



CERTIK

AladdinDAO

Security Assessment

April 17th, 2021

For :
AladdinDAO Protocol





Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.



Overview

Project Summary

Project Name	AladdinDAO
Description	Decentralized Finance Protocol
Platform	Ethereum; Solidity; Yul
Codebase	GitHub Repository
Commits	2a84dbdb3fc75b1ef75f7232f83e0e32cf9c3652

Audit Summary

Delivery Date	April. 17th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar. 28, 2021 - April. 17, 2021

Vulnerability Summary

Total Issues	9
● Total Critical	0
● Total Major	1
● Total Minor	1
● Total Informational	7
● Total Discussion	0



Executive Summary

This report has been prepared for AladdinDAO smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.



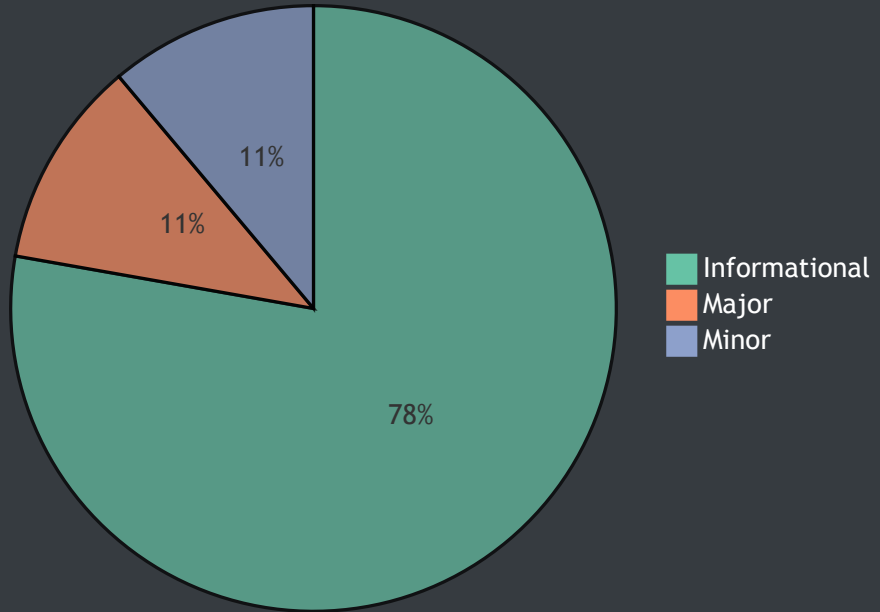
File in Scope

ID	Contract	SHA256-Checksum
DAO	DAO.sol	b8d914819016159ecc1ccb715a72b52c79c5ff1480362f802cb855d3bc33be00
TS	Treasury.sol	160451b4e029c173c80c100a934dde733a16f8fd6f08cd037edc67f4d7e92041
VT	VoteToken.sol	301981c3631eb57b5a7c14ca60fdf65bff37a9e42282fa195991afe903be40ba
BS	BaseStrategy.sol	4deec1108bbc994d0fc1a5722fbf895684dab172e68fbb4ef344507f148951e7
SCD	StrategyCompoundDAI.sol	ffa356d34cb6e0a52554a7d4792b6fbd96ccb3ce8031c79de5a217995f571
SCUC	StrategyCompoundUSDC.sol	edafa70ac8de7e0609f1f56783717c1f8371399afeed89d915014bee0668a245
SCUT	StrategyCompoundUSDT.sol	4fb702e278dec9f2b95793ad6a53524bded85b875929e4b84fe7f91729bec1ca
SCWC	StrategyCompoundWBTC.sol	3f824262ec4c2dbeb4eaae97f101d5541b43f3269fe4159aec61b80d58332971
SCWH	StrategyCompoundWETH.sol	f0a11a18a09e82d2b6bdbe91966b5f3d3193a4c63b1267ea6d7230dc6e942dfc
SCP	StrategyCurve3Pool.sol	caf71fa6def17542857c1989a8021e9668955bb725ed7c0c8cf955470117cd97
SCH	StrategyCurveHBTC.sol	695a7696656132b5181ed540e364658fd4999eec73fd2a5693571a29a9284c15
SCR	StrategyCurveRenWBTC.sol	cee333c705e2e1d00d12a48a4dd3406f5a03cf3524a887572f351cc909b17f3b
SCS	StrategyCurveSETH.sol	e148702d06bed2f11ec752bc3b93b14ab82ce179cff880fc8256c059437e13a9
SSD	StrategySushiETHDAI.sol	b1d7f9829a7949b2830abf1eed46f925dc48a6a13d63d0b06cab4ff0a84b80c6
SSEC	StrategySushiETHUSDC.sol	f74a96b5b286a0fdb73bf06fafa2aa58a6d47e578da7ed4b2db9fc06b1b6de3
SSET	StrategySushiETHUSDT.sol	4cbc24d0fd1acc23f371c4a87aeefb7d57b779a5b3e533b2ddef827ea133469c
SSEW	StrategySushiETHWBTC.sol	dfe92cda227c2c667baf4911cd24c9d1e009c7f6c41f11799813b30b3e5cdc4e
BV	BaseVault.sol	bfb599fe908dcff380e6d895dd7242202539855b5958ef347dd78bcb6d64c9a97
CL	Controller.sol	b0e4d0ab0b8830c8d1454f87c291ba378ba3d37d03fa626d9b6f129cd31f461b
MR	MultiStakingRewards.sol	fbfc6f98a0d872914354db1361d4a808796844fe0814cdb1d8a159c7f88e3ab
RD	RewardDistributor.sol	d61b1c0e8ff7094571307f241882f9323fac5f978a87d8251a9f660789998388
WE	WrappedERC20.sol	e6650b064830214793cf55d2f8f4b57da1818f66c730bedf095f116368ef247d
ALD	ALDToken.sol	dcc7afd20dfefbc6edfcf213ef6aa0433f96fd5be9cf12d9deb45d4e22f743c7
TD	TokenDistributor.sol	3e37a7ead7117c5ed97b0953e14d0323f10433393dee2d96bc082c001ab00441
TM	TokenMaster.sol	15c5c4c4e89b0c70d1b0d9aa8f54a520e9df06b1c0c0b5906e48612265941c20



Findings

Pie Chart



ID	Title	Type	Severity	Resolved
DAO-01	Proper Usage of public And external Type	Gas Optimization	● Informational	✓
VT-01	Boolean Equality	Coding Style	● Informational	Ⓜ
BS-01	Improved Checks For harvest() Operation	Gas Optimization	● Informational	✓
BV-01	A Possible Denial-of-Service Vulnerability In The deposit() Function	Logical Issue	● Minor	✓
BV-02	Unconditional Transfer	Gas Optimization	● Informational	✓
BV-03	Unlimited Call	Logical Issue	● Major	✓
BV-04	Unconditional Transfer	Gas Optimization	● Informational	✓
MR-01	Data Accuracy	Language Specific	● Informational	✓
TM-01	Missing Modifier	Logical Issue	● Informational	✓



DAO-01: Proper Usage of public And external Type

Type	Severity	Location
Gas Optimization	● Informational	DAO.sol L112

Description:

The declaration of `public` functions that are never called by the contract should be declared `external` to save gas.

For example, some functions are as follows:

```
function mint(address _to, uint256 _amount) public onlyGov {
    _mint(_to, _amount);
}

function burn(address _from, uint256 _amount) public onlyGov {
    _burn(_from, _amount);
}

function takeOut(
    address _token,
    address _destination,
    uint _amount
)
    public
    onlyGov
{
    require(_amount <= holdings(_token), "!insufficient");
    SafeERC20.safeTransfer(IERC20(_token), _destination, _amount);
}

function setGov(address _governance)
    public
    onlyGov
{
    governance = _governance;
}

function setWant(address _want)
    public
    onlyGov
{
    want = IERC20(_want);
}

function setRate(uint _rate)
    public
```




VT-01: Boolean Equality

Type	Severity	Location
Coding Style	● Informational	VoteToken.sol L39 MultiStakingRewards.sol L259 RewardDistributor.sol L91

Description:

Boolean constants can be used directly and do not need to be compared to `true` or `false`.

```
// located on VoteToken.sol
require(isMinter[msg.sender] == true, "!minter");

// located on MultiStakingRewards.sol
require(pool.isActive == false, "Cannot withdraw active reward token");

//located on RewardDistributor.sol
require(fundManager[msg.sender] == true, "!manager");
```

Recommendation:

Consider removing the equality to the boolean constant. An example revision is shown below:

```
// located on VoteToken.sol
require(isMinter[msg.sender], "!minter");

// located on MultiStakingRewards.sol
require(!pool.isActive, "Cannot withdraw active reward token");

// located on RewardDistributor.sol
require(fundManager[msg.sender], "!manager");
```

Alleviation:

No alleviation.



BS-01: Improved Checks For harvest() Operation

Type	Severity	Location
Gas Optimization	● Informational	BaseStrategy.sol L94

Description:

When the user calls the `harvest()` function, if the variable `_balance` is zero, the caller is not rewarded, and gas is consumed.

```
function harvest() external {
    _claimReward();

    uint _balance = IERC20(reward).balanceOf(address(this));

    uint256 _fee = _balance.mul(performanceFee).div(max);
    IERC20(reward).safeTransfer(strategist, _fee);

    address _vault = IController(controller).vaults(address(this));
    require(_vault != address(0), "!vault"); // additional protection so we don't burn
the funds
    IERC20(reward).safeTransfer(_vault, _balance.sub(_fee));
}
```

Recommendation:

Adding restrictions. An example revision is shown below:

```
function harvest() external {
    _claimReward();

    uint _balance = IERC20(reward).balanceOf(address(this));
    require(_balance > 0, "!_balance");
    uint256 _fee = _balance.mul(performanceFee).div(max);
    if(_fee > 0){
        IERC20(reward).safeTransfer(strategist, _fee);
    }

    address _vault = IController(controller).vaults(address(this));
    require(_vault != address(0), "!vault"); // additional protection so we don't burn
the funds
    IERC20(reward).safeTransfer(_vault, _balance.sub(_fee));
}
```




BV-01: A Possible Denial-of-Service Vulnerability In The deposit() Function

Type	Severity	Location
Logical Issue	● Minor	BaseVault.sol L90

Description:

Consider the scenario: Before user first calls `deposit` function, Eve transfers DAI token to StrategyCompoundDAI contract address. Subsequently, Bob uses `deposit` function to deposit DAI token, and the variable `shares` will be zero.

```
function balance() public view returns (uint) {
    return token.balanceOf(address(this))
        .add(IController(controller).balanceOf(address(this)));
}
.....
function deposit(uint _amount) external {
    .....
    uint _pool = balance();
    .....
    if (_pool == 0) {
        shares = _amount;
    } else {
        shares = (_amount.mul(totalSupply())).div(_pool);
    }
    _mint(msg.sender, shares);
    emit Deposit(msg.sender, _amount);
}
```

Recommendation:

Using `totalSupply() == 0` instead of `_pool == 0`. An example revision is shown below:

```
function deposit(uint _amount) external {
    .....
    uint _pool = balance();
    .....
    if (totalSupply() == 0) {
        shares = _amount;
    } else {
        shares = (_amount.mul(totalSupply())).div(_pool);
    }
    _mint(msg.sender, shares);
    emit Deposit(msg.sender, _amount);
}
```




BV-02: Unconditional Transfer

Type	Severity	Location
Coding Style	● Informational	BaseVault.sol L155

Description:

When the variable `keeperFee` is zero, the `safeTransfer` operation is not required. If so, it will consume additional gas.

```
function farm() public {
    .....
    uint keeperFee = _bal.mul(farmKeeperFeeMin).div(MAX);
    token.safeTransfer(msg.sender, keeperFee);
    .....
}
```

Recommendation:

Add extra condition, an example revision is shown below:

```
function farm() public {
    .....
    uint keeperFee = _bal.mul(farmKeeperFeeMin).div(MAX);
    if(keeperFee > 0){
        token.safeTransfer(msg.sender, keeperFee);
    }
    .....
}
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

[460521a40359c6e64c5f9c894dbe5b696f4019b7](#)



BV-03: Unlimited Call

Type	Severity	Location
Logical Issue	● Major	BaseVault.sol

Description:

Considering that `farm` function can be called by anyone without restriction, the possibility of malicious arbitrage exists.

```
function farm() public {
    uint _bal = available();

    uint keeperFee = _bal.mul(farmKeeperFeeMin).div(MAX);
    token.safeTransfer(msg.sender, keeperFee);

    uint amountLessFee = _bal.sub(keeperFee);
    token.safeTransfer(controller, amountLessFee);
    IController(controller).farm(address(this), amountLessFee);

    emit Farm(msg.sender, keeperFee, amountLessFee);
}
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

[5a949ce9a211df225d4573d9813a148c6f468af3](#)



BV-04: Unconditional Transfer

Type	Severity	Location
Gas Optimization	● Informational	BaseVault.sol L173

Description:

As in the case of BV-01 above, additional gas may be consumed here as well:

```
function harvest() external onlyEOA {  
    .....  
    uint keeperFee = harvested.mul(harvestKeeperFeeMin).div(MAX);  
    rewardToken.safeTransfer(msg.sender, keeperFee);  
    .....  
}
```

Recommendation:

Similarly, an example revision is shown below:

```
function harvest() external onlyEOA {  
    .....  
    uint keeperFee = harvested.mul(harvestKeeperFeeMin).div(MAX);  
    if(keeperFee > 0){  
        rewardToken.safeTransfer(msg.sender, keeperFee);  
    }  
    .....  
}
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

[511d50508aa3c9ce9670100ae61c22fbdefa27bb](#)



MR-01: Data Accuracy

Type	Severity	Location
Language Specific	● Informational	MultiStakingRewards.sol L194

Description:

When the value of `balance` is too small, data accuracy will be lost.

```
function notifyRewardAmount(address _rewardToken, uint256 _amount) external override
onlyRewardsDistribution updateReward(_rewardToken, address(0)) {
    .....
    require(pool.rewardRate <= balance.div(pool.rewardsDuration), "Provided reward too
high");
    .....
}
```

Recommendation:

Using multiplication instead of division. An example revision is shown below:

```
function notifyRewardAmount(address _rewardToken, uint256 _amount) external override
onlyRewardsDistribution updateReward(_rewardToken, address(0)) {
    .....
    require(pool.rewardRate.mul(pool.rewardsDuration) <= balance, "Provided reward too
high");
    .....
}
```

Alleviation:

This was resolved after thorough discussions with the developer team.



TM-01: Missing Modifier

Type	Severity	Location
Logical Issue	● Informational	TokenMaster.sol L288

Description:

We need to ensure that the argument of `_pid` is valid.

```
function set(uint256 _pid, uint256 _allocPoint) public onlyOwner {
    massUpdatePools();
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
    poolInfo[_pid].allocPoint = _allocPoint;
}
```

Recommendation:

Adding modifier of `onlyValidPool`. An example revision is shown below:

```
function set(uint256 _pid, uint256 _allocPoint) public onlyValidPool(_pid) onlyOwner {
    massUpdatePools();
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
    poolInfo[_pid].allocPoint = _allocPoint;
}
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

[7840bdec094c7f7b68f7e64c190508b0d2993e62](#)

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.


Compiler Error


Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.


Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.