Sec3™

Security Assessment Report

Wormhole Example Liquidity Layer

June 07, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Wormhole Example Liquidity Layer 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 13 issues or questions.

program	type	commit
example-liquidity-layer	EVM, Solana	`e7a987323756218a5c4d6d8ebc4a00a92291a935`

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

Table of Contents

Result Overview

Issue	Impact	Status
EXAMPLE-LIQUIDITY-LAYER		
[M-01] Closed best_offer_token may interrupt the execution flow	Medium	Resolved
[M-02] Inconsistent offer_token requirements	Medium	Resolved
[L-01] improve_offer allows unchanged offer_price under certain circumstances	Low	Resolved
[L-02] add_auction_history_entry does not handle closed initial_offer_token	Low	Acknowledged
[I-01] Rejecting a non-existent target_chain in prepare_market_order	Info	Resolved
[I-02] Auction can be created even after reserve_fast_fill_sequence_no_auction	Info	Resolved
[Q-01] Does the protocol ensure settlement even when there is no auction?	Question	Resolved
[Q-02] Inconsistent update_auction_parameters caller requirement?	Question	Resolved
[Q-03] Is the auction history function designed to be called separately?	Question	Resolved
[Q-04] Router endpoint closure instruction?	Question	Resolved
[Q-05] Inconsistent chain ids and CCTP domains?	Question	Resolved
[Q-06] Does the protocol collect a partial penalty?	Question	Resolved
[Q-07] Operations allowed when the custodian is paused?	Question	Resolved

Findings in Detail

EXAMPLE-LIQUIDITY-LAYER

[M-01] Closed best_offer_token may interrupt the execution flow

The “`reserve_fast_fill_sequence_active_auction`” requires a valid “`best_offer_token`”.

However, if the “`best_offer_token`” is closed, “`reserve_fast_fill_sequence_active_auction`” will be blocked, meaning “`ReservedFastFillSequence`” cannot be created. This subsequently blocks the execution of “`execute_fast_order_local`”. As a result, the entire execution process stalls, and users cannot receive the token.

However, attackers are not economically motivated since they may lose their tokens.

```
/* solana/programs/matching-engine/src/processor/fast_fill/reserve_sequence/active_auction.rs */
pub struct ReserveFastFillSequenceActiveAuction<'info> {
    reserve_sequence: ReserveFastFillSequence<'info>,

    #[account(
        constraint = {
            let info = auction.info.as_ref().unwrap();
            require_keys_eq!(
                best_offer_token.key(),
                info.best_offer_token,
                MatchingEngineError::BestOfferTokenMismatch
            );

            true
        }
    )]
    best_offer_token: Account<'info, token::TokenAccount>,
}
```

Resolution

This issue has been resolved by commit 4460ae3 in PR172.

Even though an attacker isn’t economically motivated, a troll may be motivated to pay to mess with someone by participating in an auction so that a user will never get his funds on the destination network.

EXAMPLE-LIQUIDITY-LAYER

[M-02] Inconsistent offer_token requirements

In "`place_initial_offer_cctp`" and "`improve_offer`", the "`offer_token`" can be any token account, and it does not necessarily have to be an ATA.

```
/* solana/programs/matching-engine/src/processor/auction/offer/place_initial/cctp.rs */
pub struct PlaceInitialOfferCctp<'info> {
    offer_token: Account<'info, token::TokenAccount>,

/* solana/programs/matching-engine/src/processor/auction/offer/improve.rs */
pub struct ImproveOffer<'info> {
    #[account(
        constraint = {
            offer_token.key() != active_auction.custody_token.key()
        } @ MatchingEngineError::InvalidOfferToken,
    )]
    offer_token: Account<'info, token::TokenAccount>,
```

However, in "`settle_auction_complete`", when the "`execute_penalty`" is "`None`", the "`best_offer_token`" must be the "`executor_token`", which should be an ATA of the "`executor`".

```
/* solana/programs/matching-engine/src/processor/auction/settle/complete.rs */
pub struct SettleAuctionComplete<'info> {
    #[account(
        mut,
        associated_token::mint = best_offer_token.mint,
        associated_token::authority = executor,
    )]
    executor_token: Account<'info, token::TokenAccount>,

fn handle_settle_auction_complete(
    ctx: Context<SettleAuctionComplete>,
    execute_penalty: Option<u64>,
) -> Result<()> {
    match execute_penalty {
        None => {
            require_keys_eq!(
                executor_token.key(),
                best_offer_token.key(),
                MatchingEngineError::ExecutorTokenMismatch
            )
        }
```

If a bidder uses a non-ATA to supply tokens for the auction, she will not be able to settle the auction and reclaim her funds.

Resolution

This issue has been resolved by commit 62adb25 in PR168.

EXAMPLE-LIQUIDITY-LAYER

[L-01] improve_offer allows unchanged offer_price under certain circumstances

The "`improve_offer`" requires that the new "`offer_price`" should not exceed the return value of the function "`compute_min_allowed_offer`".

However, if the "`active_auction.info.offer_price`" is sufficiently low (such that "`offer_price * min_offer_delta_bps / FEE_PRECISION_MAX == 0`"), the return value of "`compute_min_allowed_offer`" could equal "`info.offer_price`", which means the new price could be the same as the existing one.

This can lead to two consequences:

1. Any bidder can replace the current best_offer with the same "`offer_price`".
2. The "`transfer_authority`" may be reused, because the seeds of "`transfer_authority`" are "`[TRANSFER_AUTHORITY_SEED_PREFIX, auction.key, offer_price]`". If the amount is not precisely set during delegation, it may result in the unintended transfer of the bidder's tokens, because "`improve_offer`" does not require the owner of "`offer_token`" to be a signer.

```
/* solana/programs/matching-engine/src/processor/auction/offer/improve.rs */
pub struct ImproveOffer<'info> {
    #[account(
        seeds = [
            TRANSFER_AUTHORITY_SEED_PREFIX,
            active_auction.key().as_ref(),
            &offer_price.to_be_bytes()
        ],
        bump
    )]
    transfer_authority: UncheckedAccount<'info>,

    #[account(
        constraint = {
            // This is safe because we know that this is an active auction.
            let info = active_auction.info.as_ref().unwrap();

            require!(
                info.within_auction_duration(&active_auction.config),
                MatchingEngineError::AuctionPeriodExpired
            );

            require!(
                offer_price
```

```
                    <= utils::auction::compute_min_allowed_offer(&active_auction.config, info),
                MatchingEngineError::CarpingNotAllowed
            );

            true
        }
    )]
    active_auction: ActiveAuction<'info>,
}

/* solana/programs/matching-engine/src/utils/auction.rs */
pub fn compute_min_allowed_offer(params: &AuctionParameters, info: &AuctionInfo) -> u64 {
    info.offer_price
        .saturating_sub(mul_bps_unsafe(info.offer_price, params.min_offer_delta_bps))
}
```

Resolution

This issue has been resolved by commit cbcdd34 in PR171. The constraint on the “`offer_price`” has been changed to strictly less than.

EXAMPLE-LIQUIDITY-LAYER

[L-02] add_auction_history_entry does not handle closed initial_offer_token

In the "`add_auction_history_entry`" instruction, the current implementation does not account for the possibility that the "`initial_offer_token`" account may have already been closed.

```
/* programs/matching-engine/src/processor/auction/history/add_entry.rs */
/// CHECK: This account will either be the owner of the fee recipient token account (if there
/// was no auction) or the owner of the initial offer token account.
#[account(mut)]
beneficiary: UncheckedAccount<'info>,

#[account(
    token::authority = beneficiary,
    address = {
        match &auction.info {
            Some(info) => info.initial_offer_token,
            None => custodian.fee_recipient_token,
        }
    }
)]
beneficiary_token: Account<'info, token::TokenAccount>,
```

This prevents the corresponding "`auction`" account from being properly closed and the related records from being archived into the history. Therefore, the impact of this issue depends on the importance of the history.

Resolution

The team acknowledged this finding. This instruction helps the initial offer participant reclaim their lamports. It's acceptable if the participant cannot recover their rent because they want to close their token account.

EXAMPLE-LIQUIDITY-LAYER

[I-01] Rejecting a non-existent target_chain in prepare_market_order

In the "`prepare_market_order`" process, the "`target_chain`" is not checked, which may allow initiation of a "`prepare_market_order`" for a non-existent "`target_chain`".

This does not introduce a security issue, as the "`target_chain`" will be verified later during the "`place_market_order_cctp`" process.

However, rejecting a "`prepare_market_order`" with a non-existent "`target_chain`" from the start could save users the effort of having to close a prepared order after discovering they cannot proceed with "`place_market_order_cctp`".

```
/* solana/programs/token-router/src/processor/market_order/prepare.rs */

let PrepareMarketOrderArgs {
    amount_in,
    min_amount_out,
    target_chain,
    redeemer,
    redeemer_message,
} = args;

/* solana/programs/token-router/src/processor/market_order/place_cctp.rs */
#[account(
    seeds = [
        matching_engine::state::RouterEndpoint::SEED_PREFIX,
        router_endpoint.chain.to_be_bytes().as_ref(),
    ],
    bump = router_endpoint.bump,
    seeds::program = matching_engine::id(),
    constraint = {
        require_eq!(
            router_endpoint.chain,
            prepared_order.target_chain,
            TokenRouterError::InvalidTargetRouter,
        );

        true
    }
)]
router_endpoint: Box<Account<'info, matching_engine::state::RouterEndpoint>>,
```

Resolution

This issue has been fixed by commits 0bf3717 and bc2e336 in PR177.

EXAMPLE-LIQUIDITY-LAYER

[I-02] Auction can be created even after reserve_fast_fill_sequence_no_auction

For a request whose dst chain is Solana (the same chain as the matching-engine), when both "`fast_vaa`" and "`finalized_vaa`" have arrived, if the auction has not been created yet and "`PreparedOrderResponse`" has been set up by "`prepare_order_response_cctp`", anyone can call "`reserve_fast_fill_sequence_no_auction`" to create a "`ReservedFastFillSequence`" for this request.

However, this interface only checks if the auction doesn't exist but doesn't actually create one, which allows other users to create an "`Auction`" through "`prepare_order_response_cctp`" even after "`reserve_fast_fill_sequence_no_auction`" has been called.

```
/* solana/programs/matching-engine/src/processor/fast_fill/reserve_sequence/no_auction.rs */
009 | pub struct ReserveFastFillSequenceNoAuction<'info> {
032 |     #[account(
033 |         seeds = [
034 |             Auction::SEED_PREFIX,
035 |             prepared_order_response.seeds.fast_vaa_hash.as_ref(),
036 |         ],
037 |         bump,
038 |         constraint = auction.data_is_empty() @ MatchingEngineError::AuctionExists,
039 |     )]
040 |     auction: UncheckedAccount<'info>,
041 | }
```

After the auction ends, a "`ReservedFastFillSequence`" cannot be created using the procedure "`reserve_fast_fill_sequence_active_auction`", because the account seed is the same as that in "`reserve_fast_fill_sequence_no_auction`".

This consequently affects "`best_offer_participant`" in subsequent "`execute_fast_order_local`" operations. It won't match the expected "`best_offer_token.owner`" but will estead be equal to "`prepared_order_response.prepared_by`". This discrepancy affects the accounts that will get the returning lamports when the "`ReservedFastFillSequence`" account is closed.

```
/* solana/programs/matching-engine/src/processor/auction/execute_fast_order/local.rs */
012 | pub struct ExecuteFastOrderLocal<'info> {
043 |     /// When the reserved sequence account was created, the beneficiary was set to the best offer
044 |     /// token's owner. This account will receive the lamports from the reserved sequence account.
```

```
    ///
    /// CHECK: This account's address must equal the one encoded in the reserved sequence account.
    #[account(
        mut,
        address = reserved_sequence.beneficiary,
    )]
    best_offer_participant: UncheckedAccount<'info>,
}

pub fn execute_fast_order_local(ctx: Context<ExecuteFastOrderLocal>) -> Result<()> {
    ctx.accounts
        .reserved_sequence
        .close(ctx.accounts.best_offer_participant.to_account_info())
}
```

This won't cause serious security issues, but it might disrupt the whole process and could potentially prevent the bidder from correctly completing the auction.

Resolution

This issue was resolved by commit bd48684 in PR174.

Now the program will create the auction account in the "`reserve_fast_fill_sequence_no_auction`" instruction and change "`settle_auction_none_local`" to check that this is an "empty" auction (info is None).

EXAMPLE-LIQUIDITY-LAYER

[Q-01] Does the protocol ensure settlement even when there is no auction?

In the "`settle_auction_none`" function, both the base fee and the initial auction fee are transferred to the "`fee_recipient`".

```
/* solana/programs/matching-engine/src/processor/auction/settle/none/mod.rs */
// Pay the `fee_recipient` the base fee and init auction fee. This ensures that the protocol
// relayer is paid for relaying slow VAAs (which requires posting the fast order VAA) that do
// not have an associated auction.
let fee = prepared_order_response
    .base_fee
    .saturating_add(prepared_order_response.init_auction_fee);
token::transfer(
    CpiContext::new_with_signer(
        token_program.to_account_info(),
        token::Transfer {
            from: prepared_custody_token.to_account_info(),
            to: fee_recipient_token.to_account_info(),
            authority: prepared_order_response.to_account_info(),
        },
        &[prepared_order_response_signer_seeds],
    ),
    fee,
)?;
```

As a result, if the caller of the settle function is not the protocol itself, they will not receive any profit, which may discourage potential independent executors from performing the settle operation. If no one initiates the settlement, users will not receive their funds.

We were wondering if the protocol ensures settlement even when there is no auction.

Resolution

The team confirmed that the protocol is supposed to ensure settlement if there is no auction.

The current design discourages anyone else from doing the protocol's job of moving funds over because settling an auction is a multi-step procedure (post the VAA, which requires rent, calling the prepare order response, and finally calling a settle none instruction).

EXAMPLE-LIQUIDITY-LAYER

[Q-02] Inconsistent update_auction_parameters caller requirement?

The comment above says "This instruction can only be called by the "owner" of the proposal".

```
/* solana/programs/matching-engine/src/lib.rs */
/// This instruction is used to enact an existing auction update proposal. It can only be
/// executed after the `slot_enact_delay` has passed. This instruction can only be called by
/// the `owner` of the proposal.
///
/// # Arguments
///
/// * `ctx` - `UpdateAuctionParameters` context.
pub fn update_auction_parameters(ctx: Context<UpdateAuctionParameters>) -> Result<()> {
    processor::update_auction_parameters(ctx)
}
```

However, the "OwnerOnlyMut" and the constraints ensure that only the owner of the matching engine can call this instruction.

```
/* solana/programs/matching-engine/src/processor/admin/update/auction_parameters.rs */

admin: OwnerOnlyMut<'info>,

#[account(
    mut,
    seeds = [
        Proposal::SEED_PREFIX,
        &proposal.id.to_be_bytes(),
    ],
    bump = proposal.bump,
    constraint = {
        require_keys_eq!(
            proposal.owner, admin.owner.key()
        );
        require!(
            proposal.slot_enacted_at.is_none(),
            MatchingEngineError::ProposalAlreadyEnacted
        );

        require!(
            Clock::get().unwrap().slot >= proposal.slot_enact_delay,
            MatchingEngineError::ProposalDelayNotExpired
        );

        match &proposal.action {
            ProposalAction::UpdateAuctionParameters { id, .. } => {
                require_eq!(
                    *id,
                    // NOTE: This value is checked in `propose_auction_parameters`.
                    admin.custodian.auction_config_id.saturating_add(1),
                    MatchingEngineError::AuctionConfigMismatch
                );
            },
            _ => return err!(ErrorCode::InstructionMissing),
        };

        true
    }
)]
proposal: Account<'info, Proposal>,
```

Is the comment outdated?

Resolution

The comment is outdated and has been updated by commit 8269dd1 in PR169.

EXAMPLE-LIQUIDITY-LAYER

[Q-03] Is the auction history function designed to be called separately?

The following functions, which record auction histories, are not invoked during the process and appear to be non-mandatory after a successful auction.

```
/* solana/programs/matching-engine/src/lib.rs */
358 | pub fn create_first_auction_history(ctx: Context<CreateFirstAuctionHistory>) -> Result<()> {
359 |     processor::create_first_auction_history(ctx)
360 | }
361 |
369 | pub fn create_new_auction_history(ctx: Context<CreateNewAuctionHistory>) -> Result<()> {
370 |     processor::create_new_auction_history(ctx)
371 | }
372 |
388 | pub fn add_auction_history_entry(ctx: Context<AddAuctionHistoryEntry>) -> Result<()> {
389 |     processor::add_auction_history_entry(ctx)
390 | }
```

Is this intentional (e.g., the protocol has a separate process that calls these history recording functions)? Otherwise, some history may be missing, which may confuse users.

In addition, the first bidder always pays for each auction account. The "`add_auction_history_entry`" function allows this bidder to reclaim some lamports by closing that account. However, when the history account is not created, all bidders who have paid the auction account fee will not be able to reclaim their lamports.

Moreover, the history account cannot currently be closed, meaning that the person who created the history account also cannot reclaim their lamports.

Resolution

The team clarified that it is not mandatory for a solver to add new auction history. This process allows offers to be more competitive to the end user (since the solver that initiated the auction would get some rent back by closing the auction account to move that data to a history account).

However, these methods have not been added to the SDK yet.

EXAMPLE-LIQUIDITY-LAYER

[Q-04] Router endpoint closure instruction?

There are currently no instructions to close a "`router_endpoint`" account. Is this intentional?

Resolution

The team clarified that this is intentional so the owner assistant cannot add a new endpoint for a given chain ID after the owner closes it.

In this protocol, an owner assistant can only add new endpoints. However, any modifications, including re-enabling an endpoint, are restricted to the owner only.

EXAMPLE-LIQUIDITY-LAYER

[Q-05] Inconsistent chain ids and CCTP domains?

The “chain” and “cctp_domain” should point to the same blockchain.

```
/* solana/programs/matching-engine/src/processor/admin/router_endpoint/add/cctp.rs */
#[account(
    init,
    payer = payer,
    token::mint = usdc,
    token::authority = router_endpoint,
    seeds = [
        crate::LOCAL_CUSTODY_TOKEN_SEED_PREFIX,
        &args.chain.to_be_bytes(),
    ],
    bump,
)]
local_custody_token: Box<Account<'info, token::TokenAccount>>,

usdc: Usdc<'info>,

/// CHECK: Seeds must be \["remote_token_messenger"\, remote_domain.to_string()] (CCTP Token
/// Messenger Minter program).
#[account(
    seeds = [
        RemoteTokenMessenger::SEED_PREFIX,
        args.cctp_domain.to_string().as_ref()
    ],
    bump,
    seeds::program = token_messenger_minter_program::id(),
)]
remote_token_messenger: Account<'info, RemoteTokenMessenger>,
```

If the “chain” and “cctp_domain” do not represent the same blockchain, the message will be sent to a different chain, preventing the user from redeeming tokens on the destination chain and resulting in a loss of user funds.

Does it make sense to store the one-to-one mappings to ensure they are consistent?

Resolution

The team clarified that the network registrations between two separate bridges (Wormhole and CCTP) do not know about other bridge network IDs.

So protocol owners should read the documentation of each bridge very carefully: Wormhole Constants Reference and CCTP Supported Domains.

Also, the owner needs to be careful about some Wormhole testnet chain IDs (e.g. Polygon mainnet is 5 but Polygon Amoy testnet is 10005), while CCTP's domains are consistent between mainnet and testnet (at least for the time being).

EXAMPLE-LIQUIDITY-LAYER

[Q-06] Does the protocol collect a partial penalty?

Bidders who win but fail to confirm within the specified time are supposed to be penalized with a fine.

According to the docs, the protocol will collect a part of the penalty:

```
If the player does not fulfill the transfer after the grace period, anyone can fulfill the transfer by
↪  invoking the Liquidity Layer hub. Suppose the user gets compensated for having to wait the extra
↪  time. For ease of this example, suppose this compensation is 20% of the original fee paid and the
↪  slasher's compensation is 10% of the original fee paid. These amounts are taken out of the player's
↪  security deposit.

1. Liquidity layer transfers 19,985 USDC plus 5 USDC bonus (25 USDC * 20%) = 19,990 USDC to the
↪  destination address and pays the player half of the security deposit (12.5 USDC) plus 15 USDC for the
↪  fee he agreed on to fulfill the transfer. Whomever invoked the Liquidity Layer to fulfill the
↪  transfer will get 10% of the security deposit (2.5 USDC) and the protocol collects the difference (25
↪  USDC - 5 USDC - 12.5 USDC - 2.5 USDC = 5 USDC).

2. When the USDC transfer from the L1 is finalized, this player can redeem the transfer and collect
↪  20,000 USDC for submitting this asset VAA.

3. Player net gains 2.5 USDC for participating (depositing 20,025 USDC, withdrawing 20,027.5 USDC).
↪  Slasher gains 2.5 USDC. Protocol gains 5 USDC.
```

However, in the implementation, the protocol does not collect any penalty.

In "`prepare_order_execution`", the penalty will be divided into two parts, "`user_reward`" will be added to "`user_amount`" and sent to the user via CCTP, "`penalty`" will remain in "`custody_token`", and sent to either "`best_offer_token`" or "`executor_token`", but won't send to the protocol, whose account is "`fee_recipient_token`", so the protocol will receive no fee at all.

```rust
/* solana/programs/matching-engine/src/processor/auction/execute_fast_order/mod.rs */
fn prepare_order_execution<'info>(
    accounts: PrepareFastExecution<'_, 'info>,
) -> Result<PreparedOrderExecution<'info>> {
        let DepositPenalty {
            penalty,
            user_reward,
        } = utils::auction::compute_deposit_penalty(
            config,
            auction_info,
            current_slot,
            additional_grace_period,
        );
```

```
          // user receive part one: user_reward
        let user_amount = auction_info
            .amount_in
            .saturating_sub(auction_info.offer_price)
            .saturating_sub(init_auction_fee)
            .saturating_add(user_reward);
          // all remaining tokens in custody_token
        let mut remaining_custodied_amount = custody_token.amount.saturating_sub(user_amount);

        if best_offer_token.key() == executor_token.key() {
            token::transfer( // all remaining tokens (including penalty) are sent to best_offer_token
                CpiContext::new_with_signer(
                    token_program.to_account_info(),
                    anchor_spl::token::Transfer {
                        from: custody_token.to_account_info(),
                        to: best_offer_token.to_account_info(),
                        authority: auction.to_account_info(),
                    },
                    &[auction_signer_seeds],
                ),
                remaining_custodied_amount,
            )?;
        } else {
            if remaining_custodied_amount > 0 {
                token::transfer( // all remaining tokens (including penalty) are sent to executor_token
                    CpiContext::new_with_signer(
                        token_program.to_account_info(),
                        anchor_spl::token::Transfer {
                            from: custody_token.to_account_info(),
                            to: executor_token.to_account_info(),
                            authority: auction.to_account_info(),
                        },
                        &[auction_signer_seeds],
                    ),
                    remaining_custodied_amount,
                )?;
            }
```

We are wondering what the intended behavior would be.

Resolution

The team clarified that the docs are outdated. The protocol doesn't collect fees from liquidations.

EXAMPLE-LIQUIDITY-LAYER

[Q-07] Operations allowed when the custodian is paused?

In the Solana "`token-router`", the "`place_market_order_cctp`" checks the "`custodian.paused`" status before proceeding.

```
/* solana/programs/token-router/src/processor/market_order/place_cctp.rs */
031 | #[account(constraint = !custodian.paused @ TokenRouterError::Paused)]
032 | custodian: CheckedCustodian<'info>,
```

However, users can still call "`prepare_market_order`", "`close_prepared_order`", and "`redeem_fill`" when it's paused.

For instance, it seems that "`prepare_market_order`" should not be allowed when paused. When users place orders in this state, their tokens will be held in the PDA account until they call "`close_prepared_order`" to withdraw.

The matching engine has a similar issue too.

Are these behaviors expected?

Resolution

The team acknowledged that they could potentially pause the redeem fill when addressing an outstanding issue.

However, it's okay not to pause the preparation of market orders since those prepared orders will be blocked. This can be viewed as a process to cache preparing market orders while outbound transfers are paused, then release batches once unpaused.

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

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™