Security Assessment Report

stabble AMM

May 31, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the stabble AMM smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in a private repository.

The initial audit focused on the following versions and revealed 11 issues or questions.

program	type	commit
stabble AMM	Solana	`962d9992ff4370cbc89e8e69744d307592b43dc8`

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
stabble AMM		
[L-01] Inconsistent rounding behaviors	Low	Resolved
[L-02] Inconsistent scaling factors	Low	Resolved
[L-03] Consider rounding down taxable_amount_minus_fees	Low	Resolved
[L-04] Missing num_tokens check for initial liquidity	Low	Resolved
[L-05] Loss of precision in swap accounting	Low	Resolved
[L-06] Duplicated pool tokens	Low	Acknowledged
[I-01] Replacing UncheckedAccount with the actual account type	Info	Acknowledged
[I-02] Missing ramp_duration range check	Info	Resolved
[I-03] Consider rounding up discounted fees rate	Info	Resolved
[I-04] The mints of in and out tokens in swap can be the same	Info	Resolved
[I-05] Validate the length of user-provided vectors	Info	Resolved

Findings in Detail

stabble AMM

[L-01] Inconsistent rounding behaviors

The "checked_mul_div_up" at line 313 in "stable_math.rs" should be "checked_mul_div_down", as "b" will be a part of the divisor.

```
/* libraries/math/src/stable_math.rs */
282 | fn get_token_balance_given_invariant_n_all_other_balances(
283 |     amplification: u64,
284 |     balances: &Vec<u64>,
285 |     invariant: u64,
286 |     token_index: usize,
287 | ) -> Result<u64, StableMathError> {
307 |     // We remove the balance from c by multiplying it
308 |     let c = invariant_2
309 |         .checked_mul_div_up(amp_precision_u192(), amp_times_total * p)
310 |         .unwrap()
311 |         * uint192!(balances[token_index]);
312 |     let b = invariant
313 |         .checked_mul_div_up(amp_precision_u192(), amp_times_total)
314 |         .unwrap()
315 |         + sum;
321 |     let mut token_balance = (invariant_2 + c).checked_div_up(invariant + b).unwrap();
```

The corresponding snippet in balancer2 at StableMath.sol#L419-L423

```
/* pkg/pool-stable/contracts/StableMath.sol */
354 | function _calcTokenOutGivenExactBptIn(
362 | ) internal pure returns (uint256) {
418 |     // We remove the balance from c by multiplying it
419 |     uint256 c = Math.mul(
420 |         Math.mul(Math.divUp(inv2, Math.mul(ampTimesTotal, P_D)), _AMP_PRECISION),
421 |         balances[tokenIndex]
422 |     );
423 |     uint256 b = sum.add(Math.mul(Math.divDown(invariant, ampTimesTotal), _AMP_PRECISION));
```

Resolution

This issue has been resolved by commit 98770969aed2a3993364662cb8dc7ca16cd77a4d.

stabble AMM

[L-02] Inconsistent scaling factors

When creating a new pool, users are required to specify the "`max_caps`" for all tokens. This is used to calculate a scaling factor for the respective token amounts to prevent overflow.

In particular, in the "`weighted-math.rs`", because the balance sum (amounts after scaling) needs to be represented by "`U34F30`", the numbers after scaling should not exceed "`MAX_SAFE_BALANCE_INT`".

```
/* programs/stable-swap/src/instructions/initialize.rs */
008 | pub fn process_initialize(ctx: Context<Initialize>, amp_factor: u16, swap_fee: u64, max_caps:
↪  &Vec<u64>) -> Result<()> {
024 |     for (token_index, account) in ctx.remaining_accounts.iter().enumerate() {
030 |         let default_scaling_factor =
↪  10_u64.saturating_pow(fixed_math::SCALE.saturating_sub(decimals));
031 |         let (scaling_up, scaling_factor) = if max_caps[token_index] >
↪  stable_math::MAX_SAFE_BALANCE_INT {
032 |             let tick_size = max_caps[token_index]
033 |                 .checked_div_up(stable_math::MAX_SAFE_BALANCE_INT)
034 |                 .unwrap();
035 |             if default_scaling_factor >= tick_size {
036 |                 (true, default_scaling_factor / tick_size)
037 |             } else {
038 |                 (false, tick_size)
039 |             }
040 |         } else {
041 |             (true, default_scaling_factor)
042 |         };
```

```
/* programs/weighted-swap/src/instructions/initialize.rs */
008 | pub fn process_initialize(
009 |     ctx: Context<Initialize>,
010 |     swap_fee: u64,
011 |     weights: Vec<u64>,
012 |     max_caps: &Vec<u64>,
013 | ) -> Result<()> {
026 |     for (token_index, account) in ctx.remaining_accounts.iter().enumerate() {
034 |         let default_scaling_factor =
↪  10_u64.saturating_pow(fixed_math::SCALE.saturating_sub(decimals));
035 |         let (scaling_up, scaling_factor) = if max_caps[token_index] >
↪  weighted_math::MAX_SAFE_BALANCE_INT {
036 |             let tick_size = max_caps[token_index]
037 |                 .checked_div_up(weighted_math::MAX_SAFE_BALANCE_INT)
038 |                 .unwrap();
039 |             if default_scaling_factor >= tick_size {
040 |                 (true, default_scaling_factor / tick_size)
041 |             } else {
042 |                 (false, tick_size)
043 |             }
```

```
044 |         } else {
045 |             (true, default_scaling_factor)
046 |         };
```

Assuming the user-provided "max_caps" are the actual token amounts without any scaling, the amounts after scaling can be larger than "MAX_SAFE_BALANCE_INT".

According to the "scaling_factor" logic in both "stable-swap" and "weighted-swap":

1. When "max_caps[token_index] <= MAX_SAFE_BALANCE_INT", the wrapped amount will be "amount * default_scaling_factor", which can be larger than "MAX_SAFE_BALANCE_INT".
2. When "max_caps[token_index] > MAX_SAFE_BALANCE_INT" and "default_scaling_factor >= tick_size", the wrapped amount will be "amount * default_scaling_factor / tick_size", which is "amount * MAX_SAFE_BALANCE_INT * default_scaling_factor / max_caps[token_index]" and can be larger than "MAX_SAFE_BALANCE_INT".
3. When "max_caps[token_index] > MAX_SAFE_BALANCE_INT" and "default_scaling_factor < tick_size", the wrapped amount will be "amount / tick_size", which does not consider the "default_scaling_factor" and is inconsistent with the previous two scenarios.

Resolution

This issue has been resolved by commit 98770969aed2a3993364662cb8dc7ca16cd77a4d.

stabble AMM

[L-03] Consider rounding down taxable_amount_minus_fees

In "`calc_token_out_given_exact_pool_token_in`", although the comments specify that fees should be rounded up, the computation on line 270, which uses "`mul_up`" to round up the amount after the fees, actually rounds down the swap fee.

Notably, the reference implementation in Balancer also employs "`mulUp`". Despite this, consider changing the method to "`mul_down`" to align the rounding direction with the intended behavior as described in the comments.

```
/* libraries/math/src/weighted_math.rs */
232 | pub fn calc_token_out_given_exact_pool_token_in(
233 |     balance: u64,
234 |     normalized_weight: u64,
235 |     amount_in: u64,
236 |     pool_token_supply: u64,
237 |     swap_fee: u64,
238 | ) -> Result<u64, WeightedMathError> {
247 |     // Token out, so we round down overall. ...
266 |     // Swap fees are typically charged on 'token in', but there is no 'token in' here, so we apply
267 |     // to 'token out'. This results in slightly larger price impact. Fees are rounded up.
268 |     let taxable_amount = amount_out_without_fee.mul_up(normalized_weight.complement());
269 |     let non_taxable_amount = amount_out_without_fee - taxable_amount;
270 |     let taxable_amount_minus_fees = taxable_amount.mul_up(swap_fee.complement());
271 |
272 |     Ok(non_taxable_amount + taxable_amount_minus_fees)
273 | }
```

Resolution

This issue has been resolved by commit `237bbb5ee33a16aa4c389b7b128d4aafefa181b2`.

stabble AMM

[L-04] Missing num_tokens check for initial liquidity

In both "stable-swap" and "weighted-swap", for the initial liquidity deposit, the program does not verify the number of tokens, "num_tokens", to ensure that this number is not equal to "1". When it happens, the invariant calculation won't consider all the pools.

Although only the pool owner can provide initial liquidity, it is still recommended to implement checks to validate the quantity of "num_tokens" and ensure it's the same as the number of pool tokens.

Initial deposit in stable-swap

```
/* programs/stable-swap/src/instructions/deposit.rs */
010 | pub fn process_deposit<'a, 'b, 'c, 'info>(
011 |     ctx: Context<'_, '_, '_, 'info, Deposit<'info>>,
012 |     amounts: Vec<u64>,
013 |     minimum_amount_out: u64,
014 | ) -> Result<()> {
019 |     let amount_out = if ctx.accounts.mint.supply == 0 {
020 |         assert_eq!(ctx.accounts.user.key(), ctx.accounts.pool.owner);
022 |         // initial liquidity
023 |         stable_math::calc_invariant(
024 |             amplification,
025 |             &amounts
026 |                 .iter()
027 |                 .enumerate()
028 |                 .map(|(token_index, &amount)| {
029 |                     let mint = get_token_mint(&ctx.remaining_accounts[token_index]).unwrap();
030 |                     assert_eq!(ctx.accounts.pool.tokens[token_index].mint, mint);
031 |

126 | impl<'info> Deposit<'info> {
127 |     pub fn validate(ctx: &Context<Deposit>, amounts: &Vec<u64>) -> Result<()> {
132 |         let num_tokens = amounts.len();
133 |         assert_ne!(num_tokens, 0);
135 |         if num_tokens > 1 {
136 |             assert_eq!(num_tokens, ctx.accounts.pool.tokens.len());
137 |         }
```

Initial deposit in weighted-swap

```
/* programs/weighted-swap/src/instructions/deposit.rs */
010 | pub fn process_deposit<'a, 'b, 'c, 'info>(
011 |     ctx: Context<'_, '_, '_, 'info, Deposit<'info>>,
012 |     amounts: Vec<u64>,
```

```
013 |     minimum_amount_out: u64,
014 | ) -> Result<()> {
018 |     let amount_out = if ctx.accounts.pool.invariant == 0 {
019 |         assert_eq!(ctx.accounts.user.key(), ctx.accounts.pool.owner);
021 |         // initial liquidity
022 |         let invariant = weighted_math::calc_invariant(
023 |             &amounts
024 |                 .iter()
025 |                 .enumerate()
026 |                 .map(|(token_index, &amount)| {
027 |                     let mint = get_token_mint(&ctx.remaining_accounts[token_index]).unwrap();
028 |                     assert_eq!(ctx.accounts.pool.tokens[token_index].mint, mint);

/* programs/weighted-swap/src/instructions/deposit.rs */
118 | pub fn validate(ctx: &Context<Deposit>, amounts: &Vec<u64>) -> Result<()> {
123 |     let num_tokens = amounts.len();
124 |     assert_ne!(num_tokens, 0);
126 |     if num_tokens > 1 {
127 |         assert_eq!(num_tokens, ctx.accounts.pool.tokens.len());
128 |     }
```

Resolution

This issue has been resolved by commit 98770969aed2a3993364662cb8dc7ca16cd77a4d.

stabble AMM

[L-05] Loss of precision in swap accounting

The "swap" function calculates the "amount_out", "swap_fee_amount", and "beneficiary_fee_amount" after converting the wrapped amount into the actual transfer amount, known as the unwrapped amount. Subsequently, the sum of "amount_out" and "beneficiary_fee_amount" is converted back into a wrapped amount for accounting purposes.

In the final step, the "calc_wrapped_amount" function performs a rounding down operation, which may result in the recorded "balance_out" being smaller than the actual value transferred.

```
/* programs/weighted-swap/src/instructions/swap.rs */
054 | let amount_out_without_fee = ctx
055 |     .accounts
056 |     .pool
057 |     .calc_unwrapped_amount(balance_out_without_fee, token_out_index);
058 | let amount_out = amount_out_without_fee.mul_down(swap_fee.complement());
059 | assert!(amount_out >= minimum_amount_out); // check slippage
061 | let swap_fee_amount = amount_out_without_fee.saturating_sub(amount_out);
062 | let beneficiary_fee_amount = swap_fee_amount.mul_down(ctx.accounts.vault.beneficiary_fee);
064 | // add in token balance
065 | ctx.accounts.pool.tokens[token_in_index].balance = ctx.accounts.pool.tokens[token_in_index].balance
↪  + balance_in;
066 | // remove out token balance
067 | let balance_out = ctx
068 |     .accounts
069 |     .pool
070 |     .calc_wrapped_amount(amount_out + beneficiary_fee_amount, token_out_index);
071 | ctx.accounts.pool.tokens[token_out_index].balance =
↪  ctx.accounts.pool.tokens[token_out_index].balance - balance_out;
```

Resolution

This issue has been resolved by commit 1a81282713d2b3baa9ce2e471c37e2201097c933.

stabble AMM

[L-06] Duplicated pool tokens

When adding a new token to "`Pool.tokens`", it is necessary to check if the same mint already exists to prevent duplications.

stable-swap

```
/* programs/stable-swap/src/instructions/initialize.rs */
008 | pub fn process_initialize(ctx: Context<Initialize>, ...) -> Result<()> {
024 |     for (token_index, account) in ctx.remaining_accounts.iter().enumerate() {
044 |         ctx.accounts.pool.tokens.push(PoolToken {
045 |             mint: account.key(),
050 |         });
051 |     }
```

weighted-swap

```
/* programs/weighted-swap/src/instructions/initialize.rs */
008 | pub fn process_initialize(
009 |     ctx: Context<Initialize>,
013 | ) -> Result<()> {
026 |     for (token_index, account) in ctx.remaining_accounts.iter().enumerate() {
048 |         ctx.accounts.pool.tokens.push(PoolToken {
049 |             mint: account.key(),
055 |         });
056 |     }
```

Resolution

The team has acknowledged this issue and clarified that the respective checks will be conducted on the client side, thereby avoiding the need to further complicate the contract.

stabble AMM

[I-01] Replacing UncheckedAccount with the actual account type

All three contracts commonly use "UncheckedAccount".

It's recommended to replace them with the actual types to leverage the owner and type checks performed by Anchor.

```
/* programs/stable-swap/src/instructions/swap.rs */
147 | pub struct Swap<'info> {
150 |     /// CHECK: optional xSTB token account for swap fee discount
151 |     pub user_x_token: Option<UncheckedAccount<'info>>,
153 |     /// CHECK: OK
154 |     #[account(mut)]
155 |     pub user_token_in: UncheckedAccount<'info>,
156 |     /// CHECK: OK
157 |     #[account(mut)]
158 |     pub user_token_out: UncheckedAccount<'info>,
163 |     /// CHECK: OK
164 |     #[account(mut)]
165 |     pub vault_token_out: UncheckedAccount<'info>,
```

Resolution

The team acknowledged this issue and clarified that they deliberately use "UncheckedAccount" to conserve stack or heap spaces, provided that it does not compromise security.

stabble AMM

[I-02] Missing ramp_duration range check

In the "change_amp_factor" instruction, the pool owner can specify a new ramp duration and amp factor to adjust the amplification factor gradually. However, the program does not verify whether the "ramp_duration" is greater than zero.

If transactions occur within the same slot, it is unclear whether the "amp_initial_factor" or the "amp_target_factor" should be used.

Although the current implementation defaults to using the "amp_initial_factor", it is recommended to ensure that the "ramp_duration" is greater than zero to prevent such ambiguities.

```
/* programs/stable-swap/src/instructions/config.rs */
006 | pub fn process_change_amp_factor<'info>(
007 |     ctx: Context<OwnerOnly<'info>>,
008 |     new_amp_factor: u16,
009 |     ramp_duration: u32,
010 | ) -> Result<()> {
011 |     assert_ne!(ctx.accounts.pool.amp_target_factor, new_amp_factor);
012 |     assert!(new_amp_factor >= stable_math::MIN_AMP);
013 |     assert!(new_amp_factor <= stable_math::MAX_AMP);
014 |
015 |     ctx.accounts.pool.amp_initial_factor = u16::try_from(
016 |         ctx.accounts
017 |             .pool
018 |             .get_amplification()
019 |             .checked_div_up(stable_math::AMP_PRECISION)
020 |             .unwrap(),
021 |     )
022 |     .unwrap();
023 |     ctx.accounts.pool.amp_target_factor = new_amp_factor;
024 |     ctx.accounts.pool.ramp_start_ts = Clock::get().unwrap().unix_timestamp;
025 |     ctx.accounts.pool.ramp_stop_ts = ctx.accounts.pool.ramp_start_ts + ramp_duration as i64;
026 |
027 |     ctx.accounts.pool.emit_updated_event();
028 |
029 |     Ok(())
030 | }
```

Resolution

This issue has been resolved by commit 98770969aed2a3993364662cb8dc7ca16cd77a4d.

stabble AMM

[I-03] Consider rounding up discounted fees rate

The "`calc_swap_fee_in_discount`" function calculates the discounted trading fee rate based on the quantity of tokens held by the user. However, during the multiplication step in the calculation, rounding down is performed.

Considering that these fees are being charged to users, rounding up would be a safer practice. Nonetheless, in typical scenarios where the fee rate is set with trailing zeros, rounding may not occur at all.

```
/* libraries/math/src/swap_fee_math.rs */
003 | pub fn calc_swap_fee_in_discount(swap_fee: u64, x_amount: u64) -> u64 {
004 |     // No discount
005 |     if x_amount < 100_000_000_000_000 {
006 |         swap_fee
007 |     }
008 |     // 10% discount
009 |     else if x_amount < 200_000_000_000_000 {
010 |         swap_fee.mul_down(900_000_000)
011 |     }
012 |     // 20% discount
013 |     else if x_amount < 400_000_000_000_000 {
014 |         swap_fee.mul_down(800_000_000)
015 |     }
016 |     // 30% discount
017 |     else if x_amount < 800_000_000_000_000 {
018 |         swap_fee.mul_down(700_000_000)
019 |     }
020 |     // 40% discount
021 |     else if x_amount < 1_600_000_000_000_000 {
022 |         swap_fee.mul_down(600_000_000)
023 |     }
024 |     // 50% discount
025 |     else if x_amount < 3_200_000_000_000_000 {
026 |         swap_fee >> 1 // div by 2
027 |     }
028 |     // 60% discount
029 |     else if x_amount < 6_400_000_000_000_000 {
030 |         swap_fee.mul_down(400_000_000)
031 |     }
032 |     // 70% discount
033 |     else if x_amount < 12_800_000_000_000_000 {
034 |         swap_fee.mul_down(300_000_000)
035 |     }
036 |     // 80% discount
037 |     else if x_amount < 25_600_000_000_000_000 {
038 |         swap_fee.mul_down(200_000_000)
```

```
039 |     }
040 |     // 90% discount
041 |     else if x_amount < 51_200_000_000_000_000 {
042 |         swap_fee.mul_down(100_000_000)
043 |     }
044 |     // 100% discount
045 |     else {
046 |         0
047 |     }
048 | }
```

Resolution

This issue has been resolved by commit `98770969aed2a3993364662cb8dc7ca16cd77a4d`.

stabble AMM

[I-04] The mints of in and out tokens in swap can be the same

The "swap" functions in both "stable-swap" and "weighted-swap" do not check if "token_in_index" is the same as the "token_out_index".

```
/* programs/weighted-swap/src/instructions/swap.rs */
018 | pub fn process_swap(ctx: Context<Swap>, amount_in: Option<u64>, minimum_amount_out: u64) ->
↪  Result<()> {
019 |     let token_in_index = ctx.accounts.pool.get_token_index(ctx.accounts.vault_token_in.mint);
020 |     let token_out_index = ctx
021 |         .accounts
022 |         .pool
023 |         .get_token_index(ctx.accounts.beneficiary_token_out.mint);
```

Consider adding a mint check to ensure they are different.

Resolution

This issue has been resolved by commit 98770969aed2a3993364662cb8dc7ca16cd77a4d.

stabble AMM

[I-05] Validate the length of user-provided vectors

The length of "`remaining_accounts`" and "`max_caps`" should be the same.

```
/* programs/stable-swap/src/instructions/initialize.rs */
008 | pub fn process_initialize(ctx: Context<Initialize>, ..., max_caps: &Vec<u64>) -> Result<()> {
024 |     for (token_index, account) in ctx.remaining_accounts.iter().enumerate() {
031 |         let (scaling_up, scaling_factor) = if max_caps[token_index] > ... {
032 |             let tick_size = max_caps[token_index]
040 |         } else {
042 |         };
051 |     }
054 | }
```

The length of "`remaining_accounts`", "`weights`" and "`max_caps`" should be the same.

```
/* amm-rust-sdk/programs/weighted-swap/src/instructions/initialize.rs */
008 | pub fn process_initialize(
009 |     ctx: Context<Initialize>,
011 |     weights: Vec<u64>,
012 |     max_caps: &Vec<u64>,
013 | ) -> Result<()> {
026 |     for (token_index, account) in ctx.remaining_accounts.iter().enumerate() {
035 |         let (scaling_up, scaling_factor) = if max_caps[token_index] > ... {
036 |             let tick_size = max_caps[token_index]
044 |         } else {
046 |         };
048 |         ctx.accounts.pool.tokens.push(PoolToken {
054 |             weight: weights[token_index],
055 |         });
056 |     }
058 | }
```

Validate the length among "`remaining_accounts`", "`.pool.tokens`", and "`minimum_amounts_out`".

```
/* programs/stable-swap/src/instructions/withdraw.rs */
014 | pub fn process_withdraw<'a, 'b, 'c, 'info>(
015 |     ctx: Context<'_, '_, '_, 'info, Withdraw<'info>>,
016 |     amount: u64,
017 |     minimum_amounts_out: Vec<u64>,
018 | ) -> Result<()> {
023 |     if ctx.remaining_accounts.len() == 2 {
044 |     } else {
048 |         for (token_index, user_account) in
↪  ctx.remaining_accounts[0..balances_out.len()].iter().enumerate() {
050 |             assert_eq!(ctx.accounts.pool.tokens[token_index].mint, mint); // check token orders
059 |             assert!(amount_out >= minimum_amounts_out[token_index]); // check slippage
061 |             ctx.accounts.transfer_to_user(
```

```
064 |                 &ctx.remaining_accounts[token_index + balances_out.len()],
065 |             )?;
066 |         }
067 |     };
```

Validate the length among "`remaining_accounts`", "`.pool.tokens`", and "`minimum_amounts_out`".

```
/* amm-rust-sdk/programs/weighted-swap/src/instructions/withdraw.rs */
014 | pub fn process_withdraw<'a, 'b, 'c, 'info>(
015 |     ctx: Context<'_, '_, '_, 'info, Withdraw<'info>>,
016 |     amount: u64,
017 |     minimum_amounts_out: Vec<u64>,
018 | ) -> Result<()> {
019 |     if ctx.remaining_accounts.len() == 2 {
038 |     } else {
045 |         for (token_index, user_account) in
↪   ctx.remaining_accounts[0..balances_out.len()].iter().enumerate() {
047 |             assert_eq!(ctx.accounts.pool.tokens[token_index].mint, mint); // check token orders
056 |             assert!(amount_out >= minimum_amounts_out[token_index]); // check slippage
058 |             ctx.accounts.transfer_to_user(
061 |                 &ctx.remaining_accounts[token_index + balances_out.len()],
062 |             )?;
063 |         }
064 |     };
```

Resolution

This issue has been resolved by commit `98770969aed2a3993364662cb8dc7ca16cd77a4d`.

Appendix: Methodology and Scope of Work

The Sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the Sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and SOLA-X Ltd. d/b/a stabble (the "Client''). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, Sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At Sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.

Sec3™