HALBORN

Jet Protocol - Margin and Multisig

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 10th, 2022 - May 8th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/25/2022	Przemyslaw Swiatowiec
0.2	Draft Review	05/10/2022	Gabi Urrutia
1.0	Remediation Plan	06/01/2022	Przemyslaw Swiatowiec
1.1	Remediation Plan Review	06/02/2022	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
Przemyslaw Swiatowiec	Halborn	Przemyslaw.Swiatowiec@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Jet Protocol is an open-source, noncustodial, borrowing and lending protocol on the Solana Blockchain. A Jet user can borrow against over-collateralized debt positions and may incur debt up to governance-mandated debt ratios. If the value of a user's deposited collateral falls under the specified ratio, their position may be liquidated by external actors, such as traders or any users who can call the program.

Jet Protocol engaged Halborn to conduct a security assessment of Margin and Multisig programs on April 10th, 2022 , and ending May 8th, 2022 . This security assessment was scoped to the Solana programs included in the jet-margin and multisig repositories. Commit hashes, and further details, can be found in the Scope section of this report.

Margin programs provide borrow and lending functionality, token pool implementation, tokens swapping, and programs to perform administrative tasks to protocol. Multisig is a fork of serum multisig program with the custom feature that allows multisig owners to delegate signing permissions to other users.

1.2 AUDIT SUMMARY

The team at Halborn was provided 4 weeks for the engagement and assigned one full-time security engineer to audit the security of the programs in scope. The security engineer is a blockchain and Solana program security expert/experts with advanced penetration testing and Solana program hacking skills, and in-depth knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that program functions operate as intended
- Identify potential security issues with the programs
- In summary, Halborn identified some risks that were mostly addressed

by the Jet Protocol .

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review 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 the architecture, purpose, and use of the platform.
- Solana program manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Finding unsafe Rust code usage (cargo-geiger).
- Scanning dependencies for known vulnerabilities (cargo audit).
- Local cluster deployment.
- Scanning for common Solana vulnerabilities (soteria).

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. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that

were 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.

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

1.4 SCOPE

1. Solana programs
    (a) Repository: jet-margin
    (b) Commit ID: ed6cbadab4171ca5a23d1c3ae471d83ba75f501f
    (c) Programs in-scope:
        i. programs/control
        ii. programs/margin-pool
        iii. programs/margin-swap
        iv. programs/margin
        v. programs/metadata

    (a) Repository: multisig
    (b) Commit ID: 3071178c0bf157a44a8e01c4f5156c4b2349a7ac
    (c) Programs in-scope:
        i. programs/multisig

Out of scope: dependencies, packages and libraries.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	2	1	4	2

LIKELIHOOD

IMPACT

		(HAL-05)		(HAL-01) (HAL-02) (HAL-03)
	(HAL-06)			
(HAL-10)	(HAL-07)			(HAL-04)
	(HAL-08) (HAL-09)			
(HAL-11) (HAL-12)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) WITHDRAWAL WITHOUT THE COLLATERALIZATION RATIO CHECK	Critical	SOLVED - 05/11/2022
(HAL-02) MULTISIG OWNER PROPOSAL IMPERSONATION	Critical	SOLVED - 05/31/2022
(HAL-03) MISSING PRICE ORACLE STATUS CHECK	Critical	SOLVED - 05/22/2022
(HAL-04) LIQUIDATOR METADATA ACCOUNTS UNAVAILABLE	High	SOLVED - 05/23/2022
(HAL-05) ORACLE ADDRESSES CHECK MISSING	High	RISK ACCEPTED
(HAL-06) USING FRAMEWORK VERSION WITH KNOWN VULNERABILITIES	Medium	RISK ACCEPTED
(HAL-07) MARGIN POOLS CAN BE MISCONFIGURED	Low	RISK ACCEPTED
(HAL-08) THRESHOLD SANITY CHECK MISSING IN THE MULTISIG PROGRAM	Low	SOLVED - 05/31/2022
(HAL-09) MULTISIG CAN BE SET WITHOUT OWNERS	Low	SOLVED - 05/31/2022
(HAL-10) HARDCODED GOVERNANCE ADDRESS	Low	RISK ACCEPTED
(HAL-11) ACCOUNT TYPE CHECKS MISSING	Informational	ACKNOWLEDGED
(HAL-12) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) WITHDRAWAL WITHOUT THE COLLATERALIZATION RATIO CHECK - CRITICAL

Description:

The margin-swap program is a wrapper for Solana Token Swap and allows users to swap tokens between different margin pools. The margin program verifies the health of the position after the swap. It should not be possible to end up with positions that exceed the required collateralization ratio.

The MarginSwap instruction handler does not verify the address of the swap_program account. A malicious user could set it to the address of a malicious program and hijack the MarginSwap instruction signed by margin_account. All instructions signed by margin_accounts should go through the AdapterInvoke instruction handler. The handler checks the status of the Position; however, by replacing the swap_program account address with a malicious swap program ID, the attacker could invoke his program and send a Withdraw instruction (not the adapter's MarginWithdraw) on behalf of the margin account and withdraw all tokens registered in their other positions, skipping all position health checks.

Code Location:

Listing 1: programs/margin-swap/src/lib.rs (Line 104)

```
82 pub struct MarginSwap<'info> {
83     /// The margin account being executed on
84     #[account(signer)]
85     pub margin_account: AccountLoader<'info, MarginAccount>,
86
87     /// The account with the source deposit to be exchanged from
88     #[account(mut)]
89     pub source_account: AccountInfo<'info>,
90
91     /// The destination account to send the deposit that is
↳ exchanged into
```

```
92      #[account(mut)]
93      pub destination_account: AccountInfo<'info>,
94
95      /// Temporary account for moving tokens
96      #[account(mut)]
97      pub transit_source_account: AccountInfo<'info>,
98
99      /// Temporary account for moving tokens
100     #[account(mut)]
101     pub transit_destination_account: AccountInfo<'info>,
102
103     /// The accounts relevant to the swap pool used for the
 ↳ exchange
104     pub swap_info: SwapInfo<'info>,
105
106     /// The accounts relevant to the source margin pool
107     pub source_margin_pool: MarginPoolInfo<'info>,
```

Listing 2: programs/margin-swap/src/lib.rs (Line 168)

```
152 pub struct SwapInfo<'info> {
153     pub swap_pool: UncheckedAccount<'info>,
154     pub authority: UncheckedAccount<'info>,
155
156     #[account(mut)]
157     pub vault_into: UncheckedAccount<'info>,
158
159     #[account(mut)]
160     pub vault_from: UncheckedAccount<'info>,
161
162     #[account(mut)]
163     pub token_mint: UncheckedAccount<'info>,
164
165     #[account(mut)]
166     pub fee_account: UncheckedAccount<'info>,
167
168     pub swap_program: UncheckedAccount<'info>, // FIXME: need
 ↳ program id ?
169 }
170
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Consider introducing swap_program address checking to validate if the user provided the correct spl_token_swap program ID.

Remediation Plan:

SOLVED: The issue was fixed by introducing spl_swap program ownership check (7f296101e261b738bdd7b1db2542d99734189016).

3.2 (HAL-02) MULTISIG OWNER PROPOSAL IMPERSONATION - CRITICAL

Description:

The CreateTransaction instruction in the multisig program creates a proposal for a transaction to be signed by the multisig. The proposer's account address is registered as a vote for the proposal. This instruction is restricted to members of the multisig owners group or accounts on the owners delegate list only.

Anchor's try_deserialize function checks does not perform any account ownership checks. An attacker may use an arbitrary account with a forged multisig owner delegate list and impersonate any multisig owner. As a result, anyone could create and approve proposals on behalf of any user.

Code Location:

Listing 3: programs/multisig/src/lib.rs (Lines 98-99,101,106,111,127)

```
96 let owner_key = match ctx.remaining_accounts.get(0) {
97     Some(delegate_list_account) => {
98         let delegate_list =
99             DelegateList::try_deserialize(&mut &
↳ delegate_list_account.data.borrow()[..])?;
100
101         if delegate_list.multisig != ctx.accounts.multisig.key() {
102             msg!("delegate list isn't for this multisig");
103             return Err(ErrorCode::InvalidOwner.into());
104         }
105
106         if !delegate_list.delegates.contains(ctx.accounts.proposer
↳ .key) {
107             msg!("delegate isn't in the allowed list");
108             return Err(ErrorCode::InvalidOwner.into());
109         }
110
111         delegate_list.owner
112     }
```

```
113
114     None => *ctx.accounts.proposer.key,
115 };
116
117 let owner_index = ctx
118     .accounts
119     .multisig
120     .owners
121     .iter()
122     .position(|a| *a == owner_key)
123     .ok_or(ErrorCode::InvalidOwner)?;
124
125 let mut signers = Vec::new();
126 signers.resize(ctx.accounts.multisig.owners.len(), false);
127 signers[owner_index] = true;
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Consider checking the ownership of the DelegateList account in the create_transaction instruction handler.

Remediation Plan:

SOLVED: The issue was fixed by introducing delegated list ownership check (fb786a0fa72ae7ef8c2e271cb1b8a75b60546aa0).

3.3 (HAL-03) MISSING PRICE ORACLE STATUS CHECK - CRITICAL

Description:

The margin program uses pyth oracle in asset valuation. It should be noted that pyth requires clients to check product status (pyth best practices):

From Pyth documentation: “Its possible that Pyth will not have a valid price for a product. This situation can happen for various reasons. For example, US equity markets only trade during certain hours, and outside those hours, its not clear what an equitys price is. Alternatively, Solana congestion may prevent data publishers from being able to submit their prices. During these periods, Pyth will not have a valid price for a product. Pyths price accounts have a status field that indicates whether the price is valid. A status of trading indicates a valid price that is permissible to use in downstream applications. If the status is not trading, the Pyth price can be an arbitrary value.”

The valuation function in Margin program does not check the price account status. Even when a price feed is not trading, the program still reevaluates assets at possibly arbitrary price, which can result in incorrect asset valuation and fund loss.

Code Location:

Listing 4: programs/margin/src/adapter.rs (Lines 138-152)

```
138 for entry in price_list {
139     let twap = Number128::from_decimal(entry.twap, entry.exponent)
 ↳ ;
140     let confidence = Number128::from_decimal(entry.confidence,
 ↳ entry.exponent);
141
142     let price = match (confidence, entry.slot) {
143         (c, _) if (c / twap) > max_confidence => PriceInfo::
 ↳ new_invalid(),
144         (_, slot) if (clock.slot - slot) > MAX_ORACLE_STALENESS =>
```

```
↳ {
145             PriceInfo::new_invalid()
146         }
147         _ => PriceInfo::new_valid(
148             entry.exponent,
149             entry.value,
150             clock.unix_timestamp as u64,
151         ),
152     };
153
154     match margin_account.set_position_price(
155         &entry.mint,
156         ctx.adapter_program.key,
157         &price,
158     ) {
159         Err(Error::AnchorError(e))
160             if e.error_code_number
161                 == (ErrorCode::UnknownPosition as u32
162                     + anchor_lang::error::ERROR_CODE_OFFSET) =>
163         {
164             ()
165         }
166         Err(e) => return Err(e),
167         Ok(()) => (),
168     }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Check the price account status field before consuming the price and gracefully handle the case when pricing is currently unavailable.

Remediation Plan:

SOLVED: The issue was fixed by using official pyth library with price status check (b5c8e62c0d37b1a5a00b80cf646dc8e6b05d3379).

3.4 (HAL-04) LIQUIDATOR METADATA ACCOUNTS UNAVAILABLE - HIGH

Description:

Starting the collateral liquidation process requires a valid liquidator metadata account. Protocol governance is handled with the control program, which should be able to create such accounts.

There is no instruction to create liquidator metadata in the current implementation. As a result, an essential margin program feature is unavailable, as collateral liquidation cannot be executed.

Code Location:

Listing 5: programs/margin/src/instructions/liquidate_begin.rs (Lines 22,25-26)

```
11 #[derive(Accounts)]
12 pub struct LiquidateBegin<'info> {
13     /// The account in need of liquidation
14     #[account(mut)]
15     pub margin_account: AccountLoader<'info, MarginAccount>,
16
17     /// The address paying rent
18     #[account(mut)]
19     pub payer: Signer<'info>,
20
21     /// The liquidator account performing the liquidation actions
22     pub liquidator: Signer<'info>,
23
24     /// The metadata describing the liquidator
25     #[account(has_one = liquidator)]
26     pub liquidator_metadata: Account<'info, LiquidatorMetadata>,
```

Listing 6: programs/metadata/src/lib.rs (Line 30)

```
30 #[cfg_attr(not(feature = "devnet"), account(owner =
↳ CONTROL_PROGRAM_ID))]
```

```
31 pub authority: Signer<'info>,
32
33 /// The address paying the rent for the account
34 #[account(mut)]
35 pub payer: Signer<'info>,
36
37 pub system_program: Program<'info, System>,
```

Listing 7: programs/metadata/src/lib.rs (Line 47)

```
40 #[derive(Accounts)]
41 pub struct SetEntry<'info> {
42     /// The account containing the metadata to change
43     #[account(mut)]
44     pub metadata_account: AccountInfo<'info>,
45
46     /// The authority that must sign to make this change
47     #[cfg_attr(not(feature = "devnet"), account(owner =
↳ CONTROL_PROGRAM_ID))]
48     pub authority: Signer<'info>,
49 }
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Consider implementing instruction to set up liquidator in the control program.

Remediation Plan:

SOLVED: The issue was fixed by introducing function set_liquidator_handler in the margin control program (1692799f9d5bfea86f0465d768c3ebb7a7038a81).

3.5 (HAL-05) ORACLE ADDRESSES CHECK MISSING - HIGH

Description:

Owners of oracle accounts (pyth_price and pyth_product) are not validated. The margin program may end up using malicious price feeders. Additional vulnerable code could be found in the margin_refresh_position instruction handler, where token_price_oracle is deserialized without checking the account ownership.

What is more, the margin program does not validate if the correct pyth_product account is provided for a given pyth_price. Pyth uses an associative array structure, in which each pyth_account has a corresponding price account. In the current implementation, it's possible to set up mismatched pyth_product and pyth_account for an asset. It may be unclear and misleading to users, as both account addresses are saved in token metadata.

Code Location:

Listing 8: programs/control/src/instructions/configure_token.rs (Lines 46,47)

```
39
40      #[account(mut, has_one = token_mint)]
41      pub token_metadata: Account<'info, TokenMetadata>,
42
43      #[account(mut, constraint = deposit_metadata.
↳ underlying_token_mint == token_mint.key())]
44      pub deposit_metadata: Account<'info, PositionTokenMetadata>,
45
46      pub pyth_product: UncheckedAccount<'info>,
47      pub pyth_price: UncheckedAccount<'info>,
48
49      pub margin_pool_program: Program<'info, JetMarginPool>,
50      pub metadata_program: Program<'info, JetMetadata>,
51  }
```

Listing 9: programs/margin-pool/src/instructions/configure.rs (Lines 16,17)

```
 7 pub struct Configure<'info> {
 8     /// The pool to be configured
 9     #[account(mut)]
10     pub margin_pool: Account<'info, MarginPool>,
11
12     /// The authority allowed to modify the pool, which must sign
13     #[account(owner = CONTROL_PROGRAM_ID)]
14     pub authority: Signer<'info>,
15
16     pub pyth_product: AccountInfo<'info>,
17     pub pyth_price: AccountInfo<'info>,
18 }
```

Listing 10: programs/margin-pool/src/instructions/configure.rs (Lines 16,17,35,38)

```
16     pub pyth_product: AccountInfo<'info>,
17     pub pyth_price: AccountInfo<'info>,
18 }
19
20 pub fn configure_handler(
21     ctx: Context<Configure>,
22     fee_destination: Option<Pubkey>,
23     config: Option<MarginPoolConfig>,
24 ) -> Result<()> {
25     let pool = &mut ctx.accounts.margin_pool;
26
27     if let Some(new_fee_destination) = fee_destination {
28         pool.fee_destination = new_fee_destination;
29     }
30
31     if let Some(new_config) = config {
32         pool.config = new_config;
33     }
34
35     if *ctx.accounts.pyth_price.key != Pubkey::default() {
36         // FIXME: validate pyth product
37
38         pool.token_price_oracle = ctx.accounts.pyth_price.key();
39         msg!("oracle = {}", &pool.token_price_oracle);
40     }
```

```
41
42      Ok(())
43  }
44
```

Listing 11: programs/control/src/instructions/configure_token.rs (Lines 114,115)

```
110 if *ctx.accounts.pyth_price.key != Pubkey::default() {
111     let mut metadata = ctx.accounts.token_metadata.clone();
112     let mut data = vec![];
113
114     metadata.pyth_product = ctx.accounts.pyth_product.key();
115     metadata.pyth_price = ctx.accounts.pyth_price.key();
116
117     metadata.try_serialize(&mut data)?;
118
```

Listing 12: programs/margin-pool/src/instructions/margin_refresh_position.rs (Lines 19,27)

```
15      #[account(has_one = token_price_oracle)]
16      pub margin_pool: Account<'info, MarginPool>,
17
18      /// The pyth price account for the pool's token
19      pub token_price_oracle: AccountInfo<'info>,
20  }
21
22  pub fn margin_refresh_position_handler(ctx: Context<
↳ MarginRefreshPosition>) -> Result<()> {
23      let pool = &ctx.accounts.margin_pool;
24
25      // update the oracles with the pyth format
26      let token_oracle_data = ctx.accounts.token_price_oracle.
↳ try_borrow_data()?;
27      let token_oracle = bytemuck::from_bytes::<Price>(&
↳ token_oracle_data);
28
29      let prices = pool.calculate_prices(token_oracle);
30
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

Consider oracle accounts owner validation and verify if provided pyth_product account matches pyth_price account.

Remediation Plan:

RISK ACCEPTED: The Jet Protocol team decides to accept this risk, as it may be necessary to add alternative oracles with the same pyth format.

3.6 (HAL-06) USING FRAMEWORK VERSION WITH KNOWN VULNERABILITIES - MEDIUM

Description:

Multisig and margin programs as using Anchor from jet-labs repository- it's a fork from the original Anchor repository. It was noticed that at the time of an audit (04/27/2022):

1. Multisig is using Anchor 0.23.0.
2. Margin programs are using Anchor 0.20.1.

It's highly recommended to update corresponding Anchor repository and programs dependencies to at least Anchor 0.24.2, as all versions below are known of having critical vulnerability and should not be used.

II Missing Anchor version specification

Multisig and margin programs are missing Anchor version specification. Both anchor-lang and anchor-spl dependencies in audited programs are pointing to Anchor jet-labs fork repository to master (margin programs) and accounts branch (multisig program). It's considered as bad practice to not specify dependency version that should be used:

1. In case of critical/high error in used dependency, it's harder to verify if project is using vulnerable version - incident response process is harder to perform.
2. Missing information for which version of dependency program was tested.
3. Lack of control over dependency update and its consequences, such as breaking changes or dependency collisions. Dependency update is not requiring separate pull request and code review. It could be introduced by pulling forked repository and could be done without project collaborators approval.

Code Location:

Listing 13: programs/multisig/Cargo.rust (Line 19)

```
18 [dependencies]
19 anchor-lang = { git = "https://github.com/jet-lab/anchor", branch
↳ = "accounts" }
```

Listing 14: programs/control/Cargo.rust (Lines 22-23)

```
20 [dependencies]
21 solana-program = "1.9"
22 anchor-lang = { git = "https://github.com/jet-lab/anchor" }
23 anchor-spl = { git = "https://github.com/jet-lab/anchor" }
```

Listing 15: programs/margin/Cargo.rust (Lines 23-24)

```
23 anchor-lang = { git = "https://github.com/jet-lab/anchor" }
24 anchor-spl = { git = "https://github.com/jet-lab/anchor" }
```

Listing 16: programs/margin-pool/Cargo.rust (Lines 25-26)

```
25 anchor-lang = { git = "https://github.com/jet-lab/anchor" }
26 anchor-spl = { git = "https://github.com/jet-lab/anchor" }
```

Listing 17: programs/margin-swap/Cargo.rust (Lines 21-22)

```
19
20 [dependencies]
21 anchor-lang = { git = "https://github.com/jet-lab/anchor" }
22 anchor-spl = { git = "https://github.com/jet-lab/anchor" }
```

Listing 18: programs/metadata/Cargo.rust (Line 20)

```
19 [dependencies]
20 anchor-lang = { git = "https://github.com/jet-lab/anchor" }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Consider using proper Cargo.toml syntax to control the Anchor version that should be used by the programs. It could be done by specifying package version or expected version commit ID (ref). Update the Anchor fork and make sure that programs are using the latest Anchor version.

Update Anchor version to the newest one (by the time of writing the report - at least 0.24.2).

Remediation Plan:

RISK ACCEPTED: The Jet Protocol team decides to accept this risk as the mentioned anchor versions are vulnerable if init_if_needed is used. Multisig and margin programs are not using this function.

3.7 (HAL-07) MARGIN POOLS CAN BE MISCONFIGURED - LOW

Description:

The ConfigureToken instruction handler can be invoked by an authority to set up several token pool parameters, including the type of token and its collateral parameters (weight, max staleness), a fee vault and relevant Pyth oracle price and product accounts. Two problems were identified:

1. Margin pool can be configured without proper oracle accounts. In such case, it would not be possible to register positions. Every margin position valuation would fail, as it would not be possible to fetch the relevant asset price.
2. It's possible to create margin pools without fee destination account set. In such case, another governance voting should be done to allow for fee collection.

Code Location:

Listing 19: programs/control/src/instructions/configure_token.rs (Lines 95-97,110,128)

```
95  if *ctx.accounts.pyth_price.key != Pubkey::default()
96      || pool_param.is_some()
97      || pool_config.is_some()
98  {
99      let fee_destination = pool_param.map(|p| p.fee_destination);
100
101     jet_margin_pool::cpi::configure(
102         ctx.accounts
103             .configure_pool_context()
104             .with_signer(&[&authority]),
105         fee_destination,
106         pool_config,
107     )?;
108 }
109
110 if *ctx.accounts.pyth_price.key != Pubkey::default() {
111     let mut metadata = ctx.accounts.token_metadata.clone();
```

```
112         let mut data = vec![];
113
114         metadata.pyth_product = ctx.accounts.pyth_product.key();
115         metadata.pyth_price = ctx.accounts.pyth_price.key();
116
117         metadata.try_serialize(&mut data)?;
118
119         jet_metadata::cpi::set_entry(
120             ctx.accounts
121                 .set_metadata_context()
122                 .with_signer(&[&authority]),
123             0,
124             data,
125         )?;
126     }
127
128     if let Some(params) = metadata {
129         let mut metadata = ctx.accounts.deposit_metadata.clone();
130         let mut data = vec![];
131
132         metadata.token_kind = params.token_kind;
133         metadata.collateral_weight = params.collateral_weight;
134         metadata.collateral_max_staleness = params.
 ↳ collateral_max_staleness;
135
136         metadata.try_serialize(&mut data)?;
137
138         jet_metadata::cpi::set_entry(
139             ctx.accounts
140                 .set_deposit_metadata_context()
141                 .with_signer(&[&authority]),
142             0,
143             data,
144         )?;
145     }
146
147     Ok(())
```

Listing 20: programs/margin-pool/src/instructions/configure.rs (Lines 27,28)

```
20 pub fn configure_handler(
21     ctx: Context<Configure>,
```

```
22      fee_destination: Option<Pubkey>,
23      config: Option<MarginPoolConfig>,
24  ) -> Result<()> {
25      let pool = &mut ctx.accounts.margin_pool;
26
27      if let Some(new_fee_destination) = fee_destination {
28          pool.fee_destination = new_fee_destination;
29      }
```

Listing 21: programs/margin-pool/src/instructions/configure.rs (Line 35)

```
35  if *ctx.accounts.pyth_price.key != Pubkey::default() {
36      // FIXME: validate pyth product
37
38      pool.token_price_oracle = ctx.accounts.pyth_price.key();
39      msg!("oracle = {}", &pool.token_price_oracle);
40  }
```

Listing 22: programs/margin-pool/src/instructions/collect.rs (Lines 12,21)

```
 6  #[derive(Accounts)]
 7  pub struct Collect<'info> {
 8      /// The pool to be refreshed
 9      #[account(mut,
10                has_one = vault,
11                has_one = deposit_note_mint,
12                has_one = fee_destination)]
13      pub margin_pool: Account<'info, MarginPool>,
14
15      /// The vault for the pool, where tokens are held
16      #[account(mut)]
17      pub vault: AccountInfo<'info>,
18
19      /// The account to deposit the collected fees
20      #[account(mut)]
21      pub fee_destination: AccountInfo<'info>,
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider introducing checks to prevent creating misconfigured token pools without oracle and fee destination accounts set up properly.

Remediation Plan:

RISK ACCEPTED: The Jet Protocol team decides to accept this risk as the configure_token function should create a pool of token and update some of its parameters.

3.8 (HAL-08) THRESHOLD SANITY CHECK MISSING IN THE MULTISIG PROGRAM - LOW

Description:

In the multisig program, the `create_multisig` function verifies the minimum required threshold (above 0) only, but it's possible to set up multisigs with threshold that exceeds the number of multisig owners - proposals cannot be approved as program expects more votes than available voters.

Code Location:

Listing 23: programs/multisig/src/lib.rs (Line 42)

```
41 assert_unique_owners(&owners)?;
42 require!(threshold > 0, InvalidThreshold);
43
44 let multisig = &mut ctx.accounts.multisig;
45 multisig.owners = owners;
46 multisig.signer = ctx.accounts.signer.key();
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider verifying `threshold` does not exceed `multisig.owners.len()`.

Remediation Plan:

SOLVED: The issue was fixed by introducing check to disallow thresholds exceeding the number of owners (fb786a0fa72ae7ef8c2e271cb1b8a75b60546aa0).

3.9 (HAL-09) MULTISIG CAN BE SET WITHOUT OWNERS - LOW

Description:

Multisig accounts can be set up without owners. Such accounts cannot be used as there have to be at least one owner to propose and approve transaction.

Code Location:

Listing 24: programs/multisig/src/lib.rs (Line 45)

```
41 assert_unique_owners(&owners)?;
42 require!(threshold > 0, InvalidThreshold);
43
44 let multisig = &mut ctx.accounts.multisig;
45 multisig.owners = owners;
46 multisig.signer = ctx.accounts.signer.key();
```

Listing 25: programs/multisig/src/lib.rs (Line 201)

```
196
197 if (owners.len() as u64) < multisig.threshold {
198     multisig.threshold = owners.len() as u64;
199 }
200
201 multisig.owners = owners;
202 multisig.owner_set_seqno += 1;
```

Listing 26: programs/multisig/src/lib.rs

```
453 fn assert_unique_owners(owners: &[Pubkey]) -> Result<()> {
454     let mut uniq_owners = owners.to_vec();
455     uniq_owners.sort();
456     uniq_owners.dedup();
457     require!(owners.len() == uniq_owners.len(), UniqueOwners);
458     Ok(())
459 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider implementing a check to verify the number of owners on multi-sig account initialization. At least one owner should be required to initialize multisig account.

Remediation Plan:

SOLVED: The issue was fixed by introducing a check to prevent multisig configuration without owners (fb786a0fa72ae7ef8c2e271cb1b8a75b60546aa0).

3.10 (HAL-10) HARDCODED GOVERNANCE ADDRESS - LOW

Description:

An important governance account address is hardcoded in programs/control/src/lib.rs. In case this address is compromised, the program owner would have to redeploy the program to update it.

Code Location:

Listing 27: programs/control/src/lib.rs (Line 13)

```
13 #[cfg(not(feature = "devnet"))]
14 static ROOT_AUTHORITY: Pubkey = pubkey!("
↳ FqXoGb9Zxy4uzG12N1jvHyktNG3Zsez367vAzJeiyMF1");
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Consider making the governance address mutable and implement a function to update this address in case it is compromised.

Remediation Plan:

RISK ACCEPTED: The Jet Protocol team decides to accept this risk, as if a governance compromise or other incident, the team will update the program with new governance address. The Jet protocol team is aware of program upgrade costs.

3.11 (HAL-11) ACCOUNT TYPE CHECKS MISSING - INFORMATIONAL

Description:

The Anchor framework can check the type and ownership of user-supplied accounts. Marking accounts `UncheckedAccount<'info>` however disables this features and introduces some risk of accepting forged accounts, e.g., `mint` accounts that are not created with the SPL Token program. Code locations listed below miss account checks which do not directly present security vulnerabilities. It's recommended to check all supplied account types or addresses in all possible places though to further improve the security posture of the program.

Code Location:

Listing 28: programs/control/src/instructions/configure_token.rs (Line 35)

```
30 pub struct ConfigureToken<'info> {
31     #[cfg_attr(not(feature = "devnet"), account(address = crate::
↳ ROOT_AUTHORITY))]
32     pub requester: Signer<'info>,
33     pub authority: Account<'info, Authority>,
34
35     pub token_mint: UncheckedAccount<'info>,
36
37     #[account(mut, has_one = token_mint)]
38     pub margin_pool: Account<'info, MarginPool>,
39
40     #[account(mut, has_one = token_mint)]
41     pub token_metadata: Account<'info, TokenMetadata>,
42
43     #[account(mut, constraint = deposit_metadata.
↳ underlying_token_mint == token_mint.key())]
44     pub deposit_metadata: Account<'info, PositionTokenMetadata>,
```

Listing 29: programs/control/src/instructions/register_token.rs (Line 32)

```
29 #[account(mut)]
30 loan_note_mint: UncheckedAccount<'info>,
31
32 token_mint: UncheckedAccount<'info>,
33
34 #[account(mut)]
35 token_metadata: UncheckedAccount<'info>,
```

Listing 30: programs/margin-pool/src/instructions/deposit.rs (Lines 17,21,28,32)

```
 8 pub struct Deposit<'info> {
 9     /// The pool to deposit into
10     #[account(mut,
11               has_one = vault,
12               has_one = deposit_note_mint)]
13     pub margin_pool: Account<'info, MarginPool>,
14
15     /// The vault for the pool, where tokens are held
16     #[account(mut)]
17     pub vault: UncheckedAccount<'info>,
18
19     /// The mint for the deposit notes
20     #[account(mut)]
21     pub deposit_note_mint: UncheckedAccount<'info>,
22
23     /// The address with authority to deposit the tokens
24     pub depositor: Signer<'info>,
25
26     /// The source of the tokens to be deposited
27     #[account(mut)]
28     pub source: UncheckedAccount<'info>,
29
30     /// The destination of the deposit notes
31     #[account(mut)]
32     pub destination: UncheckedAccount<'info>,
33
34     pub token_program: Program<'info, Token>,
35 }
```

Listing 31: programs/margin-pool/src/instructions/margin_withdraw.rs (Lines 29,33,37)

```
26
27 /// The mint for the deposit notes
28 #[account(mut)]
29 pub deposit_note_mint: UncheckedAccount<'info>,
30
31 /// The source of the deposit notes to be redeemed
32 #[account(mut)]
33 pub source: UncheckedAccount<'info>,
34
35 /// The destination of the tokens withdrawn
36 #[account(mut)]
37 pub destination: UncheckedAccount<'info>,
```

Listing 32: programs/margin-pool/src/instructions/withdraw.rs (Lines 21,28,32)

```
19 /// The mint for the deposit notes
20 #[account(mut)]
21 pub deposit_note_mint: UncheckedAccount<'info>,
22
23 /// The address with authority to withdraw the deposit
24 pub depositor: Signer<'info>,
25
26 /// The source of the deposit notes to be redeemed
27 #[account(mut)]
28 pub source: UncheckedAccount<'info>,
29
30 /// The destination of the tokens withdrawn
31 #[account(mut)]
32 pub destination: UncheckedAccount<'info>,
```

Listing 33: programs/margin-swap/src/lib.rs (Line 113)

```
110 pub destination_margin_pool: MarginPoolInfo<'info>,
111
112 pub margin_pool_program: Program<'info, JetMarginPool>,
113 pub token_program: UncheckedAccount<'info>,
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider implementing proper account type checks.

Remediation Plan:

ACKNOWLEDGED: The Jet Protocol team acknowledged this finding.

3.12 (HAL-12) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE - INFORMATIONAL

Description:

The use of helper methods in Rust, such as `unwrap`, is allowed in dev and testing environment because those methods are supposed to throw an error (also known as `panic!`) when called on `Option::None` or a `Result` which is not `Ok`. However, keeping `unwrap` functions in production environment is considered bad practice because they may lead to program crashes, which are usually accompanied by insufficient or misleading error messages.

Code Location:

Listing 34: programs/margin-pool/src/state.rs (Line 221)

```
220 *self.total_uncollected_fees_mut() = Number::ZERO;
221 self.deposit_notes = self.deposit_notes.checked_add(fee_notes).
 ↳ unwrap();
```

Listing 35: programs/margin/src/adapter.rs (Line 120)

```
118 | AdapterResult::PriorBalanceChange(ref modified_accounts) => {
119     for modified in modified_accounts {
120         let account_info = account_infos.iter().find(|a| a.key ==
 ↳ modified).unwrap();
```

Listing 36: programs/margin/src/state.rs (Line 177)

```
176 for position in self.positions() {
177     let kind = PositionKind::from_integer(position.kind).unwrap();
178     let stale_reason = {
179         let balance_age = (clock.unix_timestamp a
```

Listing 37: programs/margin-pool/src/state.rs (Lines 82,83)

```
80 /// Record a deposit into the pool
81 pub fn deposit(&mut self, amount: &FullAmount) {
82     self.deposit_tokens = self.deposit_tokens.checked_add(amount.
↳ tokens).unwrap();
83     self.deposit_notes = self.deposit_notes.checked_add(amount.
↳ notes).unwrap();
84 }
```

Listing 38: programs/margin-pool/src/state.rs (Line 111)

```
110     .ok_or(ErrorCode::InsufficientLiquidity)?;
111 self.loan_notes = self.loan_notes.checked_add(amount.notes).unwrap
↳ ();
```

Listing 39: programs/margin-pool/src/state.rs (Line 120)

```
119 pub fn repay(&mut self, amount: &FullAmount) -> Result<()> {
120     self.deposit_tokens = self.deposit_tokens.checked_add(amount.
↳ tokens).unwrap();
```

Listing 40: programs/margin-pool/src/state.rs (Line 153)

```
151 *self.total_uncollected_fees_mut() += fee_to_collect;
152
153 self.accrued_until = self.accrued_until.checked_add(time_to_accrue
↳ ).unwrap();
```

Listing 41: programs/margin-pool/src/state.rs (Line 221)

```
217
218 let fee_notes = (uncollected / self.deposit_note_exchange_rate()).
↳ as_u64(0);
219
220 *self.total_uncollected_fees_mut() = Number::ZERO;
221 self.deposit_notes = self.deposit_notes.checked_add(fee_notes).
↳ unwrap();
222
223 fee_notes
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not to use the unwrap function in the production environment because its use causes panic! and may crash the contract without verbose error messages. Crashing the system will result in a loss of availability and, in some cases, even private information stored in the state. Some alternatives are possible, such as propagating the error with ? instead of unwrapping, or using the error-chain crate for errors.

Remediation Plan:

ACKNOWLEDGED: The Jet Protocol team acknowledged this finding.

AUTOMATED TESTING

4.1 AUTOMATED ANALYSIS

Description:

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

id	package	categories
RUSTSEC-2020-0071	time	memory-corruption
RUSTSEC-2020-0159	chrono	memory-corruption
RUSTSEC-2022-0013	regex	denial-of-service

4.2 AUTOMATED VULNERABILITY SCANNING

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was Soteria, a security analysis service for Solana programs. Soteria performed a scan on all the programs and sent the compiled results to the analyzers to locate any vulnerabilities.

Results:

Soteria scanner found one unsafe deserialization that was reported with HAL-02 vulnerability in previous chapter.

AUTOMATED TESTING

```
Analyzing /workspace/.coderrect/build/bpfel-unknown-unknown/release/all.ll ...
 - ✔ [00m:01s] Loading IR From File
 - ▮ [00m:00s] Running Compiler Optimization Passes
EntryPoints:
entrypoint
 - ✔ [00m:00s] Running Compiler Optimization Passes
 - ✔ [00m:00s] Running Pointer Analysis
============This account may be UNTRUSTFUL!===============
Found a potential vulnerability at line 97, column 18 in src/lib.rs
The account info is not trustful:

 91|        ctx: Context<CreateTransaction>,
 92|        pid: Pubkey,
 93|        accs: Vec<TransactionAccount>,
 94|        data: Vec<u8>,
 95|    ) -> Result<()> {
 96|        let owner_key = match ctx.remaining_accounts.get(0) {
>97|            Some(delegate_list_account) => {
 98|                let delegate_list =
 99|                    DelegateList::try_deserialize(&mut &delegate_list_account.data.borrow()[..])?;
 100|
 101|                if delegate_list.multisig != ctx.accounts.multisig.key() {
 102|                    msg!("delegate list isn't for this multisig");
 103|                    return Err(ErrorCode::InvalidOwner.into());
>>>Stack Trace:
>>>serum_multisig::dispatch::h2d6a953aad1f6030 [src/lib.rs:30]
>>>  serum_multisig::__private::__global::create_transaction::hf2857ebf93ed3513 [src/lib.rs:30]
>>>    serum_multisig::serum_multisig::create_transaction::hc65e6f7bb029c4d7 [src/lib.rs:30]

 - ✔ [00m:00s] Building Static Happens-Before Graph
 - ✔ [00m:00s] Detecting Vulnerabilities
detected 1 untrustful accounts in total.
detected 0 unsafe math operations in total.

--------The summary of potential vulnerabilities in all.ll--------

        1 untrustful account issues
```

AUTOMATED TESTING

```
--------The summary of potential vulnerabilities in jet_control.ll--------
        No vulnerabilities detected

--------The summary of potential vulnerabilities in jet_margin.ll--------
        No vulnerabilities detected

--------The summary of potential vulnerabilities in jet_margin_pool.ll--------
        No vulnerabilities detected

--------The summary of potential vulnerabilities in jet_margin_swap.ll--------
        No vulnerabilities detected

--------The summary of potential vulnerabilities in jet_metadata.ll--------
        No vulnerabilities detected
```

THANK YOU FOR CHOOSING

// HALBORN