Cyberscope

Audit Report

DEFIWAY

August 2024

File: AaveAggregatorImpl.sol

SHA256:

501121d6699edf765ae6ce365ec8bfeee8d1a05f0de0da5334119c4b1b673aed

Audited by © cyberscope

Table of Contents

Overview

The core smart contract of DEFIWAY, AaveAggregatorImpl.sol, has undergone a comprehensive audit to address security vulnerabilities, ensure business logic integrity, and optimise performance, providing users with a secure and efficient experience.

Through the aggregator, users can deposit tokens into an Aave pool. In return, the aggregator receives and holds aTokens on behalf of the users. The aggregator then tracks each user's share in the aToken pool and the rewards they accrue. Users can claim their proportional distribution of aTokens at any time, with the optionality to withdraw either a partial or the full amount of their initial deposit, ensuring both control and accessibility over their assets.

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Testing Deploy	https://testnet.bscscan.com/address/0x92ecc4ae993039a63e7cb8477c7f5f4f7b4a37fc

Audit Updates

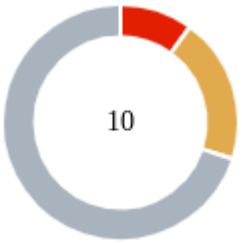
Initial Audit	08 Aug 2024

Source Files

Filename	SHA256
contracts/UUPSUpgradeAuthority.sol	7b1eea4db337e6c2e6930c844f262bc4b0108bb93f1953cfe8ad08a36ae5fcc1
contracts/AaveAggregatorImpl.sol	501121d6699edf765ae6ce365ec8bfeee8d1a05f0de0da5334119c4b1b673aed
contracts/interfaces/IAToken.sol	24d787b4352df0da369263c518457c8f3c00879f5bce2e50093393f2534ceb60
contracts/interfaces/IAPool.sol	9f9a3e5afd151f4c099cf5b7c98ab14830be31980cde2231bacba61bb23c8b93
@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol	f29ba5093b0936698c148c9bbe6d0a8b97aaf78d76c5db61741b85104eea28ce
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	a08e16324da33a9d666dc07a22ae58031c242a3869f6808e55b4b82fc70cb209
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	da01ed1f405af955a840cdbf698547faf0d607b49e338cdcad47ad615159d82f
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	a8b7eafa0fdc7cb5a644c8c61a8e4c51e031d5e1e6f268f72dbe18b768ead56e
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	9247b9ad7939d23990dbdc9274917c3762ffb37e5137ef7bbfcc2e2fba1b8dd2
@openzeppelin/contracts/utils/StorageSlot.sol	b4a5fb7ab93bfeda06509eafbd5f71fde0e0de84b6d9129553bd535a42166c15
@openzeppelin/contracts/utils/Address.sol	b3710b1712637eb8c0df81912da3450da6ff67b0b3ed18146b033ed15b1aa3b9

@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/proxy/beacon/IBeacon.sol	422eabc0e645e24c3a52898f6255b349323b013544a3ebdc4b2d3f7fc5bb7e9e
@openzeppelin/contracts/proxy/ERC1967/ERC1967Utils.sol	8850e97f15234cf93d7d1828b6289aeda7fa7167b3550b2f2a9713c8e2cecc80
@openzeppelin/contracts/interfaces/draft-IERC1822.sol	7bfe4dbf903e34b1dc61da3e27c33d601dcc496dcaf989dc6d2af610e24a145f

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	7	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	PFV	Potential Front-Running Vulnerability	Unresolved
Medium	DPI	Decimals Precision Inconsistency	Unresolved
Medium	MWD	Maximum Withdrawal Deviation	Unresolved
Minor / Informative	CO	Code Optimization	Unresolved
Minor / Informative	PAV	Pool Address Validation	Unresolved
Minor / Informative	RRA	Redundant Repeated Approvals	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved
Minor / Informative	L20	Succeeded Transfer Check	Unresolved

PFV - Potential Front-Running Vulnerability

Criticality	Critical
Location	contracts/AaveAggregatorImpl.sol#L63
Status	Unresolved

Description

Frontrunning involves exploiting transaction order in blockchain networks by submitting a transaction with higher fees to execute before another pending one, allowing attackers to gain profit or manipulate outcomes. In decentralised finance applications, this can lead to financial losses or manipulation of contract functions.

In this case, the `AaveAggregatorImpl.sol` contract includes a `supply()` function, which allows users to deposit tokens. These tokens are then deposited into an Aave pool, and in return the contract receives aTokens in approximately 1:1 ratio. Each time a user supplies tokens to the aggregator, their share of the aTokens pool increases. This share is calculated based on the current balance of aTokens held by the aggregator and the total shares accumulated from previous supplies.

However, the design of this function makes it possible for an attacker to manipulate the shares of a subsequent user when they make their deposit. Specifically, an attacker who has already supplied tokens, could frontrun a legitimate user's transaction, artificially inflating the current balance of aTokens held by the aggregator just before the new user’s transaction is processed. This manipulation could result in the new user’s shares being set to zero, even though they deposited a non-zero amount of tokens. At the same time the attacker has a non-zero number of shares which can be redeemed for more tokens after the deposit.

To demonstrate this effect see the following scenario:

```
//UserA supplies 1 wei of usdc
  userShares[usdc][UserA] = 1 wei;
  totalShares = 1 wei;

//UserA transfers 100k aUSDC to the contract
  (100_000 * 10**6)
  current = aToken.balanceOf(address(this)) =
          =  100_000 * 10**6 + 1

//UserB supplies less than 100k usdc
  userShares[usdc][UserB] = total shares (1 wei) *
 (99_999 * 10**6) / ((100_000 * 10**6) + 1) = 0

//UserA calls withdrawMax() and steals the total of the pool.
```

Recommendation

The team should revise the implementation of the `supply()` function to address scenarios where malicious users may attempt to manipulate the share distribution within the pool by front-running legitimate transactions.

DPI - Decimals Precision Inconsistency

Criticality	Medium
Location	contracts/AaveAggregatorImpl.sol#L93
Status	Unresolved

Description

The decimals field of an ERC20 token specifies the number of decimal places the token uses. In this case, there is an inconsistency in how the decimals field is managed by the contract. Specifically, in the `withdraw()` function, the input amount represents the tokens to be withdrawn. The contract calculates the corresponding shares to be destroyed from the pool in exchange for these tokens. However, due to the handling of decimal precision, certain amounts may result in a calculated number of shares that is a fraction, which is then rounded down to zero. Consequently, the withdrawal proceeds for the specified amount, but the user's share balance remains unchanged.

```
function withdraw(address token, uint amount) external nonReentrant
amountNotZero(amount) {
        ...
        uint userShare = totalShares[token] * amount / (current -
profitFee);
        if (userShare > userShares[token][msg.sender]) {
            revert TooBigAmount();
        }
        totalShares[token] -= userShare;
        userShares[token][msg.sender] -= userShare;

        pool.withdraw(token, amount + profitFee, address(this));
        IERC20(token).transfer(msg.sender, amount);
        ...
}
```

This effectively allows users to remove small fractions of the pool without having their share affected.

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. In this specific case it is advised to ensure that the number of shares is non-zero before proceeding with the withdrawal.

MWD - Maximum Withdrawal Deviation

Criticality	Medium
Location	contracts/AaveAggregatorImpl.sol#L115
Status	Unresolved

Description

The `withdrawMax()` function allows users to withdraw their entire share of the pool based on their proportional ownership. However, this function does not account for rewards that may have accumulated in the aggregator, either from yield generated by AAVE or from an increase in the balance of aTokens.

```
uint amount = userShares[token][msg.sender] * (deposited[token]
- profitFee) / totalShares[token];
```

This issue specifically arises because the withdrawable amount is calculated based on the proportion of the shares on the deposited tokens according to `deposited[token]`, rather than the current total of aTokens in the contract. Contrary, the `withdraw()` function performs the same calculation based on the `current` variable therefore accounting for the actual balance of the contract.

Recommendation

The team should modify the `withdrawMax()` function to handle cases where the current balance of aTokens exceeds the total supplied assets, ensuring that the correct portion of the pool can be withdrawn by the users.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L67
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations. In this case, `userShare` is initially set to the supplied amount. However, the assigned value is soon overwritten if `totalShares[token] != 0`, which would be the state for most operations. To avoid redundancy, it is recommended to adapt this code segment.

```
uint userShare = amount;
```

```
if (totalShares[token] != 0) {
    userShare = totalShares[token] * amount / current;
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

PAV - Pool Address Validation

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L50
Status	Unresolved

Description

The contract is missing address validation in the pool address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pool address provided as an argument. The pool address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pool address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
function initialize(address _owner, IAPool _pool, address
_upgradeAuthority) initializer public {
        __Ownable_init(_owner);
        __UUPSUpgradeable_init();
        __UpgradeAuthority_init(_upgradeAuthority);
        fee = 20;
        pool = _pool;
        emit FeeSet(fee);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pool address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pool existence in the decentralized application. Prior to interacting with the pool address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by

querying the provider's contract or utilizing external libraries that provide contract verification services.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L81
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before supply operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
IERC20(token).approve(address(pool), amount);
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L44
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
IAPool _pool
address _upgradeAuthority
address _owner
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	http://contracts/UUPSUpgradeAuthority.sol#L29
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            $.slot := UPGRADE_AUTHORITY_STORAGE
        }

assembly {
            r.slot := slot
...
            r.slot := store.slot
        }

assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L3 contracts/UUPSUpgradeAuthority.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.20;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L80,101,121,150
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(token).transferFrom(msg.sender, address(this), amount)
IERC20(token).transfer(msg.sender, amount)
IERC20(token).transfer(to, amount)
```

Recommendation

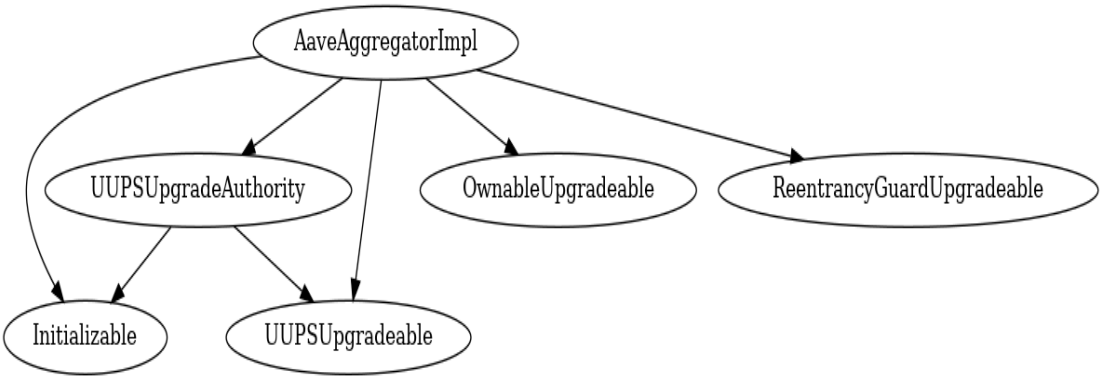
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

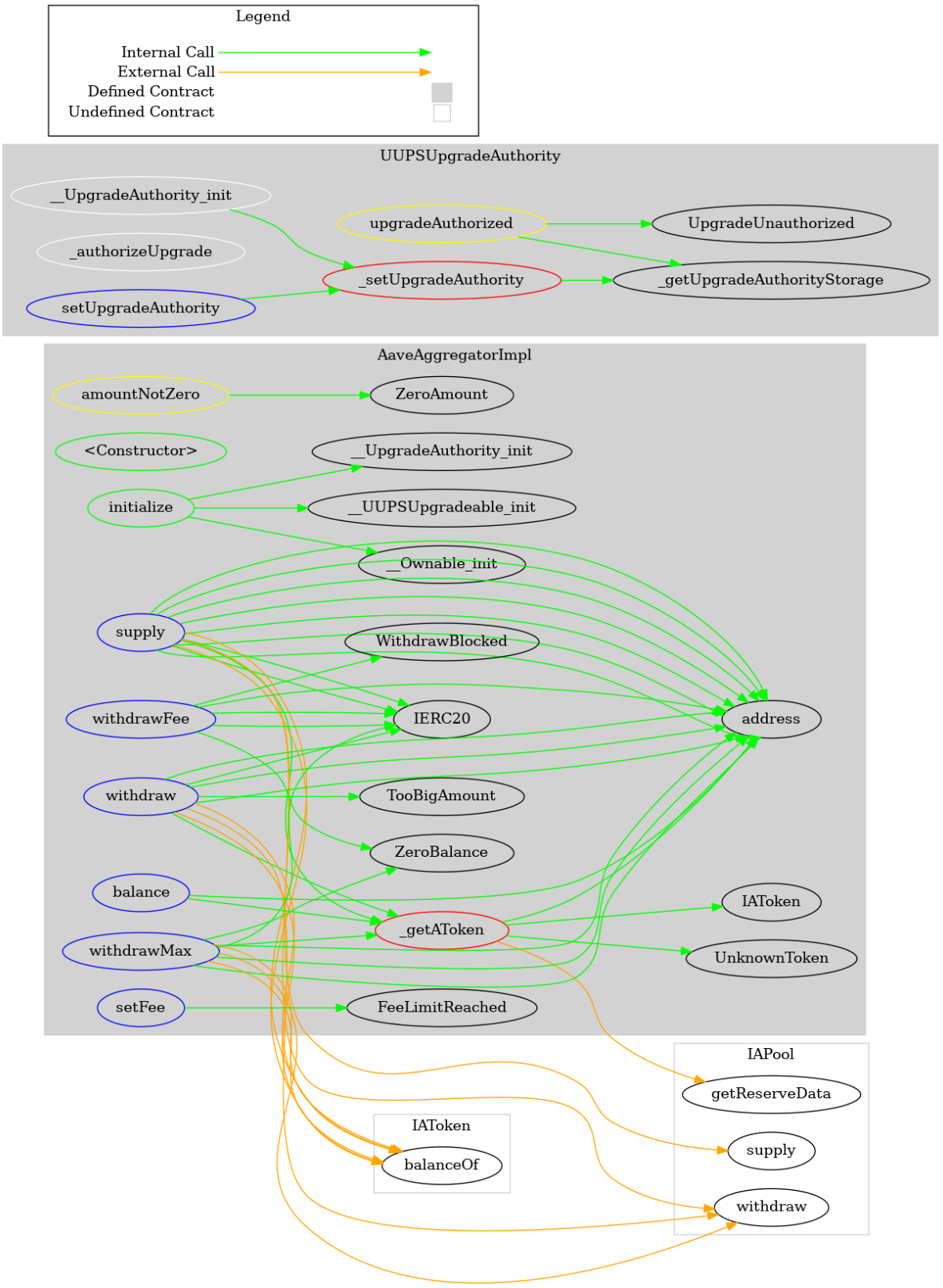
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
UUPSUpgradeAuthority	Implementation	Initializable, UUPSUpgradeable		
	_getUpgradeAuthorityStorage	Private		
	_setUpgradeAuthority	Private	✓	
	__UpgradeAuthority_init	Internal	✓	onlyInitializing
	_authorizeUpgrade	Internal	✓	upgradeAuthorized
	setUpgradeAuthority	External	✓	upgradeAuthorized
AaveAggregatorImpl	Implementation	Initializable, UUPSUpgradeable, UUPSUpgradeAuthority, OwnableUpgradeable, ReentrancyGuardUpgradeable		
		Public	✓	initializer
	initialize	Public	✓	initializer
	_getAToken	Private		
	supply	External	✓	nonReentrant amountNotZero
	withdraw	External	✓	nonReentrant amountNotZero
	withdrawMax	External	✓	nonReentrant
	balance	External		-

	setFee	External	✓	onlyOwner
	withdrawFee	External	✓	onlyOwner nonReentrant
IAToken	Interface	IERC20		
	POOL	External		-
IAPool	Interface			
	withdraw	External	✓	-
	supply	External	✓	-
	getReserveData	External		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
UUPSUpgradeAuthority
__UpgradeAuthority_init
_authorizeUpgrade
setUpgradeAuthority
upgradeAuthorized
_setUpgradeAuthority
UpgradeUnauthorized
_getUpgradeAuthorityStorage
AaveAggregatorImpl
amountNotZero
<Constructor>
initialize
supply
withdrawFee
withdraw
balance
withdrawMax
setFee
ZeroAmount
__UpgradeAuthority_init
__UUPSUpgradeable_init
__Ownable_init
WithdrawBlocked
IERC20
TooBigAmount
ZeroBalance
_getAToken
FeeLimitReached
address
IAToken
UnknownToken
IAPool
getReserveData
supply
withdraw
IAToken
balanceOf

Summary

The aggregator contract of DEFIWAY has been audited for security vulnerabilities, business concerns and overall performance. The audit identified a critical issue during the supply of new tokens, where users may suffer a front-run attack and loss of funds. Other minor issues with regards to the overall performance of the contract and its security were identified. The team is suggested to take into account these considerations to improve the security and reliability of its application.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io