CERTIK

Preliminary Comments

GoodProtocol

Jul 11th, 2021

Table of Contents

Summary

This report has been prepared for GoodDollar to discover issues and vulnerabilities in the source code of the GoodProtocol 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.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

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

Overview

Project Summary

Project Name	GoodProtocol
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/GoodDollar/GoodProtocol
Commit	435c607c972cadf1b19ae0c0d119905a8213370c

Audit Summary

Delivery Date	Jul 11, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	reserve, ubi, staking

Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
● Critical	0	0	0	0	0	0
● Major	1	1	0	0	0	0
● Medium	1	1	0	0	0	0
● Minor	6	6	0	0	0	0
● Informational	9	9	0	0	0	0
● Discussion	5	5	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum
GRG	contracts/governance/GReputation.sol	76783e18714c8f93720db50389c43f6131ed8083f0ea7658babc49f7d3b19926
EHG	contracts/reserve/ExchangeHelper.sol	89a6bd5e4672696d8fb5b4fda6a50361d54ff160c19933ae29ac34a60323098c
GMM	contracts/reserve/GoodMarketMaker.sol	aff4dca46d69c367f718d0e110a50d069ce3700f2f19956978505184aed9a123
GRC	contracts/reserve/GoodReserveCDai.sol	22f09a7e6628d8f1af5fb269a784352fd281838cdfea02cea7a28c5647315745
BSF	contracts/staking/BaseShareField.sol	e637ab90b9ed975ad90ab803fe88e51b67f5ee111004781d4f722ad6572cf8b6
DSG	contracts/staking/DonationsStaking.sol	50ac16be9eafbd4963eb405b26e3b85545bfa6e102e52205cf85c78303cd5c3f
GFM	contracts/staking/GoodFundManager.sol	a68e5e59036e35c654868129230241581b49d86b9532bbf297dd130cb53d7721
SSG	contracts/staking/SimpleStaking.sol	0518efcd05294333caefb310bed06e07a6e9b07851e1656d5dcff4dfb4fd7ebc
ASF	contracts/staking/aave/AaveStakingFactory.sol	b2f7e849526d3f65206ce35a6678b41fb8b14396ac4b8cf62488d0aea7f10be2
GAS	contracts/staking/aave/GoodAaveStaking.sol	4645d02e630fc26ec6fa909dfe8d5bec9730e79833a2fd20214c1211fcec59f4
CSF	contracts/staking/compound/CompoundStakingFactory.sol	7024ece7b7fb6bf3344c9390d6519ac72ff0f2561756981a4a1a1601ad8345f4
GCS	contracts/staking/compound/GoodCompoundStaking.sol	2c2918b40ec01da2008132efb15672cb4b60559b9d54a93bc500a44b6ea293a3
UBS	contracts/ubi/UBIScheme.sol	391c9f116e199fe33c22ec86af8e8a2729703ae26e2ed3c1ac8fab35949d72cc
DAO	contracts/utils/DAOContract.sol	cfa83fffb3112e50fcbe6ce32722c6e3e4a131f2dcb996e0f2fc0d477d52b0f0
DAU	contracts/utils/DAOUpgradeableContract.sol	9d8163fd23644b64bff6399f8a44cca829b07b2fcd6de2003d288bceab234ae5
NSG	contracts/utils/NameService.sol	50cc22edc1b0afafd47b298f28a1b9eb405946696b4d8f6ee30fda3e77eb242c
PUG	contracts/utils/ProtocolUpgrade.sol	f4df544cdbce07465bb0465ebfd6688749b199e61f9bc1776a62ae1f5bccca28
PUF	contracts/utils/ProtocolUpgradeFuse.sol	4a78bbc3e11378391184a3528047827aac045a5e6b2b9916f52952a97cf2adde
ACG	scripts/gdx/gdxAirdropCalculation.ts	c593c9b9dbacfceffcb0d081c94220aba45fa6026212998d0e185c635f490725
CGP	scripts/governance/airdropCalculation.ts	030f966ae812fff227206a3c70d72447f7838a6cd57eba096cfd07d52d893e3e

ID	file	SHA256 Checksum
TVT	scripts/upgradeToV2/upgradeToV2.ts	ed67f4fb8859d55bb1f2b65482180b4361942be663a50083c43e1aeb5a8afad2

Review Notes

Overview

The GoodProtocol contracts implement the governance, reserve, staking and UBI modules to construct an ecosystem with GoodDollar.

The governance module is built by contracts

- `ClaimerDistribution`
- `CompoundVotingMachine`
- `GovernanceStaking`
- `GReputation`
- `MultiBaseGovernanceShareField`
- `Reputation`
- `StakerDistribution`

It consists of two subsystems: the reputation system and the voting system. The reputation system mints reputation for users and determines users' voting powers, while the voting system allows users to submit proposals, vote for proposals, cancel proposals, and execute succeeded proposals.

The reserve module is built with contracts

- `ExchangeHelper`
- `GoodMarketMaker`
- `GoodReserveCDai`

It allows users to buy assets with GoodDollar or sell assets to get GoodDollar. Exchange rates are calculated and updated according to the Bancor formula.

The staking module is built with the contracts

- `BaseShareField`
- `DonationsStaking`
- `GoodFundManager`
- `SimpleStaking`
- `GoodAaveStaking`
- `GoodCompoundStaking`

Users can stake into or unstake from the staking module. When users stake into it, it sends tokens to third-party protocols (Aave and Compound) to gain interests; when users unstake from it, it withdraws tokens from third-party protocols.

The UBI module is built with the contract `UBIScheme`. It distributes daily rewards to the claimers.

Dependencies

There are a few injection dependent contracts/addresses in the current project:

- Contracts/addresses provided by `nameService`;
- `token` and `iToken` for the contract `SimpleStaking`;
- `lendingPool`, `tokenUsdOracle`, `incentiveController` and `aaveUSDOracle` for the contract `GoodAaveStaking`;
- `compUsdOracle` and `tokenUsdOracle` for the contract `GoodCompoundStaking`;
- `stakingContract` for the contract `DonationsStaking`;
- `firstClaimPool` for the contract `UBIScheme`.

We assume these dependencies are valid and non-vulnerable actors, and they are implementing proper logic to collaborate with the current project.

Privileged Roles

In the contract `GReputation`, the role **AVATAR** is authorized to set blockchain state hashes.

In the contract `GoodMarketMaker`, the roles **AVATAR** and **RESERVE** are authorized to update parameters of reserve tokens.

In the contract `GoodReserveCDai`, the role **AVATAR** is authorized to update daily expansion rate, remove minting rights, and withdraw stuck ERC20 tokens.

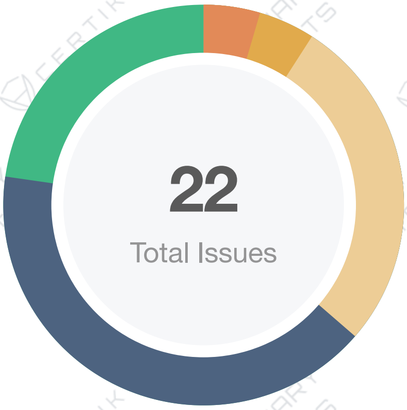
In the contract `DonationsStaking`, the role **AVATAR** is authorized to set contract status, withdraw stakes, and set the staking contract.

In the contract `SimpleStaking`, the role **AVATAR** is authorized to pause/unpause the contract and withdraw stuck ERC20 tokens.

In the contracts `GoodAaveStaking`, `GoodCompoundStaking`, `GoodFundManager` and `UBIScheme`, the role **AVATAR** is authorized to update contract configurations.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functionalities should be considered to move to the execution queue of the `CompoundVotingMachine` contract.

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**1** (4.55%)
🟨 **Medium**	**1** (4.55%)
🟨 **Minor**	**6** (27.27%)
🟦 **Informational**	**9** (40.91%)
🟩 **Discussion**	**5** (22.73%)

ID	Title	Category	Severity	Status
EHG-01	Swapping Tokens Without Approval	Logical Issue	● Major	Pending
EHG-02	Incorrect Parameters	Logical Issue	● Medium	Pending
EHG-03	Unnecessary Code	Logical Issue	● Informational	Pending
EHG-04	Lack of Check for Receiving ETH	Logical Issue	● Informational	Pending
EHG-05	Unhandled Case for `_sellPath`	Logical Issue	● Discussion	Pending
EHG-06	Lack of Check for Reentrancy	Logical Issue	● Discussion	Pending
GAS-01	Sandwich Attack Risks	Logical Issue	● Minor	Pending
GAS-02	Lack of Return Value Handling	Logical Issue	● Minor	Pending
GCS-01	Sandwich Attack Risks	Logical Issue	● Minor	Pending
GFM-01	Optimizable Boolean Comparison	Coding Style	● Informational	Pending
GFM-02	Lack of Event Emissions for Significant Transactions	Logical Issue	● Informational	Pending
GFM-03	Redundant Temporary Variable	Coding Style	● Informational	Pending
GFM-04	Redundant State Variable	Coding Style	● Informational	Pending
GMM-01	Lack of Constraint for `reserveRatioDailyExpansion`	Logical Issue	● Minor	Pending
GMM-02	Mismatch Between Code And Comment	Logical Issue	● Informational	Pending
GMM-03	Edge Situation Handling	Logical Issue	● Informational	Pending

ID	Title	Category	Severity	Status
GMM-04	Trustability of `_token.decimals()`	Logical Issue	● Discussion	Pending
SSG-01	Incompatibility with Deflationary Tokens	Logical Issue	● Minor	Pending
SSG-02	Lack of Check for Reentrancy	Logical Issue	● Minor	Pending
UBS-01	Lack of Event Emissions for Significant Transactions	Logical Issue	● Informational	Pending
UBS-02	Nullable `dailyUBIHistory`	Logical Issue	● Discussion	Pending
UBS-03	Unused State `hasWithdrawn`	Gas Optimization	● Discussion	Pending

EHG-01 | Swapping Tokens Without Approval

Category	Severity	Location	Status
Logical Issue	● Major	contracts/reserve/ExchangeHelper.sol: 311, 325	ⓘ Pending

Description

In the function `ExchangeHelper._uniswapSwap()`, the swap performed in L311 swaps tokens without approving allowance for `uniswapContract`, which means the Uniswap router will not be able to transfer `_inputPath[0]` from the contract `ExchangeHelper` to the router contract so the transaction will fail.

Also, the allowance is not approved before the swap in L325 when `isBuy` is false.

Recommendation

We recommend approving `uniswapContract`'s allowance of `_inputPath[0]` before performing swaps from non-ETH tokens.

EHG-02 | Incorrect Parameters

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/reserve/ExchangeHelper.sol: 140~141	ⓘ Pending

Description

The function `ExchangeHelper._uniswapSwap()` uses its third parameter as minimum DAI amount and fourth parameter as minimum token (other than DAI) return:

```
function _uniswapSwap(
    address[] memory _inputPath,
    uint256 _tokenAmount,
    uint256 _minDAIAmount,
    uint256 _minTokenReturn,
    address _receiver
) internal returns (uint256[] memory) {
    ...
}
```

In the function `ExchangeHelper.buy()`, `ExchangeHelper._uniswapSwap()` is called to swap the token `_buyPath[0]` to DAI. However, `_minDAIAmount` is passed as the fourth parameter rather than the third one:

```
137         uint256[] memory swap = _uniswapSwap(
138             _buyPath,
139             _tokenAmount,
140             0,
141             _minDAIAmount,
142             address(this)
143         );
```

Recommendation

We recommend passing `_minDAIAmount` as the fourth parameter of `ExchangeHelper._uniswapSwap()`:

```
        uint256[] memory swap = _uniswapSwap(
            _buyPath,
            _tokenAmount,
            _minDAIAmount,
            0,
            address(this)
        );
```

EHG-03 | Unnecessary Code

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/reserve/ExchangeHelper.sol: 116	Pending

Description

The requirement check in L112~L115 can already guarantee `_tokenAmount = msg.value` so the code in L116 can be safely omitted.

```
112            require(
113                msg.value > 0 && _tokenAmount == msg.value,
114                "you need to pay with ETH"
115            );
116            _tokenAmount = msg.value;
```

Recommendation

We advice removing the code in L116 for better code readability and gas optimization.

EHG-04 | Lack of Check for Receiving ETH

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/reserve/ExchangeHelper.sol: 117	ⓘ Pending

Description

As the only payable function in the contract `ExchangeHelper`, `ExchangeHelper.buy()` allows users to send ETH to the contract to buy GoodDollar.

When `_buyPath[0] == address(0)`, ETH will be swapped to cDAI, which will be used to buy GoodDollar in the `reserve` contract.

When `_buyPath[0] != address(0)`, users should not send ETH to the contract. However, if a user calls `ExchangeHelper.buy()` with ETH by mistake, the contract cannot do anything to the received ETH so the received ETH will be stuck in the contract.

Recommendation

We recommend checking `msg.value == 0` when `_buyPath[0] != address(0)` in the function `ExchangeHelper.buy()`.

EHG-05 | Unhandled Case for `_sellPath`

Category	Severity	Location	Status
Logical Issue	● Discussion	contracts/reserve/ExchangeHelper.sol: 196, 199, 206	ⓘ Pending

Description

In L196, when `_sellPath[0] != cDaiAddress` OR `_sellPath.length > 1`, the token will be transferred to `address(this)` via function `reserve.sell`, and the tokens might be locked into the current contract forever.

In the `if-else` block in L199~L215, the handled case are:

- when `_sellPath.length == 1 && _sellPath[0] == daiAddress`, the `if` block will be executed;
- when `_sellPath.length > 1 && _sellPath[0] == daiAddress`, the swap action in `else if` block will be executed;
- when `_sellPath[0] != cDaiAddress && _sellPath[0] != daiAddress`, it will revert;
- for all the other cases, f.e. when `_sellPath.length > 1 && _sellPath[0] == cDaiAddress`, no more action will be taken (no revert).

We hope to check with the client team and confirm if this is the intended design.

EHG-06 | Lack of Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Discussion	contracts/reserve/ExchangeHelper.sol: 96	ⓘ Pending

Description

In the function `ExchangeHelper.buy()`, there are state updates and an event emit after external calls and thus it is vulnerable to potential reentrancy attacks. It is recommended to completely eradicate all potential reentrancy. Sometimes the loss by reentrancy attack is not a direct loss, but since reentrancy would distort chain state, it could still lead to a project loss via the Butterfly Effect.

Recommendation

We recommend applying the `nonReentrant` modifier for the aforementioned function to prevent potential reentrancy attacks.

GAS-01 | Sandwich Attack Risks

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/staking/aave/GoodAaveStaking.sol: 121	ⓘ Pending

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens using the Uniswap mechanism 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 function `uniswapContract.swapExactTokensForTokens()` is called without setting restrictions on slippage or minimum output amount, so transactions triggering this function are vulnerable to sandwich attacks, especially when the input amount is large.

Recommendation

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

GAS-02 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/staking/aave/GoodAaveStaking.sol: 85, 99	ⓘ Pending

Description

The function `lendingPool.withdraw()` is not a void-returning function. Ignoring its return value might cause some unexpected exceptions.

Recommendation

We recommend checking the output of the function `lendingPool.withdraw()` before continuing processing.

GCS-01 | Sandwich Attack Risks

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/staking/compound/GoodCompoundStaking.sol: 105, 125	ⓘ Pending

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens using the Uniswap mechanism 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.

- `uniswapContract.swapExactTokensForTokens()`
- `uniswapContract.swapExactTokensForTokens()`

Recommendation

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

GFM-01 | Optimizable Boolean Comparison

Category	Severity	Location	Status
Coding Style	● Informational	contracts/staking/GoodFundManager.sol: 146~152	(!) Pending

Description

The code implementation

```
146     require(
147         false ==
148             (_isBlackListed == false &&
149                 rewardsForStakingContract[_stakingAddress].isBlackListed ==
150                 true),
151         "can't undo blacklisting"
152     );
```

can be simplified as

```
146     require(
147         _isBlackListed || !rewardsForStakingContract[_stakingAddress].isBlackListed,
148         "can't undo blacklisting"
149     );
```

GFM-02 | Lack of Event Emissions for Significant Transactions

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/staking/GoodFundManager.sol: 94, 104, 112, 122, 135	ⓘ Pending

Description

Functions changing the status of sensitive variables should emit events as notifications to the public. For example,

- `GoodFundManager.setGasCost()`;
- `GoodFundManager.setCollectInterestTimeThreshold()`;
- `GoodFundManager.setInterestMultiplier()`;
- `GoodFundManager.setGasCostExceptInterestCollect()`;
- `GoodFundManager.setStakingReward()`.

Recommendation

We recommend emitting events for all the essential state variables that are possible to be changed during the runtime.

GFM-03 | Redundant Temporary Variable

Category	Severity	Location	Status
Coding Style	● Informational	contracts/staking/GoodFundManager.sol: 284, 289	ⓘ Pending

Description

The variable `totalInterest` is only used in self-assignment on L289 after the declaration. It is never used in state updates or event emissions, so it can be removed.

Recommendation

We recommend removing the redundant temporary variable `totalInterest`.

GFM-04 | Redundant State Variable

Category	Severity	Location	Status
Coding Style	● Informational	contracts/staking/GoodFundManager.sol: 22	ⓘ Pending

Description

The state variable `lastTransferred` is never used within the contract `GoodFundManager`, so it can be removed.

Recommendation

We recommend removing the redundant state variable `lastTransferred`.

GMM-01 | Lack of Constraint for `reserveRatioDailyExpansion`

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/reserve/GoodMarketMaker.sol: 162~164	Pending

Description

According to the code implementation in L162~L164, the value of the reserve ratio has an exponential relationship with `reserveRatioDailyExpansion`.

```
162         for (uint256 i = 0; i < daysPassed; i++) {
163             ratio = rmul(ratio, reserveRatioDailyExpansion);
164         }
```

If `reserveRatioDailyExpansion` is larger than 10^{27}, `ratio` will increase exponentially daily and approaching infinity; if `reserveRatioDailyExpansion` is smaller than 10^{27}, `ratio` will decrease exponentially daily and approaching 0. Lacking check for `reserveRatioDailyExpansion` can lead to unexpected calculation result for the reserve ratio.

Recommendation

We recommend the team add an appropriate value check for `reserveRatioDailyExpansion` when it is set or updated to ensure the reserve ratio can be calculated properly as expected.

GMM-02 | Mismatch Between Code And Comment

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/reserve/GoodMarketMaker.sol: 261, 274~275	ⓘ Pending

Description

The code implementation logic of `Reserve Ratio` in L274~L275 doesn't match the comment in L261.

Recommendation

We recommend the team revisit the logic. According to our understanding, the implementation is correct and the comment should be

```
261  * new RR = Reserve supply / ((gd supply + gd mint amount) * price)
```

GMM-03 | Edge Situation Handling

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/reserve/GoodMarketMaker.sol: 302	ⓘ Pending

Description

With the current code implementation, it requires `rtoken.gdSupply` to be larger than `_gdAmount` in L302. However, according to the error message `"GD amount is higher than the total supply"`, it should include the case when `rtoken.gdSupply == _gdAmount`.

Recommendation

We recommend modifying the code in L301~L304 as

```
301        require(
302            rtoken.gdSupply >= _gdAmount,
303            "GD amount is higher than the total supply"
304        );
```

GMM-04 | Trustability of `_token.decimals()`

Category	Severity	Location	Status
Logical Issue	● Discussion	contracts/reserve/GoodMarketMaker.sol: 268, 399	ⓘ Pending

Description

The calculation in the aforementioned lines rely on the result of `decimals()` function of the input token contract. If `_token.decimals()` can not return consistent trustable value, it might introduce incorrect calculation and thus lead to unexpected loss. We recommend the team revisit the logic and ensure this is the intended design.

SSG-01 | Incompatibility with Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/staking/SimpleStaking.sol: 81~82	ⓘ Pending

Description

When users stake to and unstake from the staking contract, the token `iToken` or `token` will be transferred to the contract or users. When `iToken` or `token` is a deflationary token, the input amount may not equal the received amount due to the charged or burned transaction fees. As a result, this may not meet the assumption behind these low-level asset-transferring routines and will bring unexpected balance inconsistency.

Recommendation

We recommend keeping regulating the set of tokens supported by the staking contract, and if there is a need to support deflationary tokens, adding necessary mitigation mechanisms to keep track of accurate balances.

SSG-02 | Lack of Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/staking/SimpleStaking.sol: 180	Pending

Description

In the function `SimpleStaking.stake()`, there are state updates and an event emit after external calls and thus it is vulnerable to potential reentrancy attack.

Recommendation

We recommend applying the `nonReentrant` modifier for the aforementioned function to prevent potential reentrancy attacks.

UBS-01 | Lack of Event Emissions for Significant Transactions

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/ubi/UBIScheme.sol: 220, 512, 288~512	ⓘ Pending

Description

Functions changing the status of sensitive variables should emit events as notifications to the public. For example,

- `UBIScheme.setCycleLength()`
- `UBIScheme.setDay()`
- `UBIScheme.setShouldWithdrawFromDAO()`

Recommendation

We recommend emitting events for all the essential state variables that are possible to be changed during the runtime.

UBS-02 | Nullable `dailyUBIHistory`

Category	Severity	Location	Status
Logical Issue	● Discussion	contracts/ubi/UBIScheme.sol: 249	Pending

Description

If nobody claimed yesterday, in L249, `dailyUBIHistory[currentDay - 1]` would be 0, leading to `prevDayBalance` being 0. It will affect the value of `shouldStartEarlyCycle`. We advice the team to revisit the logic and confirm if the calculation would still work as intended in such situation.

```
249         uint256 prevDayBalance = dailyUBIHistory[currentDay - 1].openAmount;
250         bool shouldStartEarlyCycle =
251             currentBalance >= (prevDayBalance * 130) / 100 &&
252                 currentBalance > (currentCycleStartingBalance * 80) / 100;
```

UBS-03 | Unused State `hasWithdrawn`

Category	Severity	Location	Status
Gas Optimization	● Discussion	contracts/ubi/UBIScheme.sol: 79, 273	ⓘ Pending

Description

The state `hasWithdrawn` of the `Funds` instances are not used in the contract. It can be omitted if not being consumed anywhere.

```
76      struct Funds {
77          // marks if the funds for a specific day has
78          // withdrawn from avatar
79          bool hasWithdrawn;
80          // total GD held after withdrawing
81          uint256 openAmount;
82      }
```

Appendix

Finding Categories

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.

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 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 without CertiK'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 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.

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.