certora

Security Assessment

Final Report

Mamo

March 2025

Prepared for Moonwell

certora

Table of contents

Project Summary **3**
- Project Scope 3
- Project Overview 3
  - Protocol Overview 3
- Findings Summary 4
- Severity Matrix 4

Detailed Findings **5**
- **Medium Severity Issues** **7**
- M-01 Use of latestAnswer() and missing staleness threshold check allows stale prices to be consumed during swap orders 7
- M-02 backEnd has some unintended authority over users strategies 8
- M-03 withdrawal might fail for valid amounts 9
- **Low Severity Issues** **10**
- L-01 Users can addStrategy for strategies that are not the latest 10
- L-02 Missing storage __gap variable in BaseStrategy 10
- L-03 Deposits and withdrawals of strategy tokens would fail for fee-on-transfer tokens 12
- L-04 Deposits fail due to use of approve() instead of forceApprove() in function depositInternal() 14
- L-05 Calls to recoverERC20() in MamoRegistry.sol would fail for ERC20 tokens that return false on transfer failure 15
- **Informational Severity Issues** **16**
- I-01. Incomplete pausing mechanism in MamoStrategyRegistry 16
- I-02. Using view methods when possible 16
- I-03. Current implementation relies on the backEnd to initialize upgraded strategies 17
- I-04. Missing splitMToken and splitVault validation in ERC20MoonwellMorphoStrategy.initialize() 17
- I-05. Consider providing strategy owners with revoke CowSwap approval functionality 18
- I-06. Unused event StrategyRemoved in MamoStrategyRegistry.sol 18
- I-07. Modifiers whenNotPaused and whenPaused are not required on external functions 19
- I-08. Missing event emission when removing token configuration 20
- I-09. Lack of incentive for solvers to execute swap orders 20
- **Gas Optimizations** **21**
- G-01. Function setSlippage() can be optimized 21
- G-02. Consider caching array length during looping to save gas 22
- G-03. Consider commenting out redundant checks from getExpectedOutFromChainlink() function 24

Disclaimer **25**

About Certora **25**

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Mamo	Repo	220a1cba1cdadda9b795ad5694a5b4ae2e8902a0	EVM

Project Overview

This document describes the Audit of **Mamo** using Certora manual code review findings. The work was undertaken from **20/03/2025** to **25/03/2025**

The following contract list is included in our scope:

```
src/SlippagePriceChecker.sol
src/MamoStrategyRegistry.sol
src/ERC1967Proxy.sol
src/ERC20MoonwellMorphoStrategy.sol
src/BaseStrategy.sol
```

Protocol Overview

The Mamo protocol enables the deployment of personal strategy contracts that split deposits between Moonwell and Morpho Vaults for optimized yield. Users maintain complete control of their strategy contracts with direct deposit/withdrawal rights, while the Mamo backend handles allocation adjustments and reward management. This creates a managed DeFi experience without surrendering asset custody.

MamoStrategyRegistry is core to security design, using a whitelist system for trusted implementations that protects against malicious upgrades. Users decide when to upgrade their strategies, not the backend. Protocol implements slippage protection for swaps and clear permission boundaries for all operations.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	3	3	3
Low	5	5	5
Informational	9	9	4
Gas Optimizations	3	3	2
Total	20	20	11

Severity Matrix

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

certora

Detailed Findings

ID	Title	Severity	Status
M-01	Use of latestAnswer() and missing staleness threshold check allows stale prices to be consumed during swap orders	Medium	Fixed and reviewed.
M-02	backEnd has some unintended authority over users strategies	Medium	Fixed and reviewed.
M-03	withdrawal might fail for valid amounts.	Medium	Fixed and reviewed.
L-01	Users can addStrategy for strategies that are not the latest.	Low	Fixed and reviewed.
L-02	Missing storage __gap variable in BaseStrategy	Low	Fixed and reviewed.
L-03	Deposits and withdrawals of strategy tokens would fail for fee-on-transfer tokens.	Low	Acknowledged. Documentation added.
L-04	Deposits fail due to use of approve() instead of forceApprove() in function depositInternal()	Low	Fixed and reviewed.
L-05	Calls to recoverERC20() in MamoRegistry.sol would fail for ERC20 tokens that return false on transfer failure	Low	Fixed and reviewed.
I-01	Incomplete pausing mechanism	Informational	Acknowledged.

	in MamoStrategyRegistry		
I-02	Using view methods when possible	Informational	Acknowledged.
I-03	Current implementation relies on the backEnd to initialize upgrade strategies.	Informational	Acknowledged.
I-04	Missing splitMToken and splitVault validation in ERC20MoonwellMorphoStrategy. initialize()	Informational	Fixed and reviewed.
I-05	Consider providing strategy owners with revoke CowSwap approval functionality	Informational	Fixed and reviewed.
I-06	Unused event StrategyRemoved in MamoStrategyRegistry.sol	Informational	Fixed and reviewed.
I-07	Modifiers whenNotPaused and whenPaused are not required on external functions	Informational	Fixed and reviewed.
I-08	Missing event emission when removing token configuration	Informational	Fixed and reviewed.
I-09	Lack of incentive for solvers to execute swap orders	Informational	Acknowledged.
G-01	Function setSlippage() can be optimized	Gas Optimization	Fixed and reviewed.
G-02	Consider caching array length during looping to save gas	Gas Optimization	Fixed and reviewed.

G-03	Consider commenting checks in getExpectedOutFromChainlink()	Gas Optimization	Acknowledged.

Medium Severity Issues

M-01 Use of latestAnswer() and missing staleness threshold check allows stale prices to be consumed during swap orders

Severity: **Medium**	Impact: **Medium**	Likelihood: **Low**
Files: src/SlippagePriceChecker.sol	Status: Fixed and reviewed.	

Description: Function getExpectedOutFromChainlink() currently uses function latestAnswer() instead of latestRoundData().

```
C/C++
int256 _latestAnswer = _priceFeed.latestAnswer();
require(_latestAnswer > 0, "Latest answer must be positive");
```

According to the Chainlink documentation, latestAnswer() is deprecated and using latestRoundData() is recommended instead, since it provides an **updatedAt** parameter, which can be used by contracts to check when the token price was last updated by Chainlink. If the price has not been updated within the heartbeat of the feed, the price is considered stale. For example, the ETH/USD price feed on mainnet has its heartbeat as 3600 seconds, after which the price should not be considered safe to consume.

Since this stale price would be used to determine the minimum strategy tokens to receive during reward token to strategy token swap orders, it is possible for it to either be inflated or deflated

than the actual price, causing either the swap to fail or the output strategy tokens received by the strategy owner to be lesser.

Recommendations: It is recommended to use latestRoundData() and use the updatedAt parameter to perform a staleness check on the price.

Customer's response: Acknowledged.

Fix Review: Fixed in 7fb2b19.

M-02 backEnd has some unintended authority over users strategies

Severity: **Medium**	Impact: **High**	Likelihood: **very Low**
Files: src/MamoStrategyRegistry.sol	Status: Fixed and reviewed.	

Description: addStrategy lacks user signature, and upgradeStrategy does not specify the intended implementation, in combination, this means that compromised backend can front run users upgradeStrategy transaction and create a malicious latest upgrade, this in turn can drain those users funds.

Recommendations:

1. add a users signature to addStrategies
2. Add a field to upgradeStrategy address implementation, so that users can specify their intended implementation to upgrade to.

Customer's response:

Recommendation 1: The strategy contracts will be deployed in one of two ways:

- Through a web frontend with a standard wallet interaction, which could request a signature from the owner's wallet

- By an AI agent through an agentic interaction with Agent Commerce Protocol or a chat interface where an Ethereum signature can't easily be generated

Because the strategy contract must support both creation methods, we chose not to require a signature on deployment, and will endeavor to make sure that all creation methods use a programmatic way to determine the owner's Ethereum address correctly. In the case of Agent Commerce Protocol, the requesting agent knows their public address. In addition, the deposit

method can only be called by the owner, further protecting the requesting owner from losing funds by inadvertently depositing to a strategy contract that isn't owned by them.

Recommendation 2: This is acknowledged and would be fixed.

Fix Review: Fixed in 7f309ff

M-03 withdrawal might fail for valid amounts.		
Severity: **Medium**	Impact: **Medium**	Likelihood: **High**
Files: src/ERC20MoonwellMorphoStrategy.sol	Status: Fixed and reviewed.	

Description: A strategic withdrawal should allow each amount that is greater than 0 but less than the total balance denoted by getTotalBalance. However, in order to maintain the ratio in the Moonwell market and Morpho vault, that amount is split based on splitMToken and splitVault.

The assumption that the morpho vault and moonwell market maintain the ratio of assets might break in one of the following scenarios:

1. It is incorrect in cases where one of the markets yields more rewards than the other.
2. Rounding after a split does not properly account for all the assets.

This would lead users to have valid withdrawal requests fail unexpectedly. Which could lead to frustration, loss of opportunity, and locked funds.

Recommendations: There are multiple avenues that might solve this problem.

1. Maintain the ratio but limit the amount. This would mean that the withdrawal would not revert, but a smaller amount would be withdrawn to the user.
2. Allow for an alternate API that would withdraw without enforcing the ratio. This would break the dual custodial architecture and would no longer provide optimal yields.
3. Allow for users to "rebalance" the assets to the optimal ratio and then withdraw to the best effort while maintaining the optimal ratio. This would require a new API development and might still not allow for Full withdrawal, but it should provide fewer assets locked in the system until a backend can rebalance.

Customer's response: Even though the proposed scenario is not exploitable, this issue is still acknowledged, Will Fix

Fix Review: Fixed in [c0365a3](c0365a3).

Low Severity Issues

L-01 Users can addStrategy for strategies that are not the latest.		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: src/MamoStrategyRegistry.sol	Status: Fixed and reviewed.	

Description: addStrategies allows the addition of strategies that have been updated in whitelistImplementation.

Recommendations: require that the implementation is the latest implementation for that strategy ID.

Customer's response: Acknowledged.

Fix Review: Fixed in 83046b0

L-02 Missing storage __gap variable in BaseStrategy

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: src/BaseStrategy.sol	Status: Fixed and reviewed.	

Description: The Contract BaseStrategy is inherited by the ERC20MoonwellMorphoStrategy contract. Both these contracts have their own state variables declared. If in a future upgrade, a new variable is introduced in the BaseStrategy parent contract, it would cause a storage collision as this new variable would be accessing the storage slot from the child contract.

Storage gaps are a convention for reserving storage slots in a base contract, allowing future versions of that contract to use up those slots without affecting the storage layout of child contracts.

More can be read about storage slot collisions and storage gaps in the Openzeppelin documentation.

Recommendations: It is recommended to add an **uint256[49] __gap;** variable at the end of the state variable declarations in BaseStrategy. In case the ERC20MoonwellMorphoStrategy and SlippagePriceChecker contracts become parent contracts in the future, it is also recommended to add these storage gap variables in the respective contracts.

Customer's response: Acknowledged.

Fix Review: Fixed in 829ff24

L-03 Deposits and withdrawals of strategy tokens would fail for fee-on-transfer tokens

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: src/ERC20MoonwellMorphoStrategy.sol	Status: Acknowledged. Documentation added.	

Description: When depositing in the ERC20MoonwellMorphoStrategy contract, we assume that post the safeTransferFrom() operation, the contract receives exactly the parameter `amount` tokens.

```
C/C++
function deposit(uint256 amount) external onlyStrategyOwner {
        require(amount > 0, "Amount must be greater than 0");

        // Transfer tokens from the owner to this contract
        token.safeTransferFrom(msg.sender, address(this), amount); // <<

        // Deposit the funds according to the current split
        depositInternal(amount);

        emit Deposit(address(token), amount);
}
```

This is not true in case the strategy token is a fee-on-transfer ERC20 token (e.g. USDT, if fee is activated on transfers). Since the contract receives fewer tokens than intended, depositing into Moonwell and Metamorpho would fail due to the lack of tokens.

Recommendations: On deposit, it is recommended to check the pre-transfer and post-transfer balance of the contract to ensure the correct amount is recorded to deposit into Moonwell and

Morpho. On withdrawals, it is recommended to introduce a fee buffer amount to allow the strategy owner to adjust for fees.

```
C/C++
require(token.balanceOf(address(this)) >= amount - feeBuffer, "Withdrawal failed:
insufficient funds");
token.safeTransfer(msg.sender, amount - feeBuffer);
```

Customer's response: Acknowledged. We do not intend to support any fee-on-transfer tokens.

Fix Review: Added documentation in b22b7bc.

L-04 Deposits fail due to use of approve() instead of forceApprove() in function depositInternal()

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: src/ERC20MoonwellMorphoStrategy.sol	Status: Fixed and reviewed.	

Description: Some ERC20 tokens return a value of false instead and do not revert. While this token behaviour conforms to the EIP-20 specification, it is not supported in ERC20MoonwellMorphoStrategy. Due to this, failed approve() transactions are assumed to be successful since the return value is not checked. Eventually, the mint() and deposit() calls to Moonwell and Morpho would revert due to the lack of approved token.

```
C/C++
// Deposit into each protocol according to the split
        if (targetMoonwell > 0) {

            token.approve(address(mToken), targetMoonwell);

            // Mint mToken with token
            require(mToken.mint(targetMoonwell) == 0, "MToken mint failed");
        }

        if (targetMetaMorpho > 0) {
            token.approve(address(metaMorphoVault), targetMetaMorpho);

            // Deposit token into MetaMorpho
            metaMorphoVault.deposit(targetMetaMorpho, address(this));
        }
```

Recommendations: It is recommended to use forceApprove() from the SafeERC20.sol Openzeppelin library.

Customer's response: Acknowledged.

Fix Review: Fixed in [cd263ae]

certora

L-05 Calls to recoverERC20() in MamoRegistry.sol would fail for ERC20 tokens that return false on transfer failure

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: src/MamoStrategyRegistry.sol	Status: Fixed and reviewed.	

Description: Some ERC20 tokens return a value of false instead and do not revert. While this token behaviour conforms to the EIP-20 specification, it is not supported in the MamoStrategyRegistry. Due to this, failed transfer() transactions are assumed to be successful since the return value is not checked.

```
C/C++
function recoverERC20(address tokenAddress, address to, uint256 amount) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        require(to != address(0), "Cannot send to zero address");
        require(amount > 0, "Amount must be greater than 0");

        IERC20(tokenAddress).transfer(to, amount);
    }
```

Recommendations: It is recommended to use safeTransfer() from Openzeppelin's SafeERC20.sol library

Customer's response: Confirmed

Fix Review: Fixed in d1a0a18.

Informational Severity Issues

I-01. Incomplete pausing mechanism in MamoStrategyRegistry

Description: MamoStrategyRegistry allows for pausing the contract, the current pausing mechanism still allows some functionality: the view methods, recoverERC20, and whitelistImplementation.
It is recommended that if a pausing mechanism is implemented to document which features are exempt from pause, either in comments or in modifiers.
It is also considered best practice to have pausing granularity. So that the code is more future-proof and has better maintainability.

Recommendation: Allow for enumeration of pausing behaviors where guardians have the ability to fully pause the entire contract.

Customer's response: Partial Acknowledged. We would like to investigate how this might be fixed at the design level at a later date.

Fix Review: Not Applicable.

I-02. Using view methods when possible

Description: in ERC20MoonwellMorphoStrategy there are a few methods that could be used as view but are not, getTotalBalance for example, it is recommended to use view methods when possible, this would make the code easier to audit and debug.

Recommendation: Use the view for methods that don't change the storage.

Note: only getTotalBalance is missing the view modifier.

Customer's response: Fix not currently possible. We use a call to Moonwell token which is not view, so this method cannot be view.

Fix Review: Not Applicable.

I-03. Current implementation relies on the backEnd to initialize upgraded strategies.

Description: The current implementation implies that the backEnd needs to allow for a push API to initialize newly upgraded strategies, there could be other design patterns that might lift this requirement and allow initialization in the same transaction as the upgrade.

Recommendation: Document the responsibility of the backEnd, or implement a push design pattern that would initialize the strategy as it's being upgraded.

Customer's response: Acknowledged, this is intended, would not fix.

Fix Review: Not applicable.

I-04. Missing splitMToken and splitVault validation in ERC20MoonwellMorphoStrategy.initialize()

Description: In the initialize() function, we perform validation for all the members of struct InitParams except for the splitMToken and splitVault members. The sum of these two variables should be validated to be 10000 as done in the updatePosition() function.

While this does not pose a threat to the strategy tokens, it would lead to some of the strategy tokens being unallocated and sitting idle in the strategy.

Recommendation: In function initialize(), add a require or if conditional to ensure the sum of splitMToken and splitVault is 10000.

Customer's response: Acknowledged.

Fix Review: Fixed in cf38263

I-05. Consider providing strategy owners with revoke CowSwap approval functionality

Description: In ERC20MoonwellMorphoStrategy.sol, we only allow strategy owners to approve the vault relayer. If the strategy owner wants to retrieve reward tokens directly using BaseStrategy.recoverERC20(), this will not be possible since bots can always frontrun to swap the reward tokens into strategy tokens.

```
C/C++
function approveCowSwap(address tokenAddress) external onlyStrategyOwner {
        // Check if the token has a configuration in the swap checker
        require(slippagePriceChecker.isRewardToken(tokenAddress), "Token not allowed");

        // Approve the vault relayer unlimited
        IERC20(tokenAddress).forceApprove(vaultRelayer, type(uint256).max);
    }
```

Recommendation: It is recommended to provide strategy owners with the ability to revoke existing approvals to the relayer.

Customer's response: Acknowledged.

Fix Review: Fixed in cf38263 by allowing strategy owners to specify approval amount when calling function approveCowSwap().

I-06. Unused event StrategyRemoved in MamoStrategyRegistry.sol

Description: Event StrategyRemoved is never utilized in MamoStrategyRegistry. The event name creates uncertainty whether strategies should be removable or not. Therefore, it is

recommended to take appropriate action by either removing the event or implementing appropriate functionality to remove strategies.

```
C/C++
/// @notice Emitted when a strategy is removed for a user
    event StrategyRemoved(address indexed user, address strategy);
```

Recommendation: If StrategyRemoved event in MamoStrategyRegistry is not intended to be used, consider removing it. If it should be used, consider implementing functionality to remove strategies and emitting the event.

Customer's response: Acknowledged.

Fix Review: Fixed in cf38263

I-07. Modifiers whenNotPaused and whenPaused are not required on external functions

Description: We do not require the whenNotPaused and whenPaused modifiers on the external pause() and unpause() functions since they're already present on the internal _pause() and _unpause() functions.

```
C/C++
function pause() external onlyRole(GUARDIAN_ROLE) whenNotPaused {
        _pause();
}

function unpause() external onlyRole(GUARDIAN_ROLE) whenPaused {
        _unpause();
}
```

Recommendation: It is recommended to remove the modifiers from the external functions.

Customer's response: Acknowledged.

Fix Review: Fixed in cf38263

I-08. Missing event emission when removing token configuration

Description: When removeConfiguration() is called by the owner to remove a token configuration, we do not emit an event to record this change. This is contrasting to the function addTokenConfiguration(), which emits the event TokenConfigured. Emitting an event can help improvise off-chain monitoring, tracking on block explorers and help frontends to track crucial changes.

```
C/C++
function removeTokenConfiguration(address token) external onlyOwner {
        require(token != address(0), "Invalid token address");
        require(tokenOracleData[token].length > 0, "Token not configured");

        // Clear any existing configurations
        delete tokenOracleData[token];

        // Reset the maxTimePriceValid for the token
        delete maxTimePriceValid[token];
    }
```

Recommendation: It is recommended to emit an event when a token configuration is deleted.

Customer's response: Acknowledged.

Fix Review: Fixed in cf38263

I-09. Lack of incentive for solvers to execute swap orders

Description: The ERC20MoonwellMorphoStrategy contract provides an infinite reward token allowance to the CowSwap team's GPv2VaultRelayer to execute swap orders. These orders are filled by the solvers allowlisted by the Cow DAO after a bond has been posted..

When an allowlisted solver executes the order, function isValidSignature() performs validation on the swap order parameters. One of the parameters being validated is the **feeAmount**. As we can see in the code snippet below, the issue is that there is no incentive for these solvers to execute trades due to the validation performed.

```
C/C++
require(_order.feeAmount == 0, "Fee amount must be zero");
```

As per the Cow DAO documentation, when trades are executed, the Protocol collects a fee from each trade and stores it in the settlement contract.

Recommendation: It is recommended to provide strategy owners or the backend with an option to configure the maximum fees they are ready to pay on swap orders. This would create an incentive for solvers to consider executing swap orders for user strategies.

Customer's response: Acknowledged. We can assume the CowSwap fee is always zero and the incentive applies through the slippage, meaning that both the bidder and the protocol retain a percentage of the difference between the execution price and the maximum slippage set by the user. This would not hurt the ability of the order to be executed.

Fix Review: Not fixed

Gas Optimizations

G-01. Function setSlippage() can be optimized

Description: In function setSlippage(), we do not need to declare the variable **oldSlippage** as we can simply emit the event before the **allowedSlippageInBps** state variable update.

Instead of this:

```
Unset
   function setSlippage(uint256 _newSlippageInBps) external onlyStrategyOwner {
        require(_newSlippageInBps <= SPLIT_TOTAL, "Slippage exceeds maximum");

        uint256 oldSlippage = allowedSlippageInBps;
        allowedSlippageInBps = _newSlippageInBps;

        emit SlippageUpdated(oldSlippage, _newSlippageInBps);
    }
```

Use this:

```
Unset
   function setSlippage(uint256 _newSlippageInBps) external onlyStrategyOwner {
        require(_newSlippageInBps <= SPLIT_TOTAL, "Slippage exceeds maximum");

        emit SlippageUpdated(allowedSlippageInBps, _newSlippageInBps);
        allowedSlippageInBps = _newSlippageInBps;
    }
```

Recommendation: It is recommended to implement the optimization stated above.

Customer's response: Acknowledged.

Fix Review: Fixed in cf38263

G-02. Consider caching array length during looping to save gas

Description: In function getExpectedOut() of SlippagePriceChecker.sol, as the **configs** variable is a storage reference, it would load the length of the token configuration array on each iteration.

Accessing this costs an SLOAD (100 gas) per iteration. Instead, the length can be cached to an uint256 memory variable and used in the loop. Since MLOAD operations only cost 3 gas per iteration, this would save almost 97 gas per iteration.

Instead of this:

```
C/C++
for (uint256 i = 0; i < configs.length; i++) {
            priceFeeds[i] = configs[i].chainlinkFeed;
            reverses[i] = configs[i].reverse;
}
```

Use this:

```
C/C++
uint256 configsLen = configs.length;
for (uint256 i = 0; i < configsLen; i++) {
            priceFeeds[i] = configs[i].chainlinkFeed;
            reverses[i] = configs[i].reverse;
}
```

Recommendation: It is recommended to implement the optimization stated above.

Customer's response: Acknowledged.

Fix Review: Fixed in [cf38263]

G-03. Consider commenting out redundant checks from getExpectedOutFromChainlink() function

Description: The following checks in function getExpectedOutFromChainlink() are redundant. This is because the conditions are already implicitly ensured to be true when the call to function getExpectedOutFromChainlink() arrives from function getExpectedOut().

```
C/C++
require(_priceFeedsLen > 0, "Need at least one price feed");
require(_priceFeedsLen == _reverses.length, "Price feeds and reverses must have same
length");
```

Recommendation: It is recommended to comment out the checks to ensure they do not consume unnecessary gas but still serve as a good invariant reference for future developers and security researchers.

Customer's response: Acknowledged, Fix won't be implemented to maintain code readability.

Fix Review: Not Implemented.

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.