Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

WBTC Lending Strategy

Veridise Inc.

March 4, 2024

► **Prepared For:**

TIE Finance
www.tie-finance.io

► **Prepared By:**

Benjamin Sepanski
Sorawee Porncharoenwase

► **Contact Us:** contact@veridise.com

► **Version History:**

Mar. 04, 2024	V2
Jan. 30, 2024	V1
Jan. 29, 2024	Initial Draft

© 2024 Veridise Inc. All Rights Reserved.

Contents

 © 2024 Veridise Inc.

1 Executive Summary

From Jan. 22, 2024 to Jan. 24, 2024, TIE Finance engaged Veridise to review the security of their WBTC Lending Strategy smart contracts. The review covered the developers new `lendingStrategy` contract, as well as a few minor updates throughout the rest of the repository. Compared to the previous version, which Veridise has audited previously*, the new version supports a new sub-strategy leveraging Wrapped Bitcoin (WBTC) as collateral in AAVE, while using that collateral to obtain Ethereum to reinvest in other sub-strategies. The changes also contained some minor code changes and new interfaces required to support the new strategy.

Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commit `dd2a1c9b`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The TIE Finance developers provided the source code of the WBTC Lending Strategy contracts for review. The source code appears to be original code written by the developers. It contains some documentation in the form of READMEs and documentation comments on functions and storage variables. To facilitate the Veridise auditors' understanding of the code, the WBTC Lending Strategy developers provided a document describing the intended behavior of the new sub-strategy and its core functions.

The source code contained no tests. Instead the TIE Finance developers deployed a version of the code to the Polygon mainnet, and performed a few brief interactions for testing.

Summary of issues detected. The audit uncovered 18 issues. The most severe were 2 low issues: a centralization risk (V-TIE-VUL-001) and incorrect slippage protection (V-TIE-VUL-002). The auditors also identified 10 warnings and 6 informational findings. Of the 18 issues, TIE Finance has fixed 10 issues. Of the remaining issues, including V-TIE-VUL-002, TIE Finance has acknowledged them, but opted to accept the described risks.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the WBTC Lending Strategy.

Firstly, the auditors *strongly* recommend the TIE Finance developers add unit tests via one of the popular Solidity build systems such as Hardhat or Foundry before deploying the protocol. During the audit, the Veridise team wrote their own Foundry script to test proof-of-concepts locally. Creating such a local environment manually takes resources away from the audit. Further, lack of testing opens the project up to a much wider array of bugs, and a higher likelihood of bugs introduced via changes over time.

Adding regression tests for expected behaviors, to protect against known misbehaviors, and to prevent issues found in prior audits or during development from being reintroduced would provide a much higher degree of confidence in the code. This would also provide a direct

* The previous audit report can be found on Veridise's website at `https://veridise.com/audits/`

© 2024 Veridise Inc.

financial benefit to the developers, allowing them to avoid testing via deployment to mainnet. The Veridise auditors further note that deployment to testnet should be performed before deploying contracts to the mainnet in order to test functionality without spending funds.

Next, the Veridise auditors recommend careful testing of the protocol before deploying to new ecosystems. The AAVE V2 documentation for their `PriceOracle`'s `getAssetsPrices()` function[†] is slightly unclear on the price in which currencies are denominated in non-Ethereum V2 markets. The Veridise auditors validated that, as assumed by the WBTC Lending Strategy developers, the AAVE V2 price oracle returns prices denominated in wei on both Ethereum and Polygon proof-of-stake chains. This check should be performed whenever deploying the contracts to a new EVM-based chain.

Finally, the auditors recommend that the developers follow certain common Solidity coding conventions to improve code clarity. This includes capitalizing contract names, not capitalizing (non-constant) variable names, and having contract names match the file names they are defined in.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise 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.

[†] `https://docs.aave.com/developers/v/2.0/the-core-protocol/price-oracle#getassetprice`

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
WBTC Lending Strategy	`dd2a1c9b`	Solidity	Polygon

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jan. 22 - Jan. 24, 2024	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	2	0	2
Warning-Severity Issues	10	6	10
Informational-Severity Issues	6	4	6
TOTAL	18	10	18

Table 2.4: Category Breakdown.

Name	Number
Logic Error	8
Maintainability	5
Access Control	1
Frontrunning	1
Data Validation	1
Locked Funds	1
Usability Issue	1

© 2024 Veridise Inc.

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of the updates to WBTC Lending Strategy's smart contracts. In our audit, we sought to answer questions such as:

- Is the protocol vulnerable to any common Solidity vulnerabilities (such as front-running, reentrancy, and large stakeholder attacks)?
- Are funds moved properly through the protocol?
- Can any user force the protocol into insolvency/liquidation?
- Can users remove their funds to receive the expected amount?
- Are security features like slippage protection properly implemented?
- Are interactions with the AAVE protocol and Uniswap exchange performed correctly?
- Are pricing computations performed correctly?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.

Scope. The scope of this audit is limited to the following files provided by the WBTC Lending Strategy developers, which contains the changes to the smart contracts since the prior Veridise audit.

- `contracts/interfaces/IVault.sol`
- `contracts/subStrategies/interfaces/IAavePool.sol`
- `contracts/subStrategies/interfaces/IUniExchange.sol`
- `contracts/subStrategies/interfaces/IUniswapV3Router.sol`
- `contracts/subStrategies/aavePool/aavePoolV2.sol`
- `contracts/subStrategies/aavePool/aavePoolV3.sol`
- `contracts/subStrategies/exchange/UniExchange.sol`
- `contracts/subStrategies/lendingStrategy.sol`
- `contracts/subStrategies/operator.sol`

Methodology. Veridise auditors reviewed the reports of previous audits for WBTC Lending Strategy, inspected the provided tests, and read the WBTC Lending Strategy documentation. They then began a manual audit of the code assisted by both static analyzers and automated

© 2024 Veridise Inc.

testing. During the audit, the Veridise auditors regularly contacted the WBTC Lending Strategy developers via Telegram to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-TIE-VUL-001	Centralization Risk	Low	Acknowledged
V-TIE-VUL-002	Slippage protection computed at execution time	Low	Acknowledged
V-TIE-VUL-003	Potential risk in collectFee()	Warning	Acknowledged
V-TIE-VUL-004	Underflow issues	Warning	Fixed
V-TIE-VUL-005	Potential service denial from setExchange() and...	Warning	Acknowledged
V-TIE-VUL-006	Missing address zero-checks	Warning	Fixed
V-TIE-VUL-007	Imprecise math	Warning	Fixed
V-TIE-VUL-008	Deprecated functionality	Warning	Fixed
V-TIE-VUL-009	Deadline always in the future	Warning	Acknowledged
V-TIE-VUL-010	Unnecessary receive() function	Warning	Fixed
V-TIE-VUL-011	Deployment fails for AAVE V2	Warning	Acknowledged
V-TIE-VUL-012	Debt rounded down	Warning	Fixed
V-TIE-VUL-013	Unused event	Info	Fixed
V-TIE-VUL-014	Typos/Unclear Comments	Info	Acknowledged
V-TIE-VUL-015	Combining SetMLR and MLRUpdate	Info	Acknowledged
V-TIE-VUL-016	Inheriting interfaces	Info	Fixed
V-TIE-VUL-017	Gas saving opportunities	Info	Fixed
V-TIE-VUL-018	Safer decimal conversion	Info	Fixed

© 2024 Veridise Inc.

4.1 Detailed Description of Issues

4.1.1 V-TIE-VUL-001: Centralization Risk

Severity	Low	**Commit**	dd2a1c9
Type	Access Control	**Status**	Acknowledged
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Similar to many projects, TIE Finance's `lendingStrategy` contract declares an administrator role that is given special permissions. In particular, these administrators are given the following abilities:

- The `lendingStrategy owner` may set the `controller` (only once), `exchange`, `vault`, and `feePool` addresses.
- `lendingStrategy operators` may adjust the LTV of the strategy.
- A `UniExchange` owner can change the address of the Uniswap router.

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example,

- A malicious `owner` could shut down the contract by setting the `exchange` to a dummy address.
- A malicious `operator` could set the LTV too high, putting users funds at risk due to liquidation.
- A malicious `owner` could set the Uniswap exchange router to a contract which forwards part of the funds to the owner.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to a single account, which introduces a single point of failure.

Developer Response We will use a multi-sig for the time being and build a DAO for community use as the project becomes more mature.

4.1.2 V-TIE-VUL-002: Slippage protection computed at execution time

Severity	Low	**Commit**	dd2a1c9
Type	Frontrunning	**Status**	Acknowledged
File(s)	`contracts/subStrategies/lendingStrategy.sol`		
Location(s)	setMLR()		
Confirmed Fix At	N/A		

When setting the MLR, a slippage parameter is provided. This slippage parameter is intended to ensure the number of shares received from the `ethLeverage` vault is sufficient.

```
uint256 minShares = IVault(ethLeverage).convertToShares(wethAmt)*(magnifier-slipPage)
    /magnifier;
if(minShares>0){
    IVault(ethLeverage).deposit(wethAmt,minShares,address(this));
}
```

Snippet 4.1: Snippet from `raiseMLR()`, which is called by `setMLR()`.

However, the number of shares is computed at transaction execution time, rather than transaction submission time. Consequently, it will have no effect, since the `convertToShares()` function computes the number of anticipated shares directly from the deposited amount and current state of the `ethLeverage` contract.

Impact The slippage protections will not provide the operator the expected protection.

Recommendation When raising the MLR, provide the minimum expected number of shares as an argument set when the transaction is submitted. When lowering the MLR, provide the minimum number of expected assets as an argument.

Developer Response We currently have no way of knowing the actual amount to be traded, so we cannot enter the minimum output volume. We use aaveOracle's price and slippage to calculate the minimum output volume.

Updated Veridise Response The minimum output volume can be computed off-chain when the transaction is being submitted. `minShares` can then be passed directly as an argument to the function, rather than being computed by the transaction.

 © 2024 Veridise Inc.

4.1.3 V-TIE-VUL-003: Potential risk in collectFee()

Severity	Warning	**Commit**	dd2a1c9
Type	Logic Error	**Status**	Acknowledged
File(s)	`contracts/subStrategies/lendingStrategy.sol`, `contracts/subStrategies/ETHStrategy.sol`		
Location(s)	collectFee()		
Confirmed Fix At	N/A		

The modifier `collectFee` sets `harvested` to `true`, then executes the function body, and then sets `harvested` to `false`.

```
modifier collectFee(){
    (,uint256 mintAmount) = _calculateFee();
    harvested = true;
    if(mintAmount>0){
        IVault(vault).mint(mintAmount, feePool);
    }
    _;
    harvested = false;
    lastCollateral = IAavePool(IaavePool).getCollateralTo(address(this),address(
    depositAsset));
}
```

Snippet 4.2: The modifier `collectFee`

One possible risk for future development is when one function with the `collectFee` modifier, say `foo`, calls another with the `collectFee` modifier, say `bar`. After `bar` returns, `harvested` will be set to `false`, while `foo` still has not finished execution. If `foo` calls `totalAssets` at this point, the returned value could be incorrect. Note that currently, the described situation does not occur.

Impact There could be incorrect computation under the situation described above in the future version.

Recommendation One possibility is to save `harvested` and restore `harvested` to this saved value, rather than unconditionally setting it to `false`.

Developer Response We will avoid collectFee function re-entry.

4.1.4 V-TIE-VUL-004: Underflow issues

Severity	Warning	**Commit**	dd2a1c9
Type	Logic Error	**Status**	Fixed
File(s)	`contracts/subStrategies/lendingStrategy.sol`		
Location(s)	_realTotalAssets()		
Confirmed Fix At	2dd8a97		

The function `_realTotalAssets` returns total assets, accounting for collateral, debt, and the balance of this contract in the `ethLeverage` vault.

```
function _realTotalAssets() internal view returns (uint256) {
        (uint256 _collateral,uint256 _debt) = IAavePool(IaavePool).
    getCollateralAndDebtTo(address(this),address(depositAsset));
        return _collateral-_debt+IAavePool(IaavePool).convertAmount(weth,address(
    depositAsset),_totalETH());
}
```

Snippet 4.3: The function `_realTotalAssets`

Although it is unlikely to occur, it could be possible that `_collateral` is less than `_debt`. In that case, an underflow would occur, causing the execution to be reverted, without taking the balance into account.

Impact `deposit` and `withdraw` would be out of service under the described situation.

Recommendation Change the order of operations so that the debt subtraction occurs as the last step.

Developer Response We applied the recommendation.

 © 2024 Veridise Inc.

4.1.5 V-TIE-VUL-005: Potential service denial from setExchange() and setFlashLoanReceiver()

Severity	Warning	**Commit**	dd2a1c9
Type	Logic Error	**Status**	Acknowledged
File(s)	`contracts/subStrategies/lendingStrategy.sol`, `contracts/subStrategies/ETHStrategy.sol`		
Location(s)	setExchange(), setFlashLoanReceiver()		
Confirmed Fix At	N/A		

The function `setExchange` can be used to set a new `exchange` address.

```
function setExchange(address _exchange) external onlyOwner {
    require(_exchange != address(0), "INVALID_ADDRESS");
    if (exchange != address(0)){
        depositAsset.safeApprove(exchange,0);
        IERC20(weth).safeApprove(exchange,0);
    }
    exchange = _exchange;
    depositAsset.safeApprove(_exchange,type(uint256).max);
    IERC20(weth).safeApprove(_exchange,type(uint256).max);
    emit SetExchange(exchange);
}
```

Snippet 4.4: The `setExchange` function in `lendingStrategy.sol`

Calling `setExchange(controller); setExchange(exchange)` will set the `depositAsset`'s `approval` of the `controller` to 0, shutting down the contract.

For `ETHStrategy`, this issue could also occur when the address is Aave's or flash loan receiver's. Conversely, `setFlashLoanReceiver` could also cause a similar issue.

Impact Denial of service under the described situation. The contract will be very difficult to restore, since the only mechanism for re-approving the `controller` to spend `depositAsset` is calling `setExchange(controller)`. This leads to other denial-of-service issues, since the `Controller` contract is not a valid exchange.

Recommendation Require that `exchange != controller`.

Developer Response We will ensure that we do not set the `controller` and `exchange` addresses to be equal.

4.1.6 V-TIE-VUL-006: Missing address zero-checks

Severity	Warning	**Commit**	dd2a1c9
Type	Data Validation	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	2dd8a97		

Description The following functions take addresses as arguments, but do not validate that the addresses are non-zero:

- contracts/subStrategies/lendingStrategy.sol:
  - `constructor()`: `_weth`, `_IaavePool`, `_vault`, `_ethLeverage`, and `_feePool`
- contracts/subStrategies/exchange/UniExchange.sol
  - `constructor()`: `_weth`, `_leverSS`, and `_univ3Router`
    - By adding the above checks, the check of `univ3Router` on `swapExactInput` and `swapExactOutput` can be removed.
- contracts/subStrategies/aavePool/aavePoolV3.sol
  - `constructor()`: `_aave`, `_aaveOracle`, and `_weth`
- contracts/subStrategies/aavePool/aavePoolV2.sol
  - `constructor()`: `_aave` and `_aaveOracle`

Impact If zero is passed as the address, the contract may be unable to function correctly, wasting gas due to the incorrect deployment.

Developer Response We applied the recommendation.

 © 2024 Veridise Inc.

4.1.7 V-TIE-VUL-007: Imprecise math

Severity	Warning	**Commit**	dd2a1c9
Type	Logic Error	**Status**	Fixed
File(s)	`contracts/subStrategies/aavePool/aavePoolV2.sol`		
Location(s)	See issue description		
Confirmed Fix At	a99927c		

Description In the following locations, imprecise computations may lead to mathematical errors due to rounding.

- contracts/subStrategies/aavePool/aavePoolV2.sol:
  - The below computation performs a division before a multiplication. This may lead to rounding errors, as the result of integer division is truncated.

```
return _collateral*_available/(_available+_debt)*99/100;
```

Snippet 4.5: Snippet from `getCollateralMaxWithdraw()`

This pattern occurs in `getCollateralMaxWithdraw()` and `getCollateralMaxWithdrawTo()`.

- contracts/subStrategies/aavePool/aavePoolV3.sol: In this file, a similar pattern also occurs in `getCollateralMaxWithdrawTo()`.

Impact Users withdrawing via the `lendingStrategy` contract may receive less than the intended amount due to these rounding errors.

Developer Response We fixed the issue by removing the 99/100 factor.

Updated Veridise Response The 1% tolerance makes sense when withdrawing collateral. Rather than removing the 99/100 factor, you could fix the issue by moving the multiplication by 99 to the front of the computation, so that it does not occur after a division.

Note that this line is another location of 99/100 that needs to be repaired.

Updated Developer Response We removed the tolerance factor.

4.1.8 V-TIE-VUL-008: Deprecated functionality

Severity	Warning	**Commit**	dd2a1c9
Type	Maintainability	**Status**	Fixed
File(s)	`contracts/subStrategies/lendingStrategy.sol`, `contracts/subStrategies/ETHStrategy.sol`		
Location(s)	See issue description		
Confirmed Fix At	2dd8a97		

The `lendingStrategy` and `ETHStrategy` contracts use the AAVE deposit function to deposit collateral into the pool.

```
function _deposit(uint256 _amount) internal returns (uint256) {
    // Get Prev Deposit Amt
    uint256 prevAmt = _totalAssets();

    // Check Max Deposit
    require(prevAmt + _amount <= maxDeposit, "EXCEED_MAX_DEPOSIT");

    (uint256 col,uint256 debt) = IAavePool(IaavePool).getCollateralAndDebt(address(
    this));
    address aave = IAavePool(IaavePool).aave();

    // Deposit WBTC
    IAave(aave).deposit(address(depositAsset), _amount, address(this), 0);
```

Snippet 4.6: Snippet from the `_deposit()` function. Note that the referenced `aave` variable refers to the underlying AAVE pool, while `IaavePool` is a contract with convenient wrappers around pool functionality.

While `deposit()` is a function in both V2 and V3 pools, it was deprecated in V3 in favor of the supply function.

Impact Use of deprecated functionality may not be supported in future AAVE upgrades.

Recommendation Add deposit functionality to the `IAavePool` interface to hide the implementation differences between AAVE V2 and AAVE V3.

Developer Response We applied the recommendation.

 © 2024 Veridise Inc.

4.1.9 V-TIE-VUL-009: Deadline always in the future

Severity	Warning	**Commit**	dd2a1c9
Type	Logic Error	**Status**	Acknowledged
File(s)	`contracts/subStrategies/exchange/UniExchange.sol`		
Location(s)	swapExactInput(), swapExactOutput()		
Confirmed Fix At	N/A		

In `UniExchange.sol`, both `swapExactInput()` and `swapExactOutput()` use `block.timestamp+1000` as the deadline for swaps.

```
ISwapRouter.ExactOutputSingleParams
    memory params = ISwapRouter.ExactOutputSingleParams({
        tokenIn: _from,
        tokenOut: _to,
        fee: univ3Fee,
        recipient: leverSS,
        deadline:block.timestamp+1000,
        amountOut: _amountOut,
        amountInMaximum: _amountInMax,
        sqrtPriceLimitX96: 0
    });
```

Snippet 4.7: A snippet of code from the `swapExactOutput` function

However, `block.timestamp` is computed at the transaction execution time, not at the transaction submission time. Therefore, the deadline will never prevent a late swap from executing.

Impact A user's transaction may execute much later than they expect, exposing them to more risk of changing prices or market manipulation.

Recommendation Add a parameter for the deadline explicitly, so that on the transaction submission, we can indicate the deadline based on the submission time.

Developer Response We acknowledged the potential issue.

4.1.10 V-TIE-VUL-010: Unnecessary receive() function

Severity	Warning	**Commit**	dd2a1c9
Type	Locked Funds	**Status**	Fixed
File(s)	`contracts/subStrategies/exchange/UniExchange.sol`		
Location(s)	receive()		
Confirmed Fix At	2dd8a97		

The `UniExchange` function has no method to send funds, but does implement the `receive()` function.

Impact Funds sent to the contract will be locked.

Recommendation Remove the `receive` implementation.

Developer Response We applied the recommendation.

© 2024 Veridise Inc.

4.1.11 V-TIE-VUL-011: Deployment fails for AAVE V2

Severity	Warning	Commit	dd2a1c9
Type	Logic Error	Status	Acknowledged
File(s)	contracts/subStrategies/ETHStrategy.sol		
Location(s)	constructor()		
Confirmed Fix At	N/A		

The `constructor()` function calls `setUserEMode()` to 1.

```
constructor(
   // [VERIDISE] ....
) {
    // [VERIDISE] ...

    // Set Max Deposit as max uin256
    maxDeposit = type(uint256).max;
    address aave = IAavePool(_IaavePool).aave();
    baseAsset.safeApprove(aave, type(uint256).max);
    depositAsset.safeApprove(aave, type(uint256).max);
    IAave(aave).setUserEMode(1);
}
```

Snippet 4.8: Snippet from `constructor()`.

In the AAVE docs, they describe the efficiency mode (i.e. the parameter passed to `setUserEMode()`).

> The `RISK_ADMINS` and `POOL_ADMIN`, set by Aave Governance, can configure a maximum of 255 eMode categories, with each `EModeCategory` having following *risk management parameters*:
>
> LTV (Loan to value)
>
> Liquidation threshold
>
> Liquidation bonus
>
> A custom price oracle (optional)

The interpretation of the parameter depends on governance, and the specific pool.

Further, this function only exists for AAVE V3 pools.

Impact The deployment will fail for AAVE V2 pools. Depending on the pool, the behavior of this parameter may be different.

Recommendation Pass the efficiency mode as a parameter to the `constructor()`. Create a method in the `IAavePool` interface to wrap this behavior so that the V2 pools are still supported.

Developer Response We acknowledged the recommendation, but we fixed the issue by making the efficiency mode as a parameter to the `constructor()` of `ETHStrategy`. When the efficiency mode is 0, `setUserEMode` will not be called, avoiding the deployment failure.

Updated Veridise Response While the supplied fix does resolve the issue, having `setUserEMode()` as part of the `IAave` interface is an anti-pattern, since V2 AAVE pools are supposed to implement the interface.

It would be better to create a wrapper method in `IAavePool` which handles the version-specific logic.

 © 2024 Veridise Inc.

4.1.12 V-TIE-VUL-012: Debt rounded down

Severity	Warning	**Commit**	dd2a1c9
Type	Logic Error	**Status**	Fixed
File(s)	`contracts/subStrategies/lendingStrategy.sol`		
Location(s)	_withdraw()		
Confirmed Fix At	2dd8a97		

When withdrawing, the a portion of collateral in AAVE is repaid in proportion to the withdrawn amount. This `ethDebt` is computed as shown in the below snippet. When withdrawing an `_amount`,

```
uint256 ethDebt = (IAavePool(IaavePool).getDebt(address(this)) * _amount) /
    IAavePool(IaavePool).getCollateralTo(address(this),address(depositAsset));
```

Snippet 4.9: Snippet from `_withdraw()`

Note that this computation computes the debt in ETH, rounding down.

Impact The computed `ethDebt` will be slightly under-computed. For future deployments in which the `depositAsset` has 18 or more decimals, this may lead to severe under-payments to the AAVE protocol.

Recommendation Round up, rather than down, when computing `ethDebt`.

Developer Response We applied the recommendation.

4.1.13 V-TIE-VUL-013: Unused event

Severity	Info	**Commit**	dd2a1c9
Type	Maintainability	**Status**	Fixed
File(s)	`contracts/subStrategies/lendingStrategy.sol`		
Location(s)	event Harvest		
Confirmed Fix At	2dd8a97		

In the `lendingStrategy` contract, the `Harvest` event is unused.

```
1 event Harvest(uint256 prevTotal, uint256 newTotal);
```

Snippet 4.10: Definition of the unused event.

Impact Off-chain listeners may listen on the wrong event.

Recommendation Remove the unused event.

Developer Response We removed the event.

© 2024 Veridise Inc.

4.1.14 V-TIE-VUL-014: Typos/Unclear Comments

Severity	Info	**Commit**	dd2a1c9
Type	Maintainability	**Status**	Acknowledged
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	2dd8a97		

The following files contain minor typos or unclear comments.

- **contracts/subStrategies/lendingStrategy.sol**
  - The argument `_depoistAsset` in the `constructor`.
  - The comment "`uin256`" in the `constructor`.
  - The comment "`Reduce LTV`" for the function `reduceMlr`.
  - The return value of the `withdrawable()` function is incorrectly documented. In the below snippet, the nat-spec comment is unclear, and internal comments which indicate the function returns a `boolean` are out-of-date.

```
/**
    Check withdrawable status of required amount
 */
function withdrawable(
    uint256 _amount
) external view override returns (uint256) {
    // Get Current Deposit Amt
    uint256 total = _totalAssets();

    // If requested amt is bigger than total asset, return false
    if (_amount > total) return total;
    // Todo Have to check withdrawable amount
    else return _amount;
}
```

Snippet 4.11: Definition of `withdrawable()`.

Impact Developers or users may experience mild confusion or be slowed by the minor errors.

Recommendation Fix the typos.

Developer Response We fixed most typos. The rest is acknowledged.

Updated Veridise Response The inner comments of `withdrawable()` are still out-of-date, as the function does not return a `boolean` value.

4.1.15 V-TIE-VUL-015: Combining SetMLR and MLRUpdate

Severity	Info	**Commit**	dd2a1c9
Type	Usability Issue	**Status**	Acknowledged
File(s)	`contracts/subStrategies/lendingStrategy.sol`		
Location(s)	event SetMLR, event MLRUpdate, setMLR()		
Confirmed Fix At	N/A		

Both the `SetMLR` and `MLRUpdate` events are always emitted together when calling `setMLR`. It would be more convenient for listeners to listen to only one event.

```
if (recentMlr>_mlr){
    reduceMlr(st,e,slipPage);  // [VERIDISE]: This emits MLRUpdate
}else if(recentMlr<_mlr){
    raiseMlr(st,e,slipPage);   // [VERIDISE]: This emits MLRUpdate
}
emit SetMLR(oldMlr, _mlr);
```

Snippet 4.12: Snippet from the `setMLR()` function.

Impact Gas is wasted and extra code maintenance is required. Listeners are required to listen for multiple events.

Recommendation Combine both events into one.

Developer Response The recommendation is acknowledged.

 © 2024 Veridise Inc.

4.1.16 V-TIE-VUL-016: Inheriting interfaces

Severity	Info	**Commit**	dd2a1c9
Type	Maintainability	**Status**	Fixed
File(s)	`contracts/subStrategies/exchange/UniExchange.sol`, `contracts/subStrategies/aavePool/aavePoolV2.sol`, `contracts/subStrategies/aavePool/aavePoolV3.sol`		
Location(s)	See issue description		
Confirmed Fix At	2dd8a97		

- The contract `UniExchange` does not explicitly implement the`IUniExchange` interface.
- The contracts `aavePoolV2` and `aavePoolV3` do not explicitly implement the `IAavePool` interface.

Impact In future development, the contracts could become out of sync from their intended interfaces.

Recommendation Add `is <interface>` to the contract definitions.

Developer Response We applied the recommendation.

4.1.17 V-TIE-VUL-017: Gas saving opportunities

Severity	Info	**Commit**	dd2a1c9
Type	Maintainability	**Status**	Fixed
File(s)	`contracts/subStrategies/exchange/UniExchange.sol`, `contracts/core/Controller.sol`		
Location(s)	See issue description		
Confirmed Fix At	2dd8a97		

The following variables can be made `immutable`. This prevents the variables from being accidentally mutated in future development. It also may reduce gas costs.

- `contracts/subStrategies/exchange/UniExchange.sol`: `weth` and `leverSS`.
- `contracts/subStrategies/aavePool/aavePoolV2.sol`: `aave` and `aaveOracle`.
- `contracts/subStrategies/aavePool/aavePoolV3.sol`: `aave`, `aaveOracle`, and `weth`.
- `contracts/subStrategies/lendingStrategy.sol`: `depositAsset`, `weth`, `IaavePool` and `ethLeverage` .

Recommendation Mark these variables `immutable`.

Developer Response We made the variables immutable.

 © 2024 Veridise Inc.

4.1.18 V-TIE-VUL-018: Safer decimal conversion

Severity	Info	**Commit**	dd2a1c9
Type	Logic Error	**Status**	Fixed
File(s)	`contracts/subStrategies/aavePool/aavePoolV2.sol`, `contracts/subStrategies/aavePool/aavePoolV3.sol`		
Location(s)	convertAmount()		
Confirmed Fix At	2dd8a97		

The function `convertAmount` does a decimal conversion as follows:

```
return _amount*prices[0]*(10**decimalsOut)/(10**decimalsIn)/prices[1];
```

Snippet 4.13: A snippet from `convertAmount`

While this is a common pattern, there is a (small) risk for an overflow from the multiplication with `10**decimalsOut` (particularly when a token has a large decimal). It is possible to rewrite the above expression with an "equivalent" expression:

```
uint256 result = _amount * prices[0];
if (decimalsOut >= decimalsIn) {
    result *= 10**(decimalsOut - decimalsIn)
} else {
    result /= 10**(decimalsIn - decimalsOut)
}
result /= price[1];
```

which is a good effort to avoid the overflow.

Impact If an overflow does occur, it would revert the execution.

Recommendation Make a modification as suggested above.

Developer Response We applied the recommendation.

Glossary

AAVE Aave is an Open Source Protocol to create Non-Custodial Liquidity Markets to earn interest on supplying and borrowing assets. To learn more, visit https://aave.com . 1

AMM Automated Market Maker. 27

ERC 20 The famous Ethereum fungible token standard. See `https://eips.ethereum.org/EIPS/eip-20` to learn more. 27

flash loan A loan which must be repaid in the same transaction, typically offered at a much more affordable rate than traditional loans . 27

Foundry An Ethereum development environment which uses Solidity-native utilities to compile, test, and deploy EVM contracts. See `https://book.getfoundry.sh` to learn more . 1

front-running A vulnerability in which a malicious user takes advantage of information about a transaction while it is in the mempool. 5

Hardhat An Ethereum development environment which uses JavaScript/TypeScript utilities to compile, test, and deploy EVM contracts. See `https://hardhat.org` to learn more . 1

large stakeholder attack An attack which leverages a large amount of funds to provoke unexpected behavior in a smart contract. These attacks are often facilitated by flash loans . 5

reentrancy A vulnerability in which a smart contract hands off control flow to an unknown party while in an intermediate state, allowing the external party to take advantage of the situation. 5

smart contract A self-executing contract with the terms directly written into code. Hosted on a blockchain, it automatically enforces and executes the terms of an agreement between buyer and seller. Smart contracts are transparent, tamper-proof, and eliminate the need for intermediaries, making transactions more efficient and secure.. 1, 27

Solidity The standard high-level language used to develop smart contracts on the Ethereum blockchain. See `https://docs.soliditylang.org/en/v0.8.19/` to learn more. 1, 27

Uniswap One of the most famous deployed AMMs. See `https://uniswap.org` to learn more. 5

WBTC Wrapped Bitcoin. 1

Wrapped Bitcoin A ERC 20 token issued at a 1:1 exchange rate with Bitcoin. 1, 27

© 2024 Veridise Inc.