Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Enoch
Website: https://enoch.app/
Platform: Ethereum
Language: Solidity
Date: March 15th, 2023

Table of contents

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.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by Enoch to perform the Security audit of the Enoch smart contracts code. 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 March 15th, 2023.

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

- Enoch Metanomics ensures a fluid ecosystem of dual coin utility across the platform.
- A platform made to empower creative minds and 3D artists. Become the best and earn money on Enoch.
- The Enoch contract inherits the ERC20, ERC1967Proxy, Initializable, UUPSUpgradeable, ERC20Upgradeable, ERC20BurnableUpgradeable standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Email: audit@EtherAuthority.io

Audit scope

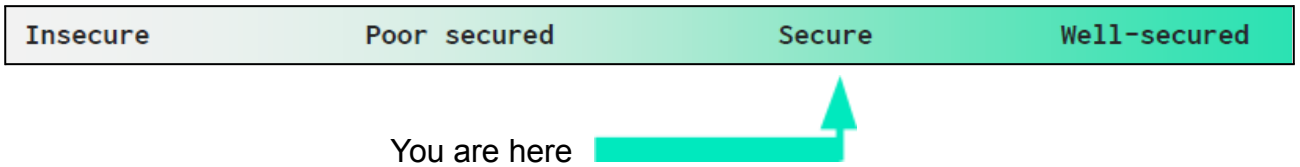
Name	Code Review and Security Analysis Report for Enoch Smart Contracts
Platform	**Ethereum / Solidity**
File 1	Enoch.sol
File 1 MD5 Hash	6E45EE220519FCE6F9D40DFEB5E2F7A2
File 1 Online Code	0x4db57D585fa82Ca32d25086DDc069d899f08D455
File 2	LoveProxy.sol
File 2 MD5 Hash	6BE3622A88F8CD75FD6144EF90C8A77D
File 3	Love.sol
File 3 MD5 Hash	DEA251BCAA81E232A17DAB919733FF2D
Updated File 3 MD5 Hash	2CBF282CDA3AEB8501A80D94CC5D45C2
File 3 Online Code	0xc215081aa55a2642745ee22b3bf5d1c775f37884
Audit Date	March 15th,2023
Revise Audit Date	May 4th,2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 Enoch.sol • Name: ENOCH • Symbol: ENOCH • Decimals: 18 • Total Supply: 54 Million • OpenZeppelin library used. **Owner Specifications:** • Admin can burn amounts. • Current admin can set a new admin address.	**YES, This is valid.**
File 2 Love.sol • Name: LOVE • Symbol: LOVE • Decimals: 18 • Total Supply: 1.5 billion • OpenZeppelin library used. **Owner Specifications:** • Mint amount by the admin. • Admin can burn amounts. • Admin can upgrade a new authorized address.	**YES, This is valid.**
File 3 LoveProxy.sol • Name: LOVE • Symbol: LOVE • Decimals: 18 • LoveProxy contract can inherit ERC1967Proxy contract from openzeppelin library.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts do contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 1 high, 0 medium and 0 low and some very low level issues.
These issues are fixed in the revised contract code.

Investors Advice: 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.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of 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	Passed
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 not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 3 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Enoch Protocol 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 Enoch Protocol.

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

Code parts are not well commented on smart contracts.

Documentation

We were given a Enoch Protocol smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. But 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.

Another source of information was its official website: https://enoch.app/ which provided rich information about the project architecture and tokenomics.

Use of Dependencies

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

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Enoch.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	burn	external	access only Admin	No Issue
3	transferAdminRole	external	access only Admin	No Issue
4	name	read	Passed	No Issue
5	symbol	read	Passed	No Issue
6	decimals	read	Passed	No Issue
7	totalSupply	read	Passed	No Issue
8	balanceOf	read	Passed	No Issue
9	transfer	write	Passed	No Issue
10	allowance	read	Passed	No Issue
11	approve	write	Passed	No Issue
12	transferFrom	write	Passed	No Issue
13	increaseAllowance	write	Passed	No Issue
14	decreaseAllowance	write	Passed	No Issue
15	_transfer	internal	Passed	No Issue
16	_mint	internal	Passed	No Issue
17	_burn	internal	Passed	No Issue
18	_approve	internal	Passed	No Issue
19	_spendAllowance	internal	Passed	No Issue
20	_beforeTokenTransfer	internal	Passed	No Issue
21	_afterTokenTransfer	internal	Passed	No Issue

Love.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializer	modifier	Passed	No Issue
3	reinitializer	modifier	Passed	No Issue
4	onlyInitializing	modifier	Passed	No Issue
5	_disableInitializers	internal	Passed	No Issue
6	_getInitializedVersion	internal	Passed	No Issue
7	_isInitializing	internal	Passed	No Issue
8	__UUPSUpgradeable_init	internal	access only Initializing	No Issue
9	__UUPSUpgradeable_init_unchained	internal	access only Initializing	No Issue
10	onlyProxy	modifier	Passed	No Issue
11	notDelegated	modifier	Passed	No Issue
12	proxiableUUID	external	Passed	No Issue

13	upgradeTo	write	access only Proxy	No Issue
14	upgradeToAndCall	write	access only Proxy	No Issue
15	_authorizeUpgrade	internal	Passed	No Issue
16	__ERC20_init	internal	access only Initializing	No Issue
17	__ERC20_init_unchained	internal	access only Initializing	No Issue
18	name	read	Passed	No Issue
19	symbol	read	Passed	No Issue
20	decimals	read	Passed	No Issue
21	totalSupply	read	Passed	No Issue
22	balanceOf	read	Passed	No Issue
23	transfer	write	Passed	No Issue
24	allowance	read	Passed	No Issue
25	approve	write	Passed	No Issue
26	transferFrom	write	Passed	No Issue
27	increaseAllowance	write	Passed	No Issue
28	decreaseAllowance	write	Passed	No Issue
29	_transfer	internal	Passed	No Issue
30	_mint	internal	Passed	No Issue
31	_burn	internal	Passed	No Issue
32	_approve	internal	Passed	No Issue
33	_spendAllowance	internal	Passed	No Issue
34	_beforeTokenTransfer	internal	Passed	No Issue
35	_afterTokenTransfer	internal	Passed	No Issue
36	__ERC20Burnable_init	internal	access only Initializing	No Issue
37	__ERC20Burnable_init_unchained	internal	access only Initializing	No Issue
38	burn	write	Passed	No Issue
39	burnFrom	write	Passed	No Issue
40	onlyAdmin	modifier	Passed	No Issue
41	initialize	external	initializer	No Issue
42	mint	internal	access only Admin	No Issue
43	_authorizeUpgrade	internal	access only Admin	No Issue
44	burn	write	access only Admin	No Issue

LoveProxy.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	_implementation	internal	Passed	No Issue
3	getImplementation	read	Passed	No Issue

Severity Definitions

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

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

(1) Pause/Unpause not working properly: **Love.sol**

Pause and Unpause functions are defined but when the admin pauses the contract then still any user can do all activities, so both functions are not affected by transfer or burn or approval, etc.

Resolution: We suggest checking for pause and Unpause functions logic to work properly. or if both functions are not in use so remove both functions.
Status: This issue is fixed in the revised contract code.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

No Very Low severity vulnerabilities were found.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

Enoch.sol

- burn: Admin can burn amount.
- transferAdminRole: Current admin can set a new admin address.

Love.sol

- initialize: Admin can initialize ids.
- mint: Mint amount by the admin.
- burn: Admin can burn amount.
- _authorizeUpgrade: admin can upgrade a new authorized address.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of files. And we have used all possible tests based on given objects as files. We had observed 1 high severity issue in the smart contracts.These issues are fixed in the revised contract code. **So, the smart contracts are ready for the mainnet deployment**.

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 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.

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

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

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 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 whitebox penetration testing. We look at the project's web site 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.

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, 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.

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 are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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.

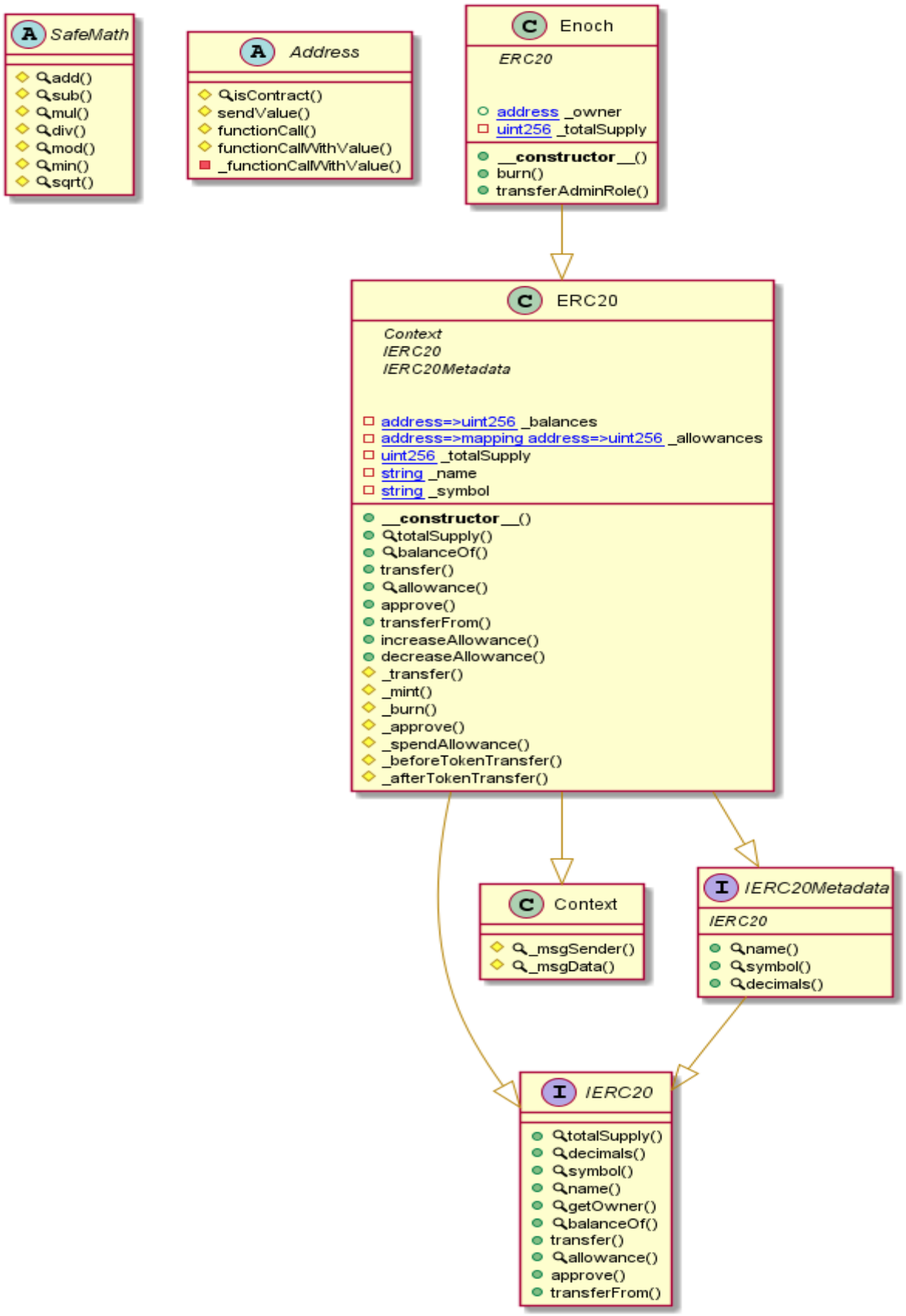
Technical Disclaimer

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 explicit security of the audited smart contracts.

Appendix

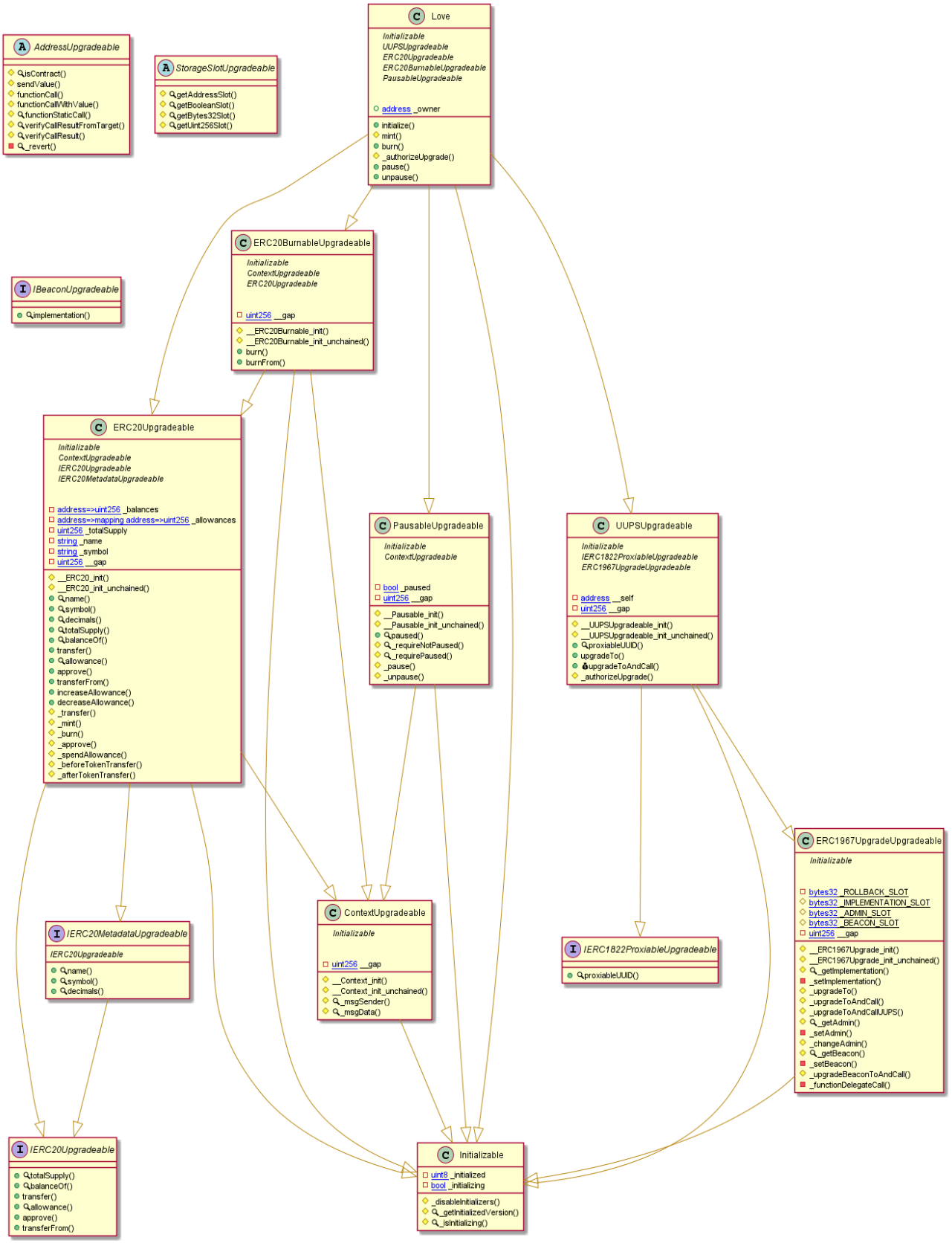
Code Flow Diagram - Enoch

Enoch Diagram

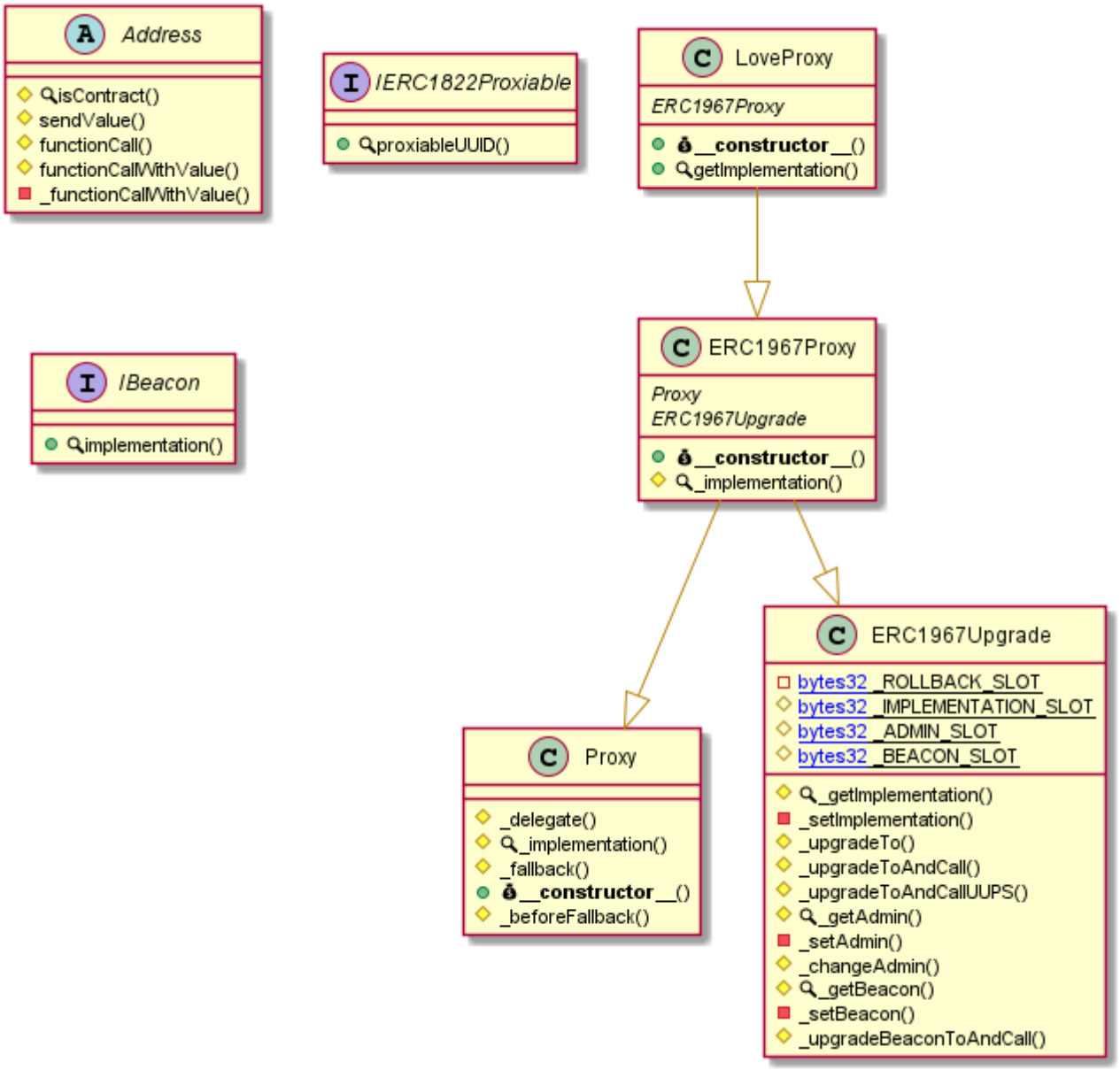


Email: audit@EtherAuthority.io

Love Diagram



LoveProxy Diagram



Slither Results Log

Slither log >> Enoch.sol

```
Enoch.transferAdminRole(address) (Enoch.sol#555-558) should emit an event for:
        - _owner = newAdmin (Enoch.sol#557)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

Address.isContract(address) (Enoch.sol#85-92) uses assembly
        - INLINE ASM (Enoch.sol#88-90)
Address._functionCallWithValue(address,bytes,uint256,string) (Enoch.sol#131-153) uses assembly
        - INLINE ASM (Enoch.sol#145-148)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
Pragma version^0.8.0 (Enoch.sol#3) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (Enoch.sol#94-99):
        - (success) = recipient.call{value: amount}() (Enoch.sol#97)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (Enoch.sol#131-153):
        - (success,returndata) = target.call{value: weiValue}(data) (Enoch.sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable Enoch._owner (Enoch.sol#535) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (Enoch.sol#191)" inContext (Enoch.sol#185-194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Enoch.slitherConstructorVariables() (Enoch.sol#533-561) uses literals with too many digits:
        - _totalSupply = 100000000000000000000000000 (Enoch.sol#536)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Enoch (Enoch.sol#533-561) does not implement functions:
        - IERC20.decimals() (Enoch.sol#158)
        - IERC20.getOwner() (Enoch.sol#164)
        - IERC20Metadata.name() (Enoch.sol#201)
        - IERC20Metadata.symbol() (Enoch.sol#206)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

Enoch._totalSupply (Enoch.sol#536) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
Enoch.sol analyzed (7 contracts with 84 detectors), 32 result(s) found
```

Slither log >> Love.sol

```
Variable 'ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (LoveV2.sol#468)' in ERC1967UpgradeUpgradea
ble._upgradeToAndCallUUPS(address,bytes,bool) (LoveV2.sol#460-475) potentially used before declaration: require(bool,string)(s
lot == _IMPLEMENTATION_SLOT,ERC1967Upgrade: unsupported proxiableUUID) (LoveV2.sol#469)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

AddressUpgradeable._revert(bytes,string) (LoveV2.sol#118-127) uses assembly
        - INLINE ASM (LoveV2.sol#120-123)
StorageSlotUpgradeable.getAddressSlot(bytes32) (LoveV2.sol#395-399) uses assembly
        - INLINE ASM (LoveV2.sol#396-398)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (LoveV2.sol#401-405) uses assembly
        - INLINE ASM (LoveV2.sol#402-404)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (LoveV2.sol#407-411) uses assembly
        - INLINE ASM (LoveV2.sol#408-410)
StorageSlotUpgradeable.getUint256Slot(bytes32) (LoveV2.sol#413-417) uses assembly
        - INLINE ASM (LoveV2.sol#414-416)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

AddressUpgradeable.functionCall(address,bytes) (LoveV2.sol#46-48) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (LoveV2.sol#50-56) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (LoveV2.sol#58-64) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (LoveV2.sol#66-75) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (LoveV2.sol#77-79) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (LoveV2.sol#81-88) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (LoveV2.sol#39-44) is never used and should be removed
AddressUpgradeable.verifyCallResultFromTarget(address,bool,bytes,string) (LoveV2.sol#90-104) is never used and should be remov
ed
ContextUpgradeable.__Context_init() (LoveV2.sol#185-186) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (LoveV2.sol#188-189) is never used and should be removed
ContextUpgradeable._msgData() (LoveV2.sol#194-196) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (LoveV2.sol#424-425) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (LoveV2.sol#427-428) is never used and should be removed
ERC1967UpgradeUpgradeable._changeAdmin(address) (LoveV2.sol#490-493) is never used and should be removed
```

```
Initializable._getInitializedVersion() (LoveV2.sol#176-178) is never used and should be removed
Initializable._isInitializing() (LoveV2.sol#180-182) is never used and should be removed
PausableUpgradeable.__Pausable_init() (LoveV2.sol#578-580) is never used and should be removed
PausableUpgradeable.__Pausable_init_unchained() (LoveV2.sol#582-584) is never used and should be removed
StorageSlotUpgradeable.getBytes32Slot(bytes32) (LoveV2.sol#407-411) is never used and should be removed
StorageSlotUpgradeable.getUint256Slot(bytes32) (LoveV2.sol#413-417) is never used and should be removed
UUPSUpgradeable.__UUPSUpgradeable_init() (LoveV2.sol#534-535) is never used and should be removed
UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (LoveV2.sol#537-538) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (LoveV2.sol#2) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Pragma version^0.8.0 (LoveV2.sol#2) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (LoveV2.sol#39-44):
        - (success) = recipient.call{value: amount}() (LoveV2.sol#42)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (LoveV2.sol#66-75):
        - (success,returndata) = target.call{value: value}(data) (LoveV2.sol#73)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (LoveV2.sol#81-88):
        - (success,returndata) = target.staticcall(data) (LoveV2.sol#86)
Low level call in ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (LoveV2.sol#524-529):
        - (success,returndata) = target.delegatecall(data) (LoveV2.sol#527)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function ContextUpgradeable.__Context_init() (LoveV2.sol#185-186) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (LoveV2.sol#188-189) is not in mixedCase
Variable ContextUpgradeable.__gap (LoveV2.sol#198) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (LoveV2.sol#211-213) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (LoveV2.sol#215-218) is not in mixedCase
Variable ERC20Upgradeable.__gap (LoveV2.sol#351) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init() (LoveV2.sol#355-356) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init_unchained() (LoveV2.sol#358-359) is not in mixedCase
Variable ERC20BurnableUpgradeable.__gap (LoveV2.sol#369) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (LoveV2.sol#424-425) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (LoveV2.sol#427-428) is not in mixedCase
Variable ERC1967UpgradeUpgradeable.__gap (LoveV2.sol#531) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init() (LoveV2.sol#534-535) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (LoveV2.sol#537-538) is not in mixedCase
Variable UUPSUpgradeable.__self (LoveV2.sol#539) is not in mixedCase
Variable UUPSUpgradeable.__gap (LoveV2.sol#568) is not in mixedCase
Function PausableUpgradeable.__Pausable_init() (LoveV2.sol#578-580) is not in mixedCase
Function PausableUpgradeable.__Pausable_init_unchained() (LoveV2.sol#582-584) is not in mixedCase
Variable PausableUpgradeable.__gap (LoveV2.sol#618) is not in mixedCase
Parameter LoveV2.initialize(string)._id (LoveV2.sol#638) is not in mixedCase
Parameter LoveV2.setID(string)._newID (LoveV2.sol#642) is not in mixedCase
Variable LoveV2._owner (LoveV2.sol#627) is not in mixedCase
```

```
Variable LoveV2._owner (LoveV2.sol#627) is not in mixedCase
Variable LoveV2.ID (LoveV2.sol#628) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

LoveV2._owner (LoveV2.sol#627) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
LoveV2.sol analyzed (14 contracts with 84 detectors), 74 result(s) found
```

Slither log >> LoveProxy.sol

```
Address.isContract(address) (LoveProxy.sol#6-13) uses assembly
        - INLINE ASM (LoveProxy.sol#9-11)
Address._functionCallWithValue(address,bytes,uint256,string) (LoveProxy.sol#52-74) uses assembly
        - INLINE ASM (LoveProxy.sol#66-69)
Proxy._delegate(address) (LoveProxy.sol#85-108) uses assembly
        - INLINE ASM (LoveProxy.sol#86-107)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address._functionCallWithValue(address,bytes,uint256,string) (LoveProxy.sol#52-74) is never used and should be removed
Address.functionCall(address,bytes) (LoveProxy.sol#22-24) is never used and should be removed
Address.functionCall(address,bytes,string) (LoveProxy.sol#26-32) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (LoveProxy.sol#34-40) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (LoveProxy.sol#42-50) is never used and should be removed
Address.sendValue(address,uint256) (LoveProxy.sol#15-20) is never used and should be removed
ERC1967Upgrade._changeAdmin(address) (LoveProxy.sol#176-179) is never used and should be removed
ERC1967Upgrade._getAdmin() (LoveProxy.sol#167-169) is never used and should be removed
ERC1967Upgrade._getBeacon() (LoveProxy.sol#185-187) is never used and should be removed
ERC1967Upgrade._setAdmin(address) (LoveProxy.sol#171-174) is never used and should be removed
ERC1967Upgrade._setBeacon(address) (LoveProxy.sol#189-196) is never used and should be removed
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (LoveProxy.sol#198-204) is never used and should be removed
ERC1967Upgrade._upgradeToAndCallUUPS(address,bytes,bool) (LoveProxy.sol#156-161) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (LoveProxy.sol#2) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (LoveProxy.sol#15-20):
        - (success) = recipient.call{value: amount}() (LoveProxy.sol#18)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (LoveProxy.sol#52-74):
        - (success,returndata) = target.call{value: weiValue}(data) (LoveProxy.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
ERC1967Upgrade._IMPLEMENTATION_SLOT (LoveProxy.sol#131) is never used in LoveProxy (LoveProxy.sol#218-230)
ERC1967Upgrade._ADMIN_SLOT (LoveProxy.sol#163) is never used in LoveProxy (LoveProxy.sol#218-230)
ERC1967Upgrade._BEACON_SLOT (LoveProxy.sol#181) is never used in LoveProxy (LoveProxy.sol#218-230)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
LoveProxy.sol analyzed (7 contracts with 84 detectors), 24 result(s) found
```

Solidity Static Analysis

Enoch.sol

Gas & Economy

Gas costs:

Gas requirement of function Enoch.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: 25:4:

Miscellaneous

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.
more
Pos: 13:8:

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.
more
Pos: 30:8:

Love.sol

Gas & Economy

Gas costs:

Gas requirement of function Love.initialize 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: 27:4:

Email: audit@EtherAuthority.io

Gas costs:

Gas requirement of function Love.pause 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: 45:4:

Gas costs:

Gas requirement of function Love.unpause 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: 49:4:

Miscellaneous

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.
more
Pos: 20:8:

Solhint Linter

Enoch.sol

```
Enoch.sol:3:1: Error: Compiler version ^0.8.0 does not satisfy the r
semver requirement
Enoch.sol:20:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
```

Love.sol

```
Love.sol:2:1: Error: Compiler version ^0.8.7 does not satisfy the r
semver requirement
Love.sol:43:88: Error: Code contains empty blocks
```

LoveProxy.sol

```
LoveProxy.sol:2:1: Error: Compiler version ^0.8.4 does not satisfy
the r semver requirement
LoveProxy.sol:8:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
LoveProxy.sol:11:5: Error: Code contains empty blocks
```

Software analysis result:

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

Email: audit@EtherAuthority.io