certora

Security Assessment

Glow Finance

Glow V1

November 2024

Prepared for Blueprint-Finance

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Glow V1	https://github.com/Blueprint-Finance/glow-v1	108ca01	Solana

Project Overview

This document describes the findings of the manual review of **Glow V1**. The work was undertaken from Nov 7 to Dec 4, 2024

The following contract list is included in our scope:

```
programs/margin-pool/src/instructions/margin_refresh_position.rs
programs/margin-pool/src/instructions/close_loan.rs
programs/margin-pool/src/instructions/configure.rs
programs/margin-pool/src/instructions/withdraw.rs
programs/margin-pool/src/instructions/repay.rs
programs/margin-pool/src/instructions/margin_borrow_v2.rs
programs/margin-pool/src/instructions/deposit.rs
programs/margin-pool/src/instructions/admin/mod.rs
programs/margin-pool/src/instructions/admin/admin_transfer_loan.rs
programs/margin-pool/src/instructions/collect.rs
programs/margin-pool/src/instructions/margin_repay.rs
programs/margin-pool/src/instructions/create_pool.rs
programs/margin-pool/src/instructions/margin_borrow.rs
programs/margin-pool/src/instructions/register_loan.rs
programs/margin-pool/src/instructions.rs
programs/margin-pool/src/events.rs
programs/margin-pool/src/util.rs
programs/margin-pool/src/lib.rs
programs/margin-pool/src/state.rs
programs/airspace/src/instructions/airspace_permit_revoke.rs
```

```
programs/airspace/src/instructions/airspace_permit_issuer_revoke.rs
programs/airspace/src/instructions/airspace_set_authority.rs
programs/airspace/src/instructions/airspace_create.rs
programs/airspace/src/instructions/create_governor_id.rs
programs/airspace/src/instructions/set_governor.rs
programs/airspace/src/instructions/mod.rs
programs/airspace/src/instructions/airspace_permit_issuer_create.rs
programs/airspace/src/instructions/airspace_permit_create.rs
programs/airspace/src/events.rs
programs/airspace/src/lib.rs
programs/airspace/src/state.rs
programs/margin/src/instructions/accounting_invoke.rs
programs/margin/src/instructions/configure/configure_account_airspace.rs
programs/margin/src/instructions/configure/configure_adapter.rs
programs/margin/src/instructions/configure/mod.rs
programs/margin/src/instructions/configure/configure_permit.rs
programs/margin/src/instructions/configure/configure_token.rs
programs/margin/src/instructions/register_position.rs
programs/margin/src/instructions/verify_healthy.rs
programs/margin/src/instructions/admin/admin_transfer_position.rs
programs/margin/src/instructions/admin/mod.rs
programs/margin/src/instructions/adapter_invoke.rs
programs/margin/src/instructions/verify_unhealthy.rs
programs/margin/src/instructions/close_position.rs
programs/margin/src/instructions/liquidate_end.rs
programs/margin/src/instructions/create_account.rs
programs/margin/src/instructions/update_position_balance.rs
programs/margin/src/instructions/liquidator_invoke.rs
programs/margin/src/instructions/liquidate_begin.rs
programs/margin/src/instructions/positions/refresh_deposit_position.rs
programs/margin/src/instructions/positions/transfer_deposit.rs
programs/margin/src/instructions/positions/create_deposit_position.rs
programs/margin/src/instructions/positions/mod.rs
programs/margin/src/instructions/positions/refresh_position_config.rs
programs/margin/src/instructions/lookup_tables/create_lookup_table.rs
programs/margin/src/instructions/lookup_tables/append_to_lookup.rs
programs/margin/src/instructions/lookup_tables/mod.rs
programs/margin/src/instructions/lookup_tables/init_lookup_registry.rs
programs/margin/src/instructions/close_account.rs
programs/margin/src/instructions.rs
```

```
programs/margin/src/adapter.rs
programs/margin/src/events.rs
programs/margin/src/util.rs
programs/margin/src/lib.rs
programs/margin/src/state/config.rs
programs/margin/src/state/account.rs
programs/margin/src/state/account/positions.rs
programs/margin/src/state.rs
programs/margin/src/seeds.rs
programs/margin/src/syscall.rs
programs/metadata/src/lib.rs
programs/control/src/instructions/create_margin_pool.rs
programs/control/src/instructions/create_authority.rs
programs/control/src/instructions/configure_margin_pool.rs
programs/control/src/instructions.rs
programs/control/src/events.rs
programs/control/src/lib.rs
```

Protocol Overview

Within this document we audited the Glow protocol. The protocol allows users to participate in non-custodial borrowing and lending marketplaces.
The protocol contains 5 programs:

- Margin
- Margin-pool
- Airspace
- Control
- Metadata

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	3	3	2
Medium	6	6	6
Low	5	5	3
Informational	4	4	3
Total	18	18	14

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
H-01	Anybody can create an issuer, since there's no access control to issuer creation	High	Fixed
H-02	Bad debt isn't socialized, causing issues for lenders	High	Acknowledged
H-03	Liquidator can steal funds from the user without repaying anything	High	Fixed
M-01	Attacker can send lamports to metadata accounts, causing an underflow during account expansion	Medium	Fixed
M-02	Understimation of reallocation size would revert the tx in metadata.set_entry()	Medium	Fixed
M-03	Borrow doesn't write adapter results, allowing liquidator to fake repayment	Medium	Fixed
M-04	Liquidation filters out borrow and token decrease from changes calculation, allowing user to fake liquidation	Medium	Fixed
M-05	Revoking permit implementation doesn't match the code comment	Medium	Fixed
M-06	Liquidator can inflate the repayment amount by borrowing and repaying again	Medium	
L-01	Signed account aren't tracked for changes, even though they might be owned by the	Low	Fixed

	margin program		
L-02	Liquidator can DoS liquidation by repeatedly registering themselves as the liquidator and not doing anything	Low	Acknowledged
L-03	Exchange rate might be zero if only uncollected fees remain in the pool	Low	Fixed
L-04	Liquidator can repay non past-due positions	Low	Acknowledged
L-05	Liquidator can repay more than necessary to make the account healthy	Low	Fixed

High Severity Issues

H-01 Anybody can create an issuer, since there's no access control to `airspace_permit_issuer_create_handler()`		
Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: programs/airspace/src/instructions/airspace_permit_issuer_create.rs	Status: Fixed	

Description: `airspace_permit_issuer_create_handler()` creates a new issuer for the airspace, this function should be called only by the authority of the airspace.
However, there's no check that the signer account 'authority' is indeed the authority of the airspace, allowing anybody to create a new issuer to any airspace.

```
Unset
    /// The airspace authority
     authority: Signer<'info>,


     /// The airspace the regulator will grant permits for
     airspace: Account<'info, Airspace>,
```

Exploit Scenario:

- Governor creates airspace X with Bob as the authority

- Eve creates a new issuer to airspace X
- Eve can now issue permits to airspace X, without any approval from Bob

Recommendations: Add a check to verify that the authority account is the airspace's authority.

Blueprint Finance's response: Fixed in 9daa625

Fix Review: Fix confirmed

H-02 Bad debt isn't socialized, causing issues for lenders		
Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: programs/margin/src/instructions/liquidator_invoke.rs	Status: Acknowledged	

Description: The `margin` and `margin-pool` programs don't have any mechanism to socialize bad debt.
Not socializing bad debt would cause issues for lenders - since instead of an even distribution of the loss, the last lenders to withdraw would take all the loss. Motivating all the lenders to withdraw their funds as soon as possible, making the lending pool unusable.

Exploit Scenario:

- The lending pool has 100K USDC from 10 lenders
- A sudden price change causes a 10K USDC loss
- All lenders rush to withdraw their deposit as soon as possible
- The last lender to withdraw takes all the loss
- At this point nobody is going to deposit any funds to the pool, since the last lender would immediately withdraw them

Recommendations: Add a mechanism to socialize bad debt

Blueprint Finance's response: Acknowledged

H-03 Liquidator can steal funds from the user without repaying anything		
Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: programs/margin/src/instructions/liquidator_invoke.rs	Status: Fixed	

Description: During liquidation, the liquidator has full freedom over the account being liquidated. The only restriction is that the equity loss at the end of the instruction shouldn't be more than 4% of the liability.
A liquidator can use that to swap or borrow more funds (and send them to themselves) from the account without repaying anything.
Borrowing more funds wouldn't only hurt the liquidated account, but also the lenders of the pool which they borrow from - since this borrow would be a bad debt that wouldn't get repaid.
The liquidator can do this multiple times per block (as much that can fit in a single tx), each time stealing an additional 4% of the liability.

Exploit Scenario:

- Bob has an account with 100K USDC borrowed
- The position is past due after some time
- Eve registers herself as the liquidator, then each round she borrows an additional 4% from the pool
  - Assuming we can run a round (begin, invoke and end liquidation) 10 times per tx, and given a block time of 0.4 seconds on Solana that means Eve can steal ~600K USDC per minute
- The pool is emptied to the pockets of Eve, causing a permanent loss of funds both to Bob and the lenders

Recommendations: Consider restricting the actions of the liquidator - don’t allow to borrow any more funds, cap the equity loss as a percentage of the repaid amount (rather than percentage of the liability)

Blueprint Finance's response: Fixed in commit e4ecd1b which prevents an available collateral decrease

Fix Review: Fix confirmed

Medium Severity Issues

M-01 Attacker can send lamports to metadata accounts, causing an underflow during account expansion		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: programs/metadata/src/lib.rs	Status: Fixed	

Description: When `metadata.set_entry()` instruction is called, we expand the account memory if needed.
During the calculation of the lamports needed to transfer we subtract the current balance from the minimum amount needed. The assumption is that the current balance would always be less than the needed balance. However, an attacker can send lamports to the account, causing an underflow in this subtraction.

```JavaScript
            let transfer_amount = rent
                .minimum_balance(data_len)
                .checked_sub(metadata_account.lamports())
                .unwrap();
```

Exploit Scenario:

- Control authority creates an account for `TokenMetadata`
- An attacker sends 0.002 SOL to the account
- When control authority tries to call `set_entry()` to set the data, the calculation above would underflow and revert the tx

Recommendations: Use `saturating_sub()` to avoid underflow

Blueprint Finance's response: Fixed in 851f3ac

Fix Review: Fix confirmed

M-02 Under-estimation of reallocation size would revert the tx in `metadata.set_entry()`		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: programs/metadata/src/lib.rs	Status: Fixed	

Description: `metadata.set_entry()` sets the `data` array at some `offset` in the account.
If the memory of the account needs to be expanded to set the data we call `realloc()` to expand the memory.
However the new account size is wrongly calculated, we account only for the length of `data` and omit the `offset`.
This would cause the tx to revert when trying to assign the data.

```
JavaScript

        let data_len = data.len();
        let account_len = metadata_account.data_len();

.......
           metadata_account.realloc(data_len, true)?;
```

Exploit Scenario:

- Control authority creates a metadata account for `PositionTokenMetadata` and sets its data
- Control authority tries to change the `value_modifier` field of this account, and passes on the `data` and the right `offset`
- The tx reverts due the wrong reallocation

Recommendations: Account for the offset as well in `data_len`

Blueprint Finance's response: Fixed in [851f3ac]()

Fix Review: Fix confirmed

M-03 Borrow doesn't write adapter results, allowing liquidator to fake repayment		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: programs/margin-pool/src/instructions/margin_borrow.rs programs/margin-pool/src/instructions/margin_borrow_v2.rs	Status: Fixed	

Description: The margin-pool program is supposed to communicate back to the margin program every change to a position (borrow, repay, closing and opening of a new position) as return data.
However, that communication is absent when borrowing (both v1 and v2).
The margin program relies on that communication to calculate the net amount repaid and prevent the user from getting liquidation fee if they repaid and then borrowed back again. The absence of this communication would allow the liquidator to fake liquidation and get a fee for it.

Exploit Scenario:

- Bob has a position with 100K USDC debt that's unhealthy/liquidatable
- Eve registers herself as the liquidator, then repays 80K and borrows it back again
- Eve gets 4K USDC as a fee despite not repaying anything

Recommendations: Communicate the change to the margin program on borrowing

Blueprint Finance's response: Fixed in 01a74b3

Fix Review: Fix confirmed

M-04 Liquidation filters out borrow and token decrease from changes calculation, allowing user to fake liquidation		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: programs/margin/src/instructions/liquidator_invoke.rs	Status: Fixed	

Description: In the `liquidator_invoke_handler()` function the function attempts to track the total increase by summing up the total increases and subtracting the total decreases, and the total repayment by summing up the repayments and subtracting the total borrows.
However, the program filters out both the borrows and the external decreases.
This means they won't be subtracted from the increases/repayments sum, and a liquidator can fake liquidation by repaying and borrowing the same amount back again.

```
JavaScript
    let fee_relevant_changes = token_changes
        .iter()
        .filter(|c| {
            c.mint == ctx.accounts.liquidator_fee_mint.key()
                && [
                    TokenBalanceChangeCause::ExternalIncrease,
                    TokenBalanceChangeCause::Repay,
                ]
                .contains(&c.change_cause)
        })
        .collect::<Vec<_>>();
   // The fee for swaps is based on the lower of the increase in the token and the repaid
amount
    let increases: i128 = fee_relevant_changes
        .iter()
        .map(|c| {
            match c.change_cause {
                TokenBalanceChangeCause::ExternalIncrease => c.tokens as i128,
                // Offset increases
                TokenBalanceChangeCause::ExternalDecrease => c.tokens as i128 * -1,
                _ => 0,
            }
        })
        .sum();

    let repayments: i128 = fee_relevant_changes
        .iter()
        .map(|c| match c.change_cause {
            TokenBalanceChangeCause::Borrow => c.tokens as i128 * -1,
            TokenBalanceChangeCause::Repay => c.tokens as i128,
            _ => 0,
        })
        .sum();
```

Exploit Scenario:

- Bob has a position with 100K USDC debt that's unhealthy/liquidatable
- Eve registers herself as the liquidator, then repays 80K and borrows it back again
- Eve gets 4K USDC as fee despite not repaying anything

Recommendations: Don't filter out external decrease and borrow

Blueprint Finance's response: Fixed in 4673562

Fix Review: Fix confirmed

M-05 Revoking permit implementation doesn't match the code comment		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: programs/airspace/src/instructions/airspace_permit_revoke.rs	Status: Fixed	

Description: The following code comment describes who can revoke a permit and under what conditions:

```
JavaScript

    ///     * the airspace authority, always
    ///     * the regulator that issued the permit, always
    ///     * any address, if the airspace is restricted and the regulator license
    ///       has been revoked
```

However, in reality we allow only the issuer and space authority to revoke the permit:

```
JavaScript
    // The airspace authority or issuing regulator is always allowed to revoke
    if authority != airspace.authority && authority != permit.issuer {
        return err!(AirspaceErrorCode::PermissionDenied);
    }
```

And on top of that we also don't allow to revoke on *any* of the following cases:

- Airspace isn't restricted
- Issuer wasn't revoked

- Permit was issued by the airspace

```
JavaScript

  // For restricted airspaces, anyone can revoke a permit from a revoked regulator.
   // For unrestricted airspaces, permits cannot be revoked
   if !airspace.is_restricted
        || !ctx.accounts.issuer_id.data_is_empty()
        || permit.issuer == airspace.key()
    {
        return err!(AirspaceErrorCode::PermissionDenied);
    }
```

Recommendations: Change the code to match the intended design.

Blueprint Finance's response: Fixed in PR #1160

Fix Review: Fix confirmed

M-06 Liquidator can inflate the repayment amount by borrowing and repaying again		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: programs/margin/src/instructions/liquidator_invoke.rs	Status:	

Description: At the end of each liquidation invocation we cancel out borrowing and repayment, that would prevent the liquidator from inflating the repayment amount by borrowing and liquidating in the same invocation.
However, the liquidator can still do this manipulation in separate invocation - repaying in one instruction, borrowing in another and then repaying in another.

Recommendations: Ensure the total of borrowing and repayment is never negative at the end of each liquidation-invoke invocation.

Blueprint Finance's response: Fixed in a6301fe

Fix Review:

Low Severity Issues

L-01 Signed accounts aren't tracked for changes, even though they might be owned by the margin program		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: programs/margin/src/adapter.rs	Status: Fixed	

Description: For each adapter invocation the program tracks balance changes and returns them to the calling function. This is used later at liquidation.
However, the program skips accounts that are signed and doesn't track their changes, assuming they wouldn't be token accounts that are owned by the margin-account.
This assumption is mostly true, however there might be an external program that would allow the users to provide their own account as a token account, as long as the token's owner/authority is the margin account (the account would be owned by the token program, but the user would still be the signer)
In that case, we'll might have an account that's owned by the margin account and also signed at this point.

```JavaScript
    if KNOWN_EXTERNAL_PROGRAMS.contains(ctx.adapter_program.key) {
        // Track balance changes if the invocation is for known external programs
        for account_info in ctx.accounts {
            // Looking for (writable) token accounts to get their balances before the
invocation.
            // Short-circuit
            if !account_info.is_writable || account_info.is_signer || account_info.executable
{
                continue;
```

Recommendations: Don't skip tracking for signed account

Blueprint Finance's response: Fixed in 894fe44

Fix Review: Fix confirmed

L-02 Liquidator can DoS liquidation by repeatedly registering themselves as the liquidator and not doing anything		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: programs/margin/src/instructions/liquidate_begin.rs	Status: Acknowledged	

Description: In order to execute liquidation the liquidator first calls the 'begin liquidation' instruction, which registers them as the liquidator and gives them exclusivity over the liquidation - nobody else can liquidate as long as 'end liquidation' wasn't called (either by the liquidator, or after time out).
A liquidator can use this to prevent liquidation - they'll just begin liquidation and do nothing. When the liquidation is about to time out they'll simply end the liquidation and begin the liquidation again.

Exploit Scenario:

- Bob has an unhealthy account with a debt of 100K USDC
- Eve is a liquidator, she is also a friend of Bob and wants to help him to prevent the liquidation of his account
- Eve begins liquidation on Bob's account but does nothing
- When the liquidation is about to time out (after 60 seconds) Eve ends the liquidation and begins the liquidation in the same transaction
- Eve keeps doing so, preventing anybody from liquidating the account

Recommendations: Require a deposit from the liquidator, if the liquidator doesn't do anything foreclose the deposit.

Blueprint Finance's response: Acknowledged, the liquidators are whitelisted and trusted not to carry out this attack.

L-03 Exchange rate might be zero if only uncollected fees remain in the pool

Severity: **Low**	Impact: **High**	Likelihood: **Very low**
Files: programs/margin-pool/src/state.rs	Status: Fixed	

Description: In order to calculate the deposit exchange rate, we take the total value that's held in the pool and divide by the total of deposit notes.
If the total value is less than 1, we take 1 as the numerator. However this check is done before we subtract the total uncollected fees.
In case that we have only uncollected fees in the pool and their total is one or more we'll end up with a zero exchange rate, leading to loss of funds to whoever deposits.

```
JavaScript
    pub fn deposit_note_exchange_rate(&self) -> Number { // tokens per notes
        let deposit_notes = std::cmp::max(1, self.deposit_notes);
        let total_value = std::cmp::max(Number::ONE, self.total_value());
        (total_value - *self.total_uncollected_fees()) / Number::from(deposit_notes)
    }
```

Exploit Scenario:

- Total uncollected fees reach 1 token
- Depositors withdraw all of their notes so only uncollected fees remain in the pool
- Bob deposits 10K USDC into the pool
- Given a zero exchange rate, Bob receives zero notes and gets no funds in return

Recommendations: Do the `max()` check after subtracting the uncollected fees

Blueprint Finance's response: Fixed in PR #1125

Fix Review: Fix confirmed

L-04 Liquidator can repay non past-due positions		
Severity: **Low**	Impact: **High**	Likelihood: **Very low**
Files: programs/margin/src/instructions/liquidator_invoke.rs	Status: Acknowledged	

Description: An account can be liquidated once it becomes unhealthy, which is if it's either insolvent (more liability than collateral) or one of the positions is past due.
Meaning once one of the positions is past due the liquidators can liquidate it. There's no check to enforce that only the past due position would be repaid, so the liquidators can liquidate also other positions and get a fee for that.

Exploit Scenario:

- Bob has a position of 3K USDC that's past due, and a position of 100K USDT that's not past due
- Eve begins liquidation, she repays the 100K USDT position and gets 5K USDT as a liquidation fee
- Bob paid a fee for a liquidation that wasn't necessary

Recommendations: If the reason for liquidation is only past due - allow to repay only the past due position.

Blueprint Finance's response: Acknowledged, currently there's no adapter that uses the 'past due' feature, we'll likely fix this in future release.

L-05 Liquidator can repay more than necessary to make the account healthy		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: programs/margin/src/instructions/liquidator_invoke.rs	Status: Fixed	

Description:

Exploit Scenario:

- Bob has 100K USDC debt that requires 20% (20K USD) worth of collateral
- Bob's collateral value drops to 19.9K USD
- Swap and liquidation fees total 8% of the repayment amount
- Eve liquidates and repays the entire 100K USDC, getting a fee of 5K USDC
  - This isn't necessary, since repaying even only 10K USDC can get the position back to being healthy

Recommendations: Limit the amount the liquidator can repay according to the account's status.

Blueprint Finance's response: Fixed in commit e4ecd1b, this limits the amount of available collateral that can be after liquidation.

Fix Review: Fix confirmed

Informational Severity Issues

I-01. Rename parameter named ‘test’ to a meaningful name

Description: The function `configure_permit()` in the `margin` program there’s a parameter named `test`. This parameter controls whether the given permission parameter (`flag`) would be added or removed from the permit account.

Recommendation: Rename the parameter to a meaningful name

Blueprint Finance's response: Fixed in 894fe44

Fix Review: Fix confirmed

I-02. When registering position revert if position already exists

Description: When a user calls the 'register position' instruction if the position already exists the instruction completes without doing anything.
This might confuse users, in case that the token config has changed since the existing position was registered the users would assume that the new config was applied, when this isn't the case.

Recommendation: Revert if the position already exists

Blueprint Finance's response: Fixed in 2406885

Fix Review: Fix confirmed

I-03. It's best practice to assign ownership to system program when closing account

Description: In the `metadata.remove_entry()` instruction we're closing the account by zeroing the discriminator and transferring all lamports from the account to another account. This works, but it's best practice to also reallocate the account size to zero and transfer ownership of the account to the system program, the same way that Anchor handles account closure.

Blueprint Finance's response: Fixed in 851f3ac

Fix Review: Fix confirmed

I-04. Uncollected fees might lead to underflow if they're lost due to bad debt

Description: Currently bad debt socialization isn't implemented, but in case it would be the total uncollected fees might be more than the total value (i.e. even the total uncollected fees might be lost due to bad debt). This would lead to an underflow and would DoS deposits to the pool.

```
JavaScript
    pub fn deposit_note_exchange_rate(&self) -> Number { // tokens per notes
        let deposit_notes = std::cmp::max(1, self.deposit_notes);
        let total_value = std::cmp::max(Number::ONE, self.total_value());
        (total_value - *self.total_uncollected_fees()) / Number::from(deposit_notes)
    }
```

Recommendation: Pay attention to this if/when implementing bad debt socialization and ensure underflow is prevented.

Blueprint Finance's response: Would fix when implementing bad debt socialization.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.