

SON-ERGY

SMART CONTRACT INITIAL AUDIT REPORT - NOV. 2022

Prepared by:

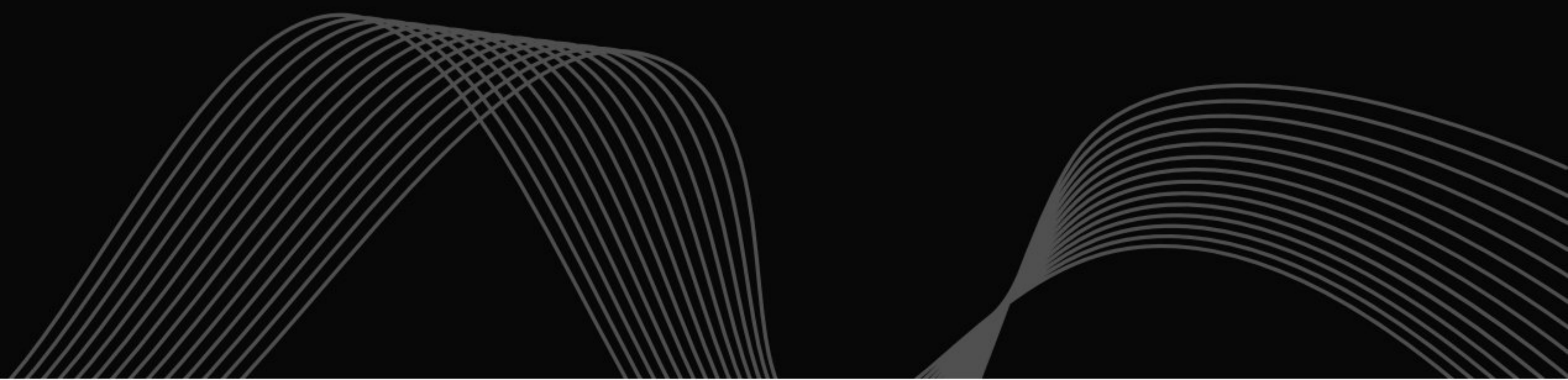
A&D Forensics

contactus@adforensics.com.ng



Table of Contents

Executive Summary	03
Project Audit Scope and Findings	04
Mode of Audit and Methodologies	05
Functional Testing	06
Report Findings	07
Closing Summary	19
Appendix	20
Disclaimer	21
Contact Information	22



Executive Summary

Sonergy is the first globally recognized blockchain-based data integrity system that is actively reshaping the research sector. Sonergy is made to let businesses and researchers interact to get high-quality, practical, and factual insights to guide their business plans

Project Audit Scope and Findings

The motive of this audit is to review the codebase of Sonergy Protocol smart contracts for the purpose of achieving secured, corrected and quality contracts.

Number of Contracts in Scope:

- Sonergy.sol Contract
- sonergySurveysNFT.sol
- surveyNFTMarket.sol
- surveysV2.sol

Link to Project codebase:

<https://github.com/SONERGY-PROTOCOL/smart-contracts/tree/main/contracts>

Github Commit: -38e41c41ee39b2a2d4b702e579fcf2f06bd01f95

Number of issues per severity

TYPE	HIGH	MEDIUM	LOW	INFORMATIONAL
Open	0	0	0	0
Acknowledged	0	0	0	0
Closed	1	9	4	6

AUDITOR:

www.github.com/davidpius95

Twitter: @davidpius10

Auditors: David Uzochukwu

Mode of Audit and Methodologies

The mode of audit carried out in this audit process is as follow:

- **Manual Review:** This is the first and principal step carried out to understand the business logic behind a project. At this point, it involves research, discussion and establishment of familiarity with contracts. Manual review is critical to understand the nitty-gritty of the contracts.
- **Automated Testing:** This is the running of tests with audit tools to further cement discoveries from the manual review. After a manual review of a contract, the audit tools which could be Slither, Echidna, or Mythril are run on the contract to find out issues.
- **Functional Testing:** Functional testing involves manually running unit, static, and dynamic testing. This is done to find out possible exploit scenarios that could be used to steal money from the contracts. This helps understand the functionality of the contracts and find out lapses in the reverts check in contract.

Tools and Platforms used for Audit

Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Report of Findings

A. Common Issues

High Severity Issues

A.1: Common Issues | Resolved

Contract Affected:

In surveysV2.sol: L35,L25 , L26, L353, L357, L17

In surveysNFTMarket.sol:

In sonergySurveyNFT.sol: L15, L16, L17

Description :

This variable is declared but not initialized which will be added to the contract byte code and it has no use.

Remediation: This variable should be removed if not in use

Status: **Closed**

Medium Severity Issues.

No issues were found

Low Severity Issues

A.2: Missing events for significant actions | Resolved

Contract Affected: Isurveys.sol, surveysNFTMarket.sol
sonergySurveysNFT.sol

Description :

Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it.

Recommendation : It is recommended to emit an event about it

Status: **Closed**

Informational Severity Issues

A.3: Floating Pragma | Resolved

Contract Affected:

In surveysV2.sol:

In surveysNFTMarket.sol:

In sonergySurveyNFT.sol:

In sonergy.sol:

Description :

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation:

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References

- Matic Smart Contract Best Practices - Lock pragmas to specific compiler version

Status: **Closed**

A.4 Missing Testcases | Resolved

Contract Affected:

lsurveysV2.sol, surveysNFTMarket.sol, sonergySurveysNFT.sol

Description :

Test cases for the code and functions have not been provided.

Remediation:

It is recommended to write test cases of all the functions. Any existing tests that fail must be resolved. Tests will help in determining if the code is working in the expected way. Unit tests, functional tests, and integration tests should have been performed to achieve good test coverage across the entire codebase.

Status: Closed

A.5 Public functions that could be declared external in order to save gas | Resolved

Contract Affected:

surveysV2.sol, surveysNFTMarket.sol, sonergySurveysNFT.sol

Description :

Whenever a function is not called internally, it is recommended to define them as external instead of the public, in order to save gas. For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If the function is only called externally, then you should explicitly mark it as external. External function parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge

Remediation:

Consider declaring the above functions as external in order to save some gas.

Status: Closed

A.6 Missing zero address validation | Resolved

Contract Affected:

In surveysV2.sol: In surveysNFTMarket.sol: In sonergySurveyNFT.sol: L30

Description :

There are multiple functions in contracts that are missing zero address validation. Adding a zero address check is necessary because, in Matic, a zero address is something to which if any funds or tokens are transferred, it can not be retrieved back. In this case, there won't be any loss of token but if the amount or address is zero it would be a wastage of gas and might cause some other issues. Hence, it is recommended to add a check for zero addresses.

Remediation:

Consider adding a required statement that validates input against zero address to mitigate the same.

Status: **Closed**

A.7 Unused Code | Resolved

Contract Affected:

surveysV2.sol, surveysNFTMarket.sol, sonergySurveysNFT.sol

Description :

While auditing, it has been found that the contracts have multiple lines of code which are not being used. It is recommended to remove the unused code in order to save gas and improve the overall code quality

Remediation:

Consider Removing the unused code

Status: **Closed**

A.8 Lack of Code Base Documentation or inline comment | Resolved

Contract Affected:

surveysV2.sol, surveysNFTMarket.sol, sonergySurveysNFT.sol

Description :

The code base doesn't have documentation or inline comment which will help for readability and a better understanding of the code base and the intention of the developer

Remediation:

It is recommended that documentation or inline comment using the NetSpec standard is done for better readability of the system.

Status: **Closed**

B. [SONERGY.sol](#)

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No Issues were found

C. [SONERGYSURVEYSNFT.sol](#)

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No Issues were found

D [SONERGYSURVEYSNFT.sol](#)

High Severity Issues

No issues were found

Medium Severity Issues

D.1: Denail of Service: | Resolved

Contract Affected:

Description: The function createMarketSale requires the user msg. value must be equal to the price which will not work

Remediation: I recommend that the operation should be changed from == to >= this implies that the user msg. sender must be greater or equal to price

Status: **Closed**

Low Severity Issues

D.2: Lack payable fallback or payable receive function | Resolved

Contract Affected:

Description: For Matic to be sent to a contract the smart contract has to implement either a payable fallback function or a payable receive function, but this contract did not implement any of this function

Remediation: It recommended that a payable fallback or payable receive function is implemented if the smart contract will receive Matic.

Status: Closed

D.3: Wrong Implementation Logic | Resolved

Contract Affected:

Description:

createMarketSale, this function is indented for a user to buy an NFT , during this transaction user will pay price of nft, transfer the price of nft to seller address and transfer listingprice to owner address(deployer of contract). But the contract is transferring listing price after it has transfer the price of nft so there will be no funds to execute the call which will transfer listing price to owner. so transaction will fail

Remediation:

This function should be reviewed and implemented base on the business logic

Status: Closed

D.5: Transfer has been Deprecated **Deprecated | Resolved**

Contract Affected:

Description:

The use of transfer has been deprecated it uses 23000 gas

Reference:

<https://www.immunebytes.com/blog/transfer-in-solidity-why-you-should-stop-using-it/>

Remediation:

It recommended to use the call method instead of send or transfer

Status: **Closed**

E. SONERGY.sol

High Severity Issues

No issues were found

Medium Severity Issues

E.1 : Locked Matic | Resolved

Contracts Affected:

Description:

The locked Matic bug occurs in contracts that can receive Matic but do not allow users to extract Matic from them (nor to destroy them).

Remediation:

It recommended to implement a withdraw function that send Matic out of the contract

Status: **Closed**

E.2 Ownership Transfer must be tow-step process. | Resolved

Contracts Affected -

Description:

The contract uses openzeppelin's ownable contract to manage ownership. The `transferOwnership()` function in ownable contract allows the current owner to transfer his privileges to another address. However, inside `transferOwnership()`, the `newOwner` is directly stored in the storage, `owner`, after validating the `newOwner` is a non-zero address, which may not be enough.

The `newOwner` is only validated against the zero address. However, if the current admin enters a wrong address by mistake, he would never be able to take the management permissions back. Besides, if the `newOwner` is the same as the current admin address stored in `_owner`, it's a waste of gas.

Remediation:

It would be much safer if the transition is managed by implementing a two-step approach: `_transferOwnership()` and `_updateOwnership()`. Specifically, the `_transferOwnership()` function keeps the new address in the storage, `_newOwner`, instead of modifying the `_owner()` directly. The `updateOwnership()` function checks whether `_newOwner` is `msg.sender`, which means `_newOwner` signs the transaction and verifies himself as the new owner. After that, `_newOwner` could be set

Status: **Closed**

E.3 Missing Zero Address Validation | Resolved

Contracts Affected: L30, L113

Description:

There are multiple functions in contracts that are missing zero address validation. Adding a zero address check is necessary because, in Matic, a zero address is something to which if any funds or tokens are transferred, it can not be retrieved back. In this case, there won't be any loss of token but if the amount or address is zero it would be a waste of gas and might cause some other issues. Hence, it is recommended to add a check for zero addresses.

Remediation:

Consider adding a require statement that validates input against zero address to mitigate the same.

Status: **Closed**

E.4: Unused functions | Resolved

Contracts Affected:

Description:

hasValidator, hasvalidatorAnswer, and isUserVerified have no use to the code. Although it is allowed, it is best practice to avoid unused variables. Unused variables can lead to a few different problems:

- Increase in computations (unnecessary gas consumption)
- Indication of bugs or malformed data structures
- Decreased code readability

Remediation:

Status: **Closed**

E.5 Unused State Variable | Resolved

Contracts Affected:

Description:

hasValidator, hasvalidatorAnswer, and isUserVerified has no use to the code. Although it is allowed, it is best practice to avoid unused variables.

Unused variables can lead to a few different problems:

- Increase in computations (unnecessary gas consumption)
- Indication of bugs or malformed data structures
- Decreased code readability

Remediation:

Remove unused variables from the code to avoid unnecessary hike in gas during deployment.

Status: **Closed**

E.6: Gas Optimization | Resolved

Contracts Affected:

Description:

Storing strings on the blockchain are expensive. some of the require statement has a long error message which will be store on the blockchain and take more gas

Remediation:

I recommend that the error message should be shorten or use of custom error

Status: **Closed**

E.7 Ignore Return Value | Resolved

Contracts Affected:

Contracts Affected: L153, 619, 640, 708

Description:

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behaviour in the subsequent program logic.

-

Remediation:

Status: **Closed**

E.8: Wrong transfer implementation | Resolved

Contracts Affected: L153

Description:

The function undoValidator revoke a validator right by setting isValidator mapping to false, line 153 is transferring the sonergy token from the user address to the same user address

Remediation: I recommend the logic of this function is reviewed and

Status: **Closed**

Closing Summary

There were discoveries of some high, medium, low and informational issues after the audit. The audit team thereafter suggested some remediation to help remedy the issues found in the contract.

Appendix

- **Audit:** The review and testing of contracts to find bugs.
- **Issues:** These are possible bugs that could lead exploits or help redefine the contracts better.
- **Slither:** A tool used to automatically find bugs in a contract.
- **Severity:** This explains the status of a bug.


Disclaimer


While the audit report is aimed at achieving a quality codebase with assured security and correctness, it should not be interpreted as a guide or or recommendation for people to invest in Sonergy contracts.

With smart contract audit being a multifaceted process, we admonish the Sonergy team to carry out further audit from other audit firms or provide a bug bounty program to ensure that more critical audit is done to the contract.


CONTACT US

    A&D Forensics

 +234 909 550 3040

 www.adforensics.com.ng

 contactus@adforensics.com.ng

 No 3. Rabat Street, Wuse Zone 6, FCT Abuja, Nigeria

