CERTIK

Dollar-Protocol

Security Assessment

January 22th, 2021

For :
Dollar-Protocol

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Adrian Hetman @ CertiK
adrian.hetman@certik.org

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	**Dollar-Protocol**
Description	Dollar Protocol is a suite of decentralized algorithmic synthetic assets (USD, EUR, YUAN) that are governed by Share holders.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. ab1fbc40f2d4075a81ab2bfc4c8f7e5ecc46a1b4 2. 6b78233ba5c4f4fdd4b7481cc4c99d8b6222a06b

Audit Summary

Delivery Date	**January 22th, 2021**
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	January 5th, 2021 - January 11th, 2021

Vulnerability Summary

Total Issues	**40**
Total Critical	0
Total Major	4
Total Medium	11
Total Minor	5
Total Informational	20

Executive Summary

We were tasked with auditing the Dollar Protocol codebase and particularly the governance, timelock and seigniorage share mechanisms.

The project is loosely derived from various de-facto standard projects, such as Compound and Yarn, with certain novel features introduced such as a new staking mechanism and variable adaptations.

Overall code quality is acceptable, however, there are quite a few areas where it can improve and become more aligned with the latest standards in the industry. For example, the project utilizes a custom mutex implementation whereas battle-tested contracts such as the `ReentrancyGuard.sol` by OpenZeppelin could have been used instead as the custom implementation does not bring novelty.

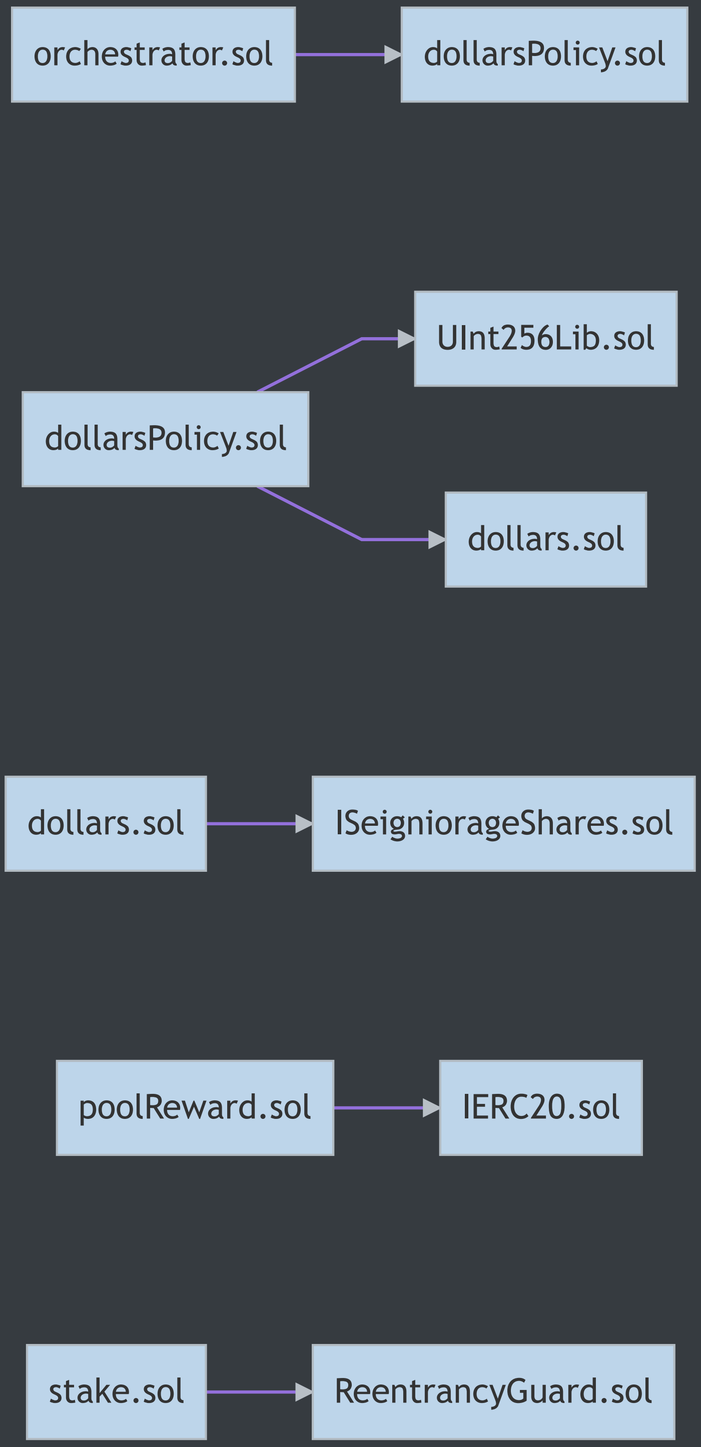
The dependencies of the codebase are relatively outdated and our advice is to upgrade them as they are no longer formally supported. Additionally, the protocol appears to heavily utilize the `onlyOwner` modifier in certain parameter tuning functions that can reduce or render obsolete the decentralization of the protocol which should be clearly conveyed to its users.

The documentation of the project was adequate and we verified the code's accordance with the facts laid out in the said documentation.

Files In Scope

ID	Contract	Location
ICH	ICash.sol	contracts/interface/ICash.sol
ISS	ISeigniorageShares.sol	contracts/interface/ISeigniorageShares.sol
SMI	SafeMathInt.sol	contracts/lib/SafeMathInt.sol
UIL	UInt256Lib.sol	contracts/lib/UInt256Lib.sol
BON	bond.sol	contracts/bond.sol
DOL	dollars.sol	contracts/dollars.sol
DOA	dollarsPolicy.sol	contracts/dollarsPolicy.sol
DOR	dollarTimelock.sol	contracts/dollarTimelock.sol
ORC	orchestrator.sol	contracts/orchestrator.sol
POO	poolReward.sol	contracts/poolReward.sol
STA	stake.sol	contracts/stake.sol
SEI	seigniorageShares.sol	contracts/seigniorageShares.sol
GAA	seigniorageGovernorAlpha.sol	contracts/seigniorageGovernorAlpha.sol

File Dependency Graph (BETA)



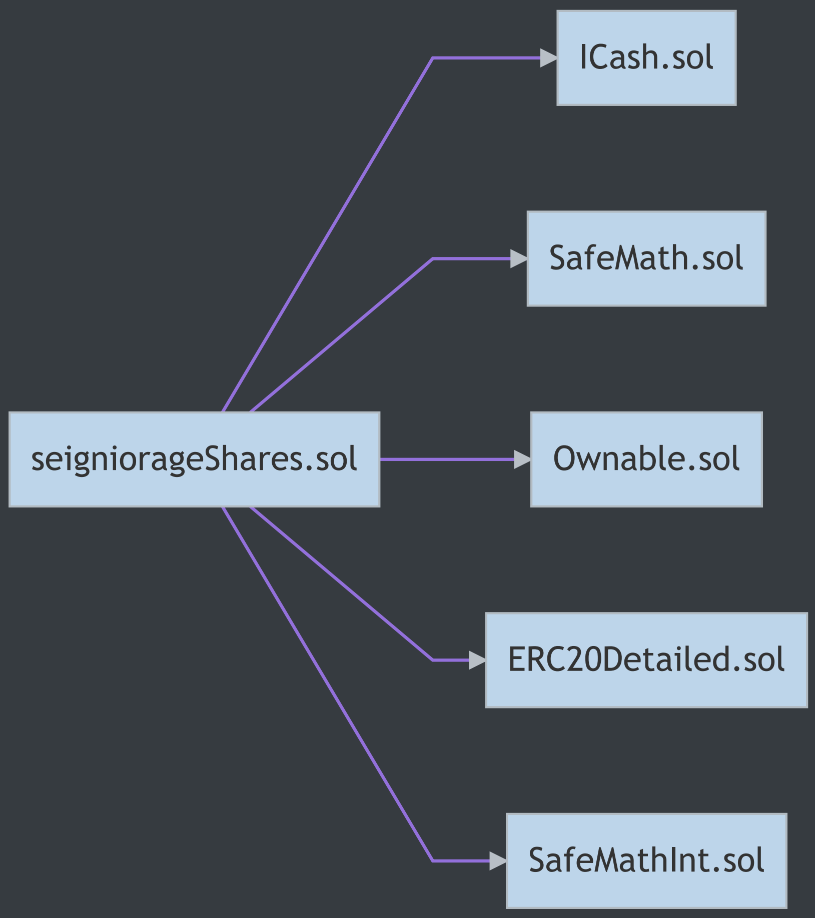
$$\begin{tikzcd}
\text{orchestrator.sol} \arrow[r] & \text{dollarsPolicy.sol}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{UInt256Lib.sol} \\
\text{dollarsPolicy.sol} \arrow[ur] \arrow[dr] & \\
 & \text{dollars.sol}
\end{tikzcd}$$

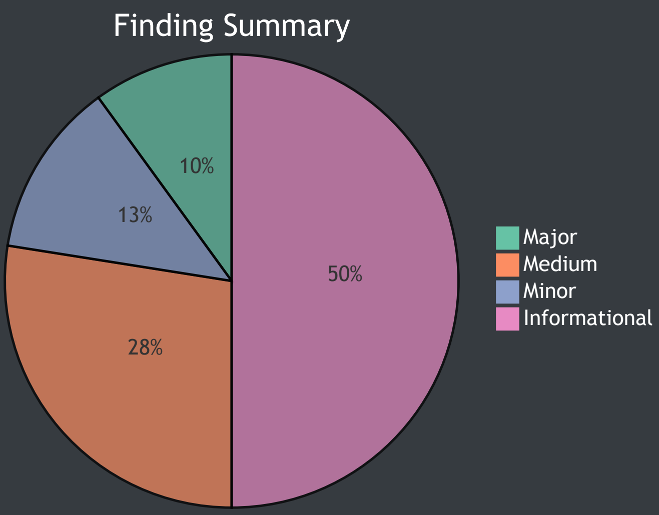
$$\begin{tikzcd}
\text{dollars.sol} \arrow[r] & \text{ISeigniorageShares.sol}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{poolReward.sol} \arrow[r] & \text{IERC20.sol}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{stake.sol} \arrow[r] & \text{ReentrancyGuard.sol}
\end{tikzcd}$$


seigniorageShares.sol
ICash.sol
SafeMath.sol
Ownable.sol
ERC20Detailed.sol
SafeMathInt.sol

Findings



ID	Title	Type	Severity	Resolved
BON-01	Unlocked Compiler Version	Language Specific	Informational	! ✓
BON-02	Unchecked value from low-level call	Volatile Code	Medium	✓
BON-03	Ignores return value of external function call	Volatile Code	Minor	! ✓
BON-04	Owner has to much power over widely used variables	Control Flow	Medium	✓
BON-05	Lacks input validation on address	Volatile Code	Medium	✓

BON-06	Unused state variables	Dead Code	Informational	✓
DOL-01	Unlocked Compiler Version	Language Specific	Informational	❗
DOL-02	Redundant Statements	Dead Code	Informational	✓
DOL-03	Create a modifier for used require statement	Coding Style	Informational	❗
DOL-04	Owner has to much power over widely used variables	Control Flow	Medium	✓
DOL-05	Own implementation of nonReentrant modifier	Coding Style	Minor	❗
DOL-06	Lack of require check for mutex on rebase() function	Control Flow	Major	✓
DOL-07	Unchecked value from low-level call	Volatile Code	Medium	❗
DOL-08	Redundant if statements	Language Specific	Informational	✓
DOL-09	SafeMath is not used	Mathematical Operations	Minor	✓
DOA-01	Unlocked Compiler Version	Language Specific	Informational	❗
DOA-02	Never initialized variables	Dead Code	Minor	✓
DOA-03	Redundant Statements	Dead Code	Informational	✓
DOA-04	Owner has to much power over widely used variables	Control Flow	Medium	✓
DOA-05	Redundant boolean value check	Gas Optimization	Informational	✓
DOA-	Typo in require error message	Coding Style	Informational	

06				✓
DOA-07	`orchestrator` address is never set	Volatile Code	Major	✓
DOA-08	Lack of input validation	Volatile Code	Medium	✓
ORC-01	Lack of input validation	Volatile Code	Medium	⏳
POO-01	Unlocked Compiler Version	Language Specific	Informational	⏳
POO-02	Lack of input validation	Volatile Code	Medium	✓
POO-03	Unused variables	Dead Code	Informational	✓
POO-04	Free gain of pool tokens	Volatile Code	Major	✓
POO-05	Owner has to much power over widely used variables	Control Flow	Medium	✓
SEI-01	Unlocked Compiler Version	Language Specific	Informational	⏳
SEI-02	Unused variables	Dead Code	Informational	✓
SEI-03	Potential for Overflow/Underflow	Mathematical Operations	Minor	✓
SEI-04	Cross-chain replay attack	Volatile Code	Major	✓
STA-01	Unlocked Compiler Version	Language Specific	Informational	⏳
STA-02	Lacks input validation on address	Volatile Code	Medium	✓
ICH-01	Unlocked Compiler Version	Language Specific	Informational	⏳

ISS-01	Unlocked Compiler Version	Language Specific	Informational	✓
SMI-01	Unlocked Compiler Version	Language Specific	Informational	✓
UIL-01	Unlocked Compiler Version	Language Specific	Informational	✓
DOR-01	Unlocked Compiler Version	Language Specific	Informational	✓

BON-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	bond.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

BON-02: Unchecked value from low-level call

Type	Severity	Location
Volatile Code	Medium	bond.sol L262

Description:

Ignores return value by `ethBondOracle.call(abi.encodeWithSignature('update()'));`

Recommendation:

If you choose to use the low-level call methods, make sure to handle the possibility that the call will fail, by checking the return value.

```
(bool success, ) = someAddress.call.value(55)("");
if(!success) {
    // handle failure code
}
```

Alleviation:

Issue was resolved

BON-03: Ignores return value of external function call

Type	Severity	Location
Volatile Code	Minor	bond.sol L261

Description:

`Dollars.claimDividends(account);` returns a value that is ignored and not acted upon.

Recommendation:

Return value of `claimDividends()` shouldn't be ignored. If any reason to call that function was to invoke the logic of `updateAccount` modifier, then that modifier should be made into the function and be called explicitly.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

BON-04: Owner has to much power over widely used variables

Type	Severity	Location
Control Flow	Medium	bond.sol L113-L123

Description:

Owner can set any values he wants using functions `setClaimableUSD` , `setConstantClaimableUSD` , `setEthBondOracle` .
This causes centralization issue on concern.

Recommendation:

Mentioned functions should be called by governance and values of these variables should be set during initialization phase, specially the address of the Oracle.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase. Client comment: "we plan on removing references to Deployer address after full decentralization - giving our compound governor the admin keys to the upgradeable contracts"

BON-05: Lacks input validation on address

Type	Severity	Location
Volatile Code	Medium	bond.sol L121, L125, L133, L137, L42

Description:

Functions doesn't checks provided address.

Recommendation:

Check that the address is not zero. You could utilize already created modifier `validRecipient` but it should be then renamed to reflect the functionality.

Alleviation:

Issue partially resolved as there isn't still an input validation on initialize function

BON-06: Unused state variables

Type	Severity	Location
Dead Code	Informational	bond.sol L22, L25

Description:

Linked variables are unused.

Recommendation:

Remove unused state variable.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase. Client comment: "these are variables from a previous proxy so we cannot remove them"

DOL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	dollars.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DOL-02: Redundant Statements

Type	Severity	Location
Dead Code	Informational	dollars.sol L107, L126-L129, L69

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase. Client comment: "these are variables from a previous proxy so we cannot remove them"

DOL-03: Create a modifier for used require statement

Type	Severity	Location
Coding Style	Informational	dollars.sol L183, L194, L160, L165

Description:

`msg.sender == timelock` is used multiple times across the file and could be used in its own modifier for cleaner code.

Recommendation:

Create a modifier for `msg.sender == timelock` for cleaner code

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DOL-04: Owner has to much power over widely used variables

Type	Severity	Location
Control Flow	Medium	dollars.sol L173, L201, L205, L284

Description:

Owner can set any values he wants using linked functions. This causes centralization issue and concern

Recommendation:

Mentioned functions should be called by governance and values of these variables should be set during initialization phase, specially the addresses.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase. Client comment: "we plan on removing references to Deployer address after full decentralization - giving our compound governor the admin keys to the upgradeable contracts"

DOL-05: Own implementation of nonReentrant modifier

Type	Severity	Location
Coding Style	Minor	dollars.sol L210, L234, L316

Description:

Function doesn't use OpenZeppelin `nonReentrant` modifier where other contracts do use it. Instead own implementation of mutex is introduced.

Recommendation:

We recommend using `openzeppelin-eth/contracts/utils/ReentrancyGuard.sol` that is already used by other contracts.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DOL-06: Lack of require check for mutex on rebase() function

Type	Severity	Location
Control Flow	Major	dollars.sol L322

Description:

`rebase()` lacks require function to check for mutex against reentrancy attack.

Recommendation:

Should add `require(!reEntrancyMutex, "dp::reentrancy");` in current implementation but as mentioned in other fnding, we recommend using `nonReentrant` modifier found in `openzeppelin-eth/contracts/utils/ReentrancyGuard.sol";` .

Alleviation:

Issue resolved

DOL-07: Unchecked value from low-level call

Type	Severity	Location
Volatile Code	Medium	dollars.sol L307, L371

Description:

Ignores return value of uniswap low-level calls.

Recommendation:

If you choose to use the low-level call methods, make sure to handle the possibility that the call will fail, by checking the return value.

```
(bool success, ) = someAddress.call.value(55)("");
  if(!success) {
    // handle failure code
}
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DOL-08: Redundant if statements

Type	Severity	Location
Language Specific	Informational	dollars.sol L444, L472

Description:

If statement is unnecessary as following safeMath `sub` operation will throw error and revert, avoiding underflow

Recommendation:

Remove the if statement

Alleviation:

Issue resolved

DOL-09: SafeMath is not used

Type	Severity	Location
Mathematical Operations	Minor	dollars.sol L581, L592

Description:

SafeMath from OpenZeppelin is not used on this two instances making it possible for overflow/underflow

Recommendation:

Use SafeMath library for all of the uint256 operations.

Alleviation:

Issue resolved

DOA-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	dollarsPolicy.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DOA-02: Never initialized variables

Type	Severity	Location
Dead Code	Minor	dollarsPolicy.sol L54-L55

Description:

Two linked variables are never initialized.

Recommendation:

Initliaze linked variables in the `initialize` function after deployment or create functions where governance can set it.

Alleviation:

Issue resolved

DOA-03: Redundant Statements

Type	Severity	Location
Dead Code	Informational	dollarsPolicy.sol L72-L73

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase. Client comment: "these are variables from a previous proxy so we cannot remove them"

DOA-04: Owner has to much power over widely used variables

Type	Severity	Location
Control Flow	Medium	dollarsPolicy.sol L93, L218-L230

Description:

Owner can set any values he wants using linked functions. This causes centralization issue and concern

Recommendation:

Mentioned functions should be called by governance and values of these variables should be set during initialization phase, specially the addresses.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase. Client comment: "we plan on removing references to Deployer address after full decentralization - giving our compound governor the admin keys to the upgradeable contracts"

DOA-05: Redundant boolean value check

Type	Severity	Location
Gas Optimization	Informational	dollarsPolicy.sol L109

Description:

Redundant boolean value check inside require.

Recommendation:

For gas optimization we recommend using sole variable in require statement, not comparing it to another bool value

Alleviation:

Issue resolved

DOA-06: Typo in require error message

Type	Severity	Location
Coding Style	Informational	dollarsPolicy.sol L108

Description:

There is a typo in `require` message.

Recommendation:

it should be `OUTSIDE_REBASE` not `OUTISDE_REBASE`

Alleviation:

Issue resolved

DOA-07: `orchestrator` address is never set

Type	Severity	Location
Volatile Code	Major	dollarsPolicy.sol L62

Description:

`orchestrator` variable is never set and is used in `onlyOrchestrator` modifier which is used in `rebase` function, marking it not usable at this point.

Recommendation:

Initialize `orchestrator` in `initialize` function or create seperate function for governance to set it.

Alleviation:

Issue resolved

DOA-08: Lack of input validation

Type	Severity	Location
Volatile Code	Medium	dollarsPolicy.sol L256, L218, L222, L226, L230, L205, L93

Description:

Functions doesn't checks provided address.

Recommendation:

Check that the address is not zero.

Alleviation:

Issue resolved apart from L205, initializeOracles() function

ORC-01: Lack of input validation

Type	Severity	Location
Volatile Code	Medium	orchestrator.sol L50

Description:

Functions doesn't checks provided address.

Recommendation:

Check that the address is not zero.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

POO-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	poolReward.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

POO-02: Lack of input validation

Type	Severity	Location
Volatile Code	Medium	poolReward.sol L75, L87, L102, L126, L222

Description:

Functions doesn't checks provided address.

Recommendation:

Check that the address is not zero.

Alleviation:

Issue resolved

POO-03: Unused variables

Type	Severity	Location
Dead Code	Informational	poolReward.sol L131-L133

Description:

`poolAllocPoint` and `dollarShareBalance` are not used in the function.

Recommendation:

Remove unneccessary variables

Alleviation:

Issue resolved

POO-04: Free gain of pool tokens

Type	Severity	Location
Volatile Code	Major	poolReward.sol L165-L198

Description:

During `withdraw` function execution, if `lpToken` is type of a token that notifies the msg.sender of succesfull token transfer, like ERC777 which is ERC20 compliant, user could make a call to the `deposit()` function before `resetuser()` would be called.
This means whatever pool tokens malicious actor have just withdrawn, could deposit again without his account info be updated and to the protocol he would only gain tokens.

Recommendation:

It is recommended to follow checks-effects-interactions pattern for cases like this. It shields public functions from re-entrancy attacks. It's always a good practice to follow this pattern. `checks-effects-interactions` pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.

Alleviation:

Issue resolved

POO-05: Owner has to much power over widely used variables

Type	Severity	Location
Control Flow	Medium	poolReward.sol L75

Description:

Owner can set any values he wants using linked functions. This causes centralization issue and concern

Recommendation:

Mentioned functions should be called by governance and values of these variables should be set during initialization phase, specially the addresses.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase. Client comment: "we plan on removing references to Deployer address after full decentralization - giving our compound governor the admin keys to the upgradeable contracts"

SEI-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	seigniorageShares.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

SEI-02: Unused variables

Type	Severity	Location
Dead Code	Informational	seigniorageShares.sol L64, L65, L68

Description:

Linked variables are unused

Recommendation:

Remove unused state variable.

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase. Client comment: "these are variables from a previous proxy so we cannot remove them"

SEI-03: Potential for Overflow/Underflow

Type	Severity	Location
Mathematical Operations	Minor	seigniorageShares.sol L196, L197, L206, L222

Description:

Raw mathematical operations should be replaced with the respective SafeMath function to prevent overflow/underflow.

Recommendation:

It's always recommended to use SafeMath's functions for any arithmetic operations.

Alleviation:

Issue resolved

SEI-04: Cross-chain replay attack

Type	Severity	Location
Volatile Code	Major	seigniorageShares.sol L560

Description:

The `delegateBySig` function of the contract utilizes the `chainid` variable acquired from `getChainId()` which returns always 1 f.e. the `chainid` of the current Ethereum chain is `1` if a fork occurs, the new chain will have a different `chainid` than that of Ethereum. The `PERMIT_TYPEHASH` uses the `chainid` variable within it to prevent a permit from being replayed on both the main chain and the side chain however, as the `chainid` is evaluated once during construction, the contracts of the main chain and the forked chain will have the same `PERMIT_TYPEHASH` thus allowing the attacker to use the permit function on both chains whereas it should have only been used on one chain.

Recommendation:

To guard contract from this type of attack it is recommended to change `getChainId()` function to look like this:

```
    function getChainId() internal pure returns (uint) {
            uint256 chainId;
            assembly { chainId := chainid() }
            return chainId;
        }
```

Alleviation:

Issue resolved

STA-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	stake.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:\n\n```Solidity\npragma solidity 0.6.2;

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

STA-02: Lacks input validation on address

Type	Severity	Location
Volatile Code	Medium	stake.sol L33, L88, L99, L106

Description:

Functions doesn't checks provided address.

Recommendation:

Check that the address is not zero. You could utilize already created modifier `validRecipient` but it should be then renamed to reflect the functionality.

Alleviation:

Issue resolved but initialize() function stil lacks input validation

ICH-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ICash.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

ISS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ISeigniorageShares.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

SMI-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	SafeMathInt.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

UIL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	UInt256Lib.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DOR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	dollarTimelock.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Dollar-Protocol development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

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 in-storage 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.