Security Assessment

DeepGo

Sept 3rd, 2021

Table of Contents

Summary

Overview

- Project Summary
- Audit Summary
- Vulnerability Summary
- Audit Scope

Findings

- Deep Go-01 : Centralization Risk
- Deep Go-02 : Potential Flashloan Attack
- Deep Go-03 : Potential Sandwich Attacks
- Deep Go-04 : Boolean Equality Optimization
- GPD-01 : Divide Before Multiply
- GPD-02 : Logic of Function `LPDepositRunning/GPDepositRunning`
- GPS-01 : Substitution of `require` Calls With Modifier
- GPW-01 : Lack of Pool Stage Check
- IPS-01 : Substitution of `require` Calls With Modifier
- LLD-01 : Divide Before Multiply
- LLD-02 : Optimization Of Judgment Conditions
- LLD-03 : Incorrect Logic In GP Liquidation
- LPL-01 : Logic of Function `LPDepositRunning/GPDepositRunning`
- LPS-01 : Substitution of `require` Calls With Modifier
- NPD-01 : Redundant usage of `whenNotPaused` modifier
- NPP-01 : Proper Usage of `public` And `external` Type
- NPP-02 : Lack of Input Validation
- NPS-01 : Mutability Specifiers Missing
- SLD-01 : Divide Before Multiply
- VSD-01 : Substitution of `require` Calls With Modifier

Appendix

Disclaimer

About

Summary

This report has been prepared for Hong Kong Deep Link Asset Management Limited to discover issues and vulnerabilities in the source code of the DeepGo project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing 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.

Additionally, this audit is based on a premise that all external smart contracts are implemented safely.

We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	DeepGo
Description	Powerful Crypto Accelerator
Platform	BSC, Ethereum, Polygon
Language	Solidity
Codebase	https://github.com/DeepGoLab/DeepGo-NudgePool/tree/CertikAudit
Commit	a92cc6185134d4b4b62351286d39aba24210642c d76fd1f33b4da59b3d806fc372441cfba3307bfa

Audit Summary

Delivery Date	Sept 03, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	2	0	0	1	0	1
Medium	1	0	0	0	0	1
Minor	7	0	0	1	0	6
Informational	10	0	0	0	0	10
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
BLD	logic/BaseLogic.sol	1d5504d266e47692984de60adb4854f26cee452a8131cb72870dcd02bd6c9b71
GPD	logic/GPDepositLogic.sol	bcf171224c1d59e1a94cf855c3a046d1278fc2d0f6655edff4b2e3a484cbacf6
GPW	logic/GPWithdrawLogic.sol	54bc1bf96a227ae5efa31d6967101ab46b21e2f8c84bfba23c4380bb2e56a1c4
IPL	logic/IPLogic.sol	976d3526038e98c1cd2d104187e65c086d8e082d41970782163cee332ddb5f4d
LPL	logic/LPLogic.sol	a77eafa324fd597890179602782bbaefdd16c8c9acf2f5234b90ee3592b81043
LLD	logic/LiquidationLogic.sol	afcf24145706856c9b37f208015033aa62051402da2ffa0a024a650728d7c63f
SLD	logic/StateLogic.sol	5f0d4d90bedd7f2e9c746bd07c46557c0df5b3049ce15d57ab849738ca1f06f2
VLD	logic/VaultLogic.sol	302db93f5491529991e9326196c5a3859e86655cc816e666ff23b1cd8497e22d
GPS	storage/GPStorage.sol	66177249a06dfc93c3230985c4d984154267d4f52d6b8abf8bf2c6ea27258148
IPS	storage/IPStorage.sol	a0fc03bf458a8ee3aab7b68e236f0cdf840bccb080976e83da985ffa9fc1e768
LPS	storage/LPStorage.sol	99a374becfed3667f43a98c3fec38fdafb8a1f4d759d2234a93276579ece992a
NPS	storage/NPStorage.sol	15cc9b7fbf3e996e7a35fc86f44f44c46fd312be0871d26bb05d69e1640b89d9
VSD	storage/VaultStorage.sol	8161042fafcd71a8e47ada684c1af5732282ce8d104b826a12887f2707393cf0
NPP	NPProxy.sol	43e983e839c5ba36b62e311bc02fb74d364cabdff0b9fea03c835ac8f4f8eca5
NPD	NudgePool.sol	1c3f8617826fb90f281b7be4e7d60b59861fe6a621f03bbad0575500a4586047

Understandings

Overview

NudgePool fulfills the functions of the supply-demand match, resource allocation, and risk tranche. It is composed of three types of users.

Three user types:

- Initial Provider (IP): Project teams who need further financing and exposure
- Great Participant (GP): Investors who are risk-tolerant and strive for excess return
- Limit Participant (LP): Investors who are risk-averse and seek a stable return

During the Ongoing period, IPs act as asset suppliers, who impawn project tokens and provide investment bids; GPs screen projects and make investment decisions; LPs invest to leverage GPs.

Operating process:

1. IPs set impawn project token amount to initiate NudgePool.
2. GPs are required to pay a purchase fee to join pools. Afterward, GPs' capital would be used to purchase project tokens from the DEX market, while a purchase fee will generate a Vault. Part of the Vault would be withdrawn by IPs as income. The remaining portion Vault will be assigned to LPs as investment return. The upper limit percent of the Vault for IP is 80%, and the lower limit percent of the Vault for LP is 20%, which means that IP can extract less than 80%, and at the same time, more Vault can be given to LP. LP brings more entry to provide liquidity.
3. The amount invested by LPs will be used to purchase project tokens and leverage GP.
4. When the pool stage changes from raising to running, IP negligence in market value management, the stage will change to liquidation and IP's assets will be transferred to `owner` instead of IP.
5. During the Ongoing period, GPs and LPs can enter and exit pools when the pool is not paused. In addition, users can switch roles freely.
6. As the pool came to an end, IPs would withdraw pledged tokens, while GPs would sell the acquired tokens and return LPs their principal.
7. During the ongoing period, it will trigger liquidation when the project token declines in price. If it triggers IP liquidation, all tokens pledged by IPs would be given to GPs, who could only retrieve the remaining project tokens after returning the investment made by LPs. If triggers GP liquidation, all GPs' project tokens will be swapped to base tokens in DEX to repay to Lps. GPs will get nothing. IP will get back his pledged tokens. Liquidation is performed by the backend program which is provided by the development team.

Privileged Functions

The project contains the following privileged functions. They are used to modify the contract configurations and address attributes. We grouped these functions below:

`owner`:

Contract `NPProxy.sol`:

- setUpgrade()
- executeUpgrade()
- rollback()

Contract `NudgePool.sol`:

- initialize()
- setPause()
- unPause()

Contract `NPStorage.sol`:

- setMinRatio()
- setAlpha()
- setRaiseRatio()
- setDuration()

`admin`:

Contract `GPStorage.sol/IPStorage.sol/LPStorage.sol/VaultStorage.sol`:

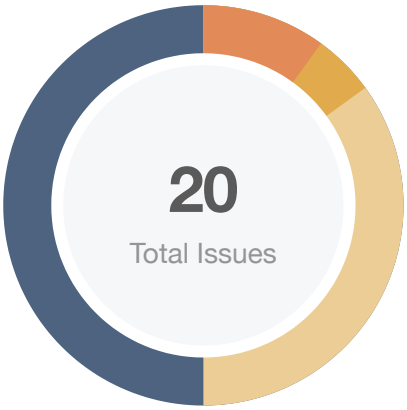
- setProxy()

Deploy Addresses

Polygon Mainnet:

- [NudgePool]: 0x52672E7a7D5427cC0631C17333c3F0ba7fdE29Ad

Findings



Critical	**0** (0.00%)
Major	**2** (10.00%)
Medium	**1** (5.00%)
Minor	**7** (35.00%)
Informational	**10** (50.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
Deep Go-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
Deep Go-02	Potential Flashloan Attack	Logical Issue	Major	Resolved
Deep Go-03	Potential Sandwich Attacks	Logical Issue	Minor	Resolved
Deep Go-04	Boolean Equality Optimization	Coding Style	Informational	Resolved
GPD-01	Divide Before Multiply	Mathematical Operations	Minor	Resolved
GPD-02	Logic of Function `LPDepositRunning/GPDepositRunning`	Logical Issue	Minor	Resolved
GPS-01	Substitution of `require` Calls With Modifier	Coding Style	Informational	Resolved
GPW-01	Lack of Pool Stage Check	Logical Issue	Informational	Resolved
IPS-01	Substitution of `require` Calls With Modifier	Coding Style	Informational	Resolved
LLD-01	Divide Before Multiply	Mathematical Operations	Minor	Resolved
LLD-02	Optimization Of Judgment Conditions	Gas Optimization	Informational	Resolved
LLD-03	Incorrect Logic In GP Liquidation	Logical Issue	Medium	Resolved
LPL-01	Logic of Function `LPDepositRunning/GPDepositRunning`	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
LPS-01	Substitution of `require` Calls With Modifier	Coding Style	Informational	Resolved
NPD-01	Redundant usage of `whenNotPaused` modifier	**Centralization / Privilege**	**Minor**	Acknowledged
NPP-01	Proper Usage of `public` And `external` Type	Gas Optimization	Informational	Resolved
NPP-02	Lack of Input Validation	Logical Issue	Informational	Resolved
NPS-01	Mutability Specifiers Missing	Gas Optimization	Informational	Resolved
SLD-01	Divide Before Multiply	Mathematical Operations	Minor	Resolved
VSD-01	Substitution of `require` Calls With Modifier	Coding Style	Informational	Resolved

Deep Go-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	Global	Acknowledged

Description

In the contract `NudgePool`, the role `owner` has the authority over the following function:

- initialize()
- setPause()
- unPause()

In the contract `NPProxy`, the role `owner` has the authority over the following function:

- setUpgrade()
- executeUpgrade()
- rollback()

In the contract `NPStorage`, the role `owner` has the authority over the following function:

- setMinRatio()
- setAlpha()
- setRaiseRatio()
- setDuration()

In these contracts `GPStorage/IPStorage/LPStorage/VaultStorage`, the role `admin` has the authority over the following function:

- setProxy()

Any compromise to these accounts may allow the hacker to manipulate the project through these functions.

Recommendation

We advise the client to carefully manage the `owner/admin` account's private key to avoid any potential risks of being hacked.

In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g.,

Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

The development team responded that as the product is in the early stage, some permissions are concentrated in the owner's hand-bound, which is more conducive to risk control than self-governance in the community. The optimization of multi-signature wallets and community voting will be reflected in subsequent product iterations, and the owner's control over the product will gradually weaken, the attributes of self-government in the community are gradually enhanced.

Deep Go-02 | Potential Flashloan Attack

Category	Severity	Location	Status
Logical Issue	Major	Global	Resolved

Description

Functions `updateMaxIPCanRaise()`, `updateGPBalance()`, `checkIPLiquidation()`, `checkGPLiquidation()`, `doIPLiquidation()`, `lendToGP()`, `reclaimFromGP()`, `updateInitIPCanRaise()` and `raisingEnd()` use library function `NPSwap.getAmountsOut()/NPSwap.getAmountIn()`, which relies on the relative token price in a single pair and is vulnerable to flashloan attacks.

Flash loans are a way to borrow large amounts of money for a certain fee. The requirement is that the loans need to be returned within the same transaction in a block. If not, the transaction will be reverted.

An attacker can use the borrowed money as the initial funds for an exploit to enlarge the profit and/or manipulate the token price in the decentralized exchanges.

We find that the used Oracle relies on price calculations that are based on-chain, meaning that they would be susceptible to flash-loan attacks by manipulating the price of given pairs to the attacker's benefit.

Recommendation

If a project requires price references, it needs to be cautious of flash loans that might manipulate token prices. To minimize the chance of this happening, we recommend the client to consider following in accordance to the project's business model.

1. Use multiple reliable on-chain price oracle sources, such as Chainlink and Uniswap.
2. Use Time-Weighted Average Price (TWAP). The TWAP represents the average price of a token over a specified time frame. If an attacker manipulates the price in one block, it will not reflect too much on the average price. Here's an example
3. If the business model allows, restrict the function caller to be a non-contract/EOA address.
4. Flash loans only allow users to borrow money within a single transaction. If the contract allows use cases, force critical transactions to span at least two blocks.

Alleviation

The development team has added account check for functions `checkRaisingEnd()`, `checkIPLiquidation()` and `checkGPLiquidation()` in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`. The other interfaces do not theoretically bring arbitrage

opportunities to attackers. `updateGPBalance` interface is used to update the weight value of the GP distribution item and will not trigger the buying and selling actions. LP can use the `lendToGP` and `reclaimFromGP` interfaces to change the head of the GP after manipulating the currency price, creating fluctuations, and causing liquidation. However, GP can withdraw from the arbitrage hedge while LP is pulling the market; on the contrary, when the LP is smashing the market, it may cause the original pledge of the personal pledge to be unable to be redeemed. Therefore, LP has no room for arbitrage. GP can use the `updateMaxIPCanRaise` interface after manipulating the currency price to change its head. However, GP's income is essentially derived from the appreciation of the currency. Therefore, GP has no arbitrage space.

Deep Go-03 | Potential Sandwich Attacks

Category	Severity	Location	Status
Logical Issue	Minor	Global	Resolved

Description

A sandwich attack may happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction being attacked) a transaction to sell the asset.

The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- `lendToGP()`
- `reclaimFromGP()`
- `raisingEnd()`
- `doIPLiquidation()`
- `doGPLiquidation()`
- `runningEnd()`
- `GPWithdrawRunning()`
- `GPDoDepositRunning()`

Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

Deep Go-04 | Boolean Equality Optimization

Category	Severity	Location	Status
Coding Style	Informational	Global	Resolved

Description

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

Recommendation

Consider removing the equality to the boolean constant as below:

```
1     require(pools[_ipt][_bst].GPM[_gp].valid, "GP Not Exist");
```

The code above is an example. Similar codes can also be modified.

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

GPD-01 | Divide Before Multiply

Category	Severity	Location	Status
Mathematical Operations	Minor	logic/GPDepositLogic.sol: 127, 130	Resolved

Description

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation

We advise the client to consider ordering multiplication before division to prevent any loss of arithmetical operation accuracy.

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

GPD-02 | Logic of Function `LPDepositRunning/GPDepositRunning`

Category	Severity	Location	Status
Logical Issue	Minor	logic/GPDepositLogic.sol: 46	Resolved

Description

If users only call function `LPDepositRunning/GPDepositRunning` and don‘t call function `LPDoDepositRunning/GPDoDepositRunning` later, how to withdraw their `_baseToken`?

Recommendation

We advise the client to add logic to withdraw `_baseToken`.

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

GPS-01 | Substitution of `require` Calls With Modifier

Category	Severity	Location	Status
Coding Style	Informational	storage/GPStorage.sol: 40, 45, 50, 55, 60, 66, 72, 78, 84, 90, 110	Resolved

Description

The `require` statements on the aforementioned lines can be substituted with a modifier to increase the legibility of the codebase.

Recommendation

We advise substituting the `require` statements on the aforementioned lines with a modifier:

```
modifier onlyProxy() {
 require(proxy == msg.sender, "Not Permit");
 _;
}
```

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

GPW-01 | Lack of Pool Stage Check

Category	Severity	Location	Status
Logical Issue	Informational	logic/GPWithdrawLogic.sol: 14	Resolved

Description

There is no validation to check whether the pool stage is `RUNNING`.

Recommendation

We advise adding validation to check this as below:

```
    function GPWithdrawRunning(
        address _ipToken,
        address _baseToken,
        uint256 _baseTokensAmount
    )
        external
        lockPool(_ipToken, _baseToken)
        returns (uint256 amount)
    {
        poolAtStage(_ipToken, _baseToken, Stages.RUNNING);
        ...
    }
```

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

IPS-01 | Substitution of `require` Calls With Modifier

Category	Severity	Location	Status
Coding Style	Informational	storage/IPStorage.sol: 55, 66, 96, 101, 106, 111, 116, 121, 126, 131, 136, 141, 146, 151, 156, 161, 166	Resolved

Description

The `require` statements on the aforementioned lines can be substituted with a modifier to increase the legibility of the codebase.

Recommendation

We advise substituting the `require` statements on the aforementioned lines with a modifier:

```
modifier onlyProxy() {
 require(proxy == msg.sender, "Not Permit");
 _;
}
```

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

LLD-01 | Divide Before Multiply

Category	Severity	Location	Status
Mathematical Operations	Minor	logic/LiquidationLogic.sol: 30, 37, 65	Resolved

Description

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation

We advise the client to consider ordering multiplication before division to prevent any loss of arithmetical operation accuracy.

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

LLD-02 | Optimization Of Judgment Conditions

Category	Severity	Location	Status
Gas Optimization	Informational	logic/LiquidationLogic.sol: 36~37	Resolved

Description

In function `LiquidationLogic.checkIPLiquidation()`, `curIPAmount.mul(price).div(inUnit)` < `IPAmount.add(curIPAmount).mul(price).div(inUnit)`, `raiseLPLossRatio` == 800000, `RATIO_FACTOR` == 1000000, so raiseLP.mul(raiseLPLossRatio).div(RATIO_FACTOR) < raiseLp. We can infer that if the condition `IPAmount.add(curIPAmount).mul(price).div(inUnit).mul(RATIO_FACTOR) <= raiseLP.mul(raiseLPLossRatio)` is met, the condition `curIPAmount.mul(price).div(inUnit) <= raiseLP` is met too. The the condition can be simplified.

Recommendation

Consider simplifying the condition as below:

```
1      if (IPAmount.add(curIPAmount).mul(price).div(inUnit).mul(RATIO_FACTOR) <=
raiseLP.mul(raiseLPLossRatio)) {
2          ...
3      } else {
4          ...
5      }
```

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

LLD-03 | Incorrect Logic In GP Liquidation

Category	Severity	Location	Status
Logical Issue	Medium	logic/LiquidationLogic.sol: 65~70	Resolved

Description

According to the comment of Line64 in `LiquidationLogic.checkGPLiquidation()`, the judgement Line65 should be equal to the judgement in `checkIPLiquidation()`, so the `ipAmount` should be `_IPS.getIPTokensAmount(_ipToken, _baseToken)`, but not `_GPS.getCurIPAmount(_ipToken, _baseToken)`.

Recommendation

Consider modifying code as below:

```
1    uint256 IPAmount = _IPS.getIPTokensAmount(_ipToken, _baseToken);
2    uint256 curIPAmount = _GPS.getCurIPAmount(_ipToken, _baseToken);
3    if (IPAmount.mul(price).div(inUnit).mul(closeLine) <= GPAmount.mul(RATIO_FACTOR))
{
4        return false;
5     } else if (curIPAmount.mul(price).div(inUnit) <= raiseLP) {
6        doGPLiquidation(_ipToken, _baseToken);
7        return true;
8     }
```

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

LPL-01 | Logic of Function `LPDepositRunning/GPDepositRunning`

Category	Severity	Location	Status
Logical Issue	Minor	logic/LPLogic.sol: 43	Resolved

Description

If users only call function `LPDepositRunning/GPDepositRunning` and don‘t call function `LPDoDepositRunning/GPDoDepositRunning` later, how to withdraw their `_baseToken`?

Recommendation

We advise the client to add logic to withdraw `_baseToken`.

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

LPS-01 | Substitution of `require` Calls With Modifier

Category	Severity	Location	Status
Coding Style	Informational	storage/LPStorage.sol: 35, 40, 46, 52, 58, 76	Resolved

Description

The `require` statements on the aforementioned lines can be substituted with a modifier to increase the legibility of the codebase.

Recommendation

We advise substituting the `require` statements on the aforementioned lines with a modifier:

```
modifier onlyProxy() {
 require(proxy == msg.sender, "Not Permit");
 _;
}
```

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

NPD-01 | Redundant usage of `whenNotPaused` modifier

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	NudgePool.sol: 211, 280	Acknowledged

Description

Function `GPWithdrawRunning()/LPWithdrawRunning()` are decorated by modifier `whenNotPause` in which the value of `paused` can be decided by calling function `setPause()` by `owner`. If the `paused` is set to `true`, users can not withdraw tokens.

Recommendation

We advise the client to make sure users will not be blocked by `paused` when they withdraw tokens.

Alleviation

The development team responded that `GPWithdrawRunning()/LPWithdrawRunning()` involves a lot of state variable reading and modification. Potential contract loopholes do not rule out the influence of this method on reading errors on the upstream or downstream production errors. At this time, it needs to be temporarily set to Paused. It will open after the contract vulnerability is repaired.

NPP-01 | Proper Usage of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	Informational	NPProxy.sol: 87	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to consider using the `external` attribute for functions never called from the contract.

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

NPP-02 | Lack of Input Validation

Category	Severity	Location	Status
Logical Issue	Informational	NPProxy.sol: 53	Resolved

Description

There is no validation to check whether `_newVersion` already exists.

Recommendation

We advise the client to add a validation as below:

```
    mapping(string => bool) public versionStatus;

    function setUpgrade(
        ...
    )
        public onlyOwner
    {
        require(_ipc != address(0) && _gpdc != address(0) && _gpwc != address(0) &&
                _lpc != address(0) && _vtc != address(0) && _stc != address(0) &&
                _lqdc != address(0), "Wrong Address");
        require(bytes(_newVersion).length > 0, "Empty Version");
        require(!versionStatus(_newVersion), "Existing Version");
        ...
        versionStatus(_newVersion) = true;
    }
```

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

NPS-01 | Mutability Specifiers Missing

Category	Severity	Location	Status
Gas Optimization	Informational	storage/NPStorage.sol: 18	Resolved

Description

The linked variables are assigned only once, either during their contract-level declaration or during the `constructor`'s execution.

Recommendation

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation

The development team has added the setup function for this variable in commit `4a86e7396308ae2cc3ec308aaf552094453d8db7`.

SLD-01 | Divide Before Multiply

Category	Severity	Location	Status
Mathematical Operations	Minor	logic/StateLogic.sol: 81, 101, 104, 127	Resolved

Description

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation

We advise the client to consider ordering multiplication before division to prevent any loss of arithmetical operation accuracy.

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

VSD-01 | Substitution of `require` Calls With Modifier

Category	Severity	Location	Status
Coding Style	Informational	storage/VaultStorage.sol: 31, 36, 41, 46	Resolved

Description

The `require` statements on the aforementioned lines can be substituted with a modifier to increase the legibility of the codebase.

Recommendation

We advise substituting the `require` statements on the aforementioned lines with a modifier:

```
modifier onlyProxy() {
 require(proxy == msg.sender, "Not Permit");
 _;
}
```

Alleviation

The development team resolved this issue in commit `d76fd1f33b4da59b3d806fc372441cfba3307bfa`.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings 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 findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

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

Coding Style

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

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

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 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 assessment. This report 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 in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

The assessment services provided by CertiK is subject to dependencies and 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.