ENTERSOFT

AUDD

Final Smart Contract Audit Report

DATE: 29th August 2023 | PREPARED BY: Entersoft Australia

Contents

Revision History & Version Control

Start Date	End Date	Author	Comments/Details
29 May 2023	15 August 2023	Charan	Final Report

Reviewed by	Released by
Nelson Garnepudi	Nishita Palaksha

Entersoft was commissioned to perform a source code review on AUDD's smart contracts. The review was conducted between 29 May 2023 to 15 August 2023. The report is organized into the following sections.

- Executive Summary: A high-level overview of the security audit findings.
- Technical analysis: Our detailed analysis of the Smart Contract code

The information in this report should be used to understand overall code quality, security, correctness, and meaning that code will work as described in the smart contract.

1.0 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to: (i) smart contract best coding practices and vulnerabilities in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2.0 Overview

2.1 Project Overview

During the period of **29 May 2023 to 15 August 2023**, Entersoft performed smart contract security audits for **AUDD**.

2.2 Scope

Entersoft is pleased to share with AUDD the outcomes of our comprehensive security audit conducted on the designated smart contracts. The primary objective of this audit was to thoroughly evaluate the code's quality, security, and accuracy, with the overarching goal of enhancing the resilience of your smart contracts. In the subsequent section of this report, we elaborate on the methodology employed, the tools harnessed, and the pivotal tasks executed throughout the audit process.

Scope and Methodology:

Our audit encompassed a thorough analysis and documentation of the smart contract codebase, with a specific focus on ensuring its quality, security, and correctness. To achieve this, we employed the following methodology, which allowed for a comprehensive evaluation:

1. Automated Scans and Static Analysis: Using cutting-edge tools, we carried out automated scans and static analysis on the entire codebase. This involved examining the code line by line to pinpoint potential vulnerabilities and security gaps.

2. Dynamic Analysis: Dynamic analysis entailed running the smart contract through various scenarios and inputs to observe its behaviour during execution. This helped unveil vulnerabilities that may only become apparent during runtime.

3. Fuzz Testing: By using tailored scripts and input variables, we conducted comprehensive fuzz testing on the contract. This entailed subjecting the contract to both anticipated and unexpected inputs to gauge its responses. This method helped expose vulnerabilities across a wide range of scenarios.

Our goal in conducting this well-rounded security audit was to ensure that the smart contract code is fortified against potential threats and functions reliably across diverse scenarios. The combination of automated scans, static analysis, dynamic analysis, and fuzz testing allowed us to comprehensively evaluate the security stance of the smart contract.

Activities Undertaken:

The audit activities were structured to comprehensively cover various aspects of the smart contract's behaviour and design:

1. Review of Smart Contract Functions' Logic: We meticulously reviewed the business logic of the smart contract functions, aligning them with the specifications outlined in the white-paper/technical paper. This was done to ensure that the functions behaved as intended and didn't exhibit any unintended behaviour.

2. Extensive Fuzz Testing on Critical Functions: Critical functions that are integral to sensitive business operations, including multi-signature actions, Ether/Token transfers, and minting/burning operations, were subjected to extensive fuzz testing.

This rigorous approach helped identify vulnerabilities that could potentially compromise the security of these critical operations.

Smart Contracts Reviewed:

As part of this audit, we reviewed the following smart contracts to assess their quality, security, and correctness:

1. Stablecoin Implementation Contract:

0xC27Ca3DC5591ff62F696E2AED6eec4549D3582b6
https://goerli.etherscan.io/address/0xC27Ca3DC5591ff62F696E2AED6eec4549D3582b6#code

2. Governance Token Implementation Contract: 0x80409AE2E895aAdf805bb17a0aBAb1876F270E18
https://goerli.etherscan.io/address/0x80409AE2E895aAdf805bb17a0aBAb1876F270E18#code

3. Stablecoin Proxy Contract: 0x2cE8E761327da36c9d2D9D97A7b06E0A40295013
https://goerli.etherscan.io/address/0x2cE8E761327da36c9d2D9D97A7b06E0A40295013#code

4. Governance Token Proxy Contract: 0xe5F931Bb332785C82b2A5B5180d87355c4746fF8
https://goerli.etherscan.io/address/0xe5F931Bb332785C82b2A5B5180d87355c4746fF8#code

5. Proxy Admin Contract: Address: 0x4146446Cfc0942A835426652D55a81C7E177F03a
https://goerli.etherscan.io/address/0x4146446Cfc0942A835426652D55a81C7E177F03a#code

OUT-OF-SCOPE: External contracts, External Oracles, other smart contracts in the repository, or imported smart contracts.

2.3 Project Summary

Project Name	No. of Smart Contract File(s)	Verified	Vulnerabilities
AUDD	4	Yes	As per report. Section 2.6

2.4 Audit Summary

Delivery Date	Method of Audit	Consultants Engaged
29 August 2023	Automated, Static and Dynamic Analysis including Fuzz Testing	3

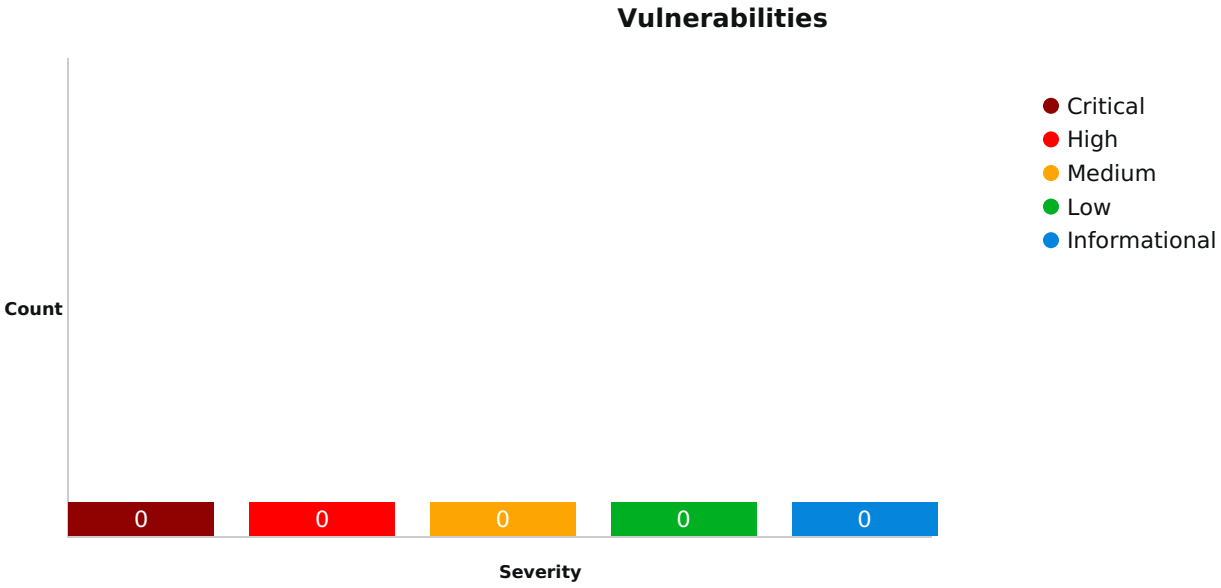
2.5 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:

		Impact				
		Minimal	Low	Medium	High	Critical
Likelihood	Critical	Minimal	Low	Medium	High	Critical
	High	Minimal	Low	Medium	High	Critical
	Medium	Minimal	Low	Medium	Medium	High
	Low	Minimal	Low	Low	Low	Medium
	Minimal	Minimal	Minimal	Minimal	Low	

2.6 Vulnerability Summary

Critical	High	Medium	Low	Informational
0	0	0	0	0



3.0 Executive Summary

Novatti specializes in delivering cutting-edge digital banking, payment, and remittance solutions, utilizing innovative technologies to ensure secure and efficient financial transactions for both businesses and individuals. By leveraging technology, Novatti aims to create seamless payment experiences and promote financial inclusion. One of their notable offerings is AUDD, Novatti's stablecoin designed to facilitate transactions and remittances between businesses and their customers. Unlike other stablecoins that have limitations on their usage, AUDD is intentionally designed to be unlimited.

During the audit, Entersoft undertook a thorough and comprehensive security assessment of the smart contract responsible for governing the token mechanism. Our approach involved a multi-pronged evaluation, which included automated scans, static analysis, dynamic analysis, and fuzz testing. This meticulous review covered a wide array of strategies to ensure the integrity and security of the smart contract code.
The subsequent files were subjected to a meticulous audit process:

- Common.sol
- StableCoin.sol
- ProxyAdmin.sol
- GovernanceToken.sol

After conducting a comprehensive analysis of the change-logs between the previously used Solidity version (0.8.17) and the recently adopted version (0.8.21), Entersoft has verified that the updates primarily consist of bug fixes, optimisations, security enhancements, new features and syntax changes. These modifications are predominantly minor enhancements that do not directly influence the operational capabilities of the existing codebase. It's important to note that the fundamental logic and functionalities of the smart contracts remain unaffected. The transition to the updated Solidity version notably bolsters the efficiency and dependability of the codebase, owing to the advancements integrated into the latest version.

The initial smart contract audit was conducted by Entersoft spanning from 29 May 2023 to 15 August 2023. Following the implementation of the recommendations by AUDD, Entersoft performed a subsequent assessment on 23 August 2023, and successfully confirmed the resolution of identified vulnerabilities. As a result, we are pleased to report that the overall security stance is now categorised as 'Secure.'

Smart Contract Audit Testing objectives	Result
Integer Overflow	Passed
Unprotected Selfdestruct	Passed
Unprotected Ether Withdrawal	Passed

Reentrancy	Passed
Weak Randomness	NA
Assert Violation	Passed
Write to Arbitrary Storage Location	Passed
Jump to Arbitrary Destination	NA
Uninitialized Storage Pointer	Passed

3.1 Intended Behavior

The smart contracts implement the multi-signature functionality for governing the token supply, whitelisting wallets, transaction control, implement thresholds, and managing signatories.

The GovernanceToken smart contract has 21 write functions and only the owner or the signatory of the GovernanceToken smart contracts can call the cancelRequest(), createSignatoryControlRequest(), createThresholdControlRequest(), createTokenSupplyControlRequest(), createTransactionControlRequest(), createWhiteListControlRequest(), updateSignatoryControlRequest(), updateThresholdControlRequest(), updateTokenSupplyControlRequest(), updateWhiteListControlRequest(), vote(), transferOwnership() and renounceOwnership() functions. All the remaining functions in the smart contract are accessible by the public users.

The StableCoin smart contract has 19 write functions and only the owner or the signatory of the GovernanceToken smart contracts can call the cancelRequest(), createSignatoryControlRequest(), createThresholdControlRequest(), createTokenSupplyControlRequest(), createTransactionControlRequest(), updateSignatoryControlRequest(), updateThresholdControlRequest(), updateTokenSupplyControlRequest(), vote(), transferOwnership() and renounceOwnership() functions.
All the remaining functions in the smart contract are accessible by the public users.The smart contracts also utilizes the ERC1967Proxy implementation to upgrade the contract functionality in the future.

3.2 Recommendations

The smart contracts reviewed under the scope for this engagement are developed securely and the developers has shown utmost skill developing the core logic securely. The developer documents and code comments have been highly informative during the code review. We recommend Novatti to periodically conduct code reviews to ensure the smart contracts are free from any future vulnerabilities.

4.0 Technical Analysis

After comprehensive Smart Contract Audit and subsequent retesting, no vulnerabilities or findings were identified in this assessment. The client has taken steps to address any potential vulnerabilities.

5.0 Static Analysis

Static analysis is carried out with the following tools:

- Slither

5.1 Slither

Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

Solidity File: GgovernanceToken.sol

```
    --> GovernanceToken.sol:2079:1:
     |
2079 | contract GovernanceToken is
     | ^ (Relevant source part starts here and spans across multiple lines).


INFO:Detectors:
StableCoin.execute(Common.RequestType,uint256).i (GovernanceToken.sol#1971) is a local variable never initialized
StableCoin.execute(Common.RequestType,uint256).i_scope_0 (GovernanceToken.sol#1996) is a local variable never initialized
GovernanceToken.getThresholds(Common.RequestType).i (GovernanceToken.sol#2218) is a local variable never initialized
GovernanceToken.execute(Common.RequestType,uint256).i (GovernanceToken.sol#2679) is a local variable never initialized
ArrayOps.isElement(address[],address)._isElement (GovernanceToken.sol#1247) is a local variable never initialized
StableCoin.getThresholds(Common.RequestType).i (GovernanceToken.sol#1614) is a local variable never initialized
GovernanceToken.execute(Common.RequestType,uint256).i_scope_0 (GovernanceToken.sol#2704) is a local variable never initialized
GovernanceToken.execute(Common.RequestType,uint256).i_scope_1 (GovernanceToken.sol#2730) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

```
INFO:Detectors:
StableCoin.initialize(string,string,uint8,address,address).governanceToken_ (GovernanceToken.sol#1560) lacks a zero-check on :
              - _governanceTokenAddress = governanceToken_ (GovernanceToken.sol#1571)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
AddressUpgradeable.verifyCallResult(bool,bytes,string) (GovernanceToken.sol#289-309) uses assembly
        - INLINE ASM (GovernanceToken.sol#301-304)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.17', '^0.8.0', '^0.8.1', '^0.8.2']
        - 0.8.17 (GovernanceToken.sol#1214)
        - 0.8.17 (GovernanceToken.sol#1525)
        - 0.8.17 (GovernanceToken.sol#2078)
        - ^0.8.0 (GovernanceToken.sol#6)
        - ^0.8.0 (GovernanceToken.sol#91)
        - ^0.8.0 (GovernanceToken.sol#455)
        - ^0.8.0 (GovernanceToken.sol#493)
        - ^0.8.0 (GovernanceToken.sol#885)
        - ^0.8.0 (GovernanceToken.sol#1001)
        - ^0.8.0 (GovernanceToken.sol#1046)
        - ^0.8.0 (GovernanceToken.sol#1140)
        - ^0.8.1 (GovernanceToken.sol#119)
        - ^0.8.2 (GovernanceToken.sol#317)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

```
INFO:Detectors:
StableCoin.execute(Common.RequestType,uint256) (GovernanceToken.sol#1929-2019) has costly operations inside a loop:
        - signatoryList = ArrayOps.deleteFromArray(signatoryList,signatoryControlRequests[id_].wallets[i]) (GovernanceToken.sol#1980-
1983)
GovernanceToken.execute(Common.RequestType,uint256) (GovernanceToken.sol#2637-2752) has costly operations inside a loop:
        - signatoryList = ArrayOps.deleteFromArray(signatoryList,signatoryControlRequests[id_].wallets[i]) (GovernanceToken.sol#2688-
2691)
GovernanceToken.execute(Common.RequestType,uint256) (GovernanceToken.sol#2637-2752) has costly operations inside a loop:
        - whiteListedUsers = ArrayOps.deleteFromArray(whiteListedUsers,whiteListControlRequests[id_].wallets[i_scope_1]) (GovernanceT
oken.sol#2738-2741)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
INFO:Detectors:
StableCoin.vote(Common.RequestType,uint256,bool) (GovernanceToken.sol#1834-1922) has a high cyclomatic complexity (13).
StableCoin.execute(Common.RequestType,uint256) (GovernanceToken.sol#1929-2019) has a high cyclomatic complexity (15).
GovernanceToken.vote(Common.RequestType,uint256,bool) (GovernanceToken.sol#2526-2630) has a high cyclomatic complexity (15).
GovernanceToken.execute(Common.RequestType,uint256) (GovernanceToken.sol#2637-2752) has a high cyclomatic complexity (18).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-complexity
```

```
INFO:Detectors:
AddressUpgradeable.functionCall(address,bytes) (GovernanceToken.sol#200-202) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (GovernanceToken.sol#210-216) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (GovernanceToken.sol#229-235) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (GovernanceToken.sol#243-254) is never used and should be remo
ved
AddressUpgradeable.functionStaticCall(address,bytes) (GovernanceToken.sol#262-264) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (GovernanceToken.sol#272-281) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (GovernanceToken.sol#175-180) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (GovernanceToken.sol#289-309) is never used and should be removed
ContextUpgradeable.__Context_init() (GovernanceToken.sol#467-468) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (GovernanceToken.sol#470-471) is never used and should be removed
ContextUpgradeable._msgData() (GovernanceToken.sol#476-478) is never used and should be removed
ERC20PausableUpgradeable.__ERC20Pausable_init_unchained() (GovernanceToken.sol#1014-1015) is never used and should be removed
PausableUpgradeable.__Pausable_init() (GovernanceToken.sol#911-913) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (GovernanceToken.sol#1174-1176) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (GovernanceToken.sol#1178-1180) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version^0.8.0 (GovernanceToken.sol#6) allows old versions
Pragma version^0.8.0 (GovernanceToken.sol#91) allows old versions
Pragma version^0.8.1 (GovernanceToken.sol#119) allows old versions
Pragma version^0.8.2 (GovernanceToken.sol#317) allows old versions
Pragma version^0.8.0 (GovernanceToken.sol#455) allows old versions
Pragma version^0.8.0 (GovernanceToken.sol#493) allows old versions
Pragma version^0.8.0 (GovernanceToken.sol#885) allows old versions
Pragma version^0.8.0 (GovernanceToken.sol#1001) allows old versions
Pragma version^0.8.0 (GovernanceToken.sol#1046) allows old versions
Pragma version^0.8.0 (GovernanceToken.sol#1140) allows old versions
Pragma version0.8.17 (GovernanceToken.sol#1214) allows old versions
Pragma version0.8.17 (GovernanceToken.sol#1525) allows old versions
Pragma version0.8.17 (GovernanceToken.sol#2078) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in AddressUpgradeable.sendValue(address,uint256) (GovernanceToken.sol#175-180):
        - (success) = recipient.call{value: amount}() (GovernanceToken.sol#178)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (GovernanceToken.sol#243-254):
        - (success,returndata) = target.call{value: value}(data) (GovernanceToken.sol#252)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (GovernanceToken.sol#272-281):
        - (success,returndata) = target.staticcall(data) (GovernanceToken.sol#279)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
Function ContextUpgradeable.__Context_init() (GovernanceToken.sol#467-468) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (GovernanceToken.sol#470-471) is not in mixedCase
Variable ContextUpgradeable.__gap (GovernanceToken.sol#485) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (GovernanceToken.sol#538-540) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (GovernanceToken.sol#542-545) is not in mixedCase
Variable ERC20Upgradeable.__gap (GovernanceToken.sol#877) is not in mixedCase
Function PausableUpgradeable.__Pausable_init() (GovernanceToken.sol#911-913) is not in mixedCase
Function PausableUpgradeable.__Pausable_init_unchained() (GovernanceToken.sol#915-917) is not in mixedCase
Variable PausableUpgradeable.__gap (GovernanceToken.sol#993) is not in mixedCase
Function ERC20PausableUpgradeable.__ERC20Pausable_init() (GovernanceToken.sol#1010-1012) is not in mixedCase
Function ERC20PausableUpgradeable.__ERC20Pausable_init_unchained() (GovernanceToken.sol#1014-1015) is not in mixedCase
Variable ERC20PausableUpgradeable.__gap (GovernanceToken.sol#1038) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (GovernanceToken.sol#1067-1069) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (GovernanceToken.sol#1071-1073) is not in mixedCase
Variable OwnableUpgradeable.__gap (GovernanceToken.sol#1132) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (GovernanceToken.sol#1174-1176) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (GovernanceToken.sol#1178-1180) is not in mixedCase
Variable ReentrancyGuardUpgradeable.__gap (GovernanceToken.sol#1208) is not in mixedCase
Variable StableCoin._decimals (GovernanceToken.sol#1539) is not in mixedCase
Variable StableCoin._governanceTokenAddress (GovernanceToken.sol#1541) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
INFO:Detectors:
Variable StableCoin.vote(Common.RequestType,uint256,bool).isApproved_scope_0 (GovernanceToken.sol#1862) is too similar to StableCoin.
vote(Common.RequestType,uint256,bool).isApproved_scope_2 (GovernanceToken.sol#1882)
Variable StableCoin.vote(Common.RequestType,uint256,bool).isApproved_scope_2 (GovernanceToken.sol#1882) is too similar to StableCoin.
vote(Common.RequestType,uint256,bool).isApproved_scope_3 (GovernanceToken.sol#1901)
Variable StableCoin.vote(Common.RequestType,uint256,bool).isApproved_scope_0 (GovernanceToken.sol#1862) is too similar to StableCoin.
vote(Common.RequestType,uint256,bool).isApproved_scope_3 (GovernanceToken.sol#1901)
Variable GovernanceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_0 (GovernanceToken.sol#2554) is too similar to Govern
anceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_2 (GovernanceToken.sol#2574)
Variable GovernanceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_2 (GovernanceToken.sol#2574) is too similar to Govern
anceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_3 (GovernanceToken.sol#2593)
Variable GovernanceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_2 (GovernanceToken.sol#2574) is too similar to Govern
anceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_4 (GovernanceToken.sol#2612)
Variable GovernanceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_0 (GovernanceToken.sol#2554) is too similar to Govern
anceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_3 (GovernanceToken.sol#2593)
Variable GovernanceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_3 (GovernanceToken.sol#2593) is too similar to Govern
anceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_4 (GovernanceToken.sol#2612)
Variable GovernanceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_0 (GovernanceToken.sol#2554) is too similar to Govern
anceToken.vote(Common.RequestType,uint256,bool).isApproved_scope_4 (GovernanceToken.sol#2612)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Slither:GovernanceToken.sol analyzed (14 contracts with 85 detectors), 79 result(s) found
```

Solidity File: ProxyAdmin.sol

```
--> proxyadmin.sol


INFO:Detectors:
ERC1967Upgrade._upgradeToAndCall(address,bytes,bool) (proxyadmin.sol#442-448) ignores return value by Address.functionDelegateCall(ne
wImplementation,data) (proxyadmin.sol#446)
ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (proxyadmin.sol#455-483) ignores return value by Address.functionDelegateC
all(newImplementation,data) (proxyadmin.sol#461)
ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (proxyadmin.sol#455-483) ignores return value by Address.functionDelegateC
all(newImplementation,abi.encodeWithSignature(upgradeTo(address),oldImplementation)) (proxyadmin.sol#469-475)
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (proxyadmin.sol#491-497) ignores return value by Address.functionDelegateC
all(IBeacon(newBeacon).implementation(),data) (proxyadmin.sol#495)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
INFO:Detectors:
Modifier TransparentUpgradeableProxy.ifAdmin() (proxyadmin.sol#640-646) does not always execute _; or revertReference: https://github
.com/crytic/slither/wiki/Detector-Documentation#incorrect-modifier
INFO:Detectors:
Reentrancy in ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (proxyadmin.sol#455-483):
        External calls:
        - Address.functionDelegateCall(newImplementation,data) (proxyadmin.sol#461)
        - Address.functionDelegateCall(newImplementation,abi.encodeWithSignature(upgradeTo(address),oldImplementation)) (proxyadmin.s
ol#469-475)
        Event emitted after the call(s):
        - Upgraded(newImplementation) (proxyadmin.sol#481)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
INFO:Detectors:
Proxy._delegate(address) (proxyadmin.sol#23-43) uses assembly
        - INLINE ASM (proxyadmin.sol#25-42)
Address.isContract(address) (proxyadmin.sol#132-141) uses assembly
        - INLINE ASM (proxyadmin.sol#139)
Address._verifyCallResult(bool,bytes,string) (proxyadmin.sol#277-294) uses assembly
        - INLINE ASM (proxyadmin.sol#286-289)
StorageSlot.getAddressSlot(bytes32) (proxyadmin.sol#349-353) uses assembly
        - INLINE ASM (proxyadmin.sol#350-352)
StorageSlot.getBooleanSlot(bytes32) (proxyadmin.sol#358-362) uses assembly
        - INLINE ASM (proxyadmin.sol#359-361)
StorageSlot.getBytes32Slot(bytes32) (proxyadmin.sol#367-371) uses assembly
        - INLINE ASM (proxyadmin.sol#368-370)
StorageSlot.getUint256Slot(bytes32) (proxyadmin.sol#376-380) uses assembly
        - INLINE ASM (proxyadmin.sol#377-379)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.2']
        - ^0.8.0 (proxyadmin.sol#5)
        - ^0.8.0 (proxyadmin.sol#91)
        - ^0.8.0 (proxyadmin.sol#109)
        - ^0.8.0 (proxyadmin.sol#301)
        - ^0.8.0 (proxyadmin.sol#574)
        - ^0.8.0 (proxyadmin.sol#605)
        - ^0.8.0 (proxyadmin.sol#725)
        - ^0.8.0 (proxyadmin.sol#752)
        - ^0.8.0 (proxyadmin.sol#821)
        - ^0.8.2 (proxyadmin.sol#387)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

```
INFO:Detectors:
Address.functionCall(address,bytes) (proxyadmin.sol#185-187) is never used and should be removed
Address.functionCall(address,bytes,string) (proxyadmin.sol#195-197) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (proxyadmin.sol#210-212) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (proxyadmin.sol#220-227) is never used and should be removed
Address.functionStaticCall(address,bytes) (proxyadmin.sol#235-237) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (proxyadmin.sol#245-251) is never used and should be removed
Address.sendValue(address,uint256) (proxyadmin.sol#159-165) is never used and should be removed
Context._msgData() (proxyadmin.sol#742-745) is never used and should be removed
ERC1967Upgrade._getBeacon() (proxyadmin.sol#550-552) is never used and should be removed
ERC1967Upgrade._setBeacon(address) (proxyadmin.sol#557-567) is never used and should be removed
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (proxyadmin.sol#491-497) is never used and should be removed
ERC1967Upgrade._upgradeTo(address) (proxyadmin.sol#432-435) is never used and should be removed
ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (proxyadmin.sol#455-483) is never used and should be removed
StorageSlot.getBooleanSlot(bytes32) (proxyadmin.sol#358-362) is never used and should be removed
StorageSlot.getBytes32Slot(bytes32) (proxyadmin.sol#367-371) is never used and should be removed
StorageSlot.getUint256Slot(bytes32) (proxyadmin.sol#376-380) is never used and should be removed
TransparentUpgradeableProxy._admin() (proxyadmin.sol#708-710) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version^0.8.0 (proxyadmin.sol#5) allows old versions
Pragma version^0.8.0 (proxyadmin.sol#91) allows old versions
Pragma version^0.8.0 (proxyadmin.sol#109) allows old versions
Pragma version^0.8.0 (proxyadmin.sol#301) allows old versions
Pragma version^0.8.2 (proxyadmin.sol#387) allows old versions
Pragma version^0.8.0 (proxyadmin.sol#574) allows old versions
Pragma version^0.8.0 (proxyadmin.sol#605) allows old versions
Pragma version^0.8.0 (proxyadmin.sol#725) allows old versions
Pragma version^0.8.0 (proxyadmin.sol#752) allows old versions
Pragma version^0.8.0 (proxyadmin.sol#821) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (proxyadmin.sol#159-165):
        - (success) = recipient.call{value: amount}() (proxyadmin.sol#163)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (proxyadmin.sol#220-227):
        - (success,returndata) = target.call{value: value}(data) (proxyadmin.sol#225)
Low level call in Address.functionStaticCall(address,bytes,string) (proxyadmin.sol#245-251):
        - (success,returndata) = target.staticcall(data) (proxyadmin.sol#249)
Low level call in Address.functionDelegateCall(address,bytes,string) (proxyadmin.sol#269-275):
        - (success,returndata) = target.delegatecall(data) (proxyadmin.sol#273)
Low level call in ProxyAdmin.getProxyImplementation(TransparentUpgradeableProxy) (proxyadmin.sol#835-841):
        - (success,returndata) = address(proxy).staticcall(0x5c60da1b) (proxyadmin.sol#838)
Low level call in ProxyAdmin.getProxyAdmin(TransparentUpgradeableProxy) (proxyadmin.sol#850-856):
        - (success,returndata) = address(proxy).staticcall(0xf851a440) (proxyadmin.sol#853)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
TransparentUpgradeableProxy (proxyadmin.sol#627-719) should inherit from IBeacon (proxyadmin.sol#96-103)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
INFO:Detectors:
Redundant expression "this (proxyadmin.sol#743)" inContext (proxyadmin.sol#737-746)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Slither:proxyadmin.sol analyzed (10 contracts with 85 detectors), 50 result(s) found
```

Solidity File: StableCoin.sol

```
    --> StableCoin.sol:1530:1:
     |
1530 | contract StableCoin is
     | ^ (Relevant source part starts here and spans across multiple lines).


INFO:Detectors:
ArrayOps.isElement(address[],address)._isElement (StableCoin.sol#1247) is a local variable never initialized
StableCoin.execute(Common.RequestType,uint256).i (StableCoin.sol#1971) is a local variable never initialized
StableCoin.getThresholds(Common.RequestType).i (StableCoin.sol#1614) is a local variable never initialized
StableCoin.execute(Common.RequestType,uint256).i_scope_0 (StableCoin.sol#1996) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

```
INFO:Detectors:
StableCoin.initialize(string,string,uint8,address,address).governanceToken_ (StableCoin.sol#1560) lacks a zero-check on :
                - _governanceTokenAddress = governanceToken_ (StableCoin.sol#1571)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
AddressUpgradeable.verifyCallResult(bool,bytes,string) (StableCoin.sol#289-309) uses assembly
        - INLINE ASM (StableCoin.sol#301-304)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.17', '^0.8.0', '^0.8.1', '^0.8.2']
        - 0.8.17 (StableCoin.sol#1214)
        - 0.8.17 (StableCoin.sol#1525)
        - ^0.8.0 (StableCoin.sol#6)
        - ^0.8.0 (StableCoin.sol#91)
        - ^0.8.0 (StableCoin.sol#455)
        - ^0.8.0 (StableCoin.sol#493)
        - ^0.8.0 (StableCoin.sol#885)
        - ^0.8.0 (StableCoin.sol#1001)
        - ^0.8.0 (StableCoin.sol#1046)
        - ^0.8.0 (StableCoin.sol#1140)
        - ^0.8.1 (StableCoin.sol#119)
        - ^0.8.2 (StableCoin.sol#317)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
StableCoin.execute(Common.RequestType,uint256) (StableCoin.sol#1929-2019) has costly operations inside a loop:
        - signatoryList = ArrayOps.deleteFromArray(signatoryList,signatoryControlRequests[id_].wallets[i]) (StableCoin.sol#1980-1983)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
INFO:Detectors:
StableCoin.vote(Common.RequestType,uint256,bool) (StableCoin.sol#1834-1922) has a high cyclomatic complexity (13).
StableCoin.execute(Common.RequestType,uint256) (StableCoin.sol#1929-2019) has a high cyclomatic complexity (15).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-complexity
INFO:Detectors:
AddressUpgradeable.functionCall(address,bytes) (StableCoin.sol#200-202) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (StableCoin.sol#210-216) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (StableCoin.sol#229-235) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (StableCoin.sol#243-254) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (StableCoin.sol#262-264) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (StableCoin.sol#272-281) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (StableCoin.sol#175-180) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (StableCoin.sol#289-309) is never used and should be removed
ContextUpgradeable.__Context_init() (StableCoin.sol#467-468) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (StableCoin.sol#470-471) is never used and should be removed
ContextUpgradeable._msgData() (StableCoin.sol#476-478) is never used and should be removed
ERC20PausableUpgradeable.__ERC20Pausable_init_unchained() (StableCoin.sol#1014-1015) is never used and should be removed
PausableUpgradeable.__Pausable_init() (StableCoin.sol#911-913) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (StableCoin.sol#1174-1176) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (StableCoin.sol#1178-1180) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version^0.8.0 (StableCoin.sol#6) allows old versions
Pragma version^0.8.0 (StableCoin.sol#91) allows old versions
Pragma version^0.8.1 (StableCoin.sol#119) allows old versions
Pragma version^0.8.2 (StableCoin.sol#317) allows old versions
Pragma version^0.8.0 (StableCoin.sol#455) allows old versions
Pragma version^0.8.0 (StableCoin.sol#493) allows old versions
Pragma version^0.8.0 (StableCoin.sol#885) allows old versions
Pragma version^0.8.0 (StableCoin.sol#1001) allows old versions
Pragma version^0.8.0 (StableCoin.sol#1046) allows old versions
Pragma version^0.8.0 (StableCoin.sol#1140) allows old versions
Pragma version0.8.17 (StableCoin.sol#1214) allows old versions
Pragma version0.8.17 (StableCoin.sol#1525) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in AddressUpgradeable.sendValue(address,uint256) (StableCoin.sol#175-180):
        - (success) = recipient.call{value: amount}() (StableCoin.sol#178)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (StableCoin.sol#243-254):
        - (success,returndata) = target.call{value: value}(data) (StableCoin.sol#252)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (StableCoin.sol#272-281):
        - (success,returndata) = target.staticcall(data) (StableCoin.sol#279)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
Function ContextUpgradeable.__Context_init() (StableCoin.sol#467-468) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (StableCoin.sol#470-471) is not in mixedCase
Variable ContextUpgradeable.__gap (StableCoin.sol#485) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (StableCoin.sol#538-540) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (StableCoin.sol#542-545) is not in mixedCase
Variable ERC20Upgradeable.__gap (StableCoin.sol#877) is not in mixedCase
Function PausableUpgradeable.__Pausable_init() (StableCoin.sol#911-913) is not in mixedCase
Function PausableUpgradeable.__Pausable_init_unchained() (StableCoin.sol#915-917) is not in mixedCase
Variable PausableUpgradeable.__gap (StableCoin.sol#993) is not in mixedCase
Function ERC20PausableUpgradeable.__ERC20Pausable_init() (StableCoin.sol#1010-1012) is not in mixedCase
Function ERC20PausableUpgradeable.__ERC20Pausable_init_unchained() (StableCoin.sol#1014-1015) is not in mixedCase
Variable ERC20PausableUpgradeable.__gap (StableCoin.sol#1038) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (StableCoin.sol#1067-1069) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (StableCoin.sol#1071-1073) is not in mixedCase
Variable OwnableUpgradeable.__gap (StableCoin.sol#1132) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (StableCoin.sol#1174-1176) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (StableCoin.sol#1178-1180) is not in mixedCase
Variable ReentrancyGuardUpgradeable.__gap (StableCoin.sol#1208) is not in mixedCase
Variable StableCoin._decimals (StableCoin.sol#1539) is not in mixedCase
Variable StableCoin._governanceTokenAddress (StableCoin.sol#1541) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable StableCoin.vote(Common.RequestType,uint256,bool).isApproved_scope_0 (StableCoin.sol#1862) is too similar to StableCoin.vote(
Common.RequestType,uint256,bool).isApproved_scope_2 (StableCoin.sol#1882)
Variable StableCoin.vote(Common.RequestType,uint256,bool).isApproved_scope_0 (StableCoin.sol#1862) is too similar to StableCoin.vote(
Common.RequestType,uint256,bool).isApproved_scope_3 (StableCoin.sol#1901)
Variable StableCoin.vote(Common.RequestType,uint256,bool).isApproved_scope_2 (StableCoin.sol#1882) is too similar to StableCoin.vote(
Common.RequestType,uint256,bool).isApproved_scope_3 (StableCoin.sol#1901)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Slither:StableCoin.sol analyzed (13 contracts with 85 detectors), 64 result(s) found
```

6.0 Dynamic Analysis

The following results are the efforts of manual analysis.

Note: The following values for “Result” mean:

- **Positive** indicates that there is no security risk.
- **Negative** indicates that there is a security risk that needs to be remediated.
- **Informational** findings should be followed as a best practice, and they are not visible from the smart contract.
- **Not Applicable** means the attack vector is Not applicable or Not available means the attack vector is Not applicable or Not available means the attack vector is Not applicable or Not available

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	increaseAllowance
Remediation	
References	
Access Specifier	public
Additional Modifiers	No
Input Parameters	spender, addedValue

Function Code	//ERC20Upgradeable.sol function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) { address owner = _msgSender(); _approve(owner, spender, allowance(owner, spender) + addedValue); return true; } function _approve(address owner, address spender, uint256 amount) internal virtual { require(owner != address(0), ""ERC20: approve from the zero address""); require(spender != address(0), ""ERC20: approve to the zero address""); _allowances[owner][spender] = amount; emit Approval(owner, spender, amount); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	updateTokenSupplyControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	id_, amount_, to_
Function Code	function updateTokenSupplyControlRequest(uint256 id_, uint256 amount_, address to_) external onlySignatory { require(tokenSupplyControlRequests[id_].owner != address(0), 'INVALID_REQUEST!'); require(tokenSupplyControlRequests[id_].owner == msg.sender, 'UNAUTHORIZED!'); require(tokenSupplyControlRequests[id_].status == RequestStatus.IN_PROGRESS, 'NOT_ACTIVE!'); require(isWhiteListed[to_], 'NOT_WHITELISTED'); tokenSupplyControlRequests[id_].amount = amount_; tokenSupplyControlRequests[id_].wallet = to_; tokenSupplyControlRequests[id_].approvals = new address[](0); emit RequestUpdated(RequestType.TOKEN_SUPPLY_CONTROL, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable

Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	updateWhiteListControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	id_, users_
Function Code	function updateWhiteListControlRequest(uint256 id_, address[] memory users_) external onlySignatory { require(whiteListControlRequests[id_].owner != address(0), 'INVALID_REQUEST!'); require(whiteListControlRequests[id_].owner == msg.sender, 'UNAUTHORIZED!'); require(whiteListControlRequests[id_].status == RequestStatus.IN_PROGRESS, 'NOT_ACTIVE!'); whiteListControlRequests[id_].wallets = users_; whiteListControlRequests[id_].approvals = new address[](0); emit RequestUpdated(RequestType.WHITELIST_CONTROL, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	vote()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory, nonReentrant
Input Parameters	reqType_, id_, approval_

Function Code

```
function vote(
    RequestType reqType_,
    uint256 id_,
    bool approval_
  ) external onlySignatory nonReentrant {
    if (reqType_ == RequestType.TOKEN_SUPPLY_CONTROL) {
      _isApprovable(tokenSupplyControlRequests[id_].owner,
tokenSupplyControlRequests[id_].status);
      bool isApproved = ArrayOps.isElement(tokenSupplyControlRequests[id_].approvals,
msg.sender);
      if (approval_) {
        require(!isApproved, 'APPROVED!');
        tokenSupplyControlRequests[id_].approvals.push(msg.sender);
      } else {
        require(isApproved, 'NOT_APPROVED!');
        address[] memory updatedApprovals = ArrayOps.deleteFromArray(
          tokenSupplyControlRequests[id_].approvals,
          msg.sender
        );
        tokenSupplyControlRequests[id_].approvals = updatedApprovals;
      }
      tokenSupplyControlRequests[id_].status = tokenSupplyControlRequests[id_].approvals.length
>=
        tokenSupplyControlThresholds[tokenSupplyControlRequests[id_].subType]
        ? RequestStatus.ACCEPTED
        : RequestStatus.IN_PROGRESS;
    } else if (reqType_ == RequestType.TRANSACTION_CONTROL) {
      _isApprovable(transactionControlRequests[id_].owner, transactionControlRequests[id_].status);
```

```
    bool isApproved = ArrayOps.isElement(transactionControlRequests[id_].approvals, msg.sender);
    if (approval_) {
      require(!isApproved, 'APPROVED!');
      transactionControlRequests[id_].approvals.push(msg.sender);
    } else {
      require(isApproved, 'NOT_APPROVED!');
      address[] memory updatedApprovals = ArrayOps.deleteFromArray(
        transactionControlRequests[id_].approvals,
        msg.sender
      );
      transactionControlRequests[id_].approvals = updatedApprovals;
    }
    transactionControlRequests[id_].status = transactionControlRequests[id_].approvals.length >=
      transactionControlThresholds[transactionControlRequests[id_].subType]
      ? RequestStatus.ACCEPTED
      : RequestStatus.IN_PROGRESS;
  } else if (reqType_ == RequestType.SIGNATORY_CONTROL) {
    _isApprovable(signatoryControlRequests[id_].owner, signatoryControlRequests[id_].status);
    bool isApproved = ArrayOps.isElement(signatoryControlRequests[id_].approvals, msg.sender);
    if (approval_) {
      require(!isApproved, 'APPROVED!');
      signatoryControlRequests[id_].approvals.push(msg.sender);
    } else {
      require(isApproved, 'NOT_APPROVED!');
      signatoryControlRequests[id_].approvals = ArrayOps.deleteFromArray(
        signatoryControlRequests[id_].approvals,
        msg.sender
      );
    }
    signatoryControlRequests[id_].status = signatoryControlRequests[id_].approvals.length >=
      signatoryControlThresholds[signatoryControlRequests[id_].subType]
      ? RequestStatus.ACCEPTED
      : RequestStatus.IN_PROGRESS;
  } else if (reqType_ == RequestType.THRESHOLD_CONTROL) {
    _isApprovable(thresholdControlRequests[id_].owner, thresholdControlRequests[id_].status);
    bool isApproved = ArrayOps.isElement(thresholdControlRequests[id_].approvals, msg.sender);
    if (approval_) {
      require(!isApproved, 'APPROVED!');
      thresholdControlRequests[id_].approvals.push(msg.sender);
    } else {
      require(isApproved, 'NOT_APPROVED!');
      thresholdControlRequests[id_].approvals = ArrayOps.deleteFromArray(
        thresholdControlRequests[id_].approvals,
        msg.sender
      );
    }
    thresholdControlRequests[id_].status = thresholdControlRequests[id_].approvals.length >=
      thresholdControlThresholds[thresholdControlRequests[id_].subType]
      ? RequestStatus.ACCEPTED
      : RequestStatus.IN_PROGRESS;
  } else {
```

```
      revert('UNKNOWN_REQUEST!');
    }
    emit RequestApproval(reqType_, id_, msg.sender, approval_);
  }
//Common.sol
function _isApprovable(address owner, RequestStatus status) internal pure {
    require(owner != address(0), 'INVALID_REQUEST!');
    require(status == RequestStatus.IN_PROGRESS || status == RequestStatus.ACCEPTED,
'NOT_ACTIVE!');
  }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	approve()
Remediation	
References	
Access Specifier	public
Additional Modifiers	No
Input Parameters	spender, amount
Function Code	//ERC20Upgradeable.sol function approve(address spender, uint256 amount) public virtual override returns (bool) { address owner = _msgSender(); _approve(owner, spender, amount); return true; } function _approve(address owner, address spender, uint256 amount) internal virtual { require(owner != address(0), "ERC20: approve from the zero address"); require(spender != address(0), "ERC20: approve to the zero address"); _allowances[owner][spender] = amount; emit Approval(owner, spender, amount); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable

Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	cancelRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory, nonReentrant
Input Parameters	reqType_, id_
Function Code	function cancelRequest(RequestType reqType_, uint256 id_) external onlySignatory nonReentrant { if (reqType_ == RequestType.TOKEN_SUPPLY_CONTROL) { _isCancellable(tokenSupplyControlRequests[id_].owner, tokenSupplyControlRequests[id_].status); tokenSupplyControlRequests[id_].status = RequestStatus.CANCELLED; } else if (reqType_ == RequestType.TRANSACTION_CONTROL) { _isCancellable(transactionControlRequests[id_].owner, transactionControlRequests[id_].status); transactionControlRequests[id_].status = RequestStatus.CANCELLED; } else if (reqType_ == RequestType.SIGNATORY_CONTROL) { _isCancellable(signatoryControlRequests[id_].owner, signatoryControlRequests[id_].status); signatoryControlRequests[id_].status = RequestStatus.CANCELLED; } else if (reqType_ == RequestType.THRESHOLD_CONTROL) { _isCancellable(thresholdControlRequests[id_].owner, thresholdControlRequests[id_].status); thresholdControlRequests[id_].status = RequestStatus.CANCELLED; } else { revert('UNKNOWN_REQUEST!'); } emit RequestCancelled(reqType_, id_); } //Common.sol function _isCancellable(address owner, RequestStatus status) internal view { require(owner != address(0), 'INVALID_REQUEST!'); require(owner == msg.sender, 'UNAUTHORIZED!'); require(status == RequestStatus.IN_PROGRESS || status == RequestStatus.ACCEPTED, 'NOT_ACTIVE!'); }
Reentrancy	Not Vulnerable

Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	createSignatoryControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqSubType_, id_, users_
Function Code	function createSignatoryControlRequest(SignatoryControlRequestType reqSubType_, uint256 id_, address[] memory users_) external onlySignatory { require(signatoryControlRequests[id_].owner == address(0), 'INVALID_REQUEST!'); signatoryControlRequests[id_].id = id_; signatoryControlRequests[id_].subType = reqSubType_; signatoryControlRequests[id_].wallets = users_; signatoryControlRequests[id_].owner = msg.sender; signatoryControlRequests[id_].status = RequestStatus.IN_PROGRESS; emit RequestCreated(RequestType.SIGNATORY_CONTROL, uint256(reqSubType_), msg.sender, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable

Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	createThresholdControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqType_, id_, thresholds_

Function Code

```
function createThresholdControlRequest(
    RequestType reqType_,
    uint256 id_,
    uint256[] memory thresholds_
  ) external onlySignatory {
    require(thresholdControlRequests[id_].owner == address(0), 'INVALID_REQUEST!');
    require(thresholds_.length == requestTypeCount[reqType_], 'INVALID_THRESHOLD_COUNTS!');
    for(uint256 i;i < thresholds_.length;i++){
      require(thresholds_[i] > 0, 'INVALID_THRESHOLD!');
    }

    thresholdControlRequests[id_].id = id_;
    thresholdControlRequests[id_].reqType = reqType_;
    thresholdControlRequests[id_].subType = ThresholdControlRequestType.UPDATE;
    thresholdControlRequests[id_].thresholds = thresholds_;
    thresholdControlRequests[id_].owner = msg.sender;
    thresholdControlRequests[id_].status = RequestStatus.IN_PROGRESS;

    emit RequestCreated(RequestType.THRESHOLD_CONTROL,
uint256(ThresholdControlRequestType.UPDATE), msg.sender, id_);
  }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable

Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	createTokenSupplyControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqSubType_, id_, amount_, to_
Function Code	function createTokenSupplyControlRequest(TokenSupplyControlRequestType reqSubType_, uint256 id_, uint256 amount_, address to_) external onlySignatory { require(tokenSupplyControlRequests[id_].owner == address(0), 'INVALID_REQUEST!'); tokenSupplyControlRequests[id_].id = id_; tokenSupplyControlRequests[id_].subType = reqSubType_; tokenSupplyControlRequests[id_].amount = amount_; tokenSupplyControlRequests[id_].wallet = to_; tokenSupplyControlRequests[id_].owner = msg.sender; tokenSupplyControlRequests[id_].status = RequestStatus.IN_PROGRESS; emit RequestCreated(RequestType.TOKEN_SUPPLY_CONTROL, uint256(reqSubType_), msg.sender, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable

Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	createTransactionControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqSubType_, id_
Function Code	function createTransactionControlRequest(TransactionControlRequestType reqSubType_, uint256 id_) external onlySignatory { require(transactionControlRequests[id_].owner == address(0), 'INVALID_REQUEST!'); transactionControlRequests[id_].id = id_; transactionControlRequests[id_].subType = reqSubType_; transactionControlRequests[id_].owner = msg.sender; transactionControlRequests[id_].status = RequestStatus.IN_PROGRESS; emit RequestCreated(RequestType.TRANSACTION_CONTROL, uint256(reqSubType_), msg.sender, id_); } event RequestCreated(RequestType indexed reqType, uint256 indexed subType, address indexed ownerAddress, uint256 reqId)
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable

Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	decreaseAllowance()
Remediation	
References	
Access Specifier	public
Additional Modifiers	No
Input Parameters	spender, subtractedValue
Function Code	

```
//ERC20Upgradeable.sol
function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool)
{
        address owner = _msgSender();
        uint256 currentAllowance = allowance(owner, spender);
        require(currentAllowance >= subtractedValue, ""ERC20: decreased allowance below zero"");
        unchecked {
            _approve(owner, spender, currentAllowance - subtractedValue);
        }
        return true;
    }

function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), ""ERC20: approve from the zero address"");
        require(spender != address(0), ""ERC20: approve to the zero address"");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable

Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	execute()
Remediation	
References	
Access Specifier	external
Additional Modifiers	No
Input Parameters	reqType_, id_

Function Code

```
function execute(RequestType reqType_, uint256 id_) external {
    if (reqType_ == RequestType.TOKEN_SUPPLY_CONTROL) {
      _isExecutable(tokenSupplyControlRequests[id_].owner,
tokenSupplyControlRequests[id_].status);

      if (TokenSupplyControlRequestType.MINT == tokenSupplyControlRequests[id_].subType) {
        _mint(tokenSupplyControlRequests[id_].wallet, tokenSupplyControlRequests[id_].amount);
      } else {
        if (tokenSupplyControlRequests[id_].wallet == tokenSupplyControlRequests[id_].owner) {
          _burn(tokenSupplyControlRequests[id_].owner, tokenSupplyControlRequests[id_].amount);
        } else if (tokenSupplyControlRequests[id_].wallet == address(this)) {
          _burn(address(this), tokenSupplyControlRequests[id_].amount);
        } else {
          require(
            allowance(
              tokenSupplyControlRequests[id_].wallet,
              tokenSupplyControlRequests[id_].owner
            ) >= tokenSupplyControlRequests[id_].amount,
            'INSUFFICIENT_ALLOWANCE!'
          );
          _spendAllowance(
            tokenSupplyControlRequests[id_].wallet,
            tokenSupplyControlRequests[id_].owner,
            tokenSupplyControlRequests[id_].amount
          );
          _burn(tokenSupplyControlRequests[id_].wallet, tokenSupplyControlRequests[id_].amount);
        }
      }
```

```
    tokenSupplyControlRequests[id_].status = RequestStatus.EXECUTED;
  } else if (reqType_ == RequestType.TRANSACTION_CONTROL) {
    _isExecutable(transactionControlRequests[id_].owner, transactionControlRequests[id_].status);

    if (TransactionControlRequestType.PAUSE == transactionControlRequests[id_].subType) {
      _pause();
    } else {
      _unpause();
    }

    transactionControlRequests[id_].status = RequestStatus.EXECUTED;
  } else if (reqType_ == RequestType.SIGNATORY_CONTROL) {
    _isExecutable(signatoryControlRequests[id_].owner, signatoryControlRequests[id_].status);

    for (uint256 i; i < signatoryControlRequests[id_].wallets.length; i++) {
      if (SignatoryControlRequestType.ADD == signatoryControlRequests[id_].subType) {
        require(!isSignatory[signatoryControlRequests[id_].wallets[i]], 'EXISTING!');
        isSignatory[signatoryControlRequests[id_].wallets[i]] = true;
        signatoryList.push(signatoryControlRequests[id_].wallets[i]);
      } else {
        require(signatoryList.length > 1, 'LAST_SIGNATORY!');
        require(isSignatory[signatoryControlRequests[id_].wallets[i]], 'UNKNOWN!');
        isSignatory[signatoryControlRequests[id_].wallets[i]] = false;
        signatoryList = ArrayOps.deleteFromArray(
          signatoryList,
          signatoryControlRequests[id_].wallets[i]
        );
      }
    }

    signatoryControlRequests[id_].status = RequestStatus.EXECUTED;
    emit SignatoriesUpdated(
      signatoryControlRequests[id_].subType,
      id_,
      signatoryControlRequests[id_].wallets
    );
  } else if (reqType_ == RequestType.THRESHOLD_CONTROL) {
    _isExecutable(thresholdControlRequests[id_].owner, thresholdControlRequests[id_].status);

    for (uint256 i; i < requestTypeCount[thresholdControlRequests[id_].reqType]; i++) {
      if (thresholdControlRequests[id_].reqType == RequestType.TOKEN_SUPPLY_CONTROL) {
        tokenSupplyControlThresholds[TokenSupplyControlRequestType(i)] =
thresholdControlRequests[
          id_
        ].thresholds[i];
      } else if (thresholdControlRequests[id_].reqType == RequestType.TRANSACTION_CONTROL) {
        transactionControlThresholds[TransactionControlRequestType(i)] =
thresholdControlRequests[
          id_
        ].thresholds[i];
```

```
        } else if (thresholdControlRequests[id_].reqType == RequestType.SIGNATORY_CONTROL) {
          signatoryControlThresholds[SignatoryControlRequestType(i)] =
thresholdControlRequests[id_]
            .thresholds[i];
        } else {
          thresholdControlThresholds[ThresholdControlRequestType(i)] =
thresholdControlRequests[id_]
            .thresholds[i];
        }
      }

      thresholdControlRequests[id_].status = RequestStatus.EXECUTED;
      emit ThresholdUpdated(thresholdControlRequests[id_].reqType, id_,
thresholdControlRequests[id_].thresholds);
    } else {
      revert('UNKNOWN_REQUEST!');
    }
  }

//Common.sol
function _isExecutable(address owner, RequestStatus status) internal view {
    require(owner != address(0), 'INVALID_REQUEST!');
    require(owner == msg.sender, 'UNAUTHORIZED!');
    require(status == RequestStatus.ACCEPTED, 'NOT_APPROVED!');
  }
//ERC20Upgradeable.sol
function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");
        _beforeTokenTransfer(address(0), account, amount);
        _totalSupply += amount;
        _balances[account] += amount;
        emit Transfer(address(0), account, amount);
        _afterTokenTransfer(address(0), account, amount);
    }

//ERC20Upgradeable.sol
function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");
        _beforeTokenTransfer(account, address(0), amount);
        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
        }
        _totalSupply -= amount;
        emit Transfer(account, address(0), amount);
        _afterTokenTransfer(account, address(0), amount);
    }
```

Reentrancy	Not Vulnerable

Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	updateThresholdControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	id_, thresholds_
Function Code	function updateThresholdControlRequest(uint256 id_, uint256[] memory thresholds_) external onlySignatory { require(thresholdControlRequests[id_].owner != address(0), 'INVALID_REQUEST!'); require(thresholdControlRequests[id_].owner == msg.sender, 'UNAUTHORIZED!'); require(thresholdControlRequests[id_].status == RequestStatus.IN_PROGRESS, 'NOT_ACTIVE!'); require(thresholds_.length == requestTypeCount[thresholdControlRequests[id_].reqType], 'INVALID_THRESHOLD_COUNTS!'); for(uint256 i;i < thresholds_.length;i++){ require(thresholds_[i] > 0, 'INVALID_THRESHOLD!'); } thresholdControlRequests[id_].thresholds = thresholds_; thresholdControlRequests[id_].approvals = new address[](0); emit RequestUpdated(RequestType.THRESHOLD_CONTROL, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable

Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	initialize()
Remediation	
References	
Access Specifier	public
Additional Modifiers	initializer
Input Parameters	name_, symbol_, decimals_, governanceToken_, owner_

Function Code

```
function initialize(
    string memory name_,
    string memory symbol_,
    uint8 decimals_,
    address governanceToken_,
    address owner_
  ) public initializer {
    __ERC20_init(name_, symbol_);
    __ERC20Pausable_init();
    __Ownable_init();

    isSignatory[owner_] = true;
    signatoryList.push(owner_);

    _decimals = decimals_;
    _governanceTokenAddress = governanceToken_;
    _setRequestTypeCount();
    _setDefaultThresholds();

    _transferOwnership(owner_);
  }

//Common.sol
function _setRequestTypeCount() internal {
    requestTypeCount[RequestType.TOKEN_SUPPLY_CONTROL] = 2;
    requestTypeCount[RequestType.TRANSACTION_CONTROL] = 2;
    requestTypeCount[RequestType.SIGNATORY_CONTROL] = 2;
    requestTypeCount[RequestType.THRESHOLD_CONTROL] = 1;
    requestTypeCount[RequestType.WHITELIST_CONTROL] = 2;
  }

//StableCoin.sol
function _setDefaultThresholds() private {
    tokenSupplyControlThresholds[TokenSupplyControlRequestType.BURN] = 1;
    tokenSupplyControlThresholds[TokenSupplyControlRequestType.MINT] = 1;
    transactionControlThresholds[TransactionControlRequestType.PAUSE] = 1;
    transactionControlThresholds[TransactionControlRequestType.UNPAUSE] = 1;
    signatoryControlThresholds[SignatoryControlRequestType.ADD] = 1;
    signatoryControlThresholds[SignatoryControlRequestType.REMOVE] = 1;
    thresholdControlThresholds[ThresholdControlRequestType.UPDATE] = 1;
  }

//OwnableUpgradeable.sol
function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	renounceOwnership()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	No
Function Code	//OwnableUpgradeable.sol function renounceOwnership() public virtual onlyOwner { _transferOwnership(address(0)); } function _transferOwnership(address newOwner) internal virtual { address oldOwner = _owner; _owner = newOwner; emit OwnershipTransferred(oldOwner, newOwner); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	swap()
Remediation	
References	
Access Specifier	external
Additional Modifiers	nonReentrant
Input Parameters	user_, amount_
Function Code	function swap(address user_, uint256 amount_) external nonReentrant { require(msg.sender == _governanceTokenAddress, 'UNAUTHORIZED!'); _mint(user_, amount_); } //ERC20Upgradeable.sol function _mint(address account, uint256 amount) internal virtual { require(account != address(0), ""ERC20: mint to the zero address""); _beforeTokenTransfer(address(0), account, amount); _totalSupply += amount; _balances[account] += amount; emit Transfer(address(0), account, amount); _afterTokenTransfer(address(0), account, amount); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable

Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	transfer()
Remediation	
References	
Access Specifier	public
Additional Modifiers	No
Input Parameters	to, amount
Function Code	

```
//ERC20Upgradeable.sol
function transfer(address to, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _transfer(owner, to, amount);
        return true;
    }

function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {
        require(from != address(0), ""ERC20: transfer from the zero address"");
        require(to != address(0), ""ERC20: transfer to the zero address"");

        _beforeTokenTransfer(from, to, amount);

        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, ""ERC20: transfer amount exceeds balance"");
        unchecked {
            _balances[from] = fromBalance - amount;
        }
        _balances[to] += amount;

        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	transferFrom()
Remediation	
References	
Access Specifier	public
Additional Modifiers	No
Input Parameters	from, to, amount

Function Code

```
//ERC20Upgradeable.sol
function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual override returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, amount);
        return true;
    }

function _spendAllowance(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, ""ERC20: insufficient allowance"");
            unchecked {
                _approve(owner, spender, currentAllowance - amount);
            }
        }
    }

//ERC20Upgradeable.sol
function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {
        require(from != address(0), ""ERC20: transfer from the zero address"");
        require(to != address(0), ""ERC20: transfer to the zero address"");

        _beforeTokenTransfer(from, to, amount);

        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, ""ERC20: transfer amount exceeds balance"");
        unchecked {
            _balances[from] = fromBalance - amount;
        }
        _balances[to] += amount;

        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	transferOwnership()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	newOwner
Function Code	//ERC20Upgradeable.sol function transferOwnership(address newOwner) public virtual onlyOwner { require(newOwner != address(0), "Ownable: new owner is the zero address""); _transferOwnership(newOwner); } function _transferOwnership(address newOwner) internal virtual { address oldOwner = _owner; _owner = newOwner; emit OwnershipTransferred(oldOwner, newOwner); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	updateSignatoryControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	id_, users_
Function Code	function updateSignatoryControlRequest(uint256 id_, address[] memory users_) external onlySignatory { require(signatoryControlRequests[id_].owner != address(0), 'INVALID_REQUEST!'); require(signatoryControlRequests[id_].owner == msg.sender, 'UNAUTHORIZED!'); require(signatoryControlRequests[id_].status == RequestStatus.IN_PROGRESS, 'NOT_ACTIVE!'); signatoryControlRequests[id_].wallets = users_; signatoryControlRequests[id_].approvals = new address[](0); emit RequestUpdated(RequestType.SIGNATORY_CONTROL, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	updateThresholdControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	id_, thresholds_
Function Code	function updateThresholdControlRequest(uint256 id_, uint256[] memory thresholds_) external onlySignatory { require(thresholdControlRequests[id_].owner != address(0), 'INVALID_REQUEST!'); require(thresholdControlRequests[id_].owner == msg.sender, 'UNAUTHORIZED!'); require(thresholdControlRequests[id_].status == RequestStatus.IN_PROGRESS, 'NOT_ACTIVE!'); require(thresholds_.length == requestTypeCount[thresholdControlRequests[id_].reqType], 'INVALID_THRESHOLD_COUNTS!'); for(uint256 i;i < thresholds_.length;i++){ require(thresholds_[i] > 0, 'INVALID_THRESHOLD!'); } thresholdControlRequests[id_].thresholds = thresholds_; thresholdControlRequests[id_].approvals = new address[](0); emit RequestUpdated(RequestType.THRESHOLD_CONTROL, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable

Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	updateTokenSupplyControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	id_, amount_, to_
Function Code	function updateTokenSupplyControlRequest(uint256 id_, uint256 amount_, address to_) external onlySignatory { require(tokenSupplyControlRequests[id_].owner != address(0), 'INVALID_REQUEST!'); require(tokenSupplyControlRequests[id_].owner == msg.sender, 'UNAUTHORIZED!'); require(tokenSupplyControlRequests[id_].status == RequestStatus.IN_PROGRESS, 'NOT_ACTIVE!'); tokenSupplyControlRequests[id_].amount = amount_; tokenSupplyControlRequests[id_].wallet = to_; tokenSupplyControlRequests[id_].approvals = new address[](0); emit RequestUpdated(RequestType.TOKEN_SUPPLY_CONTROL, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable

Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - StableCoin.sol

Result	
Description	
Location	
Observation	
Function Name	vote()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory, nonReentrant
Input Parameters	reqType_, id_, approval_
Function Code	

```
function vote(
    RequestType reqType_,
    uint256 id_,
    bool approval_
  ) external onlySignatory nonReentrant {
    if (reqType_ == RequestType.TOKEN_SUPPLY_CONTROL) {
      _isApprovable(tokenSupplyControlRequests[id_].owner,
tokenSupplyControlRequests[id_].status);

      bool isApproved = ArrayOps.isElement(tokenSupplyControlRequests[id_].approvals,
msg.sender);
      if (approval_) {
        require(!isApproved, 'APPROVED!');
        tokenSupplyControlRequests[id_].approvals.push(msg.sender);
      } else {
        require(isApproved, 'NOT_APPROVED!');
        address[] memory updatedApprovals = ArrayOps.deleteFromArray(
          tokenSupplyControlRequests[id_].approvals,
          msg.sender
        );
        tokenSupplyControlRequests[id_].approvals = updatedApprovals;
      }

      tokenSupplyControlRequests[id_].status = tokenSupplyControlRequests[id_].approvals.length
>=
        tokenSupplyControlThresholds[tokenSupplyControlRequests[id_].subType]
        ? RequestStatus.ACCEPTED
        : RequestStatus.IN_PROGRESS;
```

```
} else if (reqType_ == RequestType.TRANSACTION_CONTROL) {
 _isApprovable(transactionControlRequests[id_].owner, transactionControlRequests[id_].status);

 bool isApproved = ArrayOps.isElement(transactionControlRequests[id_].approvals, msg.sender);
 if (approval_) {
  require(!isApproved, 'APPROVED!');
  transactionControlRequests[id_].approvals.push(msg.sender);
 } else {
  require(isApproved, 'NOT_APPROVED!');
  address[] memory updatedApprovals = ArrayOps.deleteFromArray(
   transactionControlRequests[id_].approvals,
   msg.sender
  );
  transactionControlRequests[id_].approvals = updatedApprovals;
 }

 transactionControlRequests[id_].status = transactionControlRequests[id_].approvals.length >=
  transactionControlThresholds[transactionControlRequests[id_].subType]
  ? RequestStatus.ACCEPTED
  : RequestStatus.IN_PROGRESS;
} else if (reqType_ == RequestType.SIGNATORY_CONTROL) {
 _isApprovable(signatoryControlRequests[id_].owner, signatoryControlRequests[id_].status);

 bool isApproved = ArrayOps.isElement(signatoryControlRequests[id_].approvals, msg.sender);
 if (approval_) {
  require(!isApproved, 'APPROVED!');
  signatoryControlRequests[id_].approvals.push(msg.sender);
 } else {
  require(isApproved, 'NOT_APPROVED!');
  signatoryControlRequests[id_].approvals = ArrayOps.deleteFromArray(
   signatoryControlRequests[id_].approvals,
   msg.sender
  );
 }

 signatoryControlRequests[id_].status = signatoryControlRequests[id_].approvals.length >=
  signatoryControlThresholds[signatoryControlRequests[id_].subType]
  ? RequestStatus.ACCEPTED
  : RequestStatus.IN_PROGRESS;
} else if (reqType_ == RequestType.THRESHOLD_CONTROL) {
 _isApprovable(thresholdControlRequests[id_].owner, thresholdControlRequests[id_].status);

 bool isApproved = ArrayOps.isElement(thresholdControlRequests[id_].approvals, msg.sender);
 if (approval_) {
  require(!isApproved, 'APPROVED!');
  thresholdControlRequests[id_].approvals.push(msg.sender);
 } else {
  require(isApproved, 'NOT_APPROVED!');
  thresholdControlRequests[id_].approvals = ArrayOps.deleteFromArray(
   thresholdControlRequests[id_].approvals,
   msg.sender
```

```
      );
    }

    thresholdControlRequests[id_].status = thresholdControlRequests[id_].approvals.length >=
      thresholdControlThresholds[thresholdControlRequests[id_].subType]
      ? RequestStatus.ACCEPTED
      : RequestStatus.IN_PROGRESS;
  } else {
    revert('UNKNOWN_REQUEST!');
  }

  emit RequestApproval(reqType_, id_, msg.sender, approval_);
 }

//Common.sol
function _isApprovable(address owner, RequestStatus status) internal pure {
   require(owner != address(0), 'INVALID_REQUEST!');
   require(status == RequestStatus.IN_PROGRESS || status == RequestStatus.ACCEPTED,
'NOT_ACTIVE!');
 }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	createWhiteListControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqSubType_, id_, users_
Function Code	function createWhiteListControlRequest(WhiteListControlRequestType reqSubType_, uint256 id_, address[] memory users_) external onlySignatory { require(whiteListControlRequests[id_].owner == address(0), 'INVALID_REQUEST!'); whiteListControlRequests[id_].id = id_; whiteListControlRequests[id_].subType = reqSubType_; whiteListControlRequests[id_].wallets = users_; whiteListControlRequests[id_].owner = msg.sender; whiteListControlRequests[id_].status = RequestStatus.IN_PROGRESS; emit RequestCreated(RequestType.WHITELIST_CONTROL, uint256(reqSubType_), msg.sender, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable

Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - ProxyAdmin.sol

Result	
Description	
Location	
Observation	
Function Name	renounceOwnership()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	NA
Function Code	function renounceOwnership() public virtual onlyOwner { emit OwnershipTransferred(_owner, address(0)); _owner = address(0); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - ProxyAdmin.sol

Result	
Description	
Location	
Observation	
Function Name	transferOwnership()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	newOwner
Function Code	function transferOwnership(address newOwner) public virtual onlyOwner { require(newOwner != address(0), "Ownable: new owner is the zero address"); emit OwnershipTransferred(_owner, newOwner); _owner = newOwner; }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - ProxyAdmin.sol

Result	
Description	
Location	
Observation	
Function Name	upgrade()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	proxy, implementation
Function Code	function upgrade(TransparentUpgradeableProxy proxy, address implementation) public virtual onlyOwner { proxy.upgradeTo(implementation); } //ERC1967Upgrade.sol function upgradeTo(address newImplementation) external ifAdmin { _upgradeToAndCall(newImplementation, bytes(""), false); } //ERC1967Upgrade.sol function _upgradeToAndCall(address newImplementation, bytes memory data, bool forceCall) internal { _setImplementation(newImplementation); emit Upgraded(newImplementation); if (data.length > 0 || forceCall) { Address.functionDelegateCall(newImplementation, data); } } function _setImplementation(address newImplementation) private { require(Address.isContract(newImplementation), "ERC1967: new implementation is not a contract"); StorageSlot.getAddressSlot(_IMPLEMENTATION_SLOT).value = newImplementation; }
Reentrancy	Not Vulnerable

Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - ProxyAdmin.sol

Result	
Description	
Location	
Observation	
Function Name	upgradeAndCall()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	proxy, implementation, data
Function Code	function upgradeAndCall(TransparentUpgradeableProxy proxy, address implementation, bytes memory data) public payable virtual onlyOwner { proxy.upgradeToAndCall{value: msg.value}(implementation, data); } //TransparentUpgradeableProxy.sol function upgradeToAndCall(address newImplementation, bytes calldata data) external payable ifAdmin { _upgradeToAndCall(newImplementation, data, true); } //ERC1967Upgrade.sol function _upgradeToAndCall(address newImplementation, bytes memory data, bool forceCall) internal { _setImplementation(newImplementation); emit Upgraded(newImplementation); if (data.length > 0 || forceCall) { Address.functionDelegateCall(newImplementation, data); } }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable

Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	approve()
Remediation	
References	
Access Specifier	public
Additional Modifiers	NA
Input Parameters	spender, amount
Function Code	

```
//ERC20Upgradeable.sol
function approve(address spender, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }

function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable

Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	cancelRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory, nonReentrant
Input Parameters	reqType_, id_

Function Code	//Common.sol function cancelRequest(RequestType reqType_, uint256 id_) external onlySignatory nonReentrant { if (reqType_ == RequestType.TOKEN_SUPPLY_CONTROL) { _isCancellable(tokenSupplyControlRequests[id_].owner, tokenSupplyControlRequests[id_].status); tokenSupplyControlRequests[id_].status = RequestStatus.CANCELLED; } else if (reqType_ == RequestType.TRANSACTION_CONTROL) { _isCancellable(transactionControlRequests[id_].owner, transactionControlRequests[id_].status); transactionControlRequests[id_].status = RequestStatus.CANCELLED; } else if (reqType_ == RequestType.SIGNATORY_CONTROL) { _isCancellable(signatoryControlRequests[id_].owner, signatoryControlRequests[id_].status); signatoryControlRequests[id_].status = RequestStatus.CANCELLED; } else if (reqType_ == RequestType.THRESHOLD_CONTROL) { _isCancellable(thresholdControlRequests[id_].owner, thresholdControlRequests[id_].status); thresholdControlRequests[id_].status = RequestStatus.CANCELLED; } else { _isCancellable(whiteListControlRequests[id_].owner, whiteListControlRequests[id_].status); whiteListControlRequests[id_].status = RequestStatus.CANCELLED; } emit RequestCancelled(reqType_, id_); } function _isCancellable(address owner, RequestStatus status) internal view { require(owner != address(0), 'INVALID_REQUEST!'); require(owner == msg.sender, 'UNAUTHORIZED!'); require(status == RequestStatus.IN_PROGRESS || status == RequestStatus.ACCEPTED, 'NOT_ACTIVE!'); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	createSignatoryControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqSubType_, id_, users_
Function Code	function createSignatoryControlRequest(SignatoryControlRequestType reqSubType_, uint256 id_, address[] memory users_) external onlySignatory { require(signatoryControlRequests[id_].owner == address(0), 'INVALID_REQUEST!'); signatoryControlRequests[id_].id = id_; signatoryControlRequests[id_].subType = reqSubType_; signatoryControlRequests[id_].wallets = users_; signatoryControlRequests[id_].owner = msg.sender; signatoryControlRequests[id_].status = RequestStatus.IN_PROGRESS; emit RequestCreated(RequestType.SIGNATORY_CONTROL, uint256(reqSubType_), msg.sender, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable

Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	createThresholdControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqType_, id_, thresholds_

Function Code

```
function createThresholdControlRequest(
    RequestType reqType_,
    uint256 id_,
    uint256[] memory thresholds_
  ) external onlySignatory {
    require(thresholdControlRequests[id_].owner == address(0), 'INVALID_REQUEST!');
    require(thresholds_.length == requestTypeCount[reqType_], 'INVALID_THRESHOLD_COUNTS!');
    for(uint256 i;i < thresholds_.length;i++){
      require(thresholds_[i] > 0, 'INVALID_THRESHOLD!');
    }

    thresholdControlRequests[id_].id = id_;
    thresholdControlRequests[id_].reqType = reqType_;
    thresholdControlRequests[id_].subType = ThresholdControlRequestType.UPDATE;
    thresholdControlRequests[id_].thresholds = thresholds_;
    thresholdControlRequests[id_].owner = msg.sender;
    thresholdControlRequests[id_].status = RequestStatus.IN_PROGRESS;

    emit RequestCreated(RequestType.THRESHOLD_CONTROL,
uint256(ThresholdControlRequestType.UPDATE), msg.sender, id_);
  }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable

Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	createTokenSupplyControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqSubType_, id_, amount_, to_
Function Code	function createTokenSupplyControlRequest(TokenSupplyControlRequestType reqSubType_, uint256 id_, uint256 amount_, address to_) external onlySignatory { require(tokenSupplyControlRequests[id_].owner == address(0), 'INVALID_REQUEST!'); require(isWhiteListed[to_], 'NOT_WHITELISTED'); tokenSupplyControlRequests[id_].id = id_; tokenSupplyControlRequests[id_].subType = reqSubType_; tokenSupplyControlRequests[id_].amount = amount_; tokenSupplyControlRequests[id_].wallet = to_; tokenSupplyControlRequests[id_].owner = msg.sender; tokenSupplyControlRequests[id_].status = RequestStatus.IN_PROGRESS; emit RequestCreated(RequestType.TOKEN_SUPPLY_CONTROL, uint256(reqSubType_), msg.sender, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable

Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	createTransactionControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	reqSubType_, uint256 id_
Function Code	function createTransactionControlRequest(TransactionControlRequestType reqSubType_, uint256 id_) external onlySignatory { require(transactionControlRequests[id_].owner == address(0), 'INVALID_REQUEST!'); transactionControlRequests[id_].id = id_; transactionControlRequests[id_].subType = reqSubType_; transactionControlRequests[id_].owner = msg.sender; transactionControlRequests[id_].status = RequestStatus.IN_PROGRESS; emit RequestCreated(RequestType.TRANSACTION_CONTROL, uint256(reqSubType_), msg.sender, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable

Secure	Yes

Smart Contract - ProxyAdmin.sol

Result	
Description	
Location	
Observation	
Function Name	changeProxyAdmin()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	proxy, newAdmin
Function Code	function changeProxyAdmin(TransparentUpgradeableProxy proxy, address newAdmin) public virtual onlyOwner { proxy.changeAdmin(newAdmin); } //TransparentUpgradeableProxy.sol function changeAdmin(address newAdmin) external virtual ifAdmin { _changeAdmin(newAdmin); } //ERC1967Upgrade.sol function _changeAdmin(address newAdmin) internal { emit AdminChanged(_getAdmin(), newAdmin); _setAdmin(newAdmin); } //ERC1967Upgrade.sol function _setAdmin(address newAdmin) private { require(newAdmin != address(0), "ERC1967: new admin is the zero address"); StorageSlot.getAddressSlot(_ADMIN_SLOT).value = newAdmin; } function getAddressSlot(bytes32 slot) internal pure returns (AddressSlot storage r) { assembly { r.slot := slot } }
Reentrancy	Not Vulnerable

Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	decreaseAllowance()
Remediation	
References	
Access Specifier	public
Additional Modifiers	NA
Input Parameters	spender, subtractedValue
Function Code	

```
//ERC20Upgradeable.sol
function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool)
{
        address owner = _msgSender();
        uint256 currentAllowance = allowance(owner, spender);
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
        unchecked {
            _approve(owner, spender, currentAllowance - subtractedValue);
        }

        return true;
    }

function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable

Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	execute()
Remediation	
References	
Access Specifier	external
Additional Modifiers	No
Input Parameters	reqType_, id_

Function Code

```
function execute(RequestType reqType_, uint256 id_) external {
    if (reqType_ == RequestType.TOKEN_SUPPLY_CONTROL) {
      _isExecutable(tokenSupplyControlRequests[id_].owner,
tokenSupplyControlRequests[id_].status);
      if (TokenSupplyControlRequestType.MINT == tokenSupplyControlRequests[id_].subType) {
        _mint(tokenSupplyControlRequests[id_].wallet, tokenSupplyControlRequests[id_].amount);
      } else {
        if (tokenSupplyControlRequests[id_].wallet == tokenSupplyControlRequests[id_].owner) {
          _burn(tokenSupplyControlRequests[id_].owner, tokenSupplyControlRequests[id_].amount);
        } else if (tokenSupplyControlRequests[id_].wallet == address(this)) {
          _burn(address(this), tokenSupplyControlRequests[id_].amount);
        } else {
          require(
            allowance(
              tokenSupplyControlRequests[id_].wallet,
              tokenSupplyControlRequests[id_].owner
            ) >= tokenSupplyControlRequests[id_].amount,
            'INSUFFICIENT_ALLOWANCE!'
          );
          _spendAllowance(
            tokenSupplyControlRequests[id_].wallet,
            tokenSupplyControlRequests[id_].owner,
            tokenSupplyControlRequests[id_].amount
          );
          _burn(tokenSupplyControlRequests[id_].wallet, tokenSupplyControlRequests[id_].amount);
        }
      }
      tokenSupplyControlRequests[id_].status = RequestStatus.EXECUTED;
```

```
    } else if (reqType_ == RequestType.TRANSACTION_CONTROL) {
      _isExecutable(transactionControlRequests[id_].owner, transactionControlRequests[id_].status);
      if (TransactionControlRequestType.PAUSE == transactionControlRequests[id_].subType) {
        _pause();
      } else {
        _unpause();
      }
      transactionControlRequests[id_].status = RequestStatus.EXECUTED;
    } else if (reqType_ == RequestType.SIGNATORY_CONTROL) {
      _isExecutable(signatoryControlRequests[id_].owner, signatoryControlRequests[id_].status);
      for (uint256 i; i < signatoryControlRequests[id_].wallets.length; i++) {
        if (SignatoryControlRequestType.ADD == signatoryControlRequests[id_].subType) {
          require(!isSignatory[signatoryControlRequests[id_].wallets[i]], 'EXISTING!');
          isSignatory[signatoryControlRequests[id_].wallets[i]] = true;
          signatoryList.push(signatoryControlRequests[id_].wallets[i]);
        } else {
          require(signatoryList.length > 1, 'LAST_SIGNATORY!');
          require(isSignatory[signatoryControlRequests[id_].wallets[i]], 'UNKNOWN!');
          isSignatory[signatoryControlRequests[id_].wallets[i]] = false;
          signatoryList = ArrayOps.deleteFromArray(
            signatoryList,
            signatoryControlRequests[id_].wallets[i]
          );
        }
      }
      signatoryControlRequests[id_].status = RequestStatus.EXECUTED;
      emit SignatoriesUpdated(
        signatoryControlRequests[id_].subType,
        id_,
        signatoryControlRequests[id_].wallets
      );
    } else if (reqType_ == RequestType.THRESHOLD_CONTROL) {
      _isExecutable(thresholdControlRequests[id_].owner, thresholdControlRequests[id_].status);
      for (uint256 i; i < requestTypeCount[thresholdControlRequests[id_].reqType]; i++) {
        if (thresholdControlRequests[id_].reqType == RequestType.TOKEN_SUPPLY_CONTROL) {
          tokenSupplyControlThresholds[TokenSupplyControlRequestType(i)] =
thresholdControlRequests[
            id_
          ].thresholds[i];
        } else if (thresholdControlRequests[id_].reqType == RequestType.TRANSACTION_CONTROL) {
          transactionControlThresholds[TransactionControlRequestType(i)] =
thresholdControlRequests[
            id_
          ].thresholds[i];
        } else if (thresholdControlRequests[id_].reqType == RequestType.SIGNATORY_CONTROL) {
          signatoryControlThresholds[SignatoryControlRequestType(i)] =
thresholdControlRequests[id_]
            .thresholds[i];
        } else if (thresholdControlRequests[id_].reqType == RequestType.THRESHOLD_CONTROL) {
          thresholdControlThresholds[ThresholdControlRequestType(i)] =
thresholdControlRequests[id_]
```

```
          .thresholds[i];
        } else {
          whiteListControlThresholds[WhiteListControlRequestType(i)] = thresholdControlRequests[id_]
            .thresholds[i];
        }
      }
      thresholdControlRequests[id_].status = RequestStatus.EXECUTED;
      emit ThresholdUpdated(thresholdControlRequests[id_].reqType, id_,
thresholdControlRequests[id_].thresholds);
    } else {
      _isExecutable(whiteListControlRequests[id_].owner, whiteListControlRequests[id_].status);
      for (uint256 i; i < whiteListControlRequests[id_].wallets.length; i++) {
        if (WhiteListControlRequestType.ADD == whiteListControlRequests[id_].subType) {
          require(!isWhiteListed[whiteListControlRequests[id_].wallets[i]], 'EXISTING!');
          isWhiteListed[whiteListControlRequests[id_].wallets[i]] = true;
          whiteListedUsers.push(whiteListControlRequests[id_].wallets[i]);
        } else {
          require(isWhiteListed[whiteListControlRequests[id_].wallets[i]], 'UNKNOWN!');
          isWhiteListed[whiteListControlRequests[id_].wallets[i]] = false;
          whiteListedUsers = ArrayOps.deleteFromArray(
            whiteListedUsers,
            whiteListControlRequests[id_].wallets[i]
          );
        }
      }
      whiteListControlRequests[id_].status = RequestStatus.EXECUTED;
      emit WhiteListUpdated(
        whiteListControlRequests[id_].subType,
        id_,
        whiteListControlRequests[id_].wallets
      );
    }
  }

//Common.sol
function _isExecutable(address owner, RequestStatus status) internal view {
    require(owner != address(0), 'INVALID_REQUEST!');
    require(owner == msg.sender, 'UNAUTHORIZED!');
    require(status == RequestStatus.ACCEPTED, 'NOT_APPROVED!');
  }

//ERC20Upgradeable.sol
function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");
        _beforeTokenTransfer(address(0), account, amount);
        _totalSupply += amount;
        _balances[account] += amount;
        emit Transfer(address(0), account, amount);
        _afterTokenTransfer(address(0), account, amount);
    }
```

```
//ERC20Upgradeable.sol
function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");
        _beforeTokenTransfer(account, address(0), amount);
        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
        }
        _totalSupply -= amount;
        emit Transfer(account, address(0), amount);
        _afterTokenTransfer(account, address(0), amount);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	increaseAllowance()
Remediation	
References	
Access Specifier	public
Additional Modifiers	NA
Input Parameters	spender, addedValue
Function Code	

```
//ERC20Upgradeable.sol
function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, allowance(owner, spender) + addedValue);
        return true;
    }

function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable

Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	initialize()
Remediation	
References	
Access Specifier	public
Additional Modifiers	initializer
Input Parameters	name_, symbol_, decimals_, governanceToken_, owner_

Function Code

```
function initialize(
    string memory name_,
    string memory symbol_,
    uint8 decimals_,
    address owner_
  ) public initializer {
    __ERC20_init(name_, symbol_);
    __ERC20Pausable_init();
    __Ownable_init();

    isSignatory[owner_] = true;
    signatoryList.push(owner_);

    isWhiteListed[owner_] = true;
    whiteListedUsers.push(owner_);

    _decimals = decimals_;
    _setRequestTypeCount();
    _setDefaultThresholds();

    _transferOwnership(owner_);
  }

//Common.sol
function _setRequestTypeCount() internal {
    requestTypeCount[RequestType.TOKEN_SUPPLY_CONTROL] = 2;
    requestTypeCount[RequestType.TRANSACTION_CONTROL] = 2;
    requestTypeCount[RequestType.SIGNATORY_CONTROL] = 2;
    requestTypeCount[RequestType.THRESHOLD_CONTROL] = 1;
    requestTypeCount[RequestType.WHITELIST_CONTROL] = 2;
  }

//StableCoin.sol
function _setDefaultThresholds() private {
    tokenSupplyControlThresholds[TokenSupplyControlRequestType.BURN] = 1;
    tokenSupplyControlThresholds[TokenSupplyControlRequestType.MINT] = 1;
    transactionControlThresholds[TransactionControlRequestType.PAUSE] = 1;
    transactionControlThresholds[TransactionControlRequestType.UNPAUSE] = 1;
    signatoryControlThresholds[SignatoryControlRequestType.ADD] = 1;
    signatoryControlThresholds[SignatoryControlRequestType.REMOVE] = 1;
    thresholdControlThresholds[ThresholdControlRequestType.UPDATE] = 1;
  }

//OwnableUpgradeable.sol
function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
```

Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	renounceOwnership()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	No
Function Code	//OwnableUpgradeable.sol function renounceOwnership() public virtual onlyOwner { _transferOwnership(address(0)); } function _transferOwnership(address newOwner) internal virtual { address oldOwner = _owner; _owner = newOwner; emit OwnershipTransferred(oldOwner, newOwner); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	swap()
Remediation	
References	
Access Specifier	external
Additional Modifiers	No
Input Parameters	token_, amount_

Function Code	function swap(address token_, uint256 amount_) external nonReentrant { require(isWhiteListed[msg.sender], 'UNAUTHORIZED!'); _burn(msg.sender, amount_); StableCoin(token_).swap(msg.sender, amount_); emit TokenBurnedForSwap(msg.sender, amount_, token_); } //ERC20Upgradeable.sol function _burn(address account, uint256 amount) internal virtual { require(account != address(0), ""ERC20: burn from the zero address""); _beforeTokenTransfer(account, address(0), amount); uint256 accountBalance = _balances[account]; require(accountBalance >= amount, ""ERC20: burn amount exceeds balance""); unchecked { _balances[account] = accountBalance - amount; } _totalSupply -= amount; emit Transfer(account, address(0), amount); _afterTokenTransfer(account, address(0), amount); } //StableCoin.sol function swap(address user_, uint256 amount_) external nonReentrant { require(msg.sender == _governanceTokenAddress, 'UNAUTHORIZED!'); _mint(user_, amount_); } //ERC20Upgradeable.sol function _mint(address account, uint256 amount) internal virtual { require(account != address(0), ""ERC20: mint to the zero address""); _beforeTokenTransfer(address(0), account, amount); _totalSupply += amount; _balances[account] += amount; emit Transfer(address(0), account, amount); _afterTokenTransfer(address(0), account, amount); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	transfer()
Remediation	
References	
Access Specifier	public
Additional Modifiers	No
Input Parameters	to, amount
Function Code	function transfer(address to, uint256 amount) public virtual override returns (bool) { require(isWhiteListed[_msgSender()] && isWhiteListed[to], 'NOT_WHITELISTED'); _transfer(_msgSender(), to, amount); return true; } //ERC20Upgradeable.sol function _transfer(address from, address to, uint256 amount) internal virtual { require(from != address(0), "ERC20: transfer from the zero address"); require(to != address(0), "ERC20: transfer to the zero address"); _beforeTokenTransfer(from, to, amount); uint256 fromBalance = _balances[from]; require(fromBalance >= amount, "ERC20: transfer amount exceeds balance"); unchecked { _balances[from] = fromBalance - amount; } _balances[to] += amount; emit Transfer(from, to, amount); _afterTokenTransfer(from, to, amount); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable

Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	transferFrom()
Remediation	
References	
Access Specifier	public
Additional Modifiers	No
Input Parameters	from, to, amount

Function Code	function transferFrom(address from, address to, uint256 amount) public virtual override returns (bool) { address spender = _msgSender(); require(isWhiteListed[_msgSender()] && isWhiteListed[to], 'NOT_WHITELISTED'); _spendAllowance(from, spender, amount); _transfer(from, to, amount); return true; } //ERC20Upgradeable.sol function _transfer(address from, address to, uint256 amount) internal virtual { require(from != address(0), "ERC20: transfer from the zero address"); require(to != address(0), "ERC20: transfer to the zero address"); _beforeTokenTransfer(from, to, amount); uint256 fromBalance = _balances[from]; require(fromBalance >= amount, "ERC20: transfer amount exceeds balance"); unchecked { _balances[from] = fromBalance - amount; } _balances[to] += amount; emit Transfer(from, to, amount); _afterTokenTransfer(from, to, amount); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	transferOwnership()
Remediation	
References	
Access Specifier	public
Additional Modifiers	onlyOwner
Input Parameters	newOwner
Function Code	//OwnableUpgradeable.sol function transferOwnership(address newOwner) public virtual onlyOwner { require(newOwner != address(0), "Ownable: new owner is the zero address"); _transferOwnership(newOwner); } function _transferOwnership(address newOwner) internal virtual { address oldOwner = _owner; _owner = newOwner; emit OwnershipTransferred(oldOwner, newOwner); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

Smart Contract - GovernanceToken.sol

Result	
Description	
Location	
Observation	
Function Name	updateSignatoryControlRequest()
Remediation	
References	
Access Specifier	external
Additional Modifiers	onlySignatory
Input Parameters	id_, users_
Function Code	function updateSignatoryControlRequest(uint256 id_, address[] memory users_) external onlySignatory { require(signatoryControlRequests[id_].owner != address(0), 'INVALID_REQUEST!'); require(signatoryControlRequests[id_].owner == msg.sender, 'UNAUTHORIZED!'); require(signatoryControlRequests[id_].status == RequestStatus.IN_PROGRESS, 'NOT_ACTIVE!'); signatoryControlRequests[id_].wallets = users_; signatoryControlRequests[id_].approvals = new address[](0); emit RequestUpdated(RequestType.SIGNATORY_CONTROL, id_); }
Reentrancy	Not Vulnerable
Access Control	Not Vulnerable
Arithmetic	Not Vulnerable
Business Logic	Not Vulnerable
Denial of Service	Not Vulnerable
Time Manipulation	Not Vulnerable
Secure	Yes

7.0 Auditing Approach and Methodologies applied

Throughout the audit of the smart contract, care was taken to ensure:

- Overall quality of code
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Mathematical calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of token standards.
- Efficient use of gas.
- Code is safe from Re-entrancy and other vulnerabilities.

A combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

7.1 Structural Analysis

In this step we have analysed the design patterns and structure of all smart contracts. A thorough check was completed to ensure all Smart contracts are structured in a way that will not result in future problems.

7.2 Static Analysis

Static Analysis of smart contracts was undertaken to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

7.3 Code Review / Manual Analysis

Manual Analysis or review of done to identify new vulnerabilities or to verify the vulnerabilities found during the Static Analysis. The contracts were completely manually analysed, and their logic was checked and compared with the one described in the whitepaper. It should also be noted that the results of the automated analysis were verified manually.

7.4 Gas Consumption

In this step, we checked the behaviour of all smart contracts in production. Checks were completed to understand how much gas gets consumed, along with the possibilities of optimisation of code to reduce gas consumption.

7.5 Tools & Platforms Used For Audit

MythX, Remix IDE, Truffle, Solhint, Ganache, Slither

7.6 Checked Vulnerabilities

We have scanned Audd smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC-20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

8.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of AUDD and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Smart Contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of AUDD and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and AUDD governs the disclosure of this report to all other parties including product vendors and suppliers.