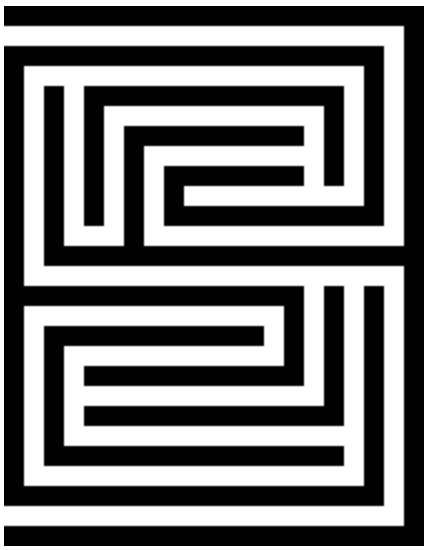

Crema Finance Public Report

PROJECT: Crema Finance
January 2022

Prepared For:

Crema Finance

henry@crema.finance

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

Executive Summary **3**
- Scope of Engagement 3
- Timeline 3
- Engagement Goals 3
- Contract Specification 3
- Overall Assessment 3
- Timeliness of Content 4

General Recommendations **5**
- Documentation requires updates 5
- Should've Used Match Instead of If Statements for TradeDirection Behavior in Swap Function 5
- Newtype Pattern to Represent Different Pricing Units and Precisions 5
- Magic Constants in SwapInstruction unpack 6
- Spelling Mistake 6
- Magic Constant for AccHead account_type and Version 6

Specific Recommendations **7**
- Arbitrary Write to Accounts and Complete Compromise of Swap 7
- Unchecked Mathematical Operations Introduce Overflow/Underflow Risk in AMMV3Curve 9
- Unchecked Mathematical Operations Introduce Overflow/Underflow Risk in tick_utils 10
- Spoofable tick_detail_info Account in Swap Instruction 10
- Missing Owner Check for swap_info in Add User Position Instruction 11
- Arbitrary Account Write in Add User Position Instruction 11

Toolset Warnings **12**
- Overview 12
- Compilation Warnings 12
- Test Coverage 12
- Static Analysis Coverage 12

Directory Structure **12**

Crema Finance Protocol Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in January of 2022 to perform a comprehensive security review of the Crema Finance smart contracts (specific contracts denoted within the appendix). Our review was conducted over a period of six days by a member of Bramah Systems, LLC. experts network.

Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: January 17th, 2022

Report Delivery: Feb 3rd, 2022

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the Crema Finance protocol, with a specific focus on trading actions. Notably, economic style attacks were not in scope of this review. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Rust code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Contract Specification

Contract specification was provided in the form of code comments and functional unit tests.

Overall Assessment

Bramah Systems was engaged to evaluate and identify any potential security concerns within the codebase of the Crema Finance Protocol. During the course of our engagement, Bramah

Systems found few instances wherein the team deviated materially from established best practices and procedures of secure software development within DLT, as our report details.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the Crema Finance Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Rust compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.

The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the Crema Finance protocol or any other relevant product, service or asset of Crema Finance or otherwise. This report is not and should not be relied upon by Crema Finance or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Bramah Systems, LLC. makes no warranties, representations, or guarantees about the Crema Finance Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Rust Development Guidelines

Documentation requires updates

The `DepositAllTokenTypes` instruction documentation didn't mark the slice in the 9th step with a leading backtick causing it to render incorrectly.

Resolution: This has been resolved as of **349734de4472265512a1fdf36e2c7e10ee718db3**.

Should've Used Match Instead of If Statements for TradeDirection Behavior in Swap Function

The swap function for `AMMV3Curve` uses two distinct `if` statements to determine the `TradeDirection` instead of an `if/else` pair or, better yet, a `match` expression. A `match` expression would be the most idiomatic way to handle this deviation in behavior.

Resolution: This has been resolved as of **349734de4472265512a1fdf36e2c7e10ee718db3**.

Newtype Pattern to Represent Different Pricing Units and Precisions

It was common to see the pattern `value * PRECISE_DECIMAL` or `value * LIQUITY_FEE_DECIMAL` or `value / PRECISE_DECIMAL` throughout the curve calculation code (as well as some other areas). This made this code hard to audit and reason about. Ideally, newtypes should be used here that represent the different units that you're working with to help ensure that bugs aren't introduced through the lack of context on what unit a variable is at any given point in time.

Operations could be implemented on these individual types to help ensure they work similarly to regular primitive number types. Rust will often just compile these extraneous types back into the same operations on the underlying primitives which means that the newtypes act as useful developer guardrails without any penalty to runtime (and therefore: cost to run).

Resolution: The Crema Finance team has accepted this risk and provided the following justification: "Due to the calculation precision, there is actually no good fixed-point computing library on Solana. Therefore, we determine our precision by scaling up by a factor of 10^n. n is a constant so it's reasonable to represent it with constant."

Magic Constants in SwapInstruction unpack

The `unpack` method used to take raw instruction bytes and convert that into `SwapInstruction` uses magic constants when interpreting tags. It's best practice to either use labeled constant for this or to create an enum to describe the instruction tags with a `From<u8>` trait implementation.

You can see this done with `CurveType` and a `TryFrom<u8>` implementation elsewhere in the codebase.

Resolution: The Crema Finance team has accepted this risk and provided the following justification: “This is pretty much just about coding style. It's not just an enum constant. It's more like a function.”

Spelling Mistake

`find_nearest_boudary_tick` should be `find_nearest_boundary_tick`

Resolution: This has been resolved as of **349734de4472265512a1fdf36e2c7e10ee718db3**.

Magic Constant for AccHead account_type and Version

Nearly every mention of `AccHead` uses inline magic constants for `account_type` and `swap_version` when they should instead use a constant or an enum that implements `Into<u8>`

```
AccHead {
    swap_version: 1u8,
    token_swap_key: *swap_info.key,
    account_type: 2,
    len: 0,
};
```

Resolution: This has been resolved as of **349734de4472265512a1fdf36e2c7e10ee718db3**.

Specific Recommendations

Unique to the Crema Finance Protocol

Arbitrary Write to Accounts and Complete Compromise of Swap

The `initialize` instruction does some preliminary work to set up a swap for two tokens. Ten different accounts are passed into this instruction which then begin to do validation and writing to the appropriate accounts for tracking and management of the swap. For example, `initialize` will validate that the `swap_info` account (*the destination for our new swap)* has not been initialized before proceeding.

While `swap_info` has validation done before being written to, that is not true for all of the accounts in this instruction. After the swap is created and written to the `swap_info` account, **both** the `tick_detail_info` and `user_position_info` accounts are written into without any prior checks. This enables a somewhat restricted write into *any* account directly owned by this program which allows an attacker to take over the swap entirely.

```
// No validation of the contents of `user_position_info` occurs before this line
// ...
// attacker swap pubkey written into new AccHead
let user_acc_head = AccHead {
    swap_version: 1u8,
    token_swap_key: *swap_info.key,
    account_type: 2u8,
    len: 0,
};

// takes the &mut [u8] without any checks that this account is uninitialized
let user_position_data = user_position_info.data.take(); // Vuln starts here
// writes our AccHead into the &mut [u8]
```

```
user_acc_head.pack_into_slice(&mut user_position_data[..AccHead::LEN]);
// ...
// writes back into "user_position_info"
user_position_info
    .data
    .swap(&Rc::new(RefCell::new(user_position_data))); // write occurs here
```

A malicious actor would begin to create their own swap. They start by creating some new accounts and assigning them to our application. Afterwards, they call the `initialize` function with their own `TokenA`, `TokenB`, and `swap_info`. Instead of passing in their own, new account for `user_position_info`, they pass in an existing `swap_info` that they wish to take over. As the initialization function lacks any validation on whether or not the underlying `user_position_info` account is the correct account type or it's already initialized, it will overwrite the data in whatever account is passed in.

Looking at the structure of `AccHead` we can see that it will write the following:

1. `0x01` (`swap_version`)
2. *attacker swap_info pub key - 32 bytes* (`token_swap_key`)
3. `0x02` (`account_type`)
4. `0x000000` (`len`)

For a total of 38 bytes. The first 38 bytes found in a properly initialized `swap_info` account looks like this:

1. Swap's public key (32 bytes)
2. `account_type` (1 byte)
3. `is_initialized` (1 byte)
4. `nonce` (1 byte)
5. first three bytes of `token_program_id` (3 bytes writable - 32 total)

We can see that the `account_type` (`0x02`) intersects with the `swap`'s `is_initialized` field. This would set the `is_initialized` field to `0x02` corrupting the `swap` so it can no longer be unpacked properly.

```
is_initialized: match is_initialized {
    [0] => false,
    [1] => true,
    _ => return Err(ProgramError::InvalidAccountData), // 0x02 goes here
},
```

When the `swap` cannot be unpacked, the `SwapVersion::is_initialized` returns `false`.

```
pub fn is_initialized(input: &[u8]) -> bool {
    match Self::unpack(input) {
        Ok(swap) => swap.is_initialized(),
        Err(_) => false, // our ProgramError::InvalidAccountData goes here
    }
}
```

This should be sufficient to completely hijack the swap. The attacker could completely reinitialize the swap with the original fields but with their own management keys in place allowing them to sign for privileged transactions.

Resolution: An initialization check has been added in **349734de4472265512a1fdf36e2c7e10ee718db3**.

Unchecked Mathematical Operations Introduce Overflow/Underflow Risk in AMMV3Curve

A majority of the mathematical operations in the `CurveCalculator` implementation for the `AMMV3Curve` fail to use checked operations which allow strong protection against overflow and underflows. Due to the density of operations performed in these functions specific instances of overflow or underflow were not articulated.

Resolution: The Crema Finance team has accepted this risk and provided the following justification: “To avoid exceeding Solana's computational limit, on one hand, we made every effort to ensure that all calculations do not overflow, and where there is a chance of overflow, we'll do checks; on the other hand, we discarded the checked operation to save the

computational resources. Therefore, the checked operation is discarded with the prerequisite that there is no overflow occurring."

Unchecked Mathematical Operations Introduce Overflow/Underflow Risk in tick_utils

A majority of the mathematical operations in `tick_utils.rs` fail to use checked operations which allow strong protection against overflow and underflows. Due to the density of operations performed in these functions specific instances of overflow or underflow were not articulated.

Resolution: The Crema Finance team has accepted this risk and provided the following justification: "To avoid exceeding Solana's computational limit, on one hand, we made every effort to ensure that all calculations do not overflow, and where there is a chance of overflow, we'll do checks; on the other hand, we discarded the checked operation to save the computational resources. Therefore, the checked operation is discarded with the prerequisite that there is no overflow occurring."

Spoofable tick_detail_info Account in Swap Instruction

The `swap` instruction does not check that the `tick_detail_info` passed in belongs to the `swap` which would allow an attacker to pass in a `tick_detail_info` account created on an attacker's swap. The attacker could prime their `tick_detail_info` account to present very favorable terms in a particular direction. The attacker could then perform a swap against the victim swap and use their primed `tick_detail_info` account.

```
// never checked if tick_detail_info belongs to the token_swap
// attacker can use their own tick_detail_info on their fake swap
let tick_info_data = tick_detail_info.data.take();
let acc_head = AccHead::unpack_from_slice(&tick_info_data[..AccHead::LEN]).unwrap();
let swap_curve = token_swap.swap_curve();
let tick_append_index: usize = acc_head.len as usize;
let mut tick_word = TickWord::new(&mut tick_info_data[AccHead::LEN..], tick_append_index);
let amount_out = swap_curve.swap(
    to_u128(amount_in)?,
    &mut tick_word,
```

```
    trade_direction,
);
```

Resolution: A check of the tick info account token swap has been added as of **349734de4472265512a1fdf36e2c7e10ee718db3**.

Missing Owner Check for swap_info in Add User Position Instruction

The Add User Position instruction is missing a check that validates that the provided `swap_info` account is owned by the contract. This means that an attacker can create an account with a `data` field that properly unpacks into a `SwapV1` eventually leading to the attacker controlled public key being written into the `user_position_info`.

Resolution: This has been resolved as of **349734de4472265512a1fdf36e2c7e10ee718db3**.

Arbitrary Account Write in Add User Position Instruction

Similar to the arbitrary write in the initialize instruction, the `user_position_info` is not checked as being associated with the swap before writing has occurred. This allows an arbitrary account that is owned by the contract (but not necessarily the `swap`) to be passed in and written to. This instruction produces the same condition that leads to `swap` reinitialization as in the arbitrary account write initialization vulnerability.

Resolution: An initialization and permission check has been added as of **349734de4472265512a1fdf36e2c7e10ee718db3**.

Toolset Warnings

Unique to the Crema Finance Protocol

Overview

In addition to our manual review, our process involves utilizing static analysis and formal methods in order to perform additional verification of the presence of security vulnerabilities (or lack thereof). An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.

The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable.

Compilation Warnings

No warnings were present at time of compilation.

Test Coverage

The contract repository possesses extensive unit test coverage throughout. This testing provides a variety of unit tests which encompass the various operational stages of the contract.

Static Analysis Coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Semgrep

In each case, the team had either mitigated relevant concerns raised by each of these tools or provided adequate justification for the risk (such as adhering to a standard), or a concern stemming from the discovered risk was elevated to a larger issue and is referenced above.

Directory Structure

At time of review, the directory structure of the Crema Finance smart contracts repository appeared as it does below. Our review, at request of Crema Finance, covers the Rust code (*.rs) as of commit-hash **a701276f9797e73abf304119e2dad8e47c3baa98** of the Crema Finance

repository.

```
.
├── Cargo.lock
├── Cargo.toml
├── Xargo.toml
├── cbindgen.toml
├── fuzz
│   ├── Cargo.toml
│   └── src
│       ├── instructions.rs
│       ├── lib.rs
│       ├── native_account_data.rs
│       ├── native_processor.rs
│       ├── native_token.rs
│       └── native_token_swap.rs
├── inc
│   └── token-swap.h
├── libraries
│   └── math
│       ├── Cargo.lock
│       ├── Cargo.toml
│       ├── Xargo.toml
│       ├── src
│       │   ├── approximations.rs
│       │   ├── checked_ceil_div.rs
│       │   ├── entrypoint.rs
│       │   ├── error.rs
│       │   ├── instruction.rs
│       │   ├── lib.rs
│       │   ├── precise_number.rs
│       │   ├── processor.rs
```

```
│   │   └── uint.rs
│   └── tests
│       └── instruction_count.rs
├── program-id.md
├── sim
│   ├── Cargo.lock
│   ├── Cargo.toml
│   ├── simulation.py
│   └── src
│       └── lib.rs
├── src
│   ├── acc_head.rs
│   ├── curve
│   │   ├── amm_v3.rs
│   │   ├── ammv3.py
│   │   ├── base.rs
│   │   ├── calculator.rs
│   │   └── mod.rs
│   ├── entrypoint.rs
│   ├── error.rs
│   ├── instruction.rs
│   ├── lib.rs
│   ├── position.rs
│   ├── processor.rs
│   ├── state.rs
│   ├── tick_utils.rs
│   └── tickmap.rs
└── token
    ├── README.md
    ├── cli
    │   ├── Cargo.toml
```

```
│   ├── README.md
│   └── src
│       ├── main.rs
│       └── sort.rs
├── js
│   ├── README.md
│   ├── babel.config.json
│   ├── babel.rollup.config.json
│   ├── cli
│   │   ├── main.js
│   │   ├── store
│   │   │   └── index.js
│   │   └── token-test.js
│   ├── client
│   │   ├── layout.js
│   │   ├── token.js
│   │   └── util
│   │       ├── new-account-with-lamports.js
│   │       ├── new-system-account-with-airdrop.js
│   │       ├── send-and-confirm-transaction.js
│   │       └── sleep.js
│   ├── cluster-devnet.env
│   ├── cluster-mainnet-beta.env
│   ├── cluster-testnet.env
│   ├── flow-typed
│   │   └── npm
│   │       ├── @solana
│   │       │   └── web3.js_vx.x.x.js
│   │       ├── bn.js_vx.x.x.js
│   │       ├── buffer-layout_vx.x.x.js
│   │       ├── buffer_vx.x.x.js
```

```
│   │   ├── chai-as-promised_vx.x.x.js
│   │   ├── chai_v4.x.x.js
│   │   ├── mkdirp_vx.x.x.js
│   │   ├── mocha_v8.x.x.js
│   │   └── mz_vx.x.x.js
│   ├── mocha.html
│   ├── module.d.ts
│   ├── module.flow.js
│   ├── package-lock.json
│   ├── package.json
│   ├── rollup.config.js
│   ├── test
│   │   ├── rollup.config.js
│   │   └── token.test.js
│   └── url.js
├── perf-monitor
│   ├── Cargo.toml
│   └── tests
│       └── assert_instruction_count.rs
└── program
    ├── Cargo.lock
    ├── Cargo.toml
    ├── Xargo.toml
    ├── inc
    │   └── token.h
    ├── program-id.md
    └── src
        ├── entrypoint.rs
        ├── error.rs
        ├── instruction.rs
        ├── lib.rs
```

```
├── native_mint.rs
├── processor.rs
└── state.rs
```

28 directories, 97 files