// HALBORN

DRAFT

CropperFinance

AMM Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 13th, 2021 - September 30th, 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	Final Draft	09/30/2021	Piotr Cielas
0.4	Draft Review	09/30/2021	Gabi Urrutia
1.0	Remediation Plan	10/04/2021	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 program beginning on September 13th, 2021 and ending September 30th, 2021. This security assessment was scoped to the AMM repository 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 smart-contract development.

In summary, Halborn identified some risks that were addressed by Cropper Finance 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 smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts 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

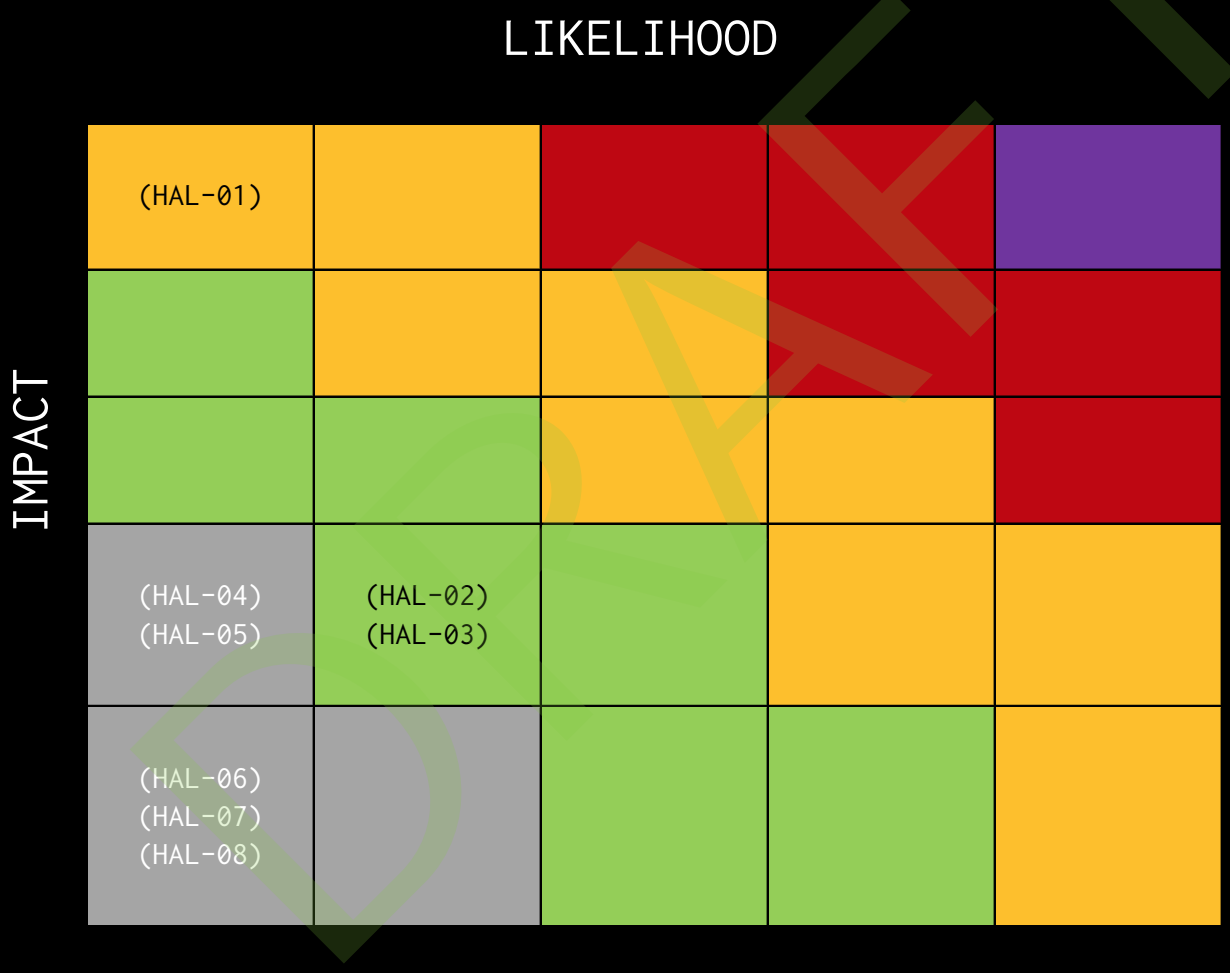
1.4 SCOPE

This review was scoped to the AMM Solana program.

1. AMM program
   (a) Repository: cropper-lp/program
   (b) Commit ID: a818ceb3d886e519b8508d509409c2038a149047

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	5



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HARDCODED GOVERNANCE ADDRESSES	Medium	SOLVED - 10/12/2021
SERUM DEX MARKET ACCOUNT OWNER VALIDATION MISSING	Low	SOLVED - 10/04/2021
USE OF UNSAFE CODE	Low	SOLVED - 10/04/2021
BYTE ARRAY LENGTH VALIDATION MISSING	Informational	SOLVED - 10/12/2021
INITIALISING SWAP WITH FROZEN TOKEN ACCOUNTS	Informational	SOLVED - 10/04/2021
MULTIPLE VULNERABILITIES IN THE FEE VALIDATE METHOD	Informational	SOLVED - 10/04/2021
GET PACKED DATA LENGTH FUNCTION RETURNS AMBIGUOUS RESULTS	Informational	SOLVED - 10/12/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) HARDCODED GOVERNANCE ADDRESSES - MEDIUM

Description:

Several important governance accounts/wallets addresses are hardcoded in constraints.rs. In case those addresses are compromised the program owner has no way of updating them thus putting users' funds at risk.

Code Location:

Listing 1: constraints.rs (Lines 59)

```
58 #[cfg(feature = "production")]
59 const OWNER_KEY: &str = env!("SWAP_PROGRAM_OWNER_FEE_ADDRESS");
```

Listing 2: constraints.rs (Lines 84)

```
80 pub const SWAP_CONSTRAINTS: Option<SwapConstraints> = {
81     #[cfg(feature = "production")]
82     {
83         Some(SwapConstraints {
84             owner_key: OWNER_KEY,
85             valid_curve_types: VALID_CURVE_TYPES,
86             fees: FEES,
87         })
88     }
```

Listing 3: processor.rs (Lines 34)

```
33 use std::convert::TryInto;
34 const FEE_WALLET_ADDRESS:&str = "2
       Pv5mjmKYAtXNpr3mcsXf7HjtS3fieJeFoWPATVT5rWa";
35 const WSOL_MINT_ADDRESS:&str = "
       So11111111111111111111111111111111111111112";
36 /// Program state handler.
37 pub struct Processor {}
```

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 691220b25ac065ca68dfffde9f9e2bac72d3da04.

3.2 (HAL-02) SERUM DEX MARKET ACCOUNT OWNER VALIDATION MISSING - LOW

Description:

One of the accounts the process_initialize function requires is a Serum DEX market ID. The account's address is one of swap properties, however the owner of this account is not validated to match the Serum DEX program ID. This allows users to initialize a swap with an arbitrary account owned by any program instead of a legitimate Serum DEX market.

Code Location:

Listing 4: processor.rs (Lines 239)

```
236 let destination_info = next_account_info(account_info_iter)?;
237 let token_program_info = next_account_info(account_info_iter)?;
238 let dex_program_info = next_account_info(account_info_iter)?;
239 let market_info = next_account_info(account_info_iter)?;
240
241 //validate account info
242 let token_program_id = *token_program_info.key;
243 if SwapVersion::is_initialized(&swap_info.data.borrow()) {
244     return Err(AmmError::AlreadyInUse.into());
245 }
```

Listing 5: processor.rs (Lines 348)

```
342 //Save the pool account info
343 let obj = SwapVersion::SwapV1(SwapV1 {
344     is_initialized: true,
345     nonce,
346     amm_id: *amm_id_info.key,
347     dex_program_id: *dex_program_info.key,
348     market_id: *market_info.key,
349     token_program_id,
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

Validate the supplied market account owner address to match the Serum DEX program ID.

Remediation Plan:

SOLVED: Fixed in commit 17d66786ce4f9c687cba9212b59773ba688c9c78.

3.3 (HAL-03) USE OF UNSAFE CODE - LOW

Description:

Although Rust language programming is memory safe by default, it allows the user to provide the unsafe keyword/feature to apply less restrictions than normal. Using unsafe code is possible to dereferencing a raw pointer, reading or writing a mutable or external static variable, accessing a field of a union other than to assign to it, calling an unsafe function or implementing an unsafe trait. The security consequences of using unsafe code in Rust increase the possibilities to be exposed to several vulnerabilities or bugs provoking memory leaks. The worst cases can expose sensitive information left in memory, or gain remote code execution by taking control of the pointer in memory, and redirecting it to malicious code execution sectors controlled by an attacker.

Code Location:

Listing 6: amm_instruction (Lines 327)

```
314 pub fn unpack<T>(input: &[u8]) -> Result<&T, ProgramError> {
315     if input.len() < size_of::<u8>() + size_of::<T>() {
316         return Err(ProgramError::InvalidAccountData);
317     }
318     #[allow(clippy::cast_ptr_alignment)]
319     let val: &T = unsafe { &*(&input[1] as *const u8 as *const T)
            };
320     Ok(val)
321 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is recommended not to use unsafe code in order to avoid exposed possible vulnerabilities or bugs triggering memory leaks.

Remediation Plan:

SOLVED: Fixed in commit 1e9ea9d6716d0314bc6c8465b4b6068944cf73c5.

3.4 (HAL-04) BYTE ARRAY LENGTH VALIDATION MISSING - INFORMATIONAL

Description:

The unpack_from_slice functions defined in curve/base.rs, curve/fees.rs , curve/constant_price, curve/offset.rs and curve/stable.rs parse user-supplied byte arrays to struct fields. Neither of those functions however verify if the user-supplied data length matches the expected ones which may lead to panics when the length is incorrect.

Code Location:

Listing 7: curve/base.rs (Lines 213)

```
209 /// Unpacks a byte buffer into a SwapCurve
210 fn unpack_from_slice(input: &[u8]) -> Result<Self, ProgramError> {
211     let input = array_ref![input, 0, 33];
212     #[allow(clippy::ptr_offset_with_cast)]
213     let (curve_type, calculator) = array_refs![input, 1, 32];
214     let curve_type = curve_type[0].try_into()?;
215     Ok(Self {
216         curve_type,
217         calculator: match curve_type {
218             CurveType::ConstantProduct => {
219                 Box::new(ConstantProductCurve::unpack_from_slice(
                        calculator)?)
220             }
221             CurveType::ConstantPrice => {
222                 Box::new(ConstantPriceCurve::unpack_from_slice(
                        calculator)?)
223             }
224             CurveType::Stable => Box::new(StableCurve::
                    unpack_from_slice(calculator)?),
225             CurveType::Offset => Box::new(OffsetCurve::
                    unpack_from_slice(calculator)?),
226         },
227     })
228 }
```

Listing 8: fees.rs (Lines 115)

```
114 fn unpack_from_slice(input: &[u8]) -> Result<Fees, ProgramError> {
115     let input = array_ref![input, 0, 24];
116     #[allow(clippy::ptr_offset_with_cast)]
117     let (
118         return_fee_numerator,
119         fixed_fee_numerator,
120         fee_denominator,
121     ) = array_refs![input, 8, 8, 8];
```

Listing 9: curve/constant_price.rs (Lines 253)

```
252 fn unpack_from_slice(input: &[u8]) -> Result<ConstantPriceCurve,
    ProgramError> {
253     let token_b_price = array_ref![input, 0, 8];
254     Ok(Self {
255         token_b_price: u64::from_le_bytes(*token_b_price),
256     })
257 }
258
259 ```{language=rust caption="curve/offset.rs" firstnumber=174 hlines
    =175}
260 fn unpack_from_slice(input: &[u8]) -> Result<OffsetCurve,
    ProgramError> {
261     let token_b_offset = array_ref![input, 0, 8];
262     Ok(Self {
263         token_b_offset: u64::from_le_bytes(*token_b_offset),
264     })
265 }
```

Listing 10: curve/stable.rs (Lines 336)

```
335 fn unpack_from_slice(input: &[u8]) -> Result<StableCurve,
    ProgramError> {
336     let amp = array_ref![input, 0, 8];
337     Ok(Self {
338         amp: u64::from_le_bytes(*amp),
339     })
340 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Validate the user-supplied data length to match the expected ones before parsing it to struct fields.

Remediation Plan:

SOLVED: Fixed in commit 691220b25ac065ca68dfffde9f9e2bac72d3da04.

3.5 (HAL-05) INITIALISING SWAP WITH FROZEN TOKEN ACCOUNTS - INFORMATIONAL

Description:

To be initialised, the token swap account requires two token accounts to be provided by the initialising user. Both accounts are checked not to have the same mint and belong to the swap authority however they are not verified not to be frozen, therefore it is possible for a malicious user to create a “frozen pool” with tokens that cannot effectively be accessed.

Code Location:

Listing 11: processor.rs (Lines 250,251)

```
250 let token_a = Self::unpack_token_account(token_a_info, &
    token_program_id)?;
251 let token_b = Self::unpack_token_account(token_b_info, &
    token_program_id)?;
252
253 let destination = Self::unpack_token_account(destination_info, &
    token_program_id)?;
254 let pool_mint = Self::unpack_mint(pool_mint_info, &
    token_program_id)?;
255 if *authority_info.key != token_a.owner {
256     return Err(AmmError::InvalidOwner.into());
257 }
258 if *authority_info.key != token_b.owner {
259     return Err(AmmError::InvalidOwner.into());
260 }
```

Listing 12: processor.rs (Lines 294,297,300,303)

```
294 if token_a.delegate.is_some() {
295     return Err(AmmError::InvalidDelegate.into());
296 }
```

```
297 if token_b.delegate.is_some() {
298     return Err(AmmError::InvalidDelegate.into());
299 }
300 if token_a.close_authority.is_some() {
301     return Err(AmmError::InvalidCloseAuthority.into());
302 }
303 if token_b.close_authority.is_some() {
304     return Err(AmmError::InvalidCloseAuthority.into());
305 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Verify if the `state` property of both token accounts is not `Frozen` before initializing the swap.

Remediation Plan:

SOLVED: Fixed in commit `5a00ce89c4a3da6dcfd6ccba123c5e81ee605308`.

3.6 (HAL-06) MULTIPLE VULNERABILITIES IN THE FEE VALIDATE METHOD - INFORMATIONAL

Description:

The Fee::validate method is tasked with verifying if the user-supplied struct fields can be used in swap/withdraw fee calculation. There is a number if issues with that method:

1. Division by zero in the first if statement: it does not ensure the fee_denominator to be greater than 0 which can lead to division by zero if used in fee calculation.
2. Integer overflow in the second if statement: 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. In this statement two u128 values are added together 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.

Although this function are not currently exploitable as the values of its arguments are hardcoded, technically it is still vulnerable and we recommend patching it.

Code Location:

Listing 13: curve/fees.rs (Lines 79,81)

```
76 /// Validate that the fees are reasonable
77 pub fn validate(&self) -> Result<(), AmmError> {
78
79     if self.fee_denominator == 0 && self.fixed_fee_numerator == 0
           && self.return_fee_numerator == 0{
80         Ok(())
```

```
81      } else if self.fixed_fee_numerator +  self.
            return_fee_numerator >= self.fee_denominator {
82          Err(AmmError::InvalidFee)
83      } else {
84          Ok(())
85      }
86  }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

In the release mode Rust does not panic on overflows and overflown values “wrapabout” without any explicit feedback to the user. It is recommended then to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system. Consider replacing the addition operator with Rust’s checked_add method and validating the denominators to be greater than 0.

Remediation:

SOLVED: Fixed in commit 6ca7e12c14edcf5fdd9d28f675b9f4b148d4258f.

3.7 (HAL-07) GET PACKED DATA LENGTH FUNCTION RETURNS AMBIGUOUS RESULTS - INFORMATIONAL

Description:

Each swap operation requires the swap account to be initialised. The account has to allocate a sufficient number of bytes for account (packed) data. This number can be determined with the get_packed_len utility function. However, Cropper prepends this data with a single byte denoting the swap version which the get_packed_len function does not consider thus returning a number one too low. Without manually increasing the allocation space by one it is impossible to create a correct account and in consequence it is impossible to create a swap.

Code Location:

Listing 14: amm_stats.rs (Lines 65)

```
60 /// Pack a swap into a byte array, based on its version
61 pub fn pack(src: Self, dst: &mut [u8]) -> Result<(), ProgramError>
       {
62     match src {
63         Self::SwapV1(swap_info) => {
64             dst[0] = 1;
65             SwapV1::pack(swap_info, &mut dst[1..])
66         }
67     }
68 }
```

Listing 15: amm_stats.rs (Lines 206)

```
202 impl Pack for SwapV1 {
203     const LEN: usize = 411;
204
205     fn pack_into_slice(&self, output: &mut [u8]) {
206         let output = array_mut_ref![output, 0, 411];
207         let (
```

```
208             is_initialized,
209             nonce,
210             amm_id,
211             dex_program_id,
212             market_id,
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Include the byte denoting swap version in the packed data so the SwapV1 ::get_packed_len function returns the correct number of bytes required.

Remediation Plan:

SOLVED: Fixed in commit 691220b25ac065ca68dfffde9f9e2bac72d3da04.

3.8 (HAL-08) INITIAL POOL LIQUIDITY EDGE CASES - INFORMATIONAL

Description:

Regardless of the actual deposit amount or selected curve type users are minted 1,000,000,000 LP tokens on swap initialisation as defined by the INITIAL_SWAP_POOL_AMOUNT constant in calculator.rs. This has at least two consequences for all other liquidity providers:

1. It will be impossible for other users to deposit less then $k * 10^n$ tokens if the initial deposit is made for an amount of $k * 10^{9+n}$ tokens ($k, n \geq 0$) and no withdraws are made because the amount of LP tokens to be minted on each subsequent deposit is calculated based on the current LP token balance.
2. if the initial deposit is for an amount of 1 token and no withdraws are made the pool will hold up to 18.5k tokens (assuming 6 decimal places)

Code Location:

Listing 16: processor.rs (Lines 329,339)

```
327 swap_curve.calculator.validate()?;
328
329 let initial_amount = swap_curve.calculator.new_pool_supply();
330
331 //Mint Initial supply
332 Self::token_mint_to(
333     swap_info.key,
334     token_program_info.clone(),
335     pool_mint_info.clone(),
336     destination_info.clone(),
337     authority_info.clone(),
338     nonce,
339     to_u64(initial_amount)?,
340 )?;
```

Listing 17: calculator.rs (Lines 101)

```
 98 /// Get the supply for a new pool
 99 /// The default implementation is a Balancer-style fixed initial
      supply
100 fn new_pool_supply(&self) -> u128 {
101     INITIAL_SWAP_POOL_AMOUNT
102 }
```

Listing 18: calculator.rs (Lines 12)

```
 8 /// Initial amount of pool tokens for swap contract, hard-coded to
      something
 9 /// "sensible" given a maximum of u128.
10 /// Note that on Ethereum, Uniswap uses the geometric mean of all
      provided
11 /// input amounts, and Balancer uses 100 * 10 ^ 18.
12 pub const INITIAL_SWAP_POOL_AMOUNT: u128 = 1_000_000_000;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Document these edge cases so that the community is more aware of the protocol limitations.

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                :~/l                                  /AMM/cropper-lp/program/fuzz/fuzz_targets$ cargo fuzz run fuzz_target_1
    Finished release [optimized] target(s) in 0.09s
    Finished release [optimized] target(s) in 0.09s
     Running `                                          /AMM/cropper-lp/program/fuzz/target/x86_64-apple-darwin/release/fuzz_target_1 -a
rtifact_prefix=,                                        'AMM/cropper-lp/program/fuzz/artifacts/fuzz_target_1/' /
                 'AMM/cropper-lp/program/fuzz/corpus/fuzz_target_1`
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 3247767838
INFO: Loaded 1 modules   (736 inline 8-bit counters): 736 [0x1071419d8, 0x107141cb8),
INFO: Loaded 1 PC tables (736 PCs): 736 [0x107141cb8,0x107144ab8),
INFO:        0 files found in                                   /AMM/cropper-lp/program/fuzz/corpus/fuzz_target_1
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: 7 ft: 7 corp: 1/1b exec/s: 0 rss: 30Mb
#2097152        pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 1048576 rss: 187Mb
#4194304        pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 1048576 rss: 345Mb
#8388608        pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 1048576 rss: 612Mb
#16777216       pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 1118481 rss: 614Mb
#33554432       pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 1048576 rss: 615Mb
#67108864       pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 1065220 rss: 616Mb
#134217728      pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 1082401 rss: 616Mb
#268435456      pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 972592 rss: 617Mb
#536870912      pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 1044495 rss: 617Mb
#1073741824     pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 979691 rss: 617Mb
#2147483648     pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 944779 rss: 617Mb
Slowest unit: 70 s:
artifact_prefix='                                       'AMM/cropper-lp/program/fuzz/artifacts/fuzz_target_1/'; Test unit written to
                                                        /AMM/cropper-lp/program/fuzz/artifacts/fuzz_target_1/slow-unit-71853c6197a6a7f222db0f1978c7cb2
32b87c5ee
Base64: Cgo=
#4294967296     pulse  cov: 7 ft: 7 corp: 1/1b lim: 4096 exec/s: 725378 rss: 617Mb
```

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