HALBORN

Clone - Protocol

Solana Program Security Assessment

Prepared by: **Halborn**

Date of Engagement: **August 28th, 2023 - October 9th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	8/28/2023
0.2	Document Updates	10/02/2023
0.3	Final Draft	10/06/2023
0.4	Draft Review	10/09/2023
0.5	Draft Review	10/09/2023
1.0	Remediation Plan	10/17/2023
1.1	Remediation Plan Updates	10/23/2023
1.2	Remediation Plan Updates	10/26/2023
1.3	Remediation Plan Review	10/26/2023
1.4	Remediation Plan Review	10/27/2023

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

The Comet Liquidity System by Clone simplifies markets liquidity provision by eliminating the requirement of owning the cloned asset, known as clAssets, (which are referred to as onAsset in the code on which the assessment was made). These clAssets faithfully replicate the price dynamics of real-world assets, offering exposure to diverse markets. The Clone Protocol extends accessibility by allowing users to participate in liquidity provision without the need to possess the underlying asset. Instead, users can deposit collateral and select their preferred commitment level for providing liquidity to specific trading pools.
The collateral supplied by liquidity providers serves to over-collateralize potential impermanent loss, ensuring the protocol's robustness. To maintain adequate collateralization, the protocol incorporates a liquidation system designed to mitigate and eliminate risky positions.

Halborn conducted a security assessment on their Solana program, beginning on August 28th, 2023 and ending on October 9th, 2023 . The security assessment was scoped to the updates to the develop branch of the clone-protocol GitHub repository. Commit hashes and further details can be found in the **Scope** section of this report.

1.2 ASSESSMENT SUMMARY

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

The purpose of this assessment is to:

- Ensure that Solana program functions operate as intended

- Identify potential security issues with the Solana program

In summary, Halborn identified some security risks that were mostly addressed by the Clone team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, 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 assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors
- 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 runtime testing (solana-test-framework)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Clone

- Repository: clone-protocol
- Commit ID:ba823ee6a7acf5bd3f6c3806f756ca467b723804
- Programs in scope:
    1. clone (programs/clone/src)
    2. clone-staking (programs/clone-staking/src)

Out-of-scope:
- third-party libraries and dependencies
- financial-related attacks

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	8

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISCALCULATION DURING LIQUIDITY WITHDRAWAL OF THE COMET	Low (3.1)	SOLVED - 10/18/2023
(HAL-02) LIQUIDATING A BORROW POSITION WITH PARTIAL DEBT SETTLEMENT	Low (3.1)	RISK ACCEPTED
(HAL-03) CHECKED ARITHMETIC MISSING	Informational (1.9)	SOLVED - 09/13/2023
(HAL-04) ORACLE CHECKS MISSING	Informational (0.6)	ACKNOWLEDGED
(HAL-05) ADMINISTRATOR ADDRESS CANNOT BE TRANSFERRED	Informational (0.5)	ACKNOWLEDGED
(HAL-06) OVERCOLLATERAL RATIOS CHECK MISSING	Informational (0.5)	SOLVED - 10/16/2023
(HAL-07) POTENCIAL DUPLICATION OF AUTHORITIES	Informational (0.0)	SOLVED - 09/14/2023
(HAL-08) ZERO AMOUNT CHECK MISSING	Informational (0.0)	SOLVED - 09/14/2023
(HAL-09) MISSING CARGO OVERFLOW CHECKS	Informational (0.0)	SOLVED - 09/12/2023
(HAL-10) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational (0.0)	SOLVED - 10/17/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) MISCALCULATION DURING LIQUIDITY WITHDRAWAL OF THE COMET - LOW (3.1)

Description:

The WithdrawLiquidityFromComet instruction allows the user to withdraw the corresponding liquidity from his position, in the instruction handler various calculations are performed to update both the pool and the user account values. However, during these calculations, the clone decimal is used to apply the value of the collateral to be withdrawn instead of the corresponding one. This results in updating the collateral_ild and onasset_ild pool fields with incorrect values, as well as the user's collateral_ild_rebate and onasset_ild_rebate fields. This leaves the pool with inconsistent values and also directly affects the execution of CollectLpRewards instructions, as well as liquidation instructions and even PayImpermanentLoss since they make use of these values.

Code Location:

Listing 1: programs/clone/src/withdraw_liquidity_from_comet.rs (Line 72)

```
66 let collateral_value_to_withdraw =
67     collateral_amount.min(comet_position.
↳ committed_collateral_liquidity);
68
69 let proportional_value = to_clone_decimal!(
↳ collateral_value_to_withdraw)
70     / collateral.to_collateral_decimal(pool.
↳ committed_collateral_liquidity)?;
71
72 let proportional_value = collateral.to_collateral_decimal(
↳ collateral_value_to_withdraw)?
73 / collateral.to_collateral_decimal(pool.
↳ committed_collateral_liquidity)?;
74
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:L/R:N/S:U (3.1)

Recommendation:

To address the issue, it is essential to ensure that the correct decimal is used during the calculation, replacing the clone decimal with the collateral decimal in `withdraw_liquidity_from_comet` instruction handler. This adjustment will ensure that accurate values are obtained, preventing discrepancies in related parameters within the pool and user accounts.

Remediation Plan:

SOLVED: The Clone team solved this finding in commit `c97115f`: the proper modification was applied within the instruction handler replacing the use of the clone decimal with the collateral decimal in the calculation process. By doing so, they ensured that the calculation mechanism now operates seamlessly, yielding the precise and expected results. This refined calculation process has far-reaching implications as it rectifies discrepancies not only for the immediate concern but across various instructions impacted by the original issue.

4.2 (HAL-02) LIQUIDATING A BORROW POSITION WITH PARTIAL DEBT SETTLEMENT - LOW (3.1)

Description:

It was observed that if, following the initialization of a user's loan position, the price of the clAsset (onAsset in the code) undergoes a significant fluctuation, and the position becomes undercollateralized, a liquidator could trigger the position's liquidation through LiquidateBorrowPosition. However, in scenarios where min_collateral_ratio value is not checked to ensure enough high and the price divergence is exceptionally wide, the liquidation could proceed with a notably low value.

Listing 2

```
collateral_value < (1 + reward_pct) * onAsset_value
```

This situation allows the liquidator to potentially realize a profit from the entire collateral of the position, even without fully liquidating the outstanding debt. Consequently, there may be limited incentive for a liquidator to complete the liquidation, as doing so could result in an unprofitable outcome, essentially incurring a loss. In such cases, if the user does not take this action calling PayBorrowDebt, it falls upon the Insurance Fund to intervene.

Code Location:

Listing 3: programs/clone/src/liquidate_borrow_position.rs

```
pub fn execute(
    ctx: Context<LiquidateBorrowPosition>,
    user: Pubkey,
    borrow_index: u8,
    amount: u64,
```

```
74 ) -> Result<()> {
75     return_error_if_false!(amount > 0, CloneError::
↳ InvalidTokenAmount);
76     let seeds = &[&[
77         CLONE_PROGRAM_SEED.as_ref(),
78         bytemuck::bytes_of(&ctx.accounts.clone.bump),
79     ][..]];
80 
81     let collateral = &ctx.accounts.clone.collateral;
82     let pools = &mut ctx.accounts.pools;
83     let oracles = &ctx.accounts.oracles;
84 
85     let borrows = &mut ctx.accounts.user_account.borrows;
86     let borrow_position = borrows[borrow_index as usize];
87     let pool_index = borrow_position.pool_index as usize;
88     let pool = &pools.pools[pool_index];
89     return_error_if_false!(
90         pool.status != Status::Frozen,
91         CloneError::StatusPreventsAction
92     );
93 
94     let pool_oracle = &oracles.oracles[pool.asset_info.
↳ oracle_info_index as usize];
95     let collateral_oracle = &oracles.oracles[collateral.
↳ oracle_info_index as usize];
96 
97     let min_overcollateral_ratio = to_ratio_decimal!(pool.
↳ asset_info.min_overcollateral_ratio);
98     let collateralization_ratio = to_ratio_decimal!(collateral.
↳ collateralization_ratio);
99 
100     let burn_amount = amount.min(borrow_position.borrowed_onasset)
↳ ;
101     let collateral_position_amount =
102         collateral.to_collateral_decimal(borrow_position.
↳ collateral_amount)?;
103 
104     // This call checks that the oracles are updated
105     let is_undercollateralized = check_mint_collateral_sufficient(
106         pool_oracle,
107         collateral_oracle,
108         to_clone_decimal!(borrow_position.borrowed_onasset),
109         min_overcollateral_ratio,
110         collateralization_ratio,
```

```
111             collateral_position_amount,
112         )
113         .is_err();
114         let is_in_liquidation_mode = pool.status == Status::
 ↳ Liquidation;
115
116         return_error_if_false!(
117             is_undercollateralized || is_in_liquidation_mode,
118             CloneError::BorrowPositionUnableToLiquidate
119         );
120
121         let borrow_liquidation_fee_rate = to_bps_decimal!(ctx.accounts
 ↳ .clone.borrow_liquidator_fee_bps);
122         let pool_price = pool_oracle.get_price() / collateral_oracle.
 ↳ get_price();
123
124         let collateral_reward = rescale_toward_zero(
125             (Decimal::one() + borrow_liquidation_fee_rate)
126                 * to_clone_decimal!(burn_amount)
127                 * pool_price,
128             collateral.scale.try_into().unwrap(),
129         )
130         .min(collateral_position_amount);
131
132         // Burn the onAsset from the liquidator
133         let cpi_accounts = Burn {
134             mint: ctx.accounts.onasset_mint.to_account_info().clone(),
135             from: ctx
136                 .accounts
137                 .liquidator_onasset_token_account
138                 .to_account_info()
139                 .clone(),
140             authority: ctx.accounts.liquidator.to_account_info().clone
 ↳ (),
141         };
142         let burn_liquidator_onasset_context = CpiContext::new(
143             ctx.accounts.token_program.to_account_info().clone(),
144             cpi_accounts,
145         );
146
147         token::burn(burn_liquidator_onasset_context, burn_amount)?;
148
149         // Send the user the collateral reward
150         let cpi_accounts = Transfer {
```

```
151         from: ctx.accounts.vault.to_account_info().clone(),
152         to: ctx
153             .accounts
154             .liquidator_collateral_token_account
155             .to_account_info()
156             .clone(),
157         authority: ctx.accounts.clone.to_account_info().clone(),
158     };
159     let send_collateral_context = CpiContext::new_with_signer(
160         ctx.accounts.token_program.to_account_info().clone(),
161         cpi_accounts,
162         seeds,
163     );
164     token::transfer(
165         send_collateral_context,
166         collateral_reward.mantissa().try_into().unwrap(),
167     )?;
168 
169     // Update data
170     borrows[borrow_index as usize].borrowed_onasset -= burn_amount
 ↳ ;
171     //TODO:en que momento se añaden los collateral rewards al
 ↳ collateral amount?
172     borrows[borrow_index as usize].collateral_amount -=
 ↳ collateral_reward.mantissa() as u64;
173 
174     // Remove position if empty
175     if borrows[borrow_index as usize].is_empty() {
176         borrows.remove(borrow_index as usize);
177     } else {
178         let borrowed_onasset = to_clone_decimal!(borrows[
 ↳ borrow_index as usize].borrowed_onasset);
179         let collateral_amount = Decimal::new(
180             borrows[borrow_index as usize]
181                 .collateral_amount
182                 .try_into()
183                 .unwrap(),
184             collateral.scale.try_into().unwrap(),
185         );
186         let max_liquidation_overcollateral_ratio =
187             to_ratio_decimal!(pool.asset_info.
 ↳ max_liquidation_overcollateral_ratio);
188         let c_ratio = collateral_amount * collateralization_ratio
 ↳ / (pool_price * borrowed_onasset);
```

```
189         return_error_if_false!(
190             c_ratio <= max_liquidation_overcollateral_ratio,
191             CloneError::InvalidMintCollateralRatio
192         );
193     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:M/R:P/S:U (3.1)

Recommendation:

To address this observed issue, it is recommended to establish a sufficiently high `min_collateral_ratio` value that ensures the collateral's worth remains significantly above the total outstanding debt, even in scenarios of considerable price divergence.

Remediation Plan:

RISK ACCEPTED: The Clone team accepted this risk, since in cases where a borrow position or a comet fails to undergo liquidation before reaching an undercollateralized state, or if the remaining liquidation reward proves insufficient to incentivize a potential liquidator, Clone DAO remains committed to executing the liquidation, even at a financial loss to our treasury. This commitment is made to guarantee that even undercollateralized positions are resolved without posing any risk to the protocol.

4.3 (HAL-03) CHECKED ARITHMETIC MISSING - INFORMATIONAL (1.9)

Description:

Multiple instances of unsafe arithmetic operations, including multiplication, division, and addition, were detected in various files and functions.

Checked arithmetic operations provide error-checking mechanisms to detect and handle arithmetic overflow or underflow conditions. By using checked arithmetic operations, you can ensure that mathematical calculations do not result in unexpected values or errors due to overflow or underflow, which is crucial for maintaining the correctness and reliability of the program.

Code Location:

- maths.rs
- states.rs
- instructions/swap.rs
- instructions/liquidate_comet_collateral_ild.rs
- instructions/liquidate_comet_onasset_ild.rs
- instructions/liquidate_borrow_position.rs:125
- instructions/withdraw_liquidity_from_comet.rs
- instructions/withdraw_collateral_from_comet.rs
- instructions/pay_impermanent_loss_debt.rs
- instructions/liquidate_borrow_position.rs
- instructions/collect_lp_rewards.rs
- instructions/pay_borrow_debt.rs
- instructions/add_collateral_to_borrow.rs
- instructions/add_collateral_to_comet.rs
- instructions/add_liquidity_to_comet.rs

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:L/Y:L/R:P/S:U (1.9)

Recommendation:

To enhance the code's robustness and safety, it is advisable to replace standard arithmetic operators with checked arithmetic operations. This substitution allows for the proactive detection and management of exceptional cases that may emerge during mathematical computations.

Remediation Plan:

SOLVED: The Clone team solved this finding in commit 9ba9c47: all standard arithmetic operators have been replaced with checked arithmetic operations, following recommended best practices.

4.4 (HAL-04) ORACLE CHECKS MISSING - INFORMATIONAL (0.6)

Description:

The update_oracles instruction handler relies on Oracle accounts, but it lacks a validation mechanism. While the administrators are responsible for providing these accounts, there is a possibility of inadvertently using incorrect oracle accounts, potentially exposing the Clone program to malicious price feeders.

Furthermore, it is worth emphasizing that the ownership of the account is not verified. Therefore, when the source value of the oracle is provided, there is a lack of validation to guarantee its correspondence with the specified account. This oversight could potentially result in issues during subsequent price updates.

Code Location:

Listing 4: programs/clone/src/update_oracles.rs (Lines 52,54,55,56)

```
42 pub fn execute(ctx: Context<UpdateOracles>, params:
↳ UpdateOracleParameters) -> Result<()> {
43     let clone = &ctx.accounts.clone;
44     let clone_auth = clone.auth;
45     let auth_key = ctx.accounts.auth.key();
46     let is_admin = auth_key.eq(&clone.admin);
47     let is_auth = clone_auth.iter().any(|auth| auth_key.eq(auth));
48
49     let oracles = &mut ctx.accounts.oracles.oracles;
50
51     match params {
52         UpdateOracleParameters::Add { address, source } => {
53             return_error_if_false!(is_admin, CloneError::
↳ Unauthorized);
54             oracles.push(OracleInfo {
55                 source,
56                 address,
57                 status: Status::Active,
```

```
58                 ..OracleInfo::default()
59             })
60         }
61         UpdateOracleParameters::Remove { index } => {
62             return_error_if_false!(is_admin, CloneError::
↳ Unauthorized);
63             oracles.remove(index.into());
64         }
65         UpdateOracleParameters::Modify {
66             index,
67             address,
68             source,
69             status,
70         } => {
71             let oracle = &mut oracles[index as usize];
72             if let Some(addr) = address {
73                 return_error_if_false!(is_admin, CloneError::
↳ Unauthorized);
74                 oracle.address = addr;
75             }
76             if let Some(src) = source {
77                 return_error_if_false!(is_admin, CloneError::
↳ Unauthorized);
78                 oracle.source = src;
79             }
80             if let Some(sts) = status {
81                 return_error_if_false!(
82                     is_admin || (is_auth && sts == Status::Frozen)
↳ ,
83                     CloneError::Unauthorized
84                 );
85                 oracle.status = sts;
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:H/D:H/Y:H/R:F/S:U (0.6)

Recommendation:

To address this security finding and enhance the robustness of the program, it is recommended to implement a validation mechanism for the Oracle

accounts used in the update_oracles instruction handler. This validation process can help prevent the inadvertent use of incorrect or unauthorized oracle accounts, which could otherwise expose the clone program to potential vulnerabilities arising from malicious price feeders.

Remediation Plan:

ACKNOWLEDGED: The Clone team acknowledged this issue since the oracle update functionality is restricted to the administrator's exclusive access, wherein only the administrator is endowed with the privilege to call the instruction for updating the oracles.

4.5 (HAL-05) ADMINISTRATOR ADDRESS CANNOT BE TRANSFERRED - INFORMATIONAL (0.5)

Description:

The program currently lacks the capability to designate a new **admin** as a privileged address. In the event that the development team needs to change the administrator address for operational reasons or suspects a security breach, the program could face a substantial loss of functionality and would be necessary to redeploy.

Code Location:

Listing 5: programs/clone/src/pay_impermanent_loss.rs

```
83 pub struct InitializeClone<'info> {
84     #[account(mut)]
85     pub admin: Signer<'info>,
86     #[account(
87         init,
88         seeds = [CLONE_PROGRAM_SEED.as_ref()],
89         space = 8 + 472,
90         bump,
91         payer = admin
92     )]
93     pub clone: Box<Account<'info, Clone>>,
94     pub collateral_mint: Account<'info, Mint>,
95     #[account(
96         token::mint = collateral_mint,
97         token::authority = clone,
98     )]
99     pub collateral_vault: Account<'info, TokenAccount>,
100    pub rent: Sysvar<'info, Rent>,
101    pub token_program: Program<'info, Token>,
102    pub system_program: Program<'info, System>,
103 }
104
105 pub fn execute(
```

```
106     ctx: Context<InitializeClone>,
107     comet_collateral_ild_liquidator_fee_bps: u16,
108     comet_onasset_ild_liquidator_fee_bps: u16,
109     borrow_liquidator_fee_bps: u16,
110     treasury_address: Pubkey,
111     collateral_oracle_index: u8,
112     collateralization_ratio: u8,
113 ) -> Result<()> {
114     return_error_if_false!(
115         comet_onasset_ild_liquidator_fee_bps < 10000 &&
 ↳ borrow_liquidator_fee_bps < 10000,
116         CloneError::InvalidValueRange
117     );
118
119     // set manager data
120     ctx.accounts.clone.admin = *ctx.accounts.admin.to_account_info
 ↳ ().key;
121     ctx.accounts.clone.bump = *ctx.bumps.get("clone").unwrap();
122     ctx.accounts.clone.treasury_address = treasury_address;
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:P/S:U (0.5)

Recommendation:

It is advisable to implement a two-step ownership transfer process. In the first step, the current owner nominates an account, and in the second step, the nominated account must confirm and accept the ownership transfer for it to be completed successfully.

Remediation Plan:

ACKNOWLEDGED: The Clone team acknowledged this issue since the Squads' functionality empowers them with the ability to efficiently modify cryptographic keys, adjust signer permissions, introduce new signers, and enact other related operational changes.

4.6 (HAL-06) OVERCOLLATERAL RATIOS CHECK MISSING - INFORMATIONAL (0.5)

Description:

The AddPool and UpdatePoolParameters instructions are accessible to the administrator, who is required to furnish a set of parameters, including max_liquidation_overcollateral_ratio and min_overcollateral_ratio. These parameters dictate the maximum collateral liquidation rate and the minimum collateral rate necessary for borrowing any asset within the system.

However, it is noteworthy that the instruction handler currently lacks a validation mechanism to ensure the integrity of these values. Consequently, there is a possibility that both parameters could be set to 0, or that the max_liquidation_overcollateral_ratio may inadvertently possess a lower value than the min_overcollateral_ratio. Such scenarios could potentially give rise to liquidity inconsistencies and borrowing position settlement issues within the system.

Code Location:

Listing 6: programs/clone/src/add_pool.rs (Lines 9,10)

```
 7 #[instruction(
 8     min_overcollateral_ratio: u16,
 9     max_liquidation_overcollateral_ratio: u16,
10     liquidity_trading_fee_bps: u16,
11     treasury_trading_fee_bps: u16,
12     il_health_score_coefficient: u16,
13     position_health_score_coefficient: u16,
14     oracle_info_index: u8,
15 )]
```

Listing 7: programs/clone/src/add_pool.rs (Lines 67,68)

```
52 pub fn execute(
53     ctx: Context<AddPool>,
54     min_overcollateral_ratio: u16,
55     max_liquidation_overcollateral_ratio: u16,
56     liquidity_trading_fee_bps: u16,
57     treasury_trading_fee_bps: u16,
58     il_health_score_coefficient: u16,
59     position_health_score_coefficient: u16,
60     oracle_info_index: u8,
61 ) -> Result<()> {
62     let asset_info = AssetInfo {
63         onasset_mint: ctx.accounts.onasset_mint.to_account_info().
↳ key(),
64         oracle_info_index,
65         il_health_score_coefficient,
66         position_health_score_coefficient,
67         min_overcollateral_ratio,
68         max_liquidation_overcollateral_ratio,
69         ..AssetInfo::default()
70     };
```

Listing 8: programs/clone/src/update_pool_parameters.rs (Lines 11,12)

```
 6 pub enum PoolParameters {
 7     Status { value: Status },
 8     TreasuryTradingFee { value: u16 },
 9     LiquidityTradingFee { value: u16 },
10     OracleInfoIndex { value: u8 },
11     MinOvercollateralRatio { value: u16 },
12     MaxLiquidationOvercollateralRatio { value: u16 },
13     IlHealthScoreCoefficient { value: u16 },
14     PositionHealthScoreCoefficient { value: u16 },
15 }
```

Listing 9: programs/clone/src/update_pool_parameters.rs (Lines 84,87)

```
 6 PoolParameters::MinOvercollateralRatio { value } => {
 7     pool.asset_info.min_overcollateral_ratio = value;
 8 }
 9 PoolParameters::MaxLiquidationOvercollateralRatio { value } => {
10     pool.asset_info.max_liquidation_overcollateral_ratio = value;
```

```
11 }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:M/D:H/Y:H/R:F/S:U (0.5)

Recommendation:

To enhance the security and reliability of the these instructions within the system, it is advised to implement robust input validation checks for the parameters `max_liquidation_overcollateral_ratio` and `min_overcollateral_ratio`. These validation checks should ensure that the provided values adhere to predefined constraints and do not fall into potentially problematic ranges.

Remediation Plan:

SOLVED: The Clone team solved this finding in commit `ca63cd1`: The `is_valid_overcollateral_ratios` function has been introduced which serves the crucial role of validating the `min_overcollateral_ratio` and `max_liquidation_overcollateral_ratio` values. Specifically, it verifies that `min_overcollateral_ratio` is greater than 100 and that `max_liquidation_overcollateral_ratio` exceeds the former. This function is invoked in both the `AddPool` and `UpdatePoolParameters` instructions to ensure that the specified parameter values align with the expected criteria.

4.7 (HAL-07) POTENCIAL DUPLICATION OF AUTHORITIES - INFORMATIONAL (0.0)

Description:

The Clone account allows administrators to append authorities using the UpdateCloneParameters call. However, it lacks a check for duplicate addresses in this array, enabling the addition of the same address multiple times. If a duplicate authority is accidentally added, it requires multiple calls to the instruction to remove each occurrence. Only the first matching address provided is removed during the elimination process.

Code Location:

Listing 10: programs/clone/src/update_clone_parameter.rs

```
33 pub fn execute(ctx: Context<UpdateCloneParameters>, params:
↳ CloneParameters) -> Result<()> {
34     let clone = &mut ctx.accounts.clone;
35     match params {
36         CloneParameters::AddAuth { address } => {
37             let auth_array = clone.auth;
38             let empty_slot = auth_array
39                 .iter()
40                 .enumerate()
41                 .find(|(_, slot)| (**slot).eq(&Pubkey::default()))
↳ ;
42             return_error_if_false!(empty_slot.is_some(),
↳ CloneError::AuthArrayFull);
43             clone.auth[empty_slot.unwrap().0] = address;
44         }
```

Listing 11: programs/clone/src/update_clone_parameter.rs

```
54  CloneParameters::RemoveAuth { address } => {
55             let auth_array = clone.auth;
56             let auth_slot = auth_array
57                 .iter()
58                 .enumerate()
```

```
59                  .find(|(_, slot)| (**slot).eq(&address));
60
61              return_error_if_false!(auth_slot.is_some(), CloneError
↳ ::AuthNotFound);
62              clone.auth[auth_slot.unwrap().0] = Pubkey::default();
63          }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.0)

Recommendation:

To enhance the security and usability of the Clone account, it is recommended to implement a check for duplicate addresses before appending authorities. This check should ensure that the same address cannot be added multiple times. This modification will not only prevent the accidental addition of duplicate authorities but also simplify the process of removing them if necessary.

Remediation Plan:

SOLVED: The Clone team solved this finding in commit ebd6c6c: a validation check has been implemented to ensure that the address intended for addition as an authority is not already present in the authority vector, thus preventing duplication.

4.8 (HAL-08) ZERO AMOUNT CHECK MISSING - INFORMATIONAL (0.0)

Description:

While certain instructions like AddCollateralToBorrow, AddCollateralToComet , BorrowMore and WithdrawCollateralFromComet enable users to add collateral to reinforce their positions, borrow additional clAssets and withdraw collateral from comet position, they lack validation checks on the provided amount. Consequently, users can execute these actions with a zero amount.

Additionally, if these transactions trigger unnecessary computation or storage operations, it could degrade the overall performance of the program.

Code Location:

- add_collateral_to_borrow.rs
- borrow_more.rs
- add_collateral_to_comet.rs
- withdraw_collateral_from_comet.rs

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.0)

Recommendation:

It is advisable to implement proper validation checks to ensure that provided amounts are valid and non-zero before allowing users to execute these instructions. This will enhance both the security and efficiency of the program.

Remediation Plan:

SOLVED: The Clone team solved this finding in commit a09d8b3: checks have been integrated into all outstanding instructions to ensure that quantities greater than zero are verified, preventing any execution without proper action.

4.9 (HAL-09) MISSING CARGO OVERFLOW CHECKS - INFORMATIONAL (0.0)

Description:

It has been observed that the changes applied to the assessment's scope do not include the implementation of the overflow-checks=true flag in the Cargo.toml files.

By default, overflow checks are disabled in optimized release builds. Consequently, any overflows occurring in release builds will remain silent, potentially causing unexpected application behavior. To ensure proper overflow handling, it is advisable to include the overflow-checks=true setting in the Cargo.toml file, even when using checked arithmetic through functions like checked_* or saturating_*.

Code Location:

- programs/clone/Cargo.toml
- programs/clone-staking/Cargo.toml

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.0)

Recommendation:

Add overflow-checks=true under the release profile in the Cargo.toml files specified for recommended behavior.

Remediation Plan:

SOLVED: The Clone team solved this finding in commit c7c2767: The Cargo.toml file has been updated to include the implementation of the overflow-checks=true flag, following the recommended best practices.

4.10 (HAL-10) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE - INFORMATIONAL (0.0)

Description:

In Rust, helper methods like unwrap are commonly used during development and testing phases. They're designed to trigger an error (a panic!) when applied to Option::None or an unsuccessful Result. However, using unwrap in a production environment is discouraged. It can potentially cause program crashes with inadequate or misleading error messages, making it a less reliable practice for production code.

Code Location:

Listing 12

```
./instructions/update_clone_parameters.rs:43:            clone.
↳ auth[empty_slot.unwrap().0] = address;
./instructions/update_clone_parameters.rs:62:            clone.
↳ auth[auth_slot.unwrap().0] = Pubkey::default();
./instructions/update_prices.rs:44:                        (info.
↳ price, (-info.expo).try_into().unwrap())
./instructions/update_prices.rs:46:                        (info.
↳ price * 10_i64.pow(info.expo.try_into().unwrap()), 0)
./instructions/update_prices.rs:57:                    result.
↳ mantissa.try_into().unwrap(),
./instructions/update_prices.rs:58:                    result.
↳ scale.try_into().unwrap(),
./instructions/initialize_clone.rs:54:    ctx.accounts.clone.bump
↳ = *ctx.bumps.get("clone").unwrap();
./instructions/withdraw_collateral_from_borrow.rs:115:
↳ pool_index: pool_index.try_into().unwrap(),
./instructions/swap.rs:93:        seeds::program =
↳ clone_staking_program.clone().unwrap().key(),
./instructions/swap.rs:99:        seeds::program =
↳ clone_staking_program.clone().unwrap().key(),
./instructions/swap.rs:131:        let user_staking_account = ctx.
↳ accounts.user_staking_account.as_ref().unwrap();
```

```
./instructions/swap.rs:132:        let clone_staking = ctx.
↳ accounts.clone_staking.as_ref().unwrap();
./instructions/swap.rs:177:        .unwrap();
./instructions/swap.rs:178:    let result_amount: u64 =
↳ swap_summary.result.mantissa().try_into().unwrap();
./instructions/swap.rs:240:            (mint_amount +
↳ treasury_fees).try_into().unwrap(),
./instructions/swap.rs:241:            -(TryInto::<i64>::try_into(
↳ transfer_amount).unwrap()),
./instructions/swap.rs:298:            -(TryInto::<i64>::try_into(
↳ burn_amount).unwrap()),
./instructions/swap.rs:299:            (transfer_amount +
↳ treasury_fees).try_into().unwrap(),
./instructions/swap.rs:331:            .unwrap(),
./instructions/swap.rs:347:        pool_price: pool_price.mantissa
↳ ().try_into().unwrap(),
./instructions/collect_lp_rewards.rs:91:
↳ collateral_reward.try_into().unwrap(),
./instructions/collect_lp_rewards.rs:117:
↳ onasset_reward.try_into().unwrap(),
./instructions/withdraw_liquidity_from_comet.rs:74:
↳ collateral.scale.try_into().unwrap(),
./instructions/withdraw_liquidity_from_comet.rs:96:
↳ pool_index: pool_index.try_into().unwrap(),
./instructions/withdraw_liquidity_from_comet.rs:112:
↳ pool_index: pool_index.try_into().unwrap(),
./instructions/withdraw_liquidity_from_comet.rs:116:
↳ pool_price: pool_price.mantissa().try_into().unwrap(),
./instructions/add_liquidity_to_comet.rs:68:        collateral.
↳ scale.try_into().unwrap(),
./instructions/add_liquidity_to_comet.rs:70:    let
↳ collateral_ild_delta: i64 = collateral_ild.mantissa().try_into().
↳ unwrap();
./instructions/add_liquidity_to_comet.rs:75:    let
↳ onasset_ild_delta: i64 = onasset_ild.mantissa().try_into().unwrap
↳ ();
./instructions/add_liquidity_to_comet.rs:109:
↳ committed_collateral_delta: collateral_amount.try_into().unwrap(),
./instructions/add_liquidity_to_comet.rs:128:        pool_price:
↳ pool_price.mantissa().try_into().unwrap(),
./instructions/initialize_borrow_position.rs:140:
↳ pool_index: pool_index.try_into().unwrap(),
./instructions/initialize_borrow_position.rs:150:
↳ collateral_delta: collateral_amount.try_into().unwrap(),
```

```
34 ./instructions/initialize_borrow_position.rs:152:
↳ borrowed_delta: onasset_amount.try_into().unwrap()
35 ./instructions/add_collateral_to_borrow.rs:63:            .unwrap
↳ (),
36 ./instructions/add_collateral_to_borrow.rs:66:
↳ collateral_delta: amount.try_into().unwrap(),
37 ./instructions/pay_borrow_debt.rs:80:            .unwrap(),
38 ./instructions/wrap_asset.rs:59:        Decimal::new(amount.
↳ try_into().unwrap(), underlying_mint_scale),
39 ./instructions/liquidate_comet_collateral_ild.rs:100:        .
↳ unwrap();
40 ./instructions/liquidate_comet_collateral_ild.rs:106:            .
↳ unwrap();
41 ./instructions/borrow_more.rs:108:        pool_index: pool_index.
↳ try_into().unwrap(),
42 ./instructions/borrow_more.rs:113:        borrowed_delta: amount.
↳ try_into().unwrap()
43 ./instructions/liquidate_comet_onasset_ild.rs:118:        let
↳ ild_rebate_increase: i64 = burn_amount.mantissa().try_into().
↳ unwrap();
44 ./instructions/liquidate_comet_onasset_ild.rs:132:
↳ ild_rebate_increase.try_into().unwrap(),
45 ./instructions/liquidate_comet_onasset_ild.rs:148:
↳ collateral_reward.mantissa().try_into().unwrap(),
46 ./instructions/liquidate_borrow_position.rs:128:        collateral
↳ .scale.try_into().unwrap(),
47 ./instructions/liquidate_borrow_position.rs:166:
↳ collateral_reward.mantissa().try_into().unwrap(),
48 ./instructions/liquidate_borrow_position.rs:182:                .
↳ unwrap(),
49 ./instructions/liquidate_borrow_position.rs:183:
↳ collateral.scale.try_into().unwrap(),
50 ./instructions/liquidate_borrow_position.rs:197:        pool_index
↳ : pool_index.try_into().unwrap(),
51 ./instructions/pay_impermanent_loss_debt.rs:90:            let
↳ ild_share: u64 = ild_share.onasset_ild_share.mantissa().try_into()
↳ .unwrap();
52 ./instructions/pay_impermanent_loss_debt.rs:119:                .
↳ unwrap();
53 ./instructions/pay_impermanent_loss_debt.rs:149:                .
↳ unwrap();
54 ./math.rs:97:        collateral.scale.try_into().unwrap(),
55 ./math.rs:178:         .unwrap();
56 ./math.rs:181:         .unwrap();
```

```
57 ./math.rs:192:             .unwrap()
58 ./math.rs:194:         collateral.scale.try_into().unwrap(),
59 ./math.rs:205:                 .unwrap(),
60 ./math.rs:206:         collateral.scale.try_into().unwrap(),
61 ./states.rs:121:             .unwrap();
62 ./states.rs:126:             .unwrap();
63 ./states.rs:231:             Ok(Decimal::new(num, self.scale.
↳ try_into().unwrap()))
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.0)

Recommendation:

In a production environment, it is best to avoid using the unwrap function. This function can potentially trigger a panic!, which may crash not only the affected module or program but also the entire runtime. Unfortunately, such crashes often lack detailed error messages, making debugging challenging. System crashes can lead to both availability loss and, in some cases, compromise private information within the system. To mitigate these risks, consider alternative approaches. You can propagate errors using the ? operator instead of unwrapping, or use the error-chain crate for effective error management.

Remediation Plan:

SOLVED: The Clone team solved this finding in commit ec94adf: The unwrap function has been entirely removed from all files in compliance with the recommended guidelines, aimed at enhancing robustness and security.

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the program for.

5.1 ORACLES

Description:

The UpdatePrice instruction is accessible to any user, serving the purpose of fetching updated prices from the oracles utilized within the program. To execute this instruction successfully, it is imperative to provide the accounts of the oracles intended for updating. These oracle accounts must already exist and correspond to those previously added via the UpdateOracles instruction by the administrator.

It is crucial to highlight that many aspects of the program's functionality rely on updating oracle data. Therefore, keeping the oracles updated is essential for seamless program operation.

Testing has been conducted to validate the correctness of this functionality and to ensure that it poses no security risks. These tests encompass a wide range of scenarios and conditions to mitigate potential vulnerabilities and guarantee the reliability of the price update process.

Results:

No vulnerabilities were identified.

```
[+] InitializeUser Instruction
[2023-10-02T13:56:53.147842000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-02T13:56:53.148252000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: InitializeUser
[2023-10-02T13:56:53.149754000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2023-10-02T13:56:53.149833000Z TRACE solana_runtime::system_instruction_processor] process_instruction: CreateAccount { lamports: 72161280, space: 10240, owner: F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v }
[2023-10-02T13:56:53.150000000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2023-10-02T13:56:53.152737000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 10124 of 200000 compute units
[2023-10-02T13:56:53.153001000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v success
user created: User { borrows: [], comet: Comet { collateral_amount: 0, positions: [] } }
[+] SetPrice usdc pyth oracle Instruction
[2023-10-02T13:56:53.199793000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW invoke [1]
[2023-10-02T13:56:53.200092000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: SetPrice
[2023-10-02T13:56:53.200834000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW consumed 1292 of 200000 compute units
[2023-10-02T13:56:53.200935000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW success
[+] SetPrice pool pyth oracle Instruction
[2023-10-02T13:56:53.216795000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW invoke [1]
[2023-10-02T13:56:53.217090000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: SetPrice
[2023-10-02T13:56:53.217457000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW consumed 1292 of 200000 compute units
[2023-10-02T13:56:53.217540000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW success
[+] UpdatePrices Instruction
[2023-10-02T13:56:53.229086000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-02T13:56:53.229434000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: UpdatePrices
[2023-10-02T13:56:53.230724000Z DEBUG solana_runtime::message_processor::stable_log] Program log: PRICE: 2 7
[2023-10-02T13:56:53.234895000Z DEBUG solana_runtime::message_processor::stable_log] Program log: UPDATED ORACLE: OracleInfo { source: PYTH, address: 1111111QLbz7JHiBTspS962RLKV8GndWFwiEaqKM, price: 2, expo: 7, status: Active, last_update_slot:
15625 }
[2023-10-02T13:56:53.235517000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 16691 of 200000 compute units
[2023-10-02T13:56:53.235623000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v success
Oracles with  updated prices: Oracles { oracles: [OracleInfo { source: PYTH, address: 1111111QLbz7JHiBTspS962RLKV8GndWFwiEaqKM, price: 2, expo: 7, status: Active, last_update_slot: 15625 }, OracleInfo { source: PYTH, address: 1111111ogCyDbaRMvkd
sHB3qfdyFYaG1WtRUAfdh, price: 0, expo: 0, status: Active, last_update_slot: 0 }] }
[+] InitializeBorrowPosition  Instruction
User Collateral Token Account amount:  ————> 100000000
Vault Collateral  Account amount:  ————> 100000000
User OnAsset Token Account amount:  ————> 100000000
[2023-10-02T13:56:53.248952000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-02T13:56:53.249808000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: InitializeBorrowPosition
[2023-10-02T13:56:53.257521000Z DEBUG solana_runtime::message_processor::stable_log] Program log: AnchorError thrown in programs/clone/src/math.rs:11. Error Code: OutdatedOracle. Error Number: 6007. Error Message: Outdated Oracle.
[2023-10-02T13:56:53.257867000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 32232 of 200000 compute units
[2023-10-02T13:56:53.257936000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v failed: custom program error: 0x1777
thread 'poc_borrow' panicked at 'called `Result::unwrap()` on an `Err` value: TransactionError(InstructionError(0, Custom(6007)))', programs/clone/tests/security.rs:1105:49
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test poc_borrow ... FAILED
```

```
[+] Initialize USDC Pyth Oracle
[2023-10-06T11:31:03.166493000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW invoke [1]
[2023-10-06T11:31:03.166767000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Initialize
[2023-10-06T11:31:03.167142000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW consumed 1375 of 200000 compute units
[2023-10-06T11:31:03.167224000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW success
[+] Initialize Pool Pyth Oracle
[2023-10-06T11:31:03.176686000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW invoke [1]
[2023-10-06T11:31:03.176962000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Initialize
[2023-10-06T11:31:03.177333000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW consumed 1375 of 200000 compute units
[2023-10-06T11:31:03.177418000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW success
[+] InitializeOracles Instruction ——>  Instruction { program_id: F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v, accounts: [AccountMeta { pubkey: 4vfHASrBFabhEuESdyd64QyVL1xGmEDzYF9He9nHcRQ, is_signer: true, is_writable: true }
, AccountMeta { pubkey: J2qGHhcUxyJ7ATtUEKxrhscYgQ4V7eKAWSYTfXMdncpm, is_signer: false, is_writable: false }, AccountMeta { pubkey: 2tKFoawguLFCQmPu1Huuczmufdt7JTEfZsctR4RFs4sLu, is_signer: false, is_writable: true }, AccountMet
a { pubkey: 11111111111111111111111111111111, is_signer: false, is_writable: false }], data: [187, 13, 76, 77, 183, 146, 201, 56] }
[2023-10-06T11:31:03.196892000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-06T11:31:03.197076000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: InitializeOracles
[2023-10-06T11:31:03.201705000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2023-10-06T11:31:03.201833000Z TRACE solana_runtime::system_instruction_processor] process_instruction: CreateAccount { lamports: 72161280, space: 10240, owner: F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v }
[2023-10-06T11:31:03.202035000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2023-10-06T11:31:03.204295000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 15014 of 200000 compute units
[2023-10-06T11:31:03.204431000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v success
[+] Oracles Initialized
[+] Update USDC Pyth Oracles Instruction to be added
[2023-10-06T11:31:03.323833000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-06T11:31:03.324285000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: UpdateOracles
[2023-10-06T11:31:03.326587000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 11746 of 200000 compute units
[2023-10-06T11:31:03.326667000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v success
[+] USDC Pyth Oracles Added
[+] UpdateOracles Instruction
[2023-10-06T11:31:03.430910000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-06T11:31:03.431287000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: UpdateOracles
[2023-10-06T11:31:03.433955000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 12533 of 200000 compute units
[2023-10-06T11:31:03.434037000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v success
[+] Oracles Parameters Updated
Oracles updated : Oracles { oracles: [OracleInfo { source: PYTH, address: 7ZEvULMr4E3uujG3Edwg3UY6o9UqyeeDQ8fS2cw4zBJL, price: 0, expo: 0, status: Active, last_update_slot: 0 }, OracleInfo { source: PYTH, address: 1111111ogCyDb
aRMvkdsHB3qfdyFYaG1WtRUAfdh, price: 0, expo: 0, status: Active, last_update_slot: 0 }] }
oracle pyth: 1111111QLbz7JHiBTspS962RLKV8GndWFwiEaqKM
oracles: 2tKFoawguLFCQmPu1HuuczmufDtJ7EfZsctR4RFs4sLu
[+] UpdatePrices Instruction
[2023-10-06T11:31:03.443049000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-06T11:31:03.443447000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: UpdatePrices
[2023-10-06T11:31:03.445147000Z DEBUG solana_runtime::message_processor::stable_log] Program log: AnchorError thrown in programs/clone/src/instructions/update_prices.rs:33. Error Code: IncorrectOracleAddress. Error Number: 6021
. Error Message: Incorrect oracle address provided.
[2023-10-06T11:31:03.445327000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 6572 of 200000 compute units
[2023-10-06T11:31:03.445397000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v failed: custom program error: 0x1785
thread 'poc_borrow' panicked at 'called `Result::unwrap()` on an `Err` value: TransactionError(InstructionError(0, Custom(6021)))', programs/clone/tests/security.rs:3117:49
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test poc_borrow ... FAILED
```

```
[+] Initialize USDC Pyth Oracle
[2023-10-06T11:34:14.050706000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW invoke [1]
[2023-10-06T11:34:14.050985000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Initialize
[2023-10-06T11:34:14.051355000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW consumed 1375 of 200000 compute units
[2023-10-06T11:34:14.051451000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW success
[+] Initialize Pool Pyth Oracle
[2023-10-06T11:34:14.062101000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW invoke [1]
[2023-10-06T11:34:14.062373000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Initialize
[2023-10-06T11:34:14.062789000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW consumed 1375 of 200000 compute units
[2023-10-06T11:34:14.062994000Z DEBUG solana_runtime::message_processor::stable_log] Program 9inbbNkKUi2VwaSH7QiXcHteeHZiXWoTkgYSdX6RmtmW success
[+] InitializeOracles Instruction ——>  Instruction { program_id: F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v, accounts: [AccountMeta { pubkey: Dvr4jjrHTG88MxnMuX9YHpi7AmawfmrHBpZTLocuEV1f, is_signer: true, is_writable: true }
, AccountMeta { pubkey: J2qGHhcUxyJ7ATtUEKxrhscYgQ4V7eKAWSYTfXMdncpm, is_signer: false, is_writable: false }, AccountMeta { pubkey: 2tKFoawguLFCQmPu1Huuczmufdt7JTEfZsctR4RFs4sLu, is_signer: false, is_writable: true }, AccountMet
a { pubkey: 11111111111111111111111111111111, is_signer: false, is_writable: false }], data: [187, 13, 76, 77, 183, 146, 201, 56] }
[2023-10-06T11:34:14.073347000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-06T11:34:14.073801000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: InitializeOracles
[2023-10-06T11:34:14.076136000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2023-10-06T11:34:14.076191000Z TRACE solana_runtime::system_instruction_processor] process_instruction: CreateAccount { lamports: 72161280, space: 10240, owner: F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v }
[2023-10-06T11:34:14.076518000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2023-10-06T11:34:14.078957000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 15014 of 200000 compute units
[2023-10-06T11:34:14.079065000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v success
[+] Oracles Initialized
[+] Update USDC Pyth Oracles Instruction to be added
[2023-10-06T11:34:14.199869000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-06T11:34:14.200247000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: UpdateOracles
[2023-10-06T11:34:14.202684000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 11746 of 200000 compute units
[2023-10-06T11:34:14.202689000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v success
[+] USDC Pyth Oracles Added
[+] UpdateOracles Instruction
[2023-10-06T11:34:14.274391000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-06T11:34:14.274767000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: UpdateOracles
[2023-10-06T11:34:14.277404000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 12533 of 200000 compute units
[2023-10-06T11:34:14.277477000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v success
[+] Oracles Parameters Updated
Oracles updated : Oracles { oracles: [OracleInfo { source: PYTH, address: CdnQzo2hRP09SUJSPQkpSyD13ZVJZEnFGbAfeN9DrMFz, price: 0, expo: 0, status: Active, last_update_slot: 0 }, OracleInfo { source: PYTH, address: 1111111ogCyDb
aRMvkdsHB3qfdyFYaG1WtRUAfdh, price: 0, expo: 0, status: Active, last_update_slot: 0 }] }
oracle pyth: 1111111QLbz7JHiBTspS962RLKV8GndWFwiEaqKM
oracles: 2tKFoawguLFCQmPu1HuuczmufDtJ7EfZsctR4RFs4sLu
[+] UpdatePrices Instruction
[2023-10-06T11:34:14.286137000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v invoke [1]
[2023-10-06T11:34:14.286516000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: UpdatePrices
[2023-10-06T11:34:14.288243000Z DEBUG solana_runtime::message_processor::stable_log] Program log: AnchorError thrown in programs/clone/src/instructions/update_prices.rs:49. Error Code: FailedToLoadPyth. Error Number: 6027. Erro
r Message: Failed to Load Pyth Price Feed.
[2023-10-06T11:34:14.288424000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v consumed 6629 of 200000 compute units
[2023-10-06T11:34:14.288489000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQ5AXKRSRHruSF55jcUxVv6S4SohkHvStd5v failed: custom program error: 0x178b
thread 'poc_borrow' panicked at 'called `Result::unwrap()` on an `Err` value: TransactionError(InstructionError(0, Custom(6027)))', programs/clone/tests/security.rs:3117:49
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test poc_borrow ... FAILED
```

5.2 WITHDRAW COLLATERAL COMET WITH UNHEALTHY SCORE

Description:

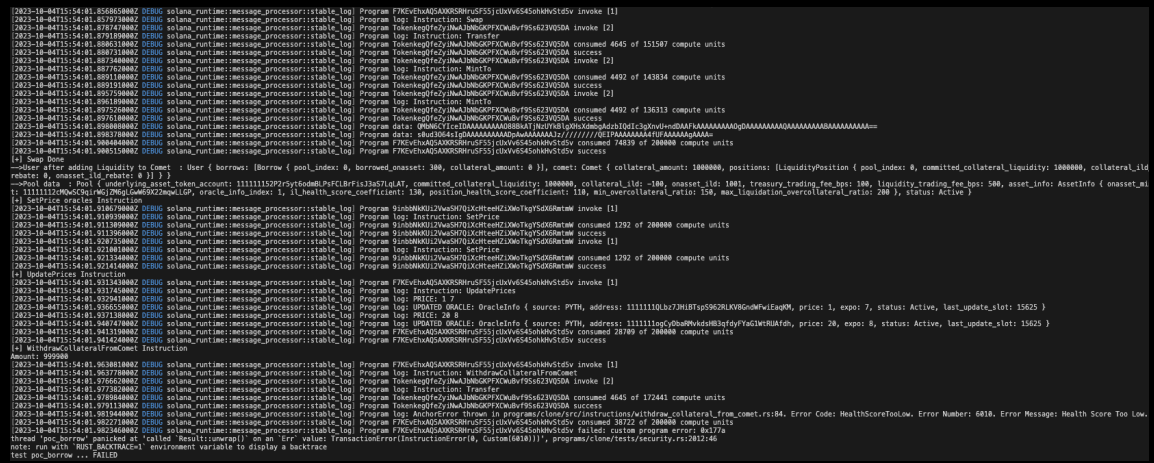
The WithdrawCollateralFromComet instruction provides users with the capability to extract collateral from their kite position. The successful execution of this instruction hinges on two conditions:

The current comet position must maintain a high healthscore.
The result of withdrawal operation must not result in the position becoming undercollateralized.

The corresponding code for this functionality has undergone review and testing to verify its security robustness and ensure the correctness of its functionality. These measures have been taken to mitigate potential security risks and uphold the integrity of the withdrawal process.

Results:

No vulnerabilities were identified.



5.3 BORROWING MORE WITH UNDERCOLLATERALIZED BORROW POSITION

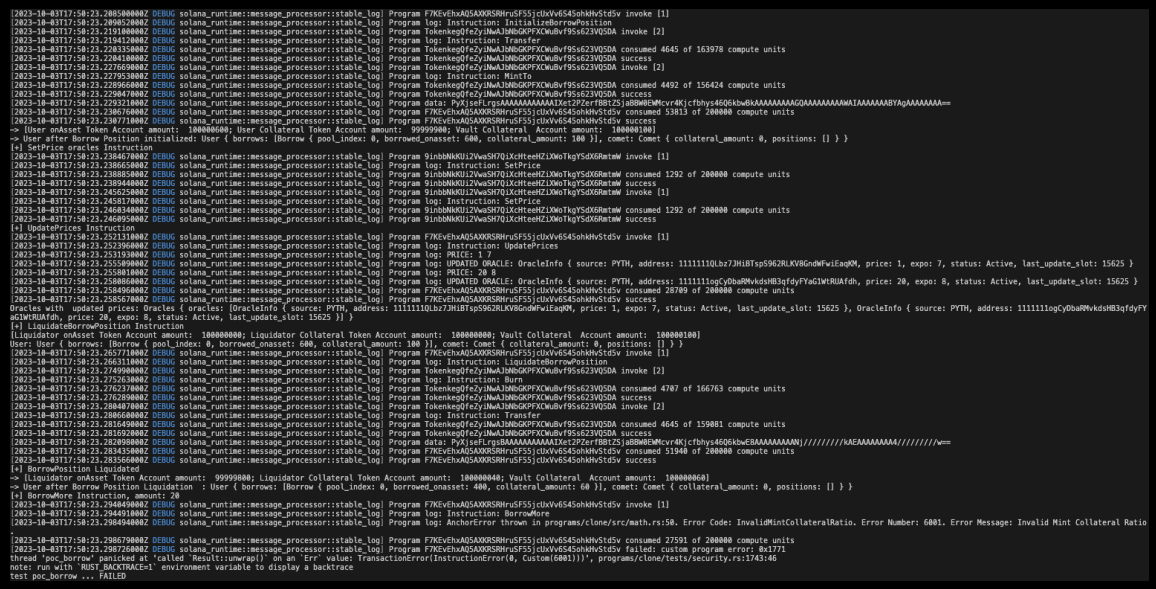
Description:

The BorrowMore instruction facilitates users with an initialized borrow position to request additional onAsset quantity. To execute this instruction successfully, it is imperative that the collateral initially provided for the loan position meets the requirement of being overcollateralized.

Testing has been conducted to validate the correct and safe functionality of the instruction to ensure the robustness of the borrowing process, minimizing potential risks and ensuring consistent, secure operations.

Results:

No vulnerabilities were identified.



5.4 PAY IMPERMANENT LOSS

Description:

The PayImpermanentLossDebt instruction provides a mechanism for settling the debt associated with a user's comet position. This debt can manifest as either **Collateral Impermanent Loss Debt** or **clAsset Impermanent Loss Debt**. To settle these debts, the instruction accepts partial or full payments in collateral or clAsset tokens, depending on the type of debt being addressed.

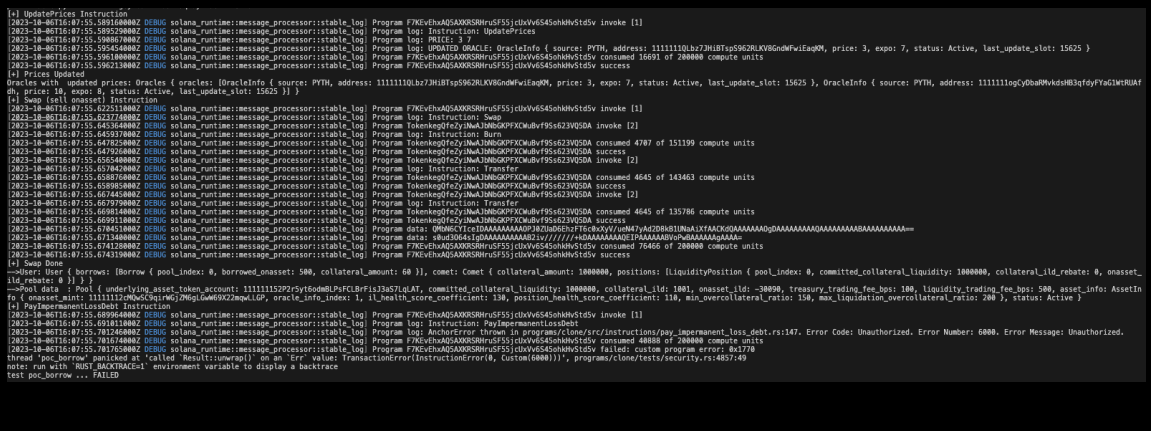
Typically, this settlement process is initiated by either the user who incurred the debt or the Insurance funds.

It is important to note that exclusively users are permitted to use their own comet's collateral to settle the collateral debt. This ensures that only the user with the associated collateral has the authority to pay down this specific type of debt.

Test procedures have been carried out to validate and ensure the security and proper functioning of the debt settlement mechanism.

Results:

No vulnerabilities were identified.



```
[2023-10-05T15:33:20.659949000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2023-10-05T15:33:20.659945000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2023-10-05T15:33:20.661847000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 132553 compute units
[2023-10-05T15:33:20.661946000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2023-10-05T15:33:20.662393000Z DEBUG solana_runtime::message_processor::stable_log] Program data: QMbN6CYIceIEAAAAAAAAAMHB8Y4L5XBnaQDmKvLiwGdVQnnvzm4mY1WhrIMN858uAADGdQAAAAAAAOgDAAAAAAAAAQAAAAAAAABAAAAAAAAA==
[2023-10-05T15:33:20.663166000Z DEBUG solana_runtime::message_processor::stable_log] Program data: s0ud3064sIgEAAAAAAAAACwFP/////91HAAAAAAAAQETPAAAAABVoPwBAAAAAgAAAA=
[2023-10-05T15:33:20.666579000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv consumed 79699 of 200000 compute units
[2023-10-05T15:33:20.666817000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv success
[+] Swap Done
-->User after adding Liquidity to Comet  : User { borrows: [Borrow { pool_index: 0, borrowed_onasset: 500, collateral_amount: 60 }], comet: Comet { collateral_amount: 1000000, positions: [LiquidityPosition { pool_index: 0, committed_collateral_liquidity: 1000000, collateral_ild_rebate: 0, onasset_ild_rebate: 0 }] } }
-->Pool data  : Pool { underlying_asset_token_account: 11111115Q2Pr5yt6odmBLPsFCLBrFis33a57LqLAT, committed_collateral_liquidity: 1000000, collateral_ild: 2002, onasset_ild: -60240, treasury_trading_fee_bps: 100, liquidity_trading_fee_bps: 500, asset_info: AssetInfo { onasset_mint: 11111112cMQwSC9qirWGjZM6gLGwW69X22mqwLLGP, oracle_info_index: 1, il_health_score_coefficient: 130, position_health_score_coefficient: 110, min_overcollateral_ratio: 150, max_liquidation_overcollateral_ratio: 200 }, status: Active }
User Collateral Token Account amount:  ------> 99001940
User OnAsset Token Account amount:  ------> 999940260
[+] CollectLpRewards Instruction
[2023-10-05T15:33:20.684232000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv invoke [1]
[2023-10-05T15:33:20.685189000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: CollectLpRewards
[2023-10-05T15:33:20.701195000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2023-10-05T15:33:20.701831000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: MintTo
[2023-10-05T15:33:20.703886000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4492 of 175582 compute units
[2023-10-05T15:33:20.704006000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2023-10-05T15:33:20.706464000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv consumed 32092 of 200000 compute units
[2023-10-05T15:33:20.706595000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv success
[+] LpRewards Collected
-->User after withdrawing Liquidity from Comet  : User { borrows: [Borrow { pool_index: 0, borrowed_onasset: 500, collateral_amount: 60 }], comet: Comet { collateral_amount: 1000000, positions: [LiquidityPosition { pool_index: 0, committed_collateral_liquidity: 1000000, collateral_ild_rebate: 0, onasset_ild_rebate: -60240 }] } }
User Collateral Token Account amount:  ------> 99001940
User OnAsset Token Account amount:  ------> 1000000580
[+] PayImpermanentLossDebt Instruction
Liquidator Collateral Token Account amount:  ------> 100000000
Vault Collateral Token Account amount:  ------> 100990858
[2023-10-05T15:33:20.725960000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv invoke [1]
[2023-10-05T15:33:20.727176000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: PayImpermanentLossDebt
[2023-10-05T15:33:20.747255000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2023-10-05T15:33:20.747902000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2023-10-05T15:33:20.750414000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 166940 compute units
[2023-10-05T15:33:20.750557000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2023-10-05T15:33:20.753360000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv consumed 40964 of 200000 compute units
[2023-10-05T15:33:20.753514000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv success
[+] ImpermanentLossDebt Collateral Paid
Liquidator Collateral Token Account amount:  ------> 99997998
Vault Collateral Token Account amount:  ------> 101000060
-->User data: User { borrows: [Borrow { pool_index: 0, borrowed_onasset: 500, collateral_amount: 60 }], comet: Comet { collateral_amount: 1000000, positions: [LiquidityPosition { pool_index: 0, committed_collateral_liquidity: 1000000, collateral_ild_rebate: 2002, onasset_ild_rebate: -60240 }] } }
[+] PayImpermanentLossDebt Instruction
[2023-10-05T15:33:20.788175000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv invoke [1]
[2023-10-05T15:33:20.789644000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: PayImpermanentLossDebt
[2023-10-05T15:33:20.820210000Z DEBUG solana_runtime::message_processor::stable_log] Program log: AnchorError thrown in programs/clone/src/instructions/pay_impermanent_loss_debt.rs:113. Error Code: InvalidPaymentType. Error Number: 6035. Error Message: Invalid Payment Type.
[2023-10-05T15:33:20.821097000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv consumed 33453 of 200000 compute units
[2023-10-05T15:33:20.821258000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv failed: custom program error: 0x1793
thread 'poc_borrow' panicked at 'called `Result::unwrap()` on an `Err` value: TransactionError(InstructionError(0, Custom(6035)))', programs/clone/tests/security.rs:2375:49
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test poc_borrow ... FAILED
```

```
[+] UpdatePrices Instruction
[2023-10-06T16:10:29.426910000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv invoke [1]
[2023-10-06T16:10:29.427235000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: UpdatePrices
[2023-10-06T16:10:29.428421000Z DEBUG solana_runtime::message_processor::stable_log] Program log: PRICE: 3 7
[2023-10-06T16:10:29.432129000Z DEBUG solana_runtime::message_processor::stable_log] Program log: UPDATED ORACLE: OracleInfo { source: PYTH, address: 11111110Lbz7JHiBTspS962RLKV8GndWFwiEaqKM, price: 3, expo: 7, status: Active, last_update_slot: 15625 }
[2023-10-06T16:10:29.432672000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv consumed 16691 of 200000 compute units
[2023-10-06T16:10:29.432758000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv success
[+] Prices Updated
Oracles with  updated prices: Oracles { oracles: [OracleInfo { source: PYTH, address: 11111110Lbz7JHiBTspS962RLKV8GndWFwiEaqKM, price: 3, expo: 7, status: Active, last_update_slot: 15625 }, OracleInfo { source: PYTH, address: 11111111ogCyDbaRMvkdsHB3qfdyFYaG1WtRUAfdh, price: 10, expo: 8, status: Active, last_update_slot: 15625 }] }
-->Users: User { borrows: [Borrow { pool_index: 0, borrowed_onasset: 500, collateral_amount: 60 }], comet: Comet { collateral_amount: 1000000, positions: [LiquidityPosition { pool_index: 0, committed_collateral_liquidity: 1000000, collateral_ild_rebate: 0, onasset_ild_rebate: 0 }] } }
-->Pool data : Pool { underlying_asset_token_account: 11111115Q2Pr5yt6odmBLPsFCLBrFis33a57LqLAT, committed_collateral_liquidity: 1000000, collateral_ild: 0, onasset_ild: 0, treasury_trading_fee_bps: 100, liquidity_trading_fee_bps: 500, asset_info: AssetInfo { onasset_mint: 11111112cMQwSC9qirWGjZM6gLGwW69X22mqwLLGP, oracle_info_index: 1, il_health_score_coefficient: 130, position_health_score_coefficient: 110, min_overcollateral_ratio: 150, max_liquidation_overcollateral_ratio: 200 }, status: Active }
[+] PayImpermanentLossDebt Instruction
[2023-10-06T16:10:29.444910000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv invoke [1]
[2023-10-06T16:10:29.445715000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: PayImpermanentLossDebt
[2023-10-06T16:10:29.453437000Z DEBUG solana_runtime::message_processor::stable_log] Program log: AnchorError thrown in programs/clone/src/instructions/pay_impermanent_loss_debt.rs:113. Error Code: InvalidPaymentType. Error Number: 6035. Error Message: Invalid Payment Type.
[2023-10-06T16:10:29.453762000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv consumed 34761 of 200000 compute units
[2023-10-06T16:10:29.453850000Z DEBUG solana_runtime::message_processor::stable_log] Program F7KEvEhxAQSAXXRSRHruSF55jcUkVv6S4SohkHvStdSv failed: custom program error: 0x1793
thread 'poc_borrow' panicked at 'called `Result::unwrap()` on an `Err` value: TransactionError(InstructionError(0, Custom(6035)))', programs/clone/tests/security.rs:4063:49
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test poc_borrow ... FAILED
```

AUTOMATED TESTING

6.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with the 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.

Results:

ID	package	Short Description
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on ‘ed25519-dalek‘

6.2 UNSAFE RUST CODE DETECTION

Description:

Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-geiger, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

Results:

No unsafe code blocks were identified in the packages in scope and their dependencies.

THANK YOU FOR CHOOSING

// HALBORN