Smart Contract

Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.12.05, the SlowMist security team received the Sunperp Dex team's security audit application for sunperp-sol-vault, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

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 was considered during the audit of the smart contract:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Unsafe External Call Audit
- Design Logic Audit
- Scoping and Declarations Audit
- Account substitution attack Audit
- Malicious Event Log Audit

3 Project Overview

3.1 Project Introduction

This is the version of the Sunperp protocol deployed on Solana, allowing users to deposit native SOL tokens as well as specific SPL tokens for custody. Withdrawals require verification of Ed25519 signatures from a trusted set of Truth Holders and are subject to an hourly withdrawal limit; if this limit is exceeded, the protocol will pause withdrawals, whereas trusted counter parties are not subject to such limits.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Lack of initialization permission control leads to premature initialization	Reordering Vulnerability	Medium	Acknowledged
N2	Missing pre-state check for state updates	Others	Suggestion	Fixed
N3	Unable to proceed with withdrawals when the claim history slots are full	Others	Information	Acknowledged
N4	Trusting externally supplied bump seeds	Design Logic Audit	Suggestion	Fixed
N5	Unlimited withdrawals allowed for counter_parties	Others	Information	Acknowledged
N6	Missing Chain ID in signature hash construction enables cross-chain replay	Replay Vulnerability	Low	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/jerkoyz/sunperp-sol-vault

commit: e51e904c1726ae99c2d3d35a866aaf1a19f37960

Fixed Version:

https://github.com/jerkoyz/sunperp-sol-vault

commit: f912a039f14c30188251bf51c390f4e0fc91b445

Audit Scope:

```
./programs/sunperp-sol-vault/src/
├── constants.rs
├── deposit.rs
├── errors.rs
├── events.rs
├── init.rs
├── lib.rs
├── sol.rs
├── token.rs
├── utils
│   ├── ed25519.rs
│   └── mod.rs
└── withdraw.rs
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist security team analyzed the visibility of major contracts during the audit, the result as follows:

init			
Function Name	Account check coverage	Auth Signer	Parameters Check
initialize	2/3	-	6/6
unstop	3/3	authority	-
stop	3/3	authority/stopper	-

init			
update_global_withdraw_enabled	3/3	authority/pauser	0/1
add_operator	3/3	authority	1/1
remove_operator	3/3	authority	1/1
add_counter_party	3/3	authority	1/1
remove_counter_party	3/3	authority	1/1
is_operator	-	-	2/2
is_counter_party	-	-	2/2
add_truth_holder	3/3	authority	1/1
remove_truth_holder	3/3	authority	1/1
change_authority	3/3	authority	0/1
change_stopper	3/3	authority	0/1
change_pauser	3/3	authority	0/1
change_business	3/3	authority	0/1

sol			
Function Name	Account check coverage	Auth Signer	Parameters Check
add_sol	4/4	authority/business	0/3
update_sol_enabled	4/4	authority/business	0/1
update_sol_hourly_limit	4/4	authority/business	0/1

SolVault			
Function Name	Account check coverage	Auth Signer	Parameters Check
has_claim_history_item	-	-	1/1

SolVault

add_claim_history_item	-	-	3/3

token

Function Name	Account check coverage	Auth Signer	Parameters Check
add_token	8/9	authority/business	0/3
update_token_enabled	8/9	authority/business	0/1
update_token_hourly_limit	8/9	authority/business	0/1

Bank

Function Name	Account check coverage	Auth Signer	Parameters Check
has_claim_history_item	-	-	1/1
add_claim_history_item	-	-	3/3

deposit

Function Name	Account check coverage	Auth Signer	Parameters Check
deposit_sol	3/4	-	1/1
deposit_token	8/10	-	1/1

withdraw

Function Name	Account check coverage	Auth Signer	Parameters Check
check_operator_enabled_for_sol	-	-	3/3
check_operator_enabled_for_token	-	-	3/3
check_deadline	-	-	2/2
build_sol_msg_hash	-	-	0/6

withdraw			
build_token_msg_hash	-	-	0/9
verify_truth_holder_signatures	-	-	4/4
withdraw_sol_by_signature	5/6	operator	4/4
do_withdraw_sol	-	-	4/5
withdraw_sol_to_counter_party	5/6	operator	4/4
withdraw_token_by_signature	10/11	operator	4/4
check_token_amount	-	-	1/2
transfer_token	-	-	0/7
do_withdraw_token	-	-	7/8
withdraw_token_to_counter_party	10/11	operator	4/4

4.3 Vulnerability Summary

[N1] [Medium] Lack of initialization permission control leads to premature initialization

Category: Reordering Vulnerability

Content

In init.rs under Initialize, the admin account is initialized using a fixed seed `constants::ADMIN`. Since there is no permission restriction on the signer (such as verifying a specific address), an attacker can invoke the `initialize` instruction before the deployer. This results in the admin account being created and controlled by the attacker (who becomes the authority), forcing the project team to redeploy the contract.

Code location: programs/sunperp-sol-vault/src/init.rs#L440

```
#[derive(Accounts)]
pub struct Initialize<'info> {
    //#[account(mut, address = crate::ID)]
    #[account(mut)]
    pub signer: Signer<'info>,
    #[account(init, payer = signer, space = 8 + std::mem::size_of::< Admin > (), seeds
= [constants::ADMIN.as_bytes()], bump)]
    pub admin: AccountLoader<'info, Admin>,
    pub system_program: Program<'info, System>,
}
```

Solution

Add an address constraint for the signer in the `Initialize` struct, restricting the initialization operation to a predefined deployer address only.

Status

Acknowledged; After communicating with the project team, they indicated that if the program is preemptively initialized, they will abandon it and deploy a new one.

[N2] [Suggestion] Missing pre-state check for state updates

Category: Others

Content

In init.rs, the stop and unstop functions directly overwrite the value of admin.stopped without verifying whether a state change is actually required (for example, stop should only be called when the current state is unstopped, and unstop should only be called when it is stopped). Although this does not pose a security risk, the absence of state validation may trigger redundant state-change events, potentially confusing off-chain monitoring systems.

Code location: programs/sunperp-sol-vault/src/init.rs#L81-L111

```
pub fn unstop(ctx: Context<UpdateAdmin>) -> Result<()> {
    ...
}

pub fn stop(ctx: Context<Stop>) -> Result<()> {
    ...
}
```

Solution

It is recommended to add checks for the stopped state within the stop and unstop functions.

Status

Fixed

[N3] [Information] Unable to proceed with withdrawals when the claim history slots are full

Category: Others

Content

In the add_claim_history_item functions within sol.rs and token.rs, a fixed-size array CLAIM_HISTORY_SIZE is used to store processed withdrawal request IDs. When all slots in this array are filled and none have expired, new withdrawal requests cannot be recorded, causing add_claim_history_item to return false. As a result, legitimate withdrawal transactions fail.

Code location:

programs/sunperp-sol-vault/src/sol.rs#L124

programs/sunperp-sol-vault/src/token.rs#L156

```
   pub fn add_claim_history_item(&mut self, idempotent: u64, dead_line: u32,
current_timestamp: u32) -> bool {
        for n in 0..CLAIM_HISTORY_SIZE {
            // clean outdated items
            if self.dead_line[n] > 0 && self.dead_line[n] <= current_timestamp - 120
{
                self.idempotent[n] = 0;
                self.dead_line[n] = 0;
            }
            // check if there is any empty slot
            if self.idempotent[n] == 0 && self.dead_line[n] == 0 {
                self.idempotent[n] = idempotent;
                self.dead_line[n] = dead_line;
                return true;
            }
        }
        return false;
    }
```

Solution

N/A

Status

Acknowledged

[N4] [Suggestion] Trusting externally supplied bump seeds

Category: Design Logic Audit

Content

In the add_token function of token.rs and the add_sol function of sol.rs, the parameters token_vault_authority_bump and sol_vault_bump are provided by the client and stored directly. If an incorrect bump value is passed in and saved, subsequent PDA signature operations (such as invoke_signed) using that bump will fail signature verification, thereby breaking the contract’s functionality.

Code location:

programs/sunperp-sol-vault/src/token.rs#L18

```
pub fn add_token(ctx: Context<AddToken>, enabled: bool, token_vault_authority_bump:
u8, hourly_limit: u64) -> Result<()> {
    ...
    bank.token_vault_authority_bump = token_vault_authority_bump;
    ...
}
```

programs/sunperp-sol-vault/src/sol.rs#L14

```
pub fn add_sol(ctx: Context<AddSol>, enabled: bool, sol_vault_bump: u8, hourly_limit:
u64) -> Result<()> {
    ...
    admin.sol_vault_bump = sol_vault_bump;
    ...
}
```

Solution

Do not read the bump value from external parameters. It is recommended to obtain the correct canonical bump directly from Anchor’s context validation mechanism using `ctx.bumps.get`.

Status

Fixed

[N5] [Information] Unlimited withdrawals allowed for counter_parties

Category: Others

Content

In the withdraw_token_to_counter_party and withdraw_sol_to_counter_party functions within withdraw.rs, as long as the recipient is verified as a legitimate counter_party, the withdrawal logic skips the hourly_limit check. In contrast, regular withdrawal operations trigger a global pause to halt withdrawals once the limit is exceeded.

Code location: programs/sunperp-sol-vault/src/withdraw.rs#L187-L195,L233,L407-L415,L451

```
fn do_withdraw_sol/do_withdraw_token<'info>(...) -> Result<()> {
    ...
    let cursor = current_timestamp / (60 * 60);
    let per_hour_value;
    if sol_vault.claim_per_hour_cursor == cursor {
        per_hour_value =
sol_vault.claim_per_hour_value.checked_add(amount).ok_or(ErrorCode::ArithmeticOverflow
)?;
    } else {
        per_hour_value = amount;
    }
    if per_hour_value > sol_vault.hourly_limit {
    ...
}

pub fn withdraw_sol_to_counter_party/withdraw_token_to_counter_party(
    ...
) -> Result<()> {
    ...
}
```

Solution

N/A

Status

Acknowledged

[N6] [Low] Missing Chain ID in signature hash construction enables cross-chain replay

Category: Replay Vulnerability

Content

In the build_sol_msg_hash and build_token_msg_hash functions within withdraw.rs, the message structure used to generate the withdrawal signature hash includes only business fields (such as amount, receiver, and idempotent), but lacks any binding to a chain identifier (Chain ID or Genesis Hash). If the contract is deployed on multiple networks—

such as Solana mainnet, testnet, or forked chains—and the same truth_holders are used for signature verification across them, an attacker could capture a valid signature from one chain and replay it on another, resulting in unintended withdrawals.

Code location: programs/sunperp-sol-vault/src/withdraw.rs#L45-L91

```
fn build_sol_msg_hash(...) -> [u8; 32] {
    keccak(&[
        ...
    ]).to_bytes()
}

fn build_token_msg_hash(
    ...
) -> [u8; 32] {
    keccak(&[
        ...
    ]).to_bytes()
}
```

Solution

Include Solana’s Genesis Hash or other uniquely identifying information of the current chain in the message data when constructing the hash, ensuring that the signature is valid only on the intended network.

Status

Acknowledged; After communicating with the project team, they have stated that they will not deploy this program on any other testnets or fork chains.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002512100001	SlowMist Security Team	2025.12.05 - 2025.12.10	Low Risk

Summary conclusion: The SlowMist security team uses a manual and the SlowMist team's analysis tool to audit the project. During the audit work, we found 1 medium risk, 1 low risk, 2 suggestions, and 2 information. All the findings were fixed or acknowledged. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist