CropperFinance

Farm Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 13th, 2021 - October 5th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/13/2021	Piotr Cielas
0.2	Document Edits	09/19/2021	Piotr Cielas
0.3	Document Edits	09/30/2021	Piotr Cielas
0.4	Final Draft	10/5/2021	Piotr Cielas
1.0	Remediation Plan	10/12/2021	Piotr Cielas
1.1	Remediation Plan Review	10/13/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

CropperFinance is introducing permissionless yield farming on Solana, enabling SPL project builders to connect their liquidity to the platform, set up the total supply that will be allocated to farming, decide the weekly emission schedule, and launch their yield farming in a few clicks.

CropperFinance engaged Halborn to conduct a security assessment on the AMM and Farming programs beginning on September 13th, 2021 and ending October 5th, 2021. This security assessment was scoped to the Farm repositories and an audit of the security risk and implications regarding the changes introduced by the development team at CropperFinance prior to its production release shortly following the assessments deadline.

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned one full time security engineer to audit the security of the program. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that program functions are intended.
- Identify potential security issues with the program.

Though this security audit's outcome is **satisfactory**, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure Solana program development.

In summary, Halborn identified few security risks that were addressed by CropperFinance team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program audit. While manual testing is recommended to uncover flaws in logic, process, 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 audit:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walkthrough.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Fuzz testing. (Halborn custom fuzzing tool)
- Checking the test coverage. (cargo tarpaulin)
- Scanning of Rust files for vulnerabilities.(cargo audit)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It’s quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

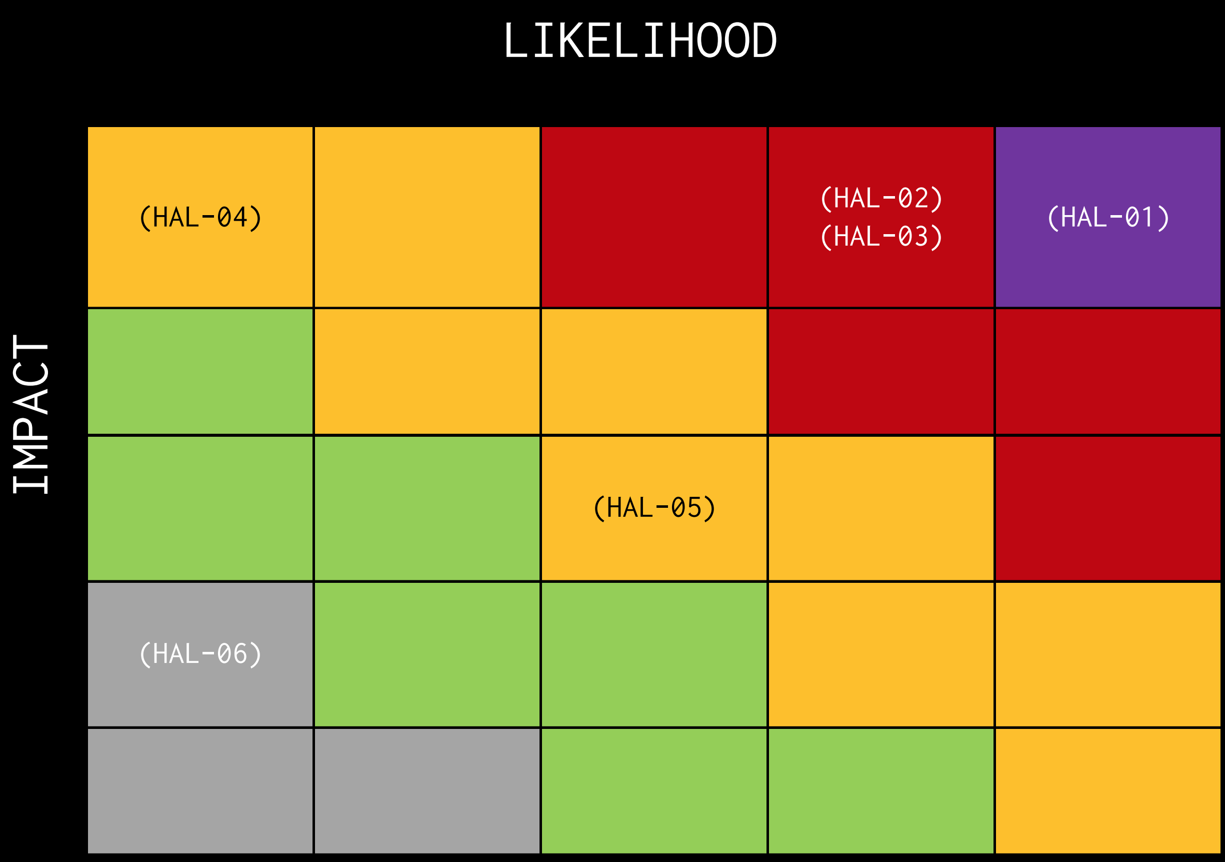
1.4 SCOPE

This review was scoped to the Farm Solana program.

1. Farm program
   (a) Repository: farms
   (b) Commit ID: 22880894983ef7b5f8d0f298d19158362965c917

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	2	2	0	1



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING ACCOUNT VALIDATION LEADS TO MULTIPLE CRITICAL VULNERABILITIES	Critical	SOLVED - 10/05/2021
FARM TAKEOVER	High	SOLVED - 10/05/2021
DELEGATE VALIDATION MISSING	High	SOLVED - 10/11/2021
HARDCODED GOVERNANCE ADDRESSES	Medium	SOLVED - 10/08/2021
INTEGER OVERFLOW	Medium	SOLVED - 10/12/2021
INITIALISING FARM WITH INVALID ACCOUNTS	Informational	SOLVED - 10/11/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING ACCOUNT VALIDATION LEADS TO MULTIPLE CRITICAL VULNERABILITIES - CRITICAL

Description:

The Cropper Farm program allows users to deposit LP tokens in farms and harvest rewards based on the deposit they made. Each farm is owned by an authority account which controls the farm's LP token pool and the reward token pool. This authority is a PDA derived from the farm account address and the farm program ID.

Whenever a user sends a transaction to the program they have to provide a number of account addresses, the most interesting of them being:

1. source token account address
2. destination token account address
3. transfer authority account address
4. fee account address

All token transfers are handled by the `token_transfer` function defined in `processor.rs`.

`token_transfer` signs **all** transactions with the farms authority account (the PDA) by default and uses the `invoke_signed` function to call the SPL Token program's `process_transfer` function to transfer the tokens. The `invoke_signed` function appends the program's signature to the transaction signers array.

Because neither deposit nor withdraw functions validate addresses and ownership of the user-supplied accounts, a number of **critical** vulnerabilities can be identified in the program, including:

1. Stealing all LP tokens in all pools

A malicious user may send a Deposit, Withdraw or an AddReward instruction and replace his transfer authority account address with the farm authority account address, the source LP token account with the farm's LP token account and the destination LP token account with their own account to transfer all tokens from the farm pool account to the user's account.

This is because the source token address is not validated not to be equal to the farm pool's account address

2. Stealing handling fees

A malicious user may send a PayFarmFee, Deposit or Withdraw instruction and replace the fee account address with their own account address to transfer the fee back to themselves. The program will confirm the fee has been paid successully and will keep processing the instruction logic.

This is because the user-supplied fee account address is not validated to match the intended one.

Code Location:

Listing 1: processor.rs (Lines 782,787,788,793,800)

```
776 /// issue a spl_token `Transfer` instruction.
777 pub fn token_transfer<'a>(
778     pool: &Pubkey,
779     token_program: AccountInfo<'a>,
780     source: AccountInfo<'a>,
781     destination: AccountInfo<'a>,
782     authority: AccountInfo<'a>,
783     nonce: u8,
784     amount: u64,
785 ) -> Result<(), ProgramError> {
786     let pool_bytes = pool.to_bytes();
787     let authority_signature_seeds = [&pool_bytes[..32], &[nonce]];
788     let signers = &[&authority_signature_seeds[..]];
789     let ix = spl_token::instruction::transfer(
```

```
        token_program.key,
        source.key,
        destination.key,
        authority.key,
        &[],
        amount,
    )?;
    invoke_signed(
        &ix,
        &[source, destination, authority, token_program],
        signers,
    )
}
```

Examples of missing account validation in process.rs.

Listing 2: process.rs

```
// authority information of this farm account
let authority_info = next_account_info(account_info_iter)?;

// depositor's wallet account information
let depositor_info = next_account_info(account_info_iter)?;

// depositor's user account information to include deposited balance, reward debt
let user_info_account_info = next_account_info(account_info_iter)?;

// depositor's transfer authority (wallet address)
let user_transfer_authority_info = next_account_info(account_info_iter)?;

// lp token account information in the depositor's wallet
let user_lp_token_account_info = next_account_info(account_info_iter)?;

// lp token account information in the farm pool
let pool_lp_token_account_info = next_account_info(account_info_iter)?;

// reward token account information in the depositor's wallet
let user_reward_token_account_info = next_account_info(account_info_iter)?;

// reward token account information in the farm pool
let pool_reward_token_account_info = next_account_info(account_info_iter)?;
```

Harvesting rewards, paying the fee and depositing tokens.

Listing 3: processor.rs (Lines 350,364,378,379,380)

```
335 // harvest user's pending rewards
336 if user_info.deposit_balance > 0 {
337
338     // pending amount
339     let pending: u64 = farm_pool.pending_rewards(&mut user_info);
340
341     if pending > 0 {
342         // harvest fee for the pending reward
343         let harvest_fee = pending * HARVEST_FEE_NUMERATOR /
            HARVEST_FEE_DENOMINATOR;
344
345         // transfer harvest fee to fee owner
346         Self::token_transfer(
347             farm_id_info.key,
348             token_program_info.clone(),
349             pool_reward_token_account_info.clone(),
350             fee_owner_info.clone(),
351             authority_info.clone(),
352             farm_pool.nonce,
353             harvest_fee
354         )?;
355
356         // user's real pending amount
357         let _pending = pending - harvest_fee;
358
359         // transfer pending reward amount from reward pool to user
              reward token account
360         Self::token_transfer(
361             farm_id_info.key,
362             token_program_info.clone(),
363             pool_reward_token_account_info.clone(),
364             user_reward_token_account_info.clone(),
365             authority_info.clone(),
366             farm_pool.nonce,
367             _pending
368         )?;
369     }
370 }
371
372 // deposit (stake lp token)
373 if amount > 0 {
```

```
    // transfer lp token amount from user's lp token account to
       pool's lp token pool
    Self::token_transfer(
        farm_id_info.key,
        token_program_info.clone(),
        user_lp_token_account_info.clone(),
        pool_lp_token_account_info.clone(),
        user_transfer_authority_info.clone(),
        farm_pool.nonce,
        amount
    )?;

    // update user's deposited balance
    user_info.deposit_balance += amount;
}
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendations:

It is of paramount importance to validate the fee, source and target token account addresses to match the intended ones in order to prevent users from stealing the tokens from the pool.

Remediation Plan:

SOLVED: Fixed in commit bcf5da93c14b4003f61705078f0f0788af866c00.

3.2 (HAL-02) FARM TAKEOVER - HIGH

Description:

Users call the process_initialize function to set farm parameters. When they pay the fee the farm is active and the is_allowed attribute is set to true. The process_initialize function however does not verify if the user-supplied farm account hasn't already been allowed, so it is possible for a malicious user to overwrite any farm by sending a transaction with the Initialize instruction and reset all farm parameters. This means the is_allowed property is also reset and the farm's inactive again.

However, because the user-supplied account with the amm_id address is not verified to be owned by the cropper-liquidity-pool program a malicious user may use a fake amm_id account with either token_a_mint or token_b_mint set to the CRP mint address and a random public key as the other of the two to **takeover and overwrite all its parameters without deactivating the farm**.

Code Location:

Listing 4: processor.rs (Lines 121)

```
117 // reward token's mint account information
118 let reward_mint_info = next_account_info(account_info_iter)?;
119
120 // amm account information what have lp token mint, token_a mint,
     token_b mint
121 let amm_id_info = next_account_info(account_info_iter)?;
122
123 // spl-token program account information
124 let token_program_info = next_account_info(account_info_iter)?;
```

Listing 5: processor.rs (Lines 167,169,171,177,181,198)

```
160 // CRP token pairing flag
161 let mut crp_token_pairing = 0;
162
```

```rust
// CRP token mint address
let crp_pubkey = Pubkey::from_str(CRP_MINT_ADDRESS).unwrap();

// other token mint address to check token pairing
let mut other_pubkey = *amm_swap.token_a_mint();

if *amm_swap.token_a_mint() == crp_pubkey {
    // this is crp token pair
    crp_token_pairing = 1;
    other_pubkey = *amm_swap.token_b_mint();
}

if *amm_swap.token_b_mint() == crp_pubkey {
    // this is crp token pair
    crp_token_pairing = 1;
}

// check if this creator can create "locked farms" specified by site owner
if crp_token_pairing == 1 {
    if  other_pubkey == Pubkey::from_str(USDC_MINT_ADDRESS).unwrap() ||
        other_pubkey == Pubkey::from_str(USDT_MINT_ADDRESS).unwrap() ||
        other_pubkey == Pubkey::from_str(SOL_MINT_ADDRESS).unwrap() ||
        other_pubkey == Pubkey::from_str(ETH_MINT_ADDRESS).unwrap() {

            // check if creator is allowed creator
            // if not returns WrongCreator error
            if *creator_info.key != Pubkey::from_str(ALLOWED_CREATOR).unwrap() {
                 return Err(FarmPoolError::WrongCreator.into());
            }
        }
}
```

Risk Level:

Likelihood - 4

Impact - 5

Recommendations:

It is recommended to verify if the farm hasn't been already allowed before updating its parameters.

Remediation Plan:

SOLVED: Fixed in commit bcf5da93c14b4003f61705078f0f0788af866c00.

3.3 (HAL-03) DELEGATE VALIDATION MISSING - HIGH

Description:

Among the accounts provided by farm creator on farm initialisation are the farm's LP token account which serves as token pool for all users depositing their tokens and the reward account which holds all the generated rewards. Both accounts authority is assigned to the farm authority. The process_initialise function does not validate however if those accounts do not have any delegates.

It is possible for a malicious user to initialize a farm with his own address as the accounts' delegate before transferring their authorities to the farm authority. Since the swap program does not support pool properties modification it is impossible to remove the delegation thus giving the malicious user **full control** over the deposited funds and generated rewards.

Code Location:

Listing 6: processor.rs (Lines 109,112)

```
105 // creator wallet account information
106 let creator_info = next_account_info(account_info_iter)?;
107
108 // lp token account information to store lp token in the pool
109 let pool_lp_token_account_info = next_account_info(
    account_info_iter)?;
110
111 // reward token account information to store reward token in the
    pool
112 let pool_reward_token_account_info = next_account_info(
    account_info_iter)?;
113
114 // lp token's mint account information
115 let pool_mint_info = next_account_info(account_info_iter)?;
116
117 // reward token's mint account information
```

```
118 let reward_mint_info = next_account_info(account_info_iter)?;
```

Listing 7: processor.rs (Lines 205,208)

```
201 // initialize fee owner with predefined wallet address
202 farm_pool.fee_owner = Pubkey::from_str(FEE_OWNER).unwrap();
203
204 // initialize lp token account to store lp token
205 farm_pool.pool_lp_token_account = *pool_lp_token_account_info.key;
206
207 // initialize reward token account to store reward token
208 farm_pool.pool_reward_token_account = *
        pool_reward_token_account_info.key;
209
210 // store nonce to authorize this farm account
211 farm_pool.nonce = nonce;
212
213 // store lp token mint address
214 farm_pool.pool_mint_address = *pool_mint_info.key;
```

Risk Level:

Likelihood - 4
Impact - 5

Recommendations:

It is recommended to verify if the farm's LP token account and reward account do not have delegates.

Remediation Plan:

SOLVED: Fixed in commit 21c2e5892a412f49149bb2530757737787315d0c.

3.4 (HAL-04) HARDCODED GOVERNANCE ADDRESSES - MEDIUM

Description:

Several important governance accounts/wallets addresses are hardcoded in cropper-lp/program/src/constraints.rs and yield-farming-v1/program/src/constants.rs. In case those addresses are compromised the program owner has no way of updating them, putting users' funds at risk.

Code Location:

Listing 8: yield-farming-v1/program/src/constants.rs (Lines 19,23)

```
16 /// Fee owner wallet address
17 /// This includes harvest fee
18 /// So this wallet address should have all token accounts of
       registered token-list
19 pub const FEE_OWNER:&str = if DEVNET_MODE {"
       BRmxAJ3ThceU2SXt6weyXarRNvAwZUtKuKbzSRneRxJn"} else {"4
       GJ3z4skEHJADz3MVeNYBg4YV8H27rBQey2YYdiPC8PA"};
20
21 /// This is allowed wallet address to create specified farms by
       site owner
22 /// Specified farms are SOL-USDC, SOL-CRP, USDT-CRP, USDC-CRP, ETH
       -USDC, ETH-CRP
23 pub const ALLOWED_CREATOR:&str = if DEVNET_MODE {"4
       GJ3z4skEHJADz3MVeNYBg4YV8H27rBQey2YYdiPC8PA"} else {"
       BRmxAJ3ThceU2SXt6weyXarRNvAwZUtKuKbzSRneRxJn"};
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendations:

Consider making the governance addresses modifiable and implement a function to update these addresses in case they are compromised.

Remediation Plan:

SOLVED: Fixed in commit 643636779a5eac3a000e217406cbd8be479f6b4e.

3.5 (HAL-05) INTEGER OVERFLOW - MEDIUM

Description:

An overflow happens when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of digits. For example, in the pending_rewards function defined in state.rs two u64 values are multiplied without checking whether the result is within the range that can be represented with a given number of bits. If it isn't, in Rust the resulting value is specified to wrap as two's complement, resulting in a value either too low or too high considering the circumstances.

Code Location:

Listing 9: state.rs (Lines 66,71)

```
62 impl FarmPool {
63     /// get current pending reward amount for a user
64     pub fn pending_rewards(&self, user_info:&mut UserInfo) -> u64{
65         let reward_per_share_net: u64 = self.reward_per_share_net;
66         return user_info.deposit_balance * reward_per_share_net /
               REWARD_MULTIPLER - user_info.reward_debt;
67     }
68 
69     /// get total reward amount for a user so far
70     pub fn get_new_reward_debt(&self, user_info:&UserInfo) -> u64{
71         return user_info.deposit_balance * self.
               reward_per_share_net / REWARD_MULTIPLER;
72     }
73 
74 }
```

Listing 10: processor.rs (Lines 669)

```
666 // update reward per second in the rest period from now
667 let duration = farm_pool.end_timestamp - cur_timestamp;
668 let added_reward_per_second = amount / duration;
```

```
farm_pool.reward_per_timestamp += added_reward_per_second;
```

Listing 11: processor.rs (Lines 770,771)

```
759 // check if valid current timestamp
760 if farm_pool.last_timestamp >= cur_timestamp {
761     return;
762 }
763 if lp_supply == 0 || farm_pool.reward_per_timestamp == 0 {
764     farm_pool.last_timestamp = cur_timestamp;
765     return;
766 }
767
768 // update reward per share net and last distributed timestamp
769 let multiplier = cur_timestamp - farm_pool.last_timestamp;
770 let reward = multiplier * farm_pool.reward_per_timestamp;
771 farm_pool.reward_per_share_net = farm_pool.reward_per_share_net +
        REWARD_MULTIPLER * reward / lp_supply;
772 farm_pool.last_timestamp = cur_timestamp;
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

Consider replacing the multiplication and subtraction operators with Rust's checked_mul and checked_sub for 64bit unsigned integers.

Remediation Plan:

SOLVED: Fixed in commit c8aaf6386305947ec1d7c3c1a9aabf448ae2a92a.

3.6 (HAL-06) INITIALISING FARM WITH INVALID TOKEN ACCOUNTS - INFORMATIONAL

Description:

To be initialised, a farm requires two token accounts to be provided by the initialising user: an LP token account and a reward account. Neither of the accounts is verified not to be frozen, to match a relevant mint, or to have the farm authority be the transfer authority, therefore it is possible for a malicious user to create a “frozen pool” with tokens that cannot effectively be accessed.

Additionally, the corresponding AMM account (also user-supplied) is not validated to be owned by the Swap program.

Code Location:

Listing 12: processor.rs (Lines 109,112,115,121)

```
105 // creator wallet account information
106 let creator_info = next_account_info(account_info_iter)?;
107
108 // lp token account information to store lp token in the pool
109 let pool_lp_token_account_info = next_account_info(
        account_info_iter)?;
110
111 // reward token account information to store reward token in the
        pool
112 let pool_reward_token_account_info = next_account_info(
        account_info_iter)?;
113
114 // lp token's mint account information
115 let pool_mint_info = next_account_info(account_info_iter)?;
116
117 // reward token's mint account information
118 let reward_mint_info = next_account_info(account_info_iter)?;
119
```

```
120 // amm account information what have lp token mint, token_a mint,
     token_b mint
121 let amm_id_info = next_account_info(account_info_iter)?;
```

Listing 13: processor.rs (Lines 207,210,216,222)

```
206 // initialize lp token account to store lp token
207 farm_pool.pool_lp_token_account = *pool_lp_token_account_info.key;
208
209 // initialize reward token account to store reward token
210 farm_pool.pool_reward_token_account = *
     pool_reward_token_account_info.key;
211
212 // store nonce to authorize this farm account
213 farm_pool.nonce = nonce;
214
215 // store lp token mint address
216 farm_pool.pool_mint_address = *pool_mint_info.key;
217
218 // store spl-token program address
219 farm_pool.token_program_id = *token_program_info.key;
220
221 // store reward token mint address
222 farm_pool.reward_mint_address = *reward_mint_info.key;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Before initialising the farm, it is recommended to verify if both token accounts have the farm PDA as authority, match the relevant mints and if the `state` property of both token accounts is not `Frozen`. Validate the AMM account owner to match the Swap program ID.

Remediation Plan:

SOLVED: Fixed in commit 3755e767c151ded394baa104994597fe615bbbf1.

FUZZ TESTING

4.1 FUZZING

Introduction:

Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

Halborn custom-built scripts leverage libFuzzer and cargo-fuzz for in-process, coverage-guided fuzz testing.

The fuzzer tracks which areas of the code are reached, and generates mutations on the corpus of input data in order to maximize the code coverage. The code coverage information is provided by LLVM's SanitizerCoverage instrumentation.

Description:

Halborn used custom fuzzing scripts, tailored to the specifics of the Solana protocol. The methods targeted were the ones accepting vectors of bytes as input because they are potentially most likely to be vulnerable to memory management issues.

PoC:

```
pc@P[redacted]:~/[redacted]/yield-farming-v1/program$ cargo fuzz run security
warning: unused import: `msg`
  --> /[redacted]/yield-farming-v1/program/src/instruction.rs:13:9
   |
13 |         msg
   |         ^^^
   |
   = note: `#[warn(unused_imports)]` on by default

warning: unused variable: `reward_mint_info`
   --> /[redacted]/yield-farming-v1/program/src/processor.rs:284:13
    |
284 |         let reward_mint_info = next_account_info(account_info_iter)?;
    |             ^^^^^^^^^^^^^^^^ help: if this is intentional, prefix it with an underscore: `_reward_mint_info`
    |
    = note: `#[warn(unused_variables)]` on by default

warning: `farm-pool` (lib) generated 2 warnings
    Finished release [optimized] target(s) in 0.36s
warning: unused import: `msg`
  --> /[redacted]/yield-farming-v1/program/src/instruction.rs:13:9
   |
13 |         msg
   |         ^^^
   |
   = note: `#[warn(unused_imports)]` on by default

warning: unused variable: `reward_mint_info`
   --> /[redacted]/yield-farming-v1/program/src/processor.rs:284:13
    |
284 |         let reward_mint_info = next_account_info(account_info_iter)?;
    |             ^^^^^^^^^^^^^^^^ help: if this is intentional, prefix it with an underscore: `_reward_mint_info`
    |
    = note: `#[warn(unused_variables)]` on by default

warning: `farm-pool` (lib) generated 2 warnings
    Finished release [optimized] target(s) in 0.09s
     Running `fuzz/target/x86_64-apple-darwin/release/security -artifact_prefix=/[redacted]
    yield-farming-v1/program/fuzz/artifacts/security/ /[redacted]/yield-farming-v1/program
/fuzz/corpus/security`
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 541144971
INFO: Loaded 1 modules   (54313 inline 8-bit counters): 54313 [0x1029fea80, 0x102a0bea9),
INFO: Loaded 1 PC tables (54313 PCs): 54313 [0x102a0beb0,0x102ae0140),
INFO:        0 files found in /[redacted]/yield-farming-v1/program/fuzz/corpus/security
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2      INITED cov: 15 ft: 15 corp: 1/1b exec/s: 0 rss: 32Mb
#5      NEW    cov: 16 ft: 16 corp: 2/5b lim: 4 exec/s: 0 rss: 32Mb L: 4/4 MS: 3 InsertByte-CopyPart-ChangeBinInt-
#8      NEW    cov: 17 ft: 17 corp: 3/8b lim: 4 exec/s: 0 rss: 33Mb L: 3/4 MS: 3 ShuffleBytes-CopyPart-EraseBytes-
#12     NEW    cov: 18 ft: 18 corp: 4/11b lim: 4 exec/s: 0 rss: 33Mb L: 3/4 MS: 4 ChangeBit-ChangeByte-CopyPart-ChangeBinInt-
#13     REDUCE cov: 18 ft: 18 corp: 4/10b lim: 4 exec/s: 0 rss: 33Mb L: 2/4 MS: 1 EraseBytes-
#37     REDUCE cov: 18 ft: 18 corp: 4/8b lim: 4 exec/s: 0 rss: 33Mb L: 2/3 MS: 4 CopyPart-ChangeByte-ChangeBit-EraseBytes-
#42     REDUCE cov: 18 ft: 18 corp: 4/7b lim: 4 exec/s: 0 rss: 33Mb L: 2/2 MS: 5 InsertByte-ChangeByte-ShuffleBytes-CopyPart-EraseBytes-
#63     REDUCE cov: 19 ft: 19 corp: 5/8b lim: 4 exec/s: 0 rss: 33Mb L: 1/2 MS: 1 EraseBytes-
#134    REDUCE cov: 19 ft: 19 corp: 5/7b lim: 4 exec/s: 0 rss: 33Mb L: 1/2 MS: 1 EraseBytes-
#187    REDUCE cov: 19 ft: 19 corp: 5/6b lim: 4 exec/s: 0 rss: 33Mb L: 1/2 MS: 3 ChangeByte-ChangeBit-EraseBytes-
#271    REDUCE cov: 20 ft: 20 corp: 6/8b lim: 4 exec/s: 0 rss: 33Mb L: 2/2 MS: 4 ChangeByte-ShuffleBytes-ChangeBinInt-ChangeBinInt-
#307    REDUCE cov: 21 ft: 21 corp: 7/9b lim: 4 exec/s: 0 rss: 33Mb L: 1/2 MS: 1 ChangeBit-
#331    REDUCE cov: 21 ft: 21 corp: 7/8b lim: 4 exec/s: 0 rss: 33Mb L: 1/2 MS: 4 ChangeBinInt-ChangeBit-ShuffleBytes-EraseBytes-
#1079   REDUCE cov: 22 ft: 22 corp: 8/18b lim: 11 exec/s: 0 rss: 33Mb L: 10/10 MS: 3 ChangeBit-ShuffleBytes-CMP- DE: "\x00\x00\x00\x00\x
00\x00\x00\x00"-
        NEW_FUNC[1/1]: 0x1020cb4a0 in core::ptr::drop_in_place$LT$alloc..vec..Vec$LT$u8$GT$$GT$::ha4b1c8bf0e553b71+0x0 (security:x86_64+
0x1000054a0)
#1121   REDUCE cov: 26 ft: 26 corp: 9/28b lim: 11 exec/s: 0 rss: 34Mb L: 10/10 MS: 2 ShuffleBytes-InsertRepeatedBytes-
#1124   REDUCE cov: 27 ft: 27 corp: 10/38b lim: 11 exec/s: 0 rss: 34Mb L: 10/10 MS: 3 ChangeByte-ChangeBinInt-CMP- DE: "\x01\x00\x00\x00
\x00\x00\x00\x00"-
```

Results:

Between the time constraints and lack of advanced memory manipulation in the source code **no issues were identified at this time**.

AUTOMATED TESTING

5.1 VULNERABILITIES AUTOMATIC DETECTION

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 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.

Results:

id	package	categories
RUSTSEC-2021-0093	crossbeam-deque	memory-corruption
RUSTSEC-2021-0079	hyper	parsing the ‘Transfer-Encoding‘ header leads to data loss
RUSTSEC-2021-0078	hyper	lenient parsing of the ‘Content-Length‘ header could allow request smuggling
RUSTSEC-2021-0072	tokio	memory-corruption
RUSTSEC-2021-0064	cpuid-bool	unmaintained
RUSTSEC-2020-0036	failure	unmaintained
RUSTSEC-2020-0016	net2	unmaintained

THANK YOU FOR CHOOSING

HALBORN