Ether Authority
etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	**Dai Stablecoin**
Website:	**makerdao.com**
Platform:	**Base Chain Network**
Language:	**Solidity**
Date:	**May 24th, 2024**

Table of contents

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Dai Stablecoin smart contract from makerdao.com was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 24th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Website Details

- MakerDAO is a decentralized autonomous organization that operates the Maker Protocol, which includes the Dai stablecoin.
- Dai is a decentralized, stable cryptocurrency pegged to the US dollar, designed to maintain a stable value without central control.
- The Maker Protocol uses smart contracts on the Ethereum blockchain and is governed by holders of the MKR token, who manage the system through voting.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Code Details

- This Solidity code defines an ERC20 token that is mintable and burnable via a bridge contract, specifically designed to work with the Optimism Layer 2 solution. Here's an overview of the key components and functionalities:
- **Core Contract:**
    - **Semver:** Handles semantic versioning with `MAJOR_VERSION`, `MINOR_VERSION`, and `PATCH_VERSION`.
    - **ERC20:**
        - Implements the standard ERC20 functionality with additional internal functions to handle minting, burning, allowances, and token transfers.
        - Functions include `name`, `symbol`, `decimals`, `totalSupply`, `balanceOf`, `transfer`, `allowance`, `approve`, `transferFrom`, `increaseAllowance`, `decreaseAllowance`, `_transfer`, `_mint`, `_burn`, `_approve`, `_spendAllowance`, `_beforeTokenTransfer`, and `_afterTokenTransfer`.
    - **OptimismMintableERC20:**
        - Extends `ERC20` and `Semver` to create a mintable and burnable token for use with the Optimism bridge.
        - The constructor takes parameters for the bridge address, remote token address, token name, and symbol.
        - The `mint` and `burn` functions can only be called by the bridge contract.
        - Implements `supportsInterface` for interface detection.

This structure ensures that the token adheres to the ERC20 standard while adding the necessary functionalities for integration with the Optimism Layer 2 solution, specifically for minting and burning tokens via a designated bridge contract.

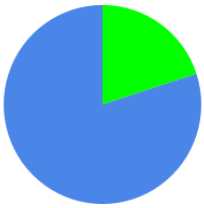
This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit scope

Name	**Code Review and Security Analysis Report for Dai Stablecoin Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	OptimismMintableERC20.sol
Smart Contract Code	0x50c5725949a6f0c72e6c4a641f24049a917db0cb
Audit Date	May 24th,2024
Audit Result	**Passed**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Code Audit History



5	0	0	0	1	4
Total Findings	Critical	High	Medium	Low	Informational

Severity Definitions

0	**Critical**		Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0	**High**		High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0	**Medium**		Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
1	**Low**		Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
4	**Lowest / Informational / Best Practice**		Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Dai Stablecoin • Symbol: DAI • Decimals: 18	**YES, This is valid.**
Ownership Control: • There are no owner functions, which makes it 100% decentralized.	**YES, This is valid.**
Key Features: • **Minting and Burning:** The `mint` and `burn` functions allow the bridge contract to control the supply of tokens on the Optimism network. • **Access Control:** Only the bridge contract can call the mint and burn functions, ensuring controlled minting and burning processes. • **Versioning:** The `Semver` contract is used to handle versioning, ensuring that different versions of the contract can be managed and identified. • **Interface Support:** The `supportsInterface` function ensures compatibility with the IERC165 standard for interface detection.	**YES, This is valid.**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**.This token contract does not have any ownership control, hence it is 100% decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 1 low, and 4 very low-level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	No
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	Yes
Is it a Proxy Contract?	No
Is it used Open Source?	No
External Call Risk?	No
Balance Modifiable?	No
Can Take Ownership?	No
Ownership Renounce?	No
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Dai Stablecoin are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Dai Stablecoin.

The EtherAuthority team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Dai Stablecoin smart contract code in the form of a basescan web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

-

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

AS-IS overview

OptimismMintableERC20.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyBridge	modifier	Passed	No Issue
3	mint	external	Bridge can mint unlimited tokens, Third-party dependency	Refer Audit Findings
4	burn	external	Bridge owner can burn anyone's token, Third-party dependency	Refer Audit Findings
5	supportsInterface	external	Passed	No Issue
6	l1Token	read	Identical value for 3 functions	Refer Audit Findings
7	l2Bridge	read	Identical value for 3 functions	Refer Audit Findings
8	remoteToken	read	Identical value for 3 functions	Refer Audit Findings
9	bridge	read	Identical value for 3 functions	Refer Audit Findings
10	name	read	Passed	No Issue
11	symbol	read	Passed	No Issue
12	decimals	read	Passed	No Issue
13	totalSupply	read	Passed	No Issue
14	balanceOf	read	Passed	No Issue
15	transfer	write	Passed	No Issue
16	allowance	read	Passed	No Issue
17	approve	write	Passed	No Issue
18	transferFrom	write	Passed	No Issue
19	increaseAllowance	write	Passed	No Issue
20	decreaseAllowance	write	Passed	No Issue
21	_transfer	internal	Passed	No Issue
22	_mint	internal	Passed	No Issue
23	_burn	internal	Passed	No Issue
24	_approve	internal	Passed	No Issue
25	_spendAllowance	internal	Passed	No Issue
26	_beforeTokenTransfer	internal	Passed	No Issue
27	_afterTokenTransfer	internal	Passed	No Issue
28	version	read	Passed	No Issue

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

[L-01] Bridge owner can burn anyone's token:

```
function burn(address _from, uint256 _amount)
        external
        virtual
        override(IOptimismMintableERC20, ILegacyMintableERC20)
        onlyBridge
    {
        _burn(_from, _amount);
        emit Burn(_from, _amount);
    }
```

Description:

Bridge owners can burn any user's tokens.

Recommendation: We suggest changing the code so only token holders can burn their own tokens and not anyone else. Not even a contract creator.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Very Low / Informational / Best practices:

[I-01] Bridge can mint unlimited tokens:

```
function mint(address _to, uint256 _amount)
        external
        virtual
        override(IOptimismMintableERC20, ILegacyMintableERC20)
        onlyBridge
    {
        _mint(_to, _amount);
        emit Mint(_to, _amount);
    }
```

Description:

There is no limit for minting tokens. Thus the minter can mint unlimited tokens to any account in OptimismMintableERC20 contract.

Recommendation: There should be a limit for minting or need to confirm, if it is a part of the plan then disregard this issue.

[I-02] Visibility can be external over the public:

Description:

Any functions which are not called internally should be declared as external. This saves some gas and is considered a good practice.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

[I-03] Identical value for 3 functions:

```
address public immutable REMOTE_TOKEN;
/** ...
function l1Token() public view returns (address) {    infinite g
    return REMOTE_TOKEN;
}
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

```
address public immutable BRIDGE;
/** ...
function l2Bridge() public view returns (address) {    infinite ga
    return BRIDGE;
}

/** ...
function bridge() public view returns (address) {    infinite gas
    return BRIDGE;
}
```

Description:

The same value has been returned by these 3 functions:

- BRIDGE
- l2Bridge()
- bridge() and remoteToken()

Recommendation: We suggest using only 1 function if the value is the same for both and the other can be removed.

[I-04] Third-party dependency:

```
function mint(address _to, uint256 _amount)    infinite gas
    external
    virtual
    override(IOptimismMintableERC20, ILegacyMintableERC20)
    onlyBridge
{
    _mint(_to, _amount);
    emit Mint(_to, _amount);
}
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

```
/** ...
function burn(address _from, uint256 _amount)    infinite gas
    external
    virtual
    override(IOptimismMintableERC20, ILegacyMintableERC20)
    onlyBridge
{
    _burn(_from, _amount);
    emit Burn(_from, _amount);
}
```

Description:

Only Bridge can Mint & Burn tokens. Bridge is a Third-party proxy contract.

Recommendation: We suggest if possible remove third-party dependency.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Centralization

The Dai Stablecoin smart contract does not have any ownership control, **hence it is 100% decentralized.**

Therefore, there is **no** centralization risk.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Conclusion

We were given a contract code in the form of a basescan web link. And we have used all possible tests based on given objects as files. We observed 1 low and 4 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

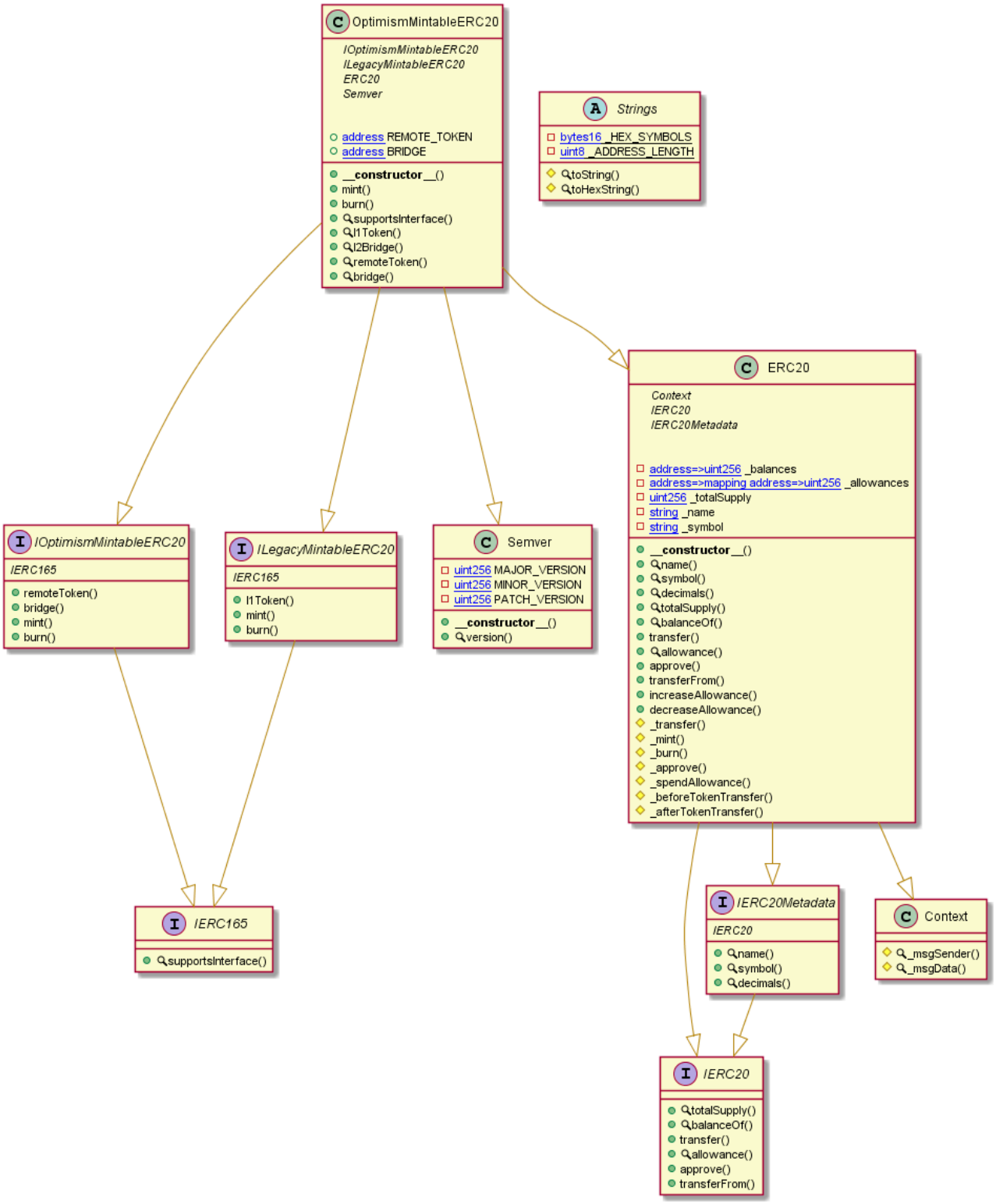
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Appendix

Code Flow Diagram - Dai Stablecoin Token



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

OptimismMintableERC20.sol

```
INFO:Detectors:
OptimismMintableERC20.constructor(address,address,string,string)._name
(OptimismMintableERC20.sol#672) shadows:
	- ERC20._name (OptimismMintableERC20.sol#274) (state variable)
OptimismMintableERC20.constructor(address,address,string,string)._symbol
(OptimismMintableERC20.sol#673) shadows:
	- ERC20._symbol (OptimismMintableERC20.sol#275) (state variable)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
OptimismMintableERC20.constructor(address,address,string,string)._remoteToken
(OptimismMintableERC20.sol#671) lacks a zero-check on :
		- REMOTE_TOKEN = _remoteToken (OptimismMintableERC20.sol#675)
OptimismMintableERC20.constructor(address,address,string,string)._bridge
(OptimismMintableERC20.sol#670) lacks a zero-check on :
		- BRIDGE = _bridge (OptimismMintableERC20.sol#676)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Context._msgData() (OptimismMintableERC20.sol#177-179) is never used and should be
removed
Strings.toHexString(address) (OptimismMintableERC20.sol#166-168) is never used and should
be removed
Strings.toHexString(uint256) (OptimismMintableERC20.sol#135-146) is never used and should
be removed
Strings.toHexString(uint256,uint256) (OptimismMintableERC20.sol#151-161) 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 (OptimismMintableERC20.sol#4) allows old versions
solc-0.8.25 is not recommended for deployment
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

```
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable Semver.MAJOR_VERSION (OptimismMintableERC20.sol#221) is not in mixedCase
Variable Semver.MINOR_VERSION (OptimismMintableERC20.sol#226) is not in mixedCase
Variable Semver.PATCH_VERSION (OptimismMintableERC20.sol#231) is not in mixedCase
Parameter OptimismMintableERC20.mint(address,uint256)._to
(OptimismMintableERC20.sol#685) is not in mixedCase
Parameter OptimismMintableERC20.mint(address,uint256)._amount
(OptimismMintableERC20.sol#685) is not in mixedCase
Parameter OptimismMintableERC20.burn(address,uint256)._from
(OptimismMintableERC20.sol#701) is not in mixedCase
Parameter OptimismMintableERC20.burn(address,uint256)._amount
(OptimismMintableERC20.sol#701) is not in mixedCase
Parameter OptimismMintableERC20.supportsInterface(bytes4)._interfaceId
(OptimismMintableERC20.sol#718) is not in mixedCase
Variable OptimismMintableERC20.REMOTE_TOKEN (OptimismMintableERC20.sol#630) is not in
mixedCase
Variable OptimismMintableERC20.BRIDGE (OptimismMintableERC20.sol#635) is not in
mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
Variable OptimismMintableERC20.REMOTE_TOKEN (OptimismMintableERC20.sol#630) is too
similar to OptimismMintableERC20.constructor(address,address,string,string)._remoteToken
(OptimismMintableERC20.sol#671)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Slither:OptimismMintableERC20.sol analyzed (10 contracts with 93 detectors), 21 result(s)
found
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

OptimismMintableERC20.sol

```
Gas costs:
Gas requirement of function OptimismMintableERC20.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 163:4:

Gas costs:
Gas requirement of function OptimismMintableERC20.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 179:4:

Constant/View/Pure functions:
OptimismMintableERC20.supportsInterface(bytes4) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
Pos: 196:4:

Similar variable names:
Semver.version() : Variables have very similar names "MAJOR_VERSION" and "MINOR_VERSION". Note: Modifiers are currently not considered by this static analysis.
Pos: 88:37:

No return:
IOptimismMintableERC20.remoteToken(): Defines a return type but never explicitly returns a value.
Pos: 19:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 135:8:
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

OptimismMintableERC20.sol

```
Compiler version >0.8.15 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Variable name must be in mixedCase
Pos: 5:49
Variable name must be in mixedCase
Pos: 5:54
Variable name must be in mixedCase
Pos: 5:59
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:66
Variable name must be in mixedCase
Pos: 5:107
Variable name must be in mixedCase
Pos: 5:112
Error message for require is too long
Pos: 9:134
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:146
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io