrawSpl2022(user1, 1000));
    balance = await PROVIDER.connection.getTokenAccountBalance(depositAta);
    console.log("user vault balance after: " + balance.value.amount);
    console.log("\n-----------------------\n");
    balance = await PROVIDER.connection.getTokenAccountBalance(depositAta);
    console.log("user vault balance before: " + balance.value.amount);
    console.log(">>> transfer by the delegate");
    await assertPromiseThrows(withdrawSpl2022Delegate(user1, 1000));
    balance = await PROVIDER.connection.getTokenAccountBalance(depositAta);
    console.log("user vault balance after: " + balance.value.amount);
  });

export async function withdrawSpl2022Delegate(
  user: anchor.web3.Keypair,
  amount: number
) {
  const [userDepositPda, bump] = getUserDepositPDA(user.publicKey);
  const userAta = await getAssociatedTokenAddress(
    SPL_TOKEN_2022_MINT.publicKey,
    user.publicKey,
    undefined,
    TOKEN_2022_PROGRAM_ID
  );
  const depositAta = await getAssociatedTokenAddress(
    SPL_TOKEN_2022_MINT.publicKey,
    userDepositPda,
    true,
    TOKEN_2022_PROGRAM_ID
  );
  await transferChecked(
    PROVIDER.connection,
    PERMANENT_DELEGATE,
    depositAta,
    SPL_TOKEN_2022_MINT.publicKey,
    userAta,
    PERMANENT_DELEGATE,
    amount,
    6,
    undefined,
    undefined,
    TOKEN_2022_PROGRAM_ID
  );
}
```

```
  vault_withdraw_spl_2022
user vault balance before: 1000
>>> withdraw_spl instruction
user vault balance after: 1000

------------------------

user vault balance before: 1000
>>> transfer by the delegate
balance before: 1000
balance after: 0
    1) Should fail to withdraw locked spl 2022 tokens after the second lock with delegate


  0 passing (8s)
  1 failing

  1) vault_withdraw_spl_2022
       Should fail to withdraw locked spl 2022 tokens after the second lock with delegate:
     AssertionError: Expected error
      at /Users/adam/Work/Audits/level3Labs/contract/solana/tests/utils.ts:143:9
      at Generator.next (<anonymous>)
      at fulfilled (tests/utils.ts:28:58)
      at processTicksAndRejections (node:internal/process/task_queues:95:5)
```

BVSS

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:C/Y:N/R:N/S:U (10.0)

Recommendation

It is recommended to ensure that the allowed SPL tokens do not have enabled extensions that might compromise program security guarantees or data consistency. This can be achieved by either maintaining a whitelist of known safe tokens or programmatically verifying the absence of enabled extensions in the token configuration.

Remediation

SOLVED: The `Proov Protocol` **team** solved this issue by implementing a whitelist of permitted extensions, effectively blocking the use of token mints with potentially harmful extensions.

Remediation Hash

3333f1c4e5c59c50e8e2e6d87020a817f9eeb647fed4304e4f787f3a25d770d5

7.2 RISK OF FRONT-RUNNING THE INITIALIZATION

// LOW

Description

The instructions `slot::initialize` and `vault::initialize` do not verify that the signing authority is a specific predefined key and thus an attacker might front-run the initialization and invoke the initialize instruction on behalf of the intended user and take control of the program.

./vault/src/instructions/initialize.rs:

```rust
pub struct Initialize<'info> {
    #[account(
        init,
        seeds = [common::VAULT_STATE_SEED.as_bytes()],
        bump,
        payer = deployer,
        space = 8 + VaultState::INIT_SPACE,
    )]
    pub vault_state: Account<'info, VaultState>,

    #[account(mut)]
    pub deployer: Signer<'info>,

    pub system_program: Program<'info, System>,
}
```

./slot/src/instructions/initialize.rs:

```rust
pub struct Initialize<'info> {
    #[account(
        init,
        seeds = [common::SLOT_STATE_SEED.as_bytes()],
        bump,
        payer = authority,
        space = 8 + SlotState::INIT_SPACE,
    )]
    pub slot_state: Account<'info, SlotState>,

    #[account(
        init,
        seeds = [common::SLOT_CONFIG_SEED.as_bytes()],
        bump,
        payer = authority,
        space = 8 + SlotConfig::INIT_SPACE,
    )]
    pub slot_config: Account<'info, SlotConfig>,

    #[account(mut)]
    pub authority: Signer<'info>,

    pub system_program: Program<'info, System>,
}
```

BVSS

AO:A/AC:L/AX:L/C:C/I:M/A:N/D:N/Y:N/R:F/S:U (2.8)

Recommendation

It is recommended to verify the address of the signing initialization authorities being the expected addresses.

Remediation

RISK ACCEPTED: The `Proov Protocol` **team** accepted the risk of this finding as the program will not be used until the authority is set, so the risk is very limited.

7.3 MULTI-STEP AUTHORITY TRANSFER NOT ENFORCED

// LOW

Description

The `transfer_authority` instruction does not enforce a multi-step authority transfer process, nor does it require the new authority to provide its signature. As a result, accidentally transferring authority to an incorrect public key—especially one without access to the corresponding private key—would lead to a loss of control over the protocol.

transfer_authority.rs:

```rust
pub fn transfer_authority(ctx: Context<TransferAuthority>, new_authority: Pubkey) -> Result<()> {
    let slot_state = &mut ctx.accounts.slot_state;

    slot_state.set_authority(new_authority);

    msg!(
        "Transferred authority from {} to {}",
        &ctx.accounts.authority.key,
        new_authority,
    );
    Ok(())
}
```

BVSS

AO:S/AC:L/AX:L/C:C/I:N/A:M/D:C/Y:N/R:N/S:U (2.8)

Recommendation

To address this issue, it is recommended to require the new authority to provide its signature during the transfer process, whether in a single or multi-step instruction. This ensures that the new authority retains control over the protocol and prevents accidental transfers.

Remediation

SOLVED: The `Proov Protocol` **team** solved this issue by requiring the new authority to provide its signature during the transfer process.

Remediation Hash

3333f1c4e5c59c50e8e2e6d87020a817f9eeb647fed4304e4f787f3a25d770d5

7.4 UNNECESSARY PASSING OF FULL ACCOUNTINFO STRUCTURES

// LOW

Description

The instructions `settle_user_*`, `settle_loss_spl` and `remove_approval` expect the `user` account to be passed as `AccountInfo` account. However, these instructions do not need to read or write any data to the `user` account and need only the account's public key to derive relative PDAs.

Passing the `user` account as **AccountInfo** only increases transaction size and increases overhead, resulting in higher transaction fees compared to passing only the user's public key as an instruction parameter.

remove_approval.rs:

```rust
#[derive(Accounts)]
pub struct RemoveApproval<'info> {
    #[account(
        seeds = [common::VAULT_STATE_SEED.as_bytes()],
        bump,
    )]
    pub vault_state: Account<'info, VaultState>,

    #[account(
        mut,
        seeds = [common::USER_DEPOSIT_SEED.as_bytes(), user.key().as_ref()],
        bump,
    )]
    pub user_deposit: Account<'info, UserDeposit>,

    #[account()]
    /// CHECK: reference to the user account
    pub user: AccountInfo<'info>,
    // We allow to remove approvals only by the authority, cause settlement is done asynchronously.
    // This is to prevent the user from removing the approval before the settlement is done onchain.
    #[account(
        constraint = vault_state.is_authorized(* authority.key) @ ValidationError::Unauthorized,
    )]
    pub authority: Signer<'info>,
}
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (2.5)

Recommendation

It is recommended to pass the public keys of accounts as instruction parameters when those accounts are not being read from or written to. This approach avoids unnecessary overhead.

Remediation

SOLVED: The`Proov Protocol` **team** solved this issue by passing public keys as instruction parameters for accounts that are not being read from or written to.

Remediation Hash

3333f1c4e5c59c50e8e2e6d87020a817f9eeb647fed4304e4f787f3a25d770d5

7.5 RISK OF INCORRECT USER VAULT ATA DETERMINATION

// LOW

Description

The `initialize_token_allowance` instruction does not validate that the token program matches the owner of the mint account. These accounts are later used to compute the user's deposit vault's ATA. If an incorrect token program is passed, it results in an invalid ATA calculation. Despite this, the instruction completes successfully, returning an incorrect user deposit vault ATA. This discrepancy can lead to backend inconsistencies.

initialize_token_allowance.rs:

```rust
pub struct InitializeTokenAllowance<'info> {
    #[account(
        init,
        seeds = [common::USER_TOKEN_ALLOWANCE_SEED.as_bytes(), user.key().as_ref(), token_mint.key().as_ref
        bump,
        payer = signer,
        space = 8 + UserTokenAllowance::INIT_SPACE,
    )]
    pub user_token_allowance: Account<'info, UserTokenAllowance>,

    #[account()]
    /// CHECK: reference to the user account
    pub user: AccountInfo<'info>,

    #[account(mut)]
    pub signer: Signer<'info>,

    pub token_mint: InterfaceAccount<'info, Mint>,

    /// CHECK: reference to the token program
    pub token_program: AccountInfo<'info>,

    pub system_program: Program<'info, System>,
}
```

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:L/Y:N/R:F/S:U (2.0)

Recommendation

To address this issue, it is recommended to ensure that the passed token program corresponds to the owner of the passed mint account. This can be done by adding the following anchor constraint`#[account(mint::token_program = token_program )]` to the passed mint account.

Remediation

SOLVED: The `Proov Protocol` **team** solved this issue by adding a constraint that ensures that the provided token programs corresponds to the provided mint account owner.

Remediation Hash

3333f1c4e5c59c50e8e2e6d87020a817f9eeb647fed4304e4f787f3a25d770d5

7.6 RELIANCE ON OFF-CHAIN CRITICAL LOGIC

// INFORMATIONAL

Description

The on-chain programs are used for settlement only. Critical actions, such as initiating and accepting bets, determining game outcomes, and settling those outcomes, are handled by off-chain backend servers that are out of scope of this assessment. While the settlement instructions require signatures from all involved backend servers, users must trust that the core logic is correctly implemented and that the backend servers will not collude or misuse their authority.

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:H/Y:N/R:N/S:U (1.5)

Recommendation

It is recommended to provide detailed documentation outlining the high-level architecture and functionality of the entire protocol, including the role of backend servers. This transparency will enable users to make informed decisions about whether to trust the platform.

Remediation

ACKNOWLEDGED: The `Proov Protocol` **team** acknowledged this finding as this reliance on off-chain logic is by design. The detailed documentation will be provided.

7.7 PASSING BUMPS AS INSTRUCTION PARAMETERS

// INFORMATIONAL

Description

All `settle_user_*` instructions and `withdraw_from_bankroll` instruction require PDA bumps to be passed as instruction parameters. This is however redundant and not necessary, as the bumps can be read from Anchor's context.

While this is not a security issue in this specific implementation, allowing users to provide custom bump values introduces risks. Notably, multiple valid bump values can exist for the same seed combination. If mishandled, this could lead to unintended behavior or security vulnerabilities. For this reason, it is generally advised to avoid passing bump values directly in user-facing instructions.

settle_user_gain.rs:

```rust
pub fn settle_user_gain(
    ctx: Context<SettleUserGain>,
    start_nonce: u64,
    next_nonce: u64,
    wagered: u64,
    won: u64,
    decimals: u8,
    slot_state_bump: u8,
) -> Result<SettlementResponse> {
```

Score

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.0)

Recommendation

To address this issue, it is recommended to avoid passing PDA bumps as instruction parameters and rather user Anchor's `ctx.bumps` to get the correct bump value.

Remediation

SOLVED: The `Proov Protocol` **team** solved this issue by removing the redundant bump instruction parameters and using the bumps provided in the Anchor's context.

Remediation Hash

3333f1c4e5c59c50e8e2e6d87020a817f9eeb647fed4304e4f787f3a25d770d5

7.8 LACK OF SPECIFIC ERROR CODES

// INFORMATIONAL

Description

The programs extensively use the generic error code `ValidationError::NotAllowed`. However, this error code is applied in diverse scenarios where it fails to describe the specific issue encountered. As a result, it provides insufficient information to users, making it challenging for them to understand and resolve the error effectively.

By using more descriptive and context-specific error codes, the programs can improve user experience, facilitate debugging, and enhance the clarity of error messages.

vault_state.rs:

```rust
pub fn add_authority(&mut self, authority: Pubkey) -> Result<()> {
    require!(!self.is_authorized(authority), ValidationError::NotAllowed);
    require!(
        self.authorities.len() < MAX_AUTHORITIES,
        ValidationError::NotAllowed
    );
```

```rust
require!(
    !self.is_game_contract_approved(contract),
    ValidationError::NotAllowed
);
require!(
    self.game_contracts.len() < MAX_GAME_CONTRACTS,
    ValidationError::NotAllowed
);
```

settle_user_charge.rs:

```rust
pub fn settle_user_charge(
    ctx: Context<SettleUserCharge>,
    current_total_charge: u64,
    charge: u64,
    user_token_allowance_bump: u8,
    user_deposit_bump: u8,
    decimals: u8,
) -> Result<SettlementResponse> {
    utils::validate_multisig(ctx.remaining_accounts)?;
    require!(charge > 0, ValidationError::NotAllowed)
```

Score

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.0)

Recommendation

It is recommended to introduce specific error codes and messages for various error scenarios.

Remediation

SOLVED: The `Proov Protocol` **team** resolved this issue by introducing new specific error codes and messages for various error scenarios.

Remediation Hash

3333f1c4e5c59c50e8e2e6d87020a817f9eeb647fed4304e4f787f3a25d770d5

8. AUTOMATED TESTING

STATIC ANALYSIS REPORT

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Cargo Audit Results

ID	CRATE	DESCRIPTION
RUSTSEC-2024-0344	curve25519-dalek	Timing variability in `curve25519-dalek`'s `Scalar29::sub`/`Scalar52::sub`
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project’s integrity and addressing potential vulnerabilities introduced by code modifications.