CRED SHiELDS

CredShields
Smart Contract Audit

July 25th, 2023 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Lync World between May 4th, 2023, and May 27th, 2023. And a retest was performed on July 17th, 2023.

Author

Shashank (Co-founder, CredShields)
shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)
aditya@CredShields.com

Prepared for

Lync World

Table of Contents

1. Executive Summary

Lync World engaged CredShields to perform a smart contract audit from May 4th, 2023, to May 27th, 2023. During this timeframe, Twenty-six (26) vulnerabilities were identified. **A retest was performed on July 17th, 2023, and all the bugs have been addressed.**

During the audit, one (1) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Lync World" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
Lync Smart Contracts	1	0	4	8	4	9	**26**
	1	**0**	**4**	**8**	**4**	**9**	**26**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Smart Contract's scope during the testing window while abiding by the policies set forth by Smart Contract's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both Lync World's internal security and development teams to not only identify specific vulnerabilities, but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at Lync World can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Lync World can future-proof its security posture and protect its assets.

2. Methodology

Lync World engaged CredShields to perform a Lync World Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

The CredShields team meticulously reviewed all provided documents and comments in the smart-contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from May 4th, 2023, to May 27th, 2023, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed-upon:

IN SCOPE ASSETS
https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/tree/main/contracts

Table: List of Files in Scope

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

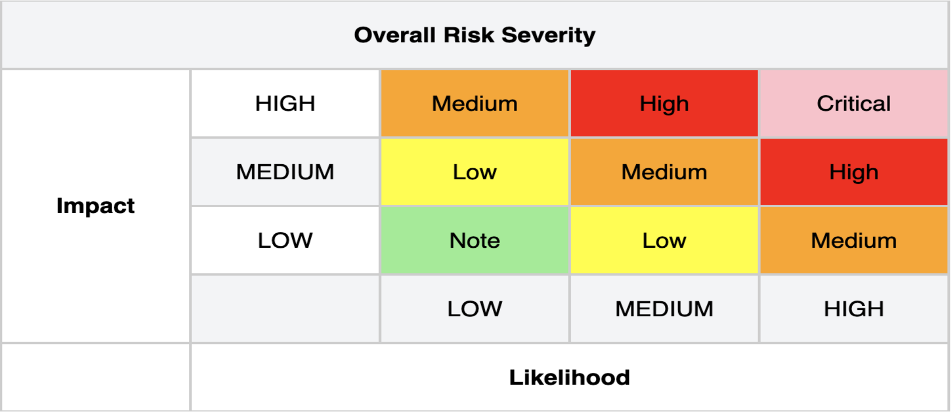
CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting phase

Lync World is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.



Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. Informational

We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. **Low**

   Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. **Medium**

   Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
    - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have around the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, Twenty-six (26) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Missing State Variable Visibility	Informational	Missing Best Practices
Missing Zero Address Validations	Low	Missing Input Validation
Missing Events in Functions	Low	Missing Events in Functions
Boolean Equality	Gas	Gas Optimization
Missing NatSpec Comments	Informational	Missing best practices
Variables should be Immutable	Gas	Gas Optimization

Modifier Created But Never Used	Gas	Dead Code
Hardcoded Static Address	Informational	Missing Best Practises
Functions should be declared External	Gas	Gas Optimization
Superfluous Event Field	Gas	Gas Optimization
Public Constants can be Private	Gas	Gas Optimization
Gas Optimization in Require Statements	Gas	Gas Optimization
Potential Signature Replay Attack	Medium	Signature Reply attack
Use Multisig for Admin Functions	Low	Business Logic
Gas Optimization in For Loops	Gas	Gas Optimization
Unnecessary Multiple Payable Functions	Low	Missing Best Practices
Missing Zero Address Validations	Low	Missing Input Validation
Missing Input Validation	Medium	Missing Input Validation
Lack of Unique Signer Verification	Medium	Business Logic
Missing Zero Address Validations For recover() Function	Medium	Missing Validation
Signature Replay Attack	Critical	Business Logic
Functions should use Multisig	Low	Business Logic
Use Call instead of Transfer	Informational	Best Practices

Unused Imports	Gas	Gas Optimization
Locked Ether	Low	Business Logic

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^.8.0 and above is used
SWC-103	Floating Pragma	Not Vulnerable	Contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0

SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found

SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.
SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

Lync World is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on July 17th, 2023, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Floating and Outdated Pragma	Low	**Fixed [17/07/2023]**
Missing State Variable Visibility	Informational	**Fixed [17/07/2023]**
Missing Zero Address Validations	Low	**Fixed [17/07/2023]**
Missing Events in Functions	Low	**Fixed [17/07/2023]**
Boolean Equality	Gas	**Fixed [17/07/2023]**
Missing NatSpec Comments	Informational	**Fixed [17/07/2023]**
Variables should be Immutable	Gas	**Fixed [17/07/2023]**
Modifier Created But Never Used	Gas	**Fixed [17/07/2023]**

Hardcoded Static Address	Informational	**Won’t Fix**
Functions should be declared External	Gas	**Fixed [17/07/2023]**
Superfluous Event Field	Gas	**Fixed [17/07/2023]**
Public Constants can be Private	Gas	**Fixed [17/07/2023]**
Gas Optimization in Require Statements	Gas	**Fixed [17/07/2023]**
Potential Signature Replay Attack	Medium	**Fixed [17/07/2023]**
Use Multisig for Admin Functions	Low	**Fixed [17/07/2023]**
Gas Optimization in For Loops	Gas	**Fixed [17/07/2023]**
Unnecessary Multiple Payable Functions	Low	**Fixed [17/07/2023]**
Missing Zero Address Validations	Low	**Fixed [17/07/2023]**
Missing Input Validation	Medium	**Fixed [17/07/2023]**
Lack of Unique Signer Verification	Medium	**Fixed [17/07/2023]**
Missing Zero Address Validations For recover() Function	Medium	**Fixed [17/07/2023]**
Signature Replay Attack	Critical	**Fixed**

		[17/07/2023]
Functions should use Multisig	Low	**Won’t Fix**
Use Call instead of Transfer	Informational	**Fixed [17/07/2023]**
Unused Imports	Gas	**Fixed [17/07/2023]**
Locked Ether	Low	**Fixed [17/07/2023]**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1 [Fixed]

Floating and Outdated Pragma

Vulnerability Type
Floating Pragma (SWC-103)

Severity
Low

Description
Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.
The contract was allowing floating or unlocked pragma to be used, i.e., **>=0.8 <0.9.0.**
This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

Impacts
If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.
Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is really low therefore this is only informational.

Remediation
Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.18 pragma version
Reference: https://swcregistry.io/docs/SWC-103

Retest:

Strict pragma is in use everywhere now.

Bug ID #2 [Fixed]

Missing State Variable Visibility

Vulnerability Type
Missing Best Practices

Severity
Informational

Description
In Solidity, the visibility of state variables is important as it determines how those variables can be accessed and modified by other contracts or functions.
The contract defined state variables that were missing a visibility modifier.

Affected Code

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendAutomation.sol#L9
  - address _owner

Impacts
If the visibility of a state variable is accidentally left out, it can cause unexpected behavior and security vulnerabilities. For example, if a state variable is supposed to be private and is accidentally declared without any visibility keyword, it will be treated as "internal" by default, which may lead to it being accessible by other contracts or functions outside the intended scope. This can lead to a potential attack vector for malicious actors.

Remediation
Explicitly define visibility for all state variables. These variables can be specified as public, internal, or private.

Retest:
Variable visibility has been added.

https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/RentAndLend/RentLendAutomation.sol#L17

Bug ID #3 [Fixed]

Missing Zero Address Validations

Vulnerability Type

Missing Input Validation

Severity

Low

Description:

The contracts were found to be setting new addresses without proper validations for zero addresses.

Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.

Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Variables and Line Numbers

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendAutomation.sol#L35 - setLyncRentLendMarketplace() - _newAddress
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L144 - SetSignerAddress() - _newSignerAddress

Impacts

If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation

Add a zero address validation to all the functions where addresses are being set.

Retest

Zero address validation has been added.
https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/RentAndLend/RentLendAutomation.sol#L65
https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L234

Bug ID#4 [Fixed]

Missing Events in Functions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.

The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

Affected Code
The following functions were affected -

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L52-L54 - setFeesForAdmin()
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol - setLyncAddress(), setlyncFeePercent(), SetSignerAddress(), setBaseURI(), setMaxSupply(), setCost(), setMaxMintPerUser()
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol - setLyncAddress(), setlyncFeePercent(), SetSignerAddress(), setURI(), setMaxSupply(), setCost(), setMaxMintPerUser()
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyER

C721.sol - SetSignerAddress(), setMaxSupply(), setBaseURI(), setMaxMintAllowedPerUser()

Impacts

Events are used to track the transactions off-chain, and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Remediation

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

Retest

Events are emitted for all the functions mentioned above.

Bug ID#5 [Fixed]

Boolean Equality

Vulnerability Type

Gas Optimization

Severity

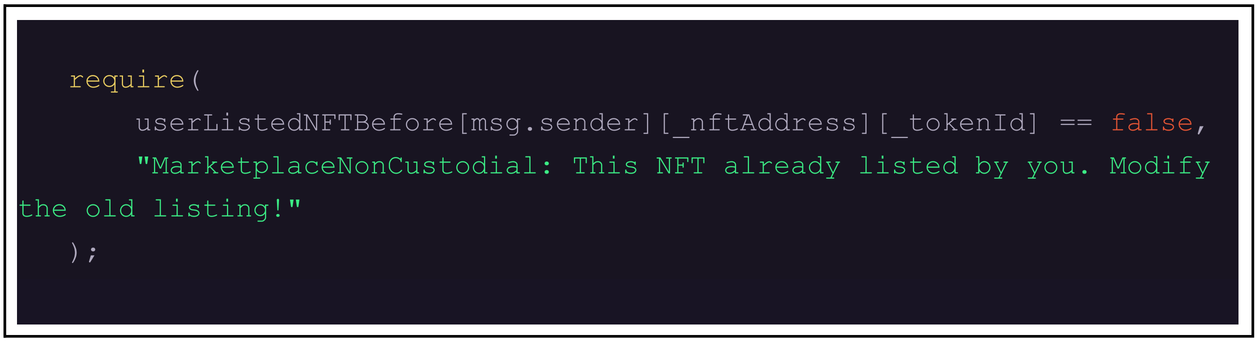
Gas

Description

The contract was found to be equating variables with a boolean constant inside a “require()” statement which is not recommended and is unnecessary.
Boolean constants can be used directly in conditionals.

Affected Code

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L98
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L265
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L61
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L399
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L596
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L661



```
   require(
       userListedNFTBefore[msg.sender][_nftAddress][_tokenId] == false,
       "MarketplaceNonCustodial: This NFT already listed by you. Modify
the old listing!"
   );
```

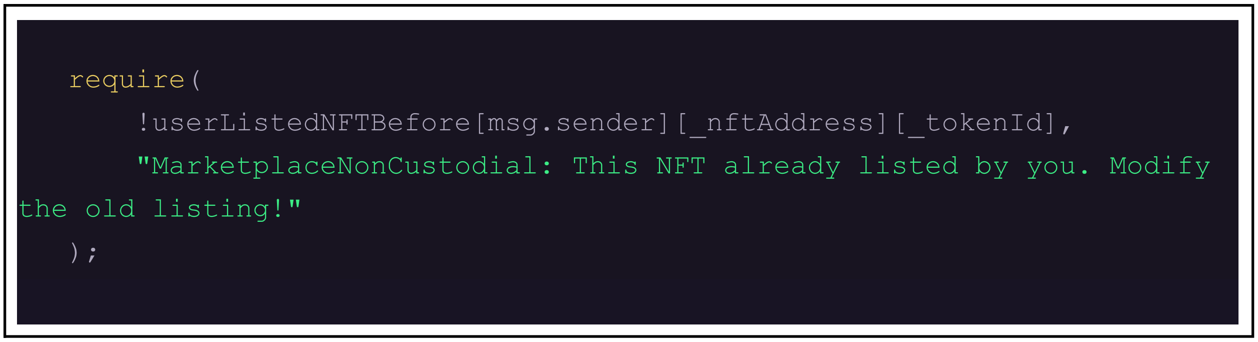
Impacts

Equating the values to boolean constants in conditions cost gas and can be used directly.

Remediation

It is recommended to use boolean constants directly. It is not required to equate them to true or false.

Eg:



```
   require(
       !userListedNFTBefore[msg.sender][_nftAddress][_tokenId],
       "MarketplaceNonCustodial: This NFT already listed by you. Modify
the old listing!"
   );
```

Retest

This is fixed.

Bug ID#6 [Fixed]

Missing NatSpec Comments

Vulnerability Type

Missing best practices

Severity

Informational

Description:

Solidity contracts use a special form of comments to document code. This special form is named the Ethereum Natural Language Specification Format (NatSpec).

The document is divided into descriptions for developers and end-users along with the title and the author.

The contract is missing NatSpec comments in the code which makes it difficult for the auditors and future developers to understand the code.

Affected Code:

- All contracts

Impacts:

Missing NatSpec comments and documentation about a library or a contract affect the audit and future development of the smart contracts.

Remediation:

Add necessary NatSpec comments inside the library along with documentation specifying what it’s for and how it’s implemented.

Retest:

NatSpec comments have been added everywhere in all the contracts

Bug ID #7 [Fixed]

Variables should be Immutable

Vulnerability Type
Gas Optimization

Severity
Gas

Description:
Declaring state variables that are not updated following deployment as immutable can save gas costs in smart contract deployments and function executions. Immutable state variables are those that cannot be changed once they are initialized, and their values are set permanently.

By declaring state variables as immutable, the compiler can optimize their storage in a way that reduces gas costs. Specifically, the compiler can store the value directly in the bytecode of the contract, rather than in storage, which is a more expensive operation.

Affected Code:

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendAutomation.sol#L9

Impacts:
Gas usage is increased if the variables that are not updated outside of the constructor are not set as immutable.

Remediation:
An "`immutable`" attribute should be added in the parameters that are never updated outside of the constructor to save the gas.

Retest
This won't be needed as the owner variable is being updated in the same contract.

Bug ID #8 [Fixed]

Modifier Created But Never Used

Vulnerability Type

Dead Code

Severity

Gas

Description:

Solidity is a language that uses gas on deployment. Each unit of gas costs real money or ether. This makes it crucially important that there's no instance of dead or unused code throughout the smart contract.
The contract was defining a modifier but none of the functions were found to be using it.

Affected Code:

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L33-L36

Impacts:

Missing modifier usage shows either critical missing access control or the presence of dead code.

Remediation:

It is recommended to go through the code to check if there is a missing requirement for a modifier on any critical function.
If this is not required, remove the definition from the code as well.

Retest

The dead code has been removed.

Bug ID#9 [Won’t Fix]

Hardcoded Static Address

Vulnerability Type

Missing Best Practises

Severity

Informational

Description

The contracts were found to be using hardcoded addresses on multiple lines. This could have been optimized using dynamic address update techniques along with proper access control to aid in address upgrade at a later stage.

Affected Code

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155Collection.sol#L19
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/newBicoERC721.sol#L17
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollection.sol#L19
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L30
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L20
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L16

Impacts

Hardcoding address variables in the contract make it difficult for it to be modified at a later stage in the contract as everything will need to be deployed again at a different address if there’s a code upgrade.

Remediation

It is recommended to create dynamic functions to address upgrades so that it becomes easier for developers to make changes at a later stage if necessary.

The said function should have proper access controls to ensure only administrators can call that function using access control modifiers.

There should also be a zero address validation in the function to ensure inconsistencies are not introduced.

If the address is supposed to be hardcoded, it is advisable to make it a constant if its value is not getting updated.

Retest:

The fix would increase gas cost and the possible exploitation scenario is negligible.

Bug ID#10 [Fixed]

Functions should be declared External

Vulnerability Type

Gas Optimization

Severity

Gas

Description

Public functions that are never called by a contract should be declared external in order to conserve gas.

The following functions were declared as public but were not called anywhere in the contract, making public visibility useless.

Affected Code

The following functions were affected -

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L160-L165
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L81-L100
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L140-L142

Impacts

Smart Contracts are required to have effective Gas usage as they cost real money and each function should be monitored for the amount of gas it costs to make it gas efficient.

“**public**” functions cost more Gas than “**external**” functions.

Remediation

Use the “**external**” state visibility for functions that are never called from inside the contract.

Retest

The required functions have been marked external now

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L259-L264
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L226-L228
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L161-L178

Bug ID#11 [Fixed]

Superfluous Event Field

Vulnerability Type

Gas Optimization

Severity

Gas

Description

block.timestamp and block.number is by default added to event information. Adding them manually costs extra gas.

Affected Code

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L310-L319

Impacts

Emitting block values in event costs extra gas as it is not required and available by default.

Remediation

block.timestamp do not need to be added manually. Consider removing it from the emitted events.

Retest

Superfluous fields have been removed.
https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/RentAndLend/RentLendMarketplace.sol#L368-L374

Bug ID#12 [Fixed]

Public Constants can be Private

Vulnerability Type

Gas Optimization

Severity

Gas

Description

Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.

Affected Code

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L25
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L12
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L13
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L13
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L14

Impacts

Public constants are more costly due to the default getter functions created for them, increasing the overall gas cost.

Remediation

If reading the values for the constants is not necessary, consider changing the public visibility to private.

Retest

The public constants have been updated to private.

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L43
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/marketplace/MarketplaceNonCustodial.sol#L20
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/marketplace/MarketplaceNonCustodial.sol#L23
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/RentAndLend/RentLendMarketplace.sol#L20
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/1aea47fa35fc38fec67ad74ecf1bf3283a22f586/contracts/RentAndLend/RentLendMarketplace.sol#L23

Bug ID#13 [Fixed]

Gas Optimization in Require Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The **require()** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset, and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Vulnerable Code

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/creator-contracts/ERC721ACreator.sol#L46
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#l125
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L149
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L102
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L143

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L154
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/marketplace/MarketplaceNonCustodial.sol#L220
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L32
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L63
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L71
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L95
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L107
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L125
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L129
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L133
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L137

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L141
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L159
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L169
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L188
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L42
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L74
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L82
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L90
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L110
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L137
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L141

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L145
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L153
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L153
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L183
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/creator-contracts/ERC1155Creator.sol#L47
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L60
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L69
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L85
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L182
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L191
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L198

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L209
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L218
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L250
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L255
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L291
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L370
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L480
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L526
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L535

Impacts

Having longer require strings than 32 bytes costs a significant amount of gas.

Remediation

It is recommended to shorten the strings passed inside **require()** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest

The require statements have been shortened to less than 32 bytes.

Bug ID#14 [Fixed]

Potential Signature Replay Attack

Vulnerability Type
Signature Reply attack

Severity
Medium

Description
The NFT contracts use a function called “recoverSigner()” to check if the signature and the hash match.
This function lacks certain validations:

- There is no deadline for signatures. NFTs can be minted anytime in the future.
- The signature can be replayed on other EVM-compatible chains on which the NFT contracts are deployed.
- The signature can be replayed on multiple instances of the NFT contracts if they are deployed on the same chain.

Vulnerable Code

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L81-L100 - mintNewNFT()
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L118-L149 - mintNFT()
- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L129-L161 - mintNFT()

Impacts

There’s no time validation which could allow anyone to mint NFTs anytime in the future and on multiple chains and instances if they are deployed. This would allow unintentional minting of NFTs.

Remediation

Consider including deadline, chainid and NFT contract’s address in the signature message. Ideally signatures should be created according to the EIP712 standard.

Retest

This is fixed. A getHash() function is introduced to get the current hash using the contract’s address, timestamp, chainID, and nonce.

Bug ID#15 [Fixed]

Use Multisig for Admin Functions

Vulnerability Type

Business Logic

Severity

Low

Description

Smart contracts can have privileged roles that are not the ideal but are often necessary in the early lifecycle of a project. Over time, a project can become more and more decentralized, either disabling privileged roles or placing them under the control of the community.

In the meantime, admin accounts with such privileged roles need to be protected. If such an account were to fall in the hands of a malicious attacker, they could wreak havoc on your system. Therefore, it is recommended to use a Multisig address for admin only functions. A multisig is a contract that can execute actions, as long as a predefined number of trusted members agree upon it.

Impacts

If the only admin account address gets compromised, the contract could be destroyed or the funds could be stolen.

Remediation

It is recommended to implement a Multisig in admin privileged functions so multiple approvals are required when updating critical values.

Retest

Multisig has been implemented for admin functions in the Marketplace and RentLend contracts.

Bug ID#16 [Fixed]

Gas Optimization in For Loops

Vulnerability Type

Gas Optimization

Severity

Gas

Description

Loops are in most cases bounded by definition (the bounding is represented by the exit condition). Therefore in the vast majority of cases, checking for overflows is really not needed, and can get very gas expensive. Here’s an example:

```
pragma solidity ^0.8.0;

contract Test1 {


    function loop() public pure {

        for(uint256 i = 0; i < 100; i++) {
        }

    }
}
```

```
pragma solidity ^0.8.0;



contract Test {

    function loop() public pure {

        for(uint256 i = 0; i < 100;) {

            unchecked {

                i++;

            }

        }

    }

}
```

loop() in Test1 costs more than 31K gas, vs 25.5K gas for loop() in Test2.

Impacts

Removing overflow validations using unchecked blocks will save gas in the loops.

Remediation
It is recommended to implement unchecked blocks in for loops wherever possible since they are already bounded by an upper length and there’s a very rare chance that it might overflow.

Retest
Loop increment variables have ben marked unchecked to save gas.

Bug ID#17 [Fixed]

Unnecessary Multiple Payable Functions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
The contracts define empty receive and fallback payable functions that do not serve any other purpose but to receive Ether into the contract. This is redundant and unnecessary and it is recommended to keep only one of these functions.

Affected Code

- biconomyERC721.sol - https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/biconomy/biconomyERC721.sol#L171-L175
- ERC721ACollectionBase.sol - https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC721ACollectionBase.sol#L191-L193
- ERC1155CollectionBase.sol - https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/deployer-contracts/launch-collection/ERC1155CollectionBase.sol#L186-L189
- RentLendMarketplace.sol - https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/0e38c97a643beec75e8857270a16a3f36d9aee5b/contracts/RentAndLend/RentLendMarketplace.sol#L518-L522

Impacts

Smart Contracts are required to have effective Gas usage as they cost real money and each function should be monitored for the amount of gas it costs to make it gas efficient.
Having redundant code in the production codebase just increases the overall size and deployment costs.

Remediation

If the purpose of the contracts is just to receive Ether, it is recommended to keep only the receive() function. The deposit() function can also be kept if the developers want users to deposit Ether by calling another function.

Retest

Unnecessary payable functions are removed.

Bug ID #18 [Fixed]

Missing Zero Address Validations

Vulnerability Type
Missing Input Validation

Severity
Low

Description:
The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Variables and Line Numbers

- LyncMultiSig.withdrawAnyFunds - _to
- MarketplaceNonCustodial.withdrawFunds - _to

Impacts
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest
Zero address validation has been added.

Bug ID #19 [Fixed]

Missing Function to Update Owners

Vulnerability Type
Business Logic

Severity
Medium

Description:
The contract LyncMultiSig has an array of owners which store the administrators who will call the privileged functions. This array is initialized and updated inside the contract's constructor. However, there's no function to update this array to remove the owners. This could lead to privilege compromise in case the owner goes rogue.

Affected Variables and Line Numbers

- https://github.com/LYNC-WORLD/Smart-Contracts-Auditing/blob/be85cc5eebe4e607d84161423e2cd350deb8c93b/contracts/multisig/LyncMultiSig.sol

Impacts
Due to the missing function to update or remove an owner, the array will never be updated after the initial contract deployment and it might get compromised in case one of the owners go rogue.

Remediation
It is recommended to add a function to remove an owner address from the owner's array.

Retest
A function has been added to delete all previous owners and update them with the new values.

Bug ID #20 [Fixed]

Lack of Unique Signer Verification

Vulnerability Type

Business Logic

Severity

Medium

Description:

The _verifySignatures function in the provided Solidity code lacks a proper check to ensure that each signature is associated with a unique signer. This vulnerability allows a single wallet to sign multiple signatures, potentially bypassing the requirement of having distinct signers for each signature.

Affected Variables and Line Numbers

- LyncMultiSig.getSigner - signerAddress

Impacts

The impact of this vulnerability can vary depending on the specific use case and the significance of the multi-signature functionality. However, allowing a single wallet to sign multiple signatures can undermine the security assumptions and compromise the integrity of the multi-signature mechanism. It can lead to unauthorized or unintended execution of actions, potentially resulting in financial loss or unauthorized access to sensitive resources

Remediation

To address this vulnerability, it is recommended to implement a mechanism that ensures each signature is associated with a unique signer. This can be achieved by keeping track of the signers in a separate mapping and verifying that each signer can only sign once.

Retest

This is fixed by implementing signedAlready mapping.

Bug ID #21 [Fixed]

Missing Zero Address Validations For recover() Function

Vulnerability Type

Missing Validation

Severity

Medium

Description:

The `recover` function is used to extract the signer address from a message hash and a corresponding signature. If the signature is invalid or an error occurs during the recovery process, it will return the zero address. You can check the official documentation of Openzeppelin.

Affected Variables and Line Numbers

- LyncMultiSig.getSigner - signerAddress
- gasLesslync.verifySignature - ethSignedHash.recover(_sig)
- ERC721ACollectionBase.verifySignature - ethSignedHash.recover(_sig)
- ERC1155CollectionBase.verifySignature - ethSignedHash.recover(_sig)

Impacts

In function _verifySignatures it checks whether the returned address is the owner or not in this case provided signature will lead to a used signature in usedSigs mapping although this signature is a wrong signature.

Remediation

Add a zero address validation for the variable signerAddress in the signerAddress() function.

Retest

This is fixed. 0 address validation has been added wherever needed.

Bug ID #22 [Fixed]

Signature Replay Attack

Vulnerability Type

Business Logic

Severity

Critical

Description:

The `verifySignature()` function in the Solidity code does not adequately verify whether the provided signature has been previously used or not. The function primarily relies on the `recover()` function to verify the signer of the signature. However, it does not incorporate a check to ensure that the provided signature has not been used before

Affected Variables and Line Numbers

- gasLesslync.mintNFT()
- ERC721ACollectionBase.mintNFT()
- ERC1155CollectionBase.mintNFT()

Impacts

Attackers can spend the same NFT multiple times by replaying a valid signature, leading to financial losses for collectors and investors who unknowingly acquire counterfeit or devalued NFTs. Exploiting the vulnerability allows attackers to create multiple identical NFTs by replaying a valid signature. This dilutes the uniqueness and scarcity of the original NFT, potentially harming its value and reputation

Remediation

To address this vulnerability, it is recommended to implement a mechanism that ensures signature has been used or not. This can be achieved by keeping track of the signature with bool value.

Retest

This is fixed. The code now maintains and validates the used signatures and reverts if someone reuses the signature.

Bug ID #23 [Won’t Fix]

Functions should use Multisig

Vulnerability Type
Business Logic

Severity
Low

Description:
Smart contracts can have privileged roles that are not ideal but are often necessary in the early lifecycle of a project. Over time, a project can become more and more decentralized, either disabling privileged roles or placing them under the control of the community.
In the meantime, admin accounts with such privileged roles need to be protected. If such an account were to fall in the hands of a malicious attacker, they could wreak havoc on your system. Therefore, it is recommended to use a Multisig address for admin-only functions. A multisig is a contract that can execute actions, as long as a predefined number of trusted members agree upon it.

Affected Code

- onlyOwner functions used across the contract code

Impacts
If the only admin account address gets compromised, the contract could be destroyed or the funds could be stolen.

Remediation
It is recommended to implement a Multisig in admin-privileged functions so multiple approvals are required when updating critical values.

Retest
This contract doesn’t require multisig as per business requirements.

Bug ID #24 [Fixed]

Use Call instead of Transfer

Vulnerability Type

Best Practices

Severity

Informational

Description:

Using Solidity's transfer function has some notable shortcomings when the withdrawer is a smart contract, which can render ETH deposits impossible to withdraw. Specifically, the withdrawal will inevitably fail when:

- The withdrawer smart contract does not implement a payable fallback function.
- The withdrawer smart contract implements a payable fallback function which uses more than 2300 gas units.
- The withdrawer smart contract implements a payable fallback function which needs less than 2300 gas units but is called through a proxy that raises the call's gas usage above 2300.

Affected Code

- gasLesslync.withdrawFunds()
- ERC721ACollectionBase._chargeFee()
- ERC1155CollectionBase._chargeFee()
- MarketplaceNonCustodial._splitFunds(), _sellerAdminSplit()
- RentLendMarketplace.returnRented(), _splitFunds(), _returnRentedUsingAutomation()

Impacts

The transfer function has some restrictions when it comes to sending ETH to contracts in terms of gas which could lead to transfer failure in some cases.

Remediation

It is recommended to transfer ETH using the call() function, handle the return value using require statement, and use the nonreentrant modifier wherever necessary to prevent reentrancy.
Ref: https://solidity-by-example.org/sending-ether/

Retest
All the transfer methods have been updated to call for sending ETH.

Bug ID#25 [Fixed]

Unused Imports

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Solidity is a Gas-constrained language. Having unused code or import statements incurs extra gas usage when deploying the contract.
The contracts were found to be importing the file “Strings.sol” which is not used anywhere in the code.

Affected Code

- ERC721ACreator.sol
- ERC1155Creator.sol
- ERC721AGameLauncher.sol
- ERC1155CollectionBase.sol

Impacts
Having unnecessary dead code in the production contract costs extra gas and makes it difficult for auditors and developers.

Remediation
It is recommended to remove the import statement if it’s not supposed to be used.

Retest
Unused imports have been removed.

Bug ID#26 [Fixed]

Locked Ether

Vulnerability Type
Business Logic

Severity
Low

Description
The contract RentLendMarketplace has a payable receive() function which is capable of receiving ETH.
However, there's no functionality in the contract to withdraw the ETH out, essentially locking the ETH in the contract forever.

Affected Code

- RentLendMarketplace.receive()

Impacts
This issue will lock the ETH received by the contract using the receive() function with no way of withdrawing it from the contract.

Remediation
It is recommended to add a withdraw function that can only be called by owners to be able to withdraw the contract balance.

Retest
The receive function is now adding sent ETH to the withdrawableAmount that can be withdrawn by the admin.

6. Disclosure

The Reports provided by CredShields is not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.