Cyberscope

Audit Report

DEFIWAY

September 2024

Networks: ETH, BSC, Polygon, Arbitrum One, Base, Avalanche, Optimism

Address: 0xc69a4950Ef0e2121572225Fef341197D657A3969

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

Explorers	https://etherscan.io/address/0xc69a4950ef0e2121572225fef341197d657a3969 https://bscscan.com/address/0xc69a4950ef0e2121572225fef341197d657a3969 https://polygonscan.com/address/0xc69a4950ef0e2121572225fef341197d657a3969 https://arbiscan.io/address/0xc69a4950ef0e2121572225fef341197d657a3969 https://basescan.org/address/0xc69a4950ef0e2121572225fef341197d657a3969 https://snowtrace.io/address/0xc69a4950Ef0e2121572225Fef341197D657A3969 https://optimistic.etherscan.io/address/0xc69a4950Ef0e2121572225Fef341197D657A3969

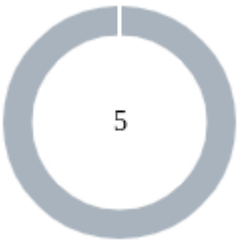
Audit Updates

Initial Audit	08 Aug 2024 https://github.com/cyberscope-io/audits/blob/main/4-defi/v1/audit.pdf
Corrected Phase 2	30 Aug 2024 https://github.com/cyberscope-io/audits/blob/main/4-defi/v2/audit.pdf
Corrected Phase 3	26 Sep 2024

Source Files

Filename	SHA256
UUPSUpgradeAuthority.sol	7b1eea4db337e6c2e6930c844f262bc4b0108bb93f1953cfe8ad08a36ae5fcc1
AaveAggregatorImpl.sol	8247589e2296d50d7201bacd4066ccac6a82a02c1b0ff46be385ce9fa2363af8
interfaces/IDecimals.sol	5c615a2b03808c3884cffeea8e301e399cfde79fe7a86e9dc1c87fb755d35667
interfaces/IAPool.sol	9f9a3e5afd151f4c099cf5b7c98ab14830be31980cde2231bacba61bb23c8b93

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	3	0	0	2

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	DPI	Decimals Precision Inconsistency	SemiResolved
	PAV	Pool Address Validation	Unresolved
	PFV	Potential Front-Running Vulnerability	SemiResolved
	L17	Usage of Solidity Assembly	Unresolved
	L19	Stable Compiler Version	Unresolved

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L93
Status	SemiResolved

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).safeTransfer(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.

Team Update

The team has updated the code to mitigate all practical instances of this vulnerability under the current conditions. However, it remains theoretically possible that this issue could arise in the future assuming different token characteristics in the pool; therefore, the finding is marked as semiResolved.

PAV - Pool Address Validation

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L49
Status	Unresolved

Description

The contract is missing proper 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. In this instance, the contract verifies if the contract at the initPool address implements the getReserveData function. However, this verification is insufficient, as any contract could implement a function with the same name.

```
function initialize(address initOwner, IAPool initPool, address
initUpgradeAuthority) initializer public {
        __Ownable_init(initOwner);
        __UUPSUpgradeable_init();
        __UpgradeAuthority_init(initUpgradeAuthority);
        fee = 20;
        pool = initPool;
        pool.getReserveData(address(0));
        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 utilising external libraries that provide contract verification services. The team is advised to check the AAVE documentation for the immutable PoolAddressesProvider contract.
https://docs.aave.com/developers/core-contracts/pooladdressesprovider

PFV - Potential Front-Running Vulnerability

Criticality	Minor / Informative
Location	contracts/AaveAggregatorImpl.sol#L69
Status	SemiResolved

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 smart contracts, 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 token
userShares[token][UserA] = 1 * 10 ** 18
totalShares = 1 * 10 ** 18
```

```
// UserB attempts to supply x * 10 **18 tokens &
// UserA frontruns them by transferring
   (1.1 * x * 10 **18) * 10 ** 18 atokens to the contract
current = 1.1x * 10**36 + 1
current = 1.04x * 10 ** 36 + 1 // minus the fee
userShares[token][UserB] =
(1 * 10 **18) * (x * 10 ** 18) / (1.04 * x * 10**36 + 1) = 0
```

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

This scenario becomes feasible only if the attacker can transfer an amount of aTokens equal to the amount supplied by UserB multiplied by 1E18. At the time of this report, AAVE supports tokens with less supply than that needed to perform this attack. In cases of tokens with larger supply this attack could become feasible.

Recommendation

The team should consider future cases where malicious users may attempt to manipulate the share distribution within the pool by front-running legitimate transactions.

Team Update

The team has updated the code to mitigate all practical instances of this vulnerability under the current conditions. However, it remains theoretically possible that this issue could arise in the future assuming different token characteristics in the pool; therefore, the finding is marked as semiResolved.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	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

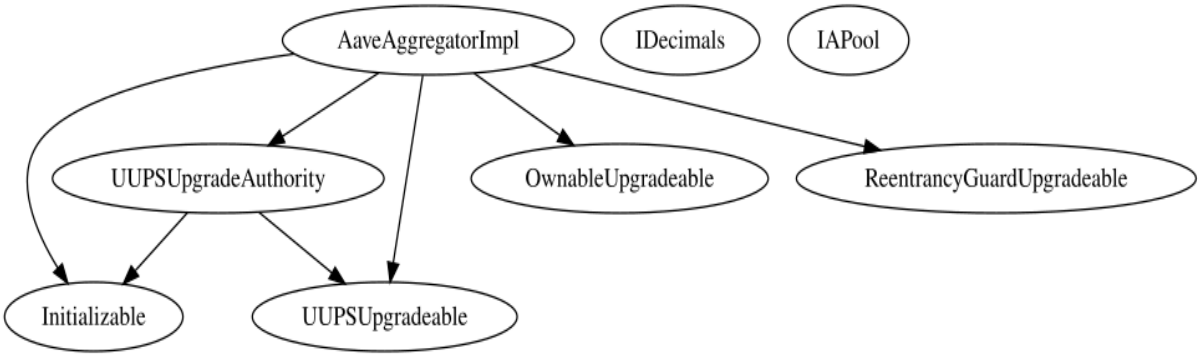
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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.

Functions Analysis

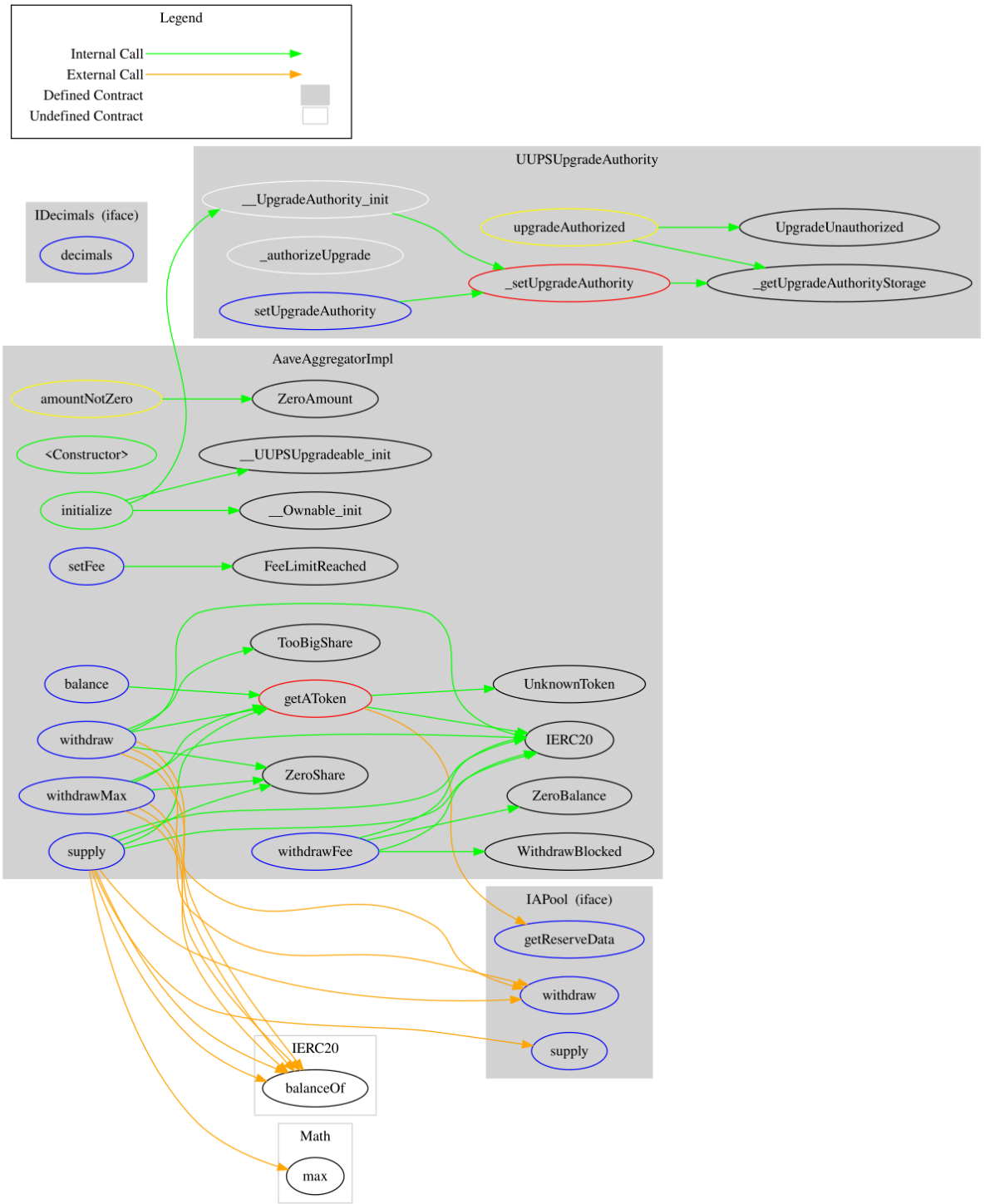
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
UUPSUpgradeAuthority	Implementation	Initializable, UUPSUpgradeable		
	_getUpgradeAuthorityStorage	Private		
	_setUpgradeAuthority	Private	✓	
	__UpgradeAuthority_init	Internal	✓	onlyInitializing
	_authorizeUpgrade	Internal	✓	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
IDecimals (iface)
decimals
AaveAggregatorImpl
amountNotZero
ZeroAmount
<Constructor>
__UUPSUpgradeable_init
initialize
__Ownable_init
setFee
FeeLimitReached
TooBigShare
balance
getAToken
UnknownToken
withdraw
IERC20
ZeroShare
withdrawMax
ZeroBalance
supply
withdrawFee
WithdrawBlocked
IAPool (iface)
getReserveData
withdraw
supply
IERC20
balanceOf
Math
max

Summary

The aggregator contract of DEFIWAY has been audited for security vulnerabilities, business concerns and overall performance. The audit identified minor issues affecting the supply and withdrawal of tokens. 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