d3ploy

d3ploy

SECURITY ASSESSMENT

ProxyMeta Labs

September 21st 2023

TABLE OF

Contents

LEGAL

Disclaimer

D3ploy audits are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts d3ploy to perform a security review. D3ploy does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

D3ploy audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort. The report is provided only for the contract(s) mentioned in the report and does not include any other potential additions and/or contracts deployed by Owner. The report does not provide a review for contract(s), applications and/or operations, that are out of this report scope.

D3ploy’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

D3ploy represents an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. D3ploy’s position is that each company and individual are responsible for their own due diligence and continuous security. The security audit is not meant to replace functional testing done before a software release. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits and a public bug bounty program to ensure the security of the smart contracts.

D3PLOY

Introduction

D3ploy is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Secure your project with d3ploy

We offer field-proven audits with in-depth reporting and a range of suggestions to improve and avoid contract vulnerabilities. Industry-leading comprehensive and transparent smart contract auditing on all public and private blockchains.

Vunerability checking

A crucial manual inspection carried out to eliminate any code flaws and security loopholes. This is vital to avoid vulnerabilities and exposures incurring costly errors at a later stage.

Contract verification

A thorough and comprehensive review in order to verify the safety of a smart contract and ensure it is ready for launch and built to protect the end-user

Risk assessment

Analyse the architecture of the blockchain system to evaluate, assess and eliminate probable security breaches. This includes a full assessment of risk and a list of expert suggestions.

In-depth reporting

A truly custom exhaustive report that is transparent and depicts details of any identified threats and vulnerabilities and classifies those by severity.

Fast turnaround

We know that your time is valuable and therefore provide you with the fastest turnaround times in the industry to ensure that both your project and community are at ease.

Best-of-class blockchain engineers

Our engineers combine both experience and knowledge stemming from a large pool of developers at our disposal. We work with some of the brightest minds that have audited countless smart contracts over the last 4 years.

PROJECT

Introduction

Proxymeta Labs is a comprehensive toolset for web3 developers, that accelerates projects' web3 development and reduces costs. One can generate customizable smart contracts that meet all project requirements. The toolset is not a coding solution, rather a completely free of charge toolset, with no hidden fees.

Project Name	*ProxyMeta Labs*
Contract Name	-
Contract Address	-
Contract Chain	*Not yet deployed on mainnet*
Contract Type	*Smart Contract*
Platform	*EVM*
Language	*Solidity*
Network	*BNB Chain (BEP20)*
Codebase	*Private GitHub Repository*
Total Supply	-

INFO

Social

https://proxymetalabs.com/

https://twitter.com/Proxymeta_Labs

-

-

-

-

-

92

PASS

AUDIT

Score

✦ Issues	8
✦ Critical	0
✦ Major	1
✦ Medium	1
✦ Minor	5
✦ Informational	1
✦ Discussion	0

All issues are described in further detail on the following pages.

AUDIT Scope

GITHUB REPOSITORY	LOCATION
Sonny-CX/SC-Generation/contracts	✦ Private GitHub Repository
Sonny-CX/SC-Generation/app	✦ Private GitHub Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for ProxyMeta Labs to discover issues and vulnerabilities in the source code of the ProxyMeta Labs project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic, Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective in the comments below.

TIMESTAMP

Version v1.0

Date 2023/08/23

Descrption Layout project

Architecture / Manual review / Static & dynamic security testing

Summary

Version v1.1

Date 2023/09/21

Descrption Re-audit addressed issues

Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
Excess Eth Not Refunded To Buyer	✦ Medium	Fixed
Use of Ownable2Step for Safer Ownership Transfer	✦ Minor	Fixed
Missing Events	✦ Minor	Fixed
Use of Floating Pragma in Solidity Files	✦ Minor	Fixed
The function Should Be External	✦ Minor	Fixed
Gas Optimization In Increments	✦ Minor	Fixed
Gas Optimization in Require Statements	✦ Gas	Fixed
Path Traversal and File Overwrite	✦ Major	Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

Upon reviewing the code, it has been observed that the mint() & buyToken() function, which facilitates the purchase of NFTs, does not include a mechanism to refund any excess Ether sent by the buyer. While the function correctly checks whether the received Ether is greater than or equal to the required amount, it does not handle situations where the buyer sends an excessive amount of Ether.

AFFECTED CODE

- mint() & payment.js#buyToken()

IMPACTS

Buyers who accidentally send more Ether than required will not have the excess amount refunded. This can lead to a loss of funds for users and negatively impact their trust in the protocol.

Issue : Excess Eth Not Refunded To Buyer

Level : Medium

Type : Loss of Funds

Remediation : Modify the mint() & buyToken() functions to include a refund mechanism that calculates and refunds any excess Ether sent by the buyer beyond the required amount.

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

The smart contract has incorporated the use of the Ownable2Step pattern for managing ownership. This pattern enhances the safety of ownership transfers compared to the standard Ownable pattern. In the Ownable2Step pattern, ownership transfer is a two-step process where the new owner must explicitly accept ownership before the transfer is completed. This mitigates the risk of accidental or unauthorized ownership transfers to mistyped or unintended addresses

AFFECTED CODE

- ERC721.sol and ERC1155.sol

IMPACTS

The two-step ownership transfer process significantly lowers the risk of accidental ownership transfers to incorrect or mistyped addresses. The new owner must confirm the transfer, ensuring a deliberate and secure change of ownership.

Issue : Use of Ownable2Step for Safer Ownership Transfer

Level : Minor

Type : Business Logic

Remediation : Use Ownable2Step instead of Ownable

Alleviation / Retest : Fixed.
https://github.com/Sonny-CX/SC-Generation/blob/808ce04b1233b0b0d1a2fba9284998faf9ca3af2/contracts/templates/ERC1155/erc1155.js#L11

IN - DEPTH Vulnerabilities

DESCRIPTION

During the code review, it has come to light that the smart contract lacks the use of events within the functions. Events are an integral part of contract functionality, serving as a means to store arguments in the transaction's log, which is essential for off-chain transaction tracking. The absence of events within the functions may hinder the ability of developers and auditors to effectively monitor and analyze transactions associated with this operation

AFFECTED CODE

- withdrawAll()

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.

Issue : Missing Events

Level : Minor

Type : Missing Best Practices

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

Alleviation / Retest : Fixed
https://github.com/Sonny-CX/SC-Generation/blob/808ce04b1233b0b0d1a2fba9284998faf9ca3af2/contracts/templates/ERC1155/erc1155.js#L75

IN - DEPTH Vulnerabilities

DESCRIPTION

Upon reviewing the codebase, it has been observed that the smart contract uses a floating pragma directive in the Solidity source file. A floating pragma does not specify a fixed compiler version and instead relies on the specified version and any compatible later versions. This approach allows the contract to be compiled with various compiler versions, which may introduce inconsistencies or compatibility issues over time.

AFFECTED CODE

- ERC721.sol and ERC1155.sol

IMPACTS

Relying on a floating pragma means the contract may be compiled with different compiler versions, potentially leading to variations in behavior and output across different versions.

Issue : Use of Floating Pragma in Solidity Files

Level : Minor

Type : Floating Pragma

Remediation : To ensure consistent behavior, stability, and security of the contract, it is recommended to avoid the use of floating pragma in Solidity source files. Instead, adopt a specific compiler version that is thoroughly tested and verified

Alleviation / Retest : The Pragma version has been fixed and updated.
https://github.com/Sonny-CX/SC-Generation/blob/808ce04b1233b0b0d1a2fba9284998faf9ca3af2/contracts/templates/ERC1155/erc1155.js#L4

IN - DEPTH Vulnerabilities

DESCRIPTION

During the analysis of the code, it was identified that a function with a public visibility modifier has been used that is not called internally within the contract. The public visibility modifier makes a function accessible both internally within the contract and externally from other contracts. However, since this function is not invoked internally, it could be optimized for gas usage by changing its visibility to external.

AFFECTED CODE

- buyToken() in both the contracts

IMPACTS

Functions with public visibility involve copying arguments to memory, which can result in higher gas consumption, especially when dealing with large arrays of data. This can lead to increased gas costs for transactions that interact with the function.

Issue : The function Should Be External

Level : Minor

Type : Missing Best Practices

Remediation : To optimize gas usage and improve the efficiency of the contract, it is recommended to change the visibility modifier of the identified function from public to external

Alleviation / Retest : The functions have been made external.
https://github.com/Sonny-CX/SC-Generation/blob/808ce04b1233b0b0d1a2fba9284998faf9ca3af2/contracts/templates/ERC1155/components/payments.js#L20-L49

IN - DEPTH Vulnerabilities

DESCRIPTION

Upon reviewing the code, it has been identified that the contract uses various increment operations (i++, i += 1) for unsigned integer variables. However, it's worth noting that the ++i increment operation is more gas-efficient compared to i++ or i += 1. This is due to the compiler's optimization behavior, where ++i directly increments the value and returns it, resulting in lower gas consumption.

AFFECTED CODE

- ERC721.sol & ERC1155.sol#constructor & mint()

IMPACTS

The use of i++ or i += 1 increment operations can result in higher gas costs compared to ++i. This can contribute to increased transaction fees and operational costs.

Issue : Gas Optimization In Increments

Level : Minor

Type : Gas Optimization

Remediation : To optimize gas usage and improve the efficiency of the contract, it is recommended to replace instances of i++ and i += 1 with ++i for unsigned integer variables

Alleviation / Retest : This is fixed. +i is being used to save gas.
https://github.com/Sonny-CX/SC-Generation/blob/808ce04b1233b0b0d1a2fba9284998faf9ca3af2/contracts/templates/ERC1155/erc1155.js#L48

IN - DEPTH Vulnerabilities

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.

AFFECTED CODE

- Require statements inside the constructor and buyToken

IMPACTS

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

Issue : Gas Optimization in Require Statements

Level : Gas

Type : Gas Optimization

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.

Alleviation / Retest : All require statements are updated to fit under 32 bytes

IN - DEPTH Vulnerabilities

DESCRIPTION

The API is taking “user” and “name” parameters as input from the user. These parameters are being used to create files and directories for the generated smart contracts. Due to missing input validation on these parameters, it is possible to provide malicious inputs such as “../” which will lead to folder and file creation at the attacker-defined path.

PROOF OF CONCEPT

- 1. Send an API request using a manipulate “user” and “name” parameter as shown below.
- 2. It can be seen that the file was created at a different location.

IMPACTS

This vulnerability could allow attackers to create directories and files in the choice of their own folders anywhere on the server. They could also cause the application to crash by supplying invalid payloads creating a permanent Denial of Service.

Issue : Path Traversal and File Overwrite

Level : Major

Type : High

Remediation : Implement input validation on all the user input parameters. Ensure malicious characters are blocked and encoded properly before being used. Also, make sure that the server does not crash when invalid arguments are supplied. Implement proper error handling.

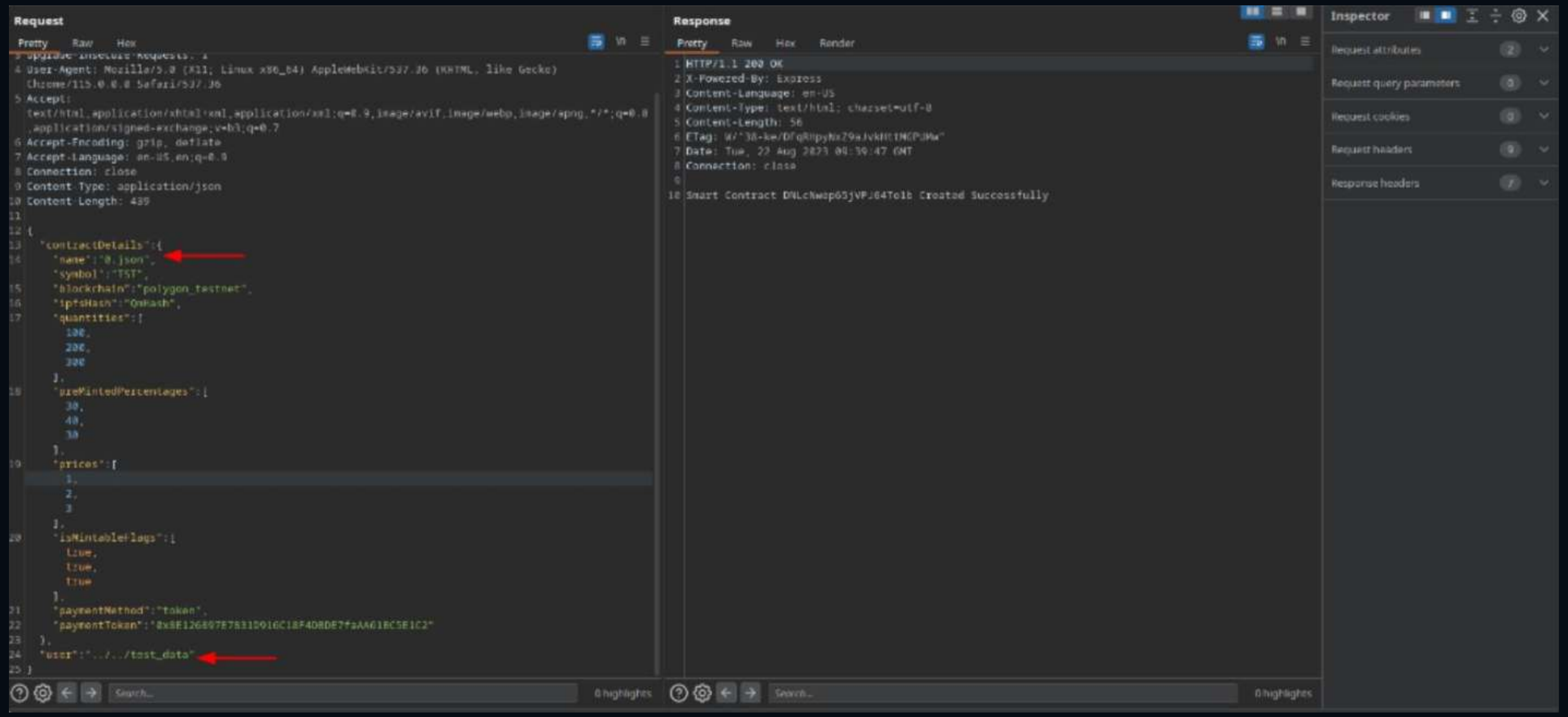
Alleviation / Retest : Validations have been added that validate name and address parameters which prevent path traversal

IN - DEPTH Vulnerabilities

PROOF OF CONCEPT

```
POST /contract/create HTTP/1.1
Host: 192.168.29.100:3000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/115.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Type: application/json
Content-Length: 439

{
  "contractDetails": {
    "name": "0.json","symbol": "TST",
    "blockchain": "polygon_testnet",
    "ipfsHash": "QmHash",
    "quantities": [100, 200, 300],
    "preMintedPercentages": [30, 40, 30],
    "prices": [1, 2, 3],
    "isMintableFlags": [true, true, true],
    "paymentMethod": "token",
    "paymentToken": "0x8E126897E7831D916C18F4DBDE7faAA61BC5E1C2"
  },
  "user": "../../test_data"
}
```

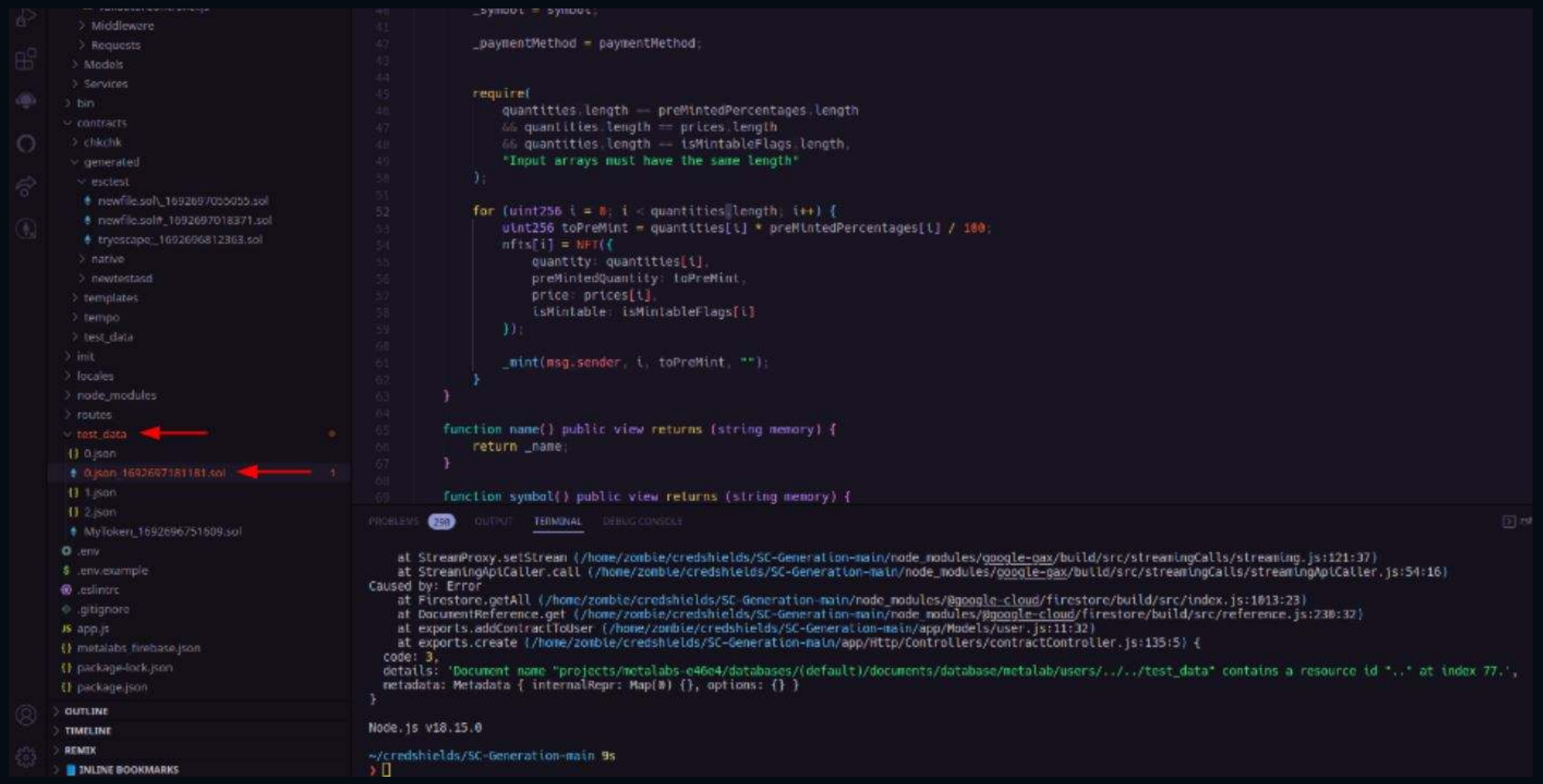


```
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/115.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Type: application/json
Content-Length: 439

{
  "contractDetails":{
    "name":"0.json",
    "symbol":"TST",
    "blockchain":"polygon_testnet",
    "ipfsHash":"QmHash",
    "quantities":[
      100,
      200,
      300
    ],
    "preMintedPercentages":[
      30,
      40,
      30
    ],
    "prices":[
      1,
      2,
      3
    ],
    "isMintableFlags":[
      true,
      true,
      true
    ],
    "paymentMethod":"token",
    "paymentToken":"0x8E126897E7831D916C18F4DBDE7faAA61BC5E1C2"
  },
  "user":"../../test_data"
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Language: en-US
Content-Type: text/html; charset=utf-8
Content-Length: 56
ETag: W/"38-ke/DfqRhpyNxZ9aJvkHt1MGPJMw"
Date: Tue, 22 Aug 2023 09:39:47 GMT
Connection: close

Smart Contract DNLcNwsp65jVPJ64Tolb Created Successfully
```



```
        _symbol = symbol;

        _paymentMethod = paymentMethod;


        require(
            quantities.length == preMintedPercentages.length
            && quantities.length == prices.length
            && quantities.length == isMintableFlags.length,
            "Input arrays must have the same length"
        );

        for (uint256 i = 0; i < quantities.length; i++) {
            uint256 toPreMint = quantities[i] * preMintedPercentages[i] / 100;
            nfts[i] = NFT({
                quantity: quantities[i],
                preMintedQuantity: toPreMint,
                price: prices[i],
                isMintable: isMintableFlags[i]
            });

            _mint(msg.sender, i, toPreMint, "");
        }
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
```

```
    at StreamProxy.setStream (/home/zombie/credshields/SC-Generation-main/node_modules/google-gax/build/src/streamingCalls/streaming.js:121:37)
    at StreamingApiCaller.call (/home/zombie/credshields/SC-Generation-main/node_modules/google-gax/build/src/streamingCalls/streamingApiCaller.js:54:16)
Caused by: Error
    at Firestore.getAll (/home/zombie/credshields/SC-Generation-main/node_modules/@google-cloud/firestore/build/src/index.js:1013:23)
    at DocumentReference.get (/home/zombie/credshields/SC-Generation-main/node_modules/@google-cloud/firestore/build/src/reference.js:230:32)
    at exports.addContractToUser (/home/zombie/credshields/SC-Generation-main/app/Models/user.js:11:32)
    at exports.create (/home/zombie/credshields/SC-Generation-main/app/Http/Controllers/contractController.js:135:5) {
  code: 3,
  details: 'Document name "projects/metalabs-e46e4/databases/(default)/documents/database/metalab/users/../../test_data" contains a resource id ".." at index 77.',
  metadata: Metadata { internalRepr: Map(0) {}, options: {} }
}

Node.js v18.15.0

~/credshields/SC-Generation-main 9s
❯
```

SOURCE Code

Private GitHub Repository

Sonny-CX/SC-Generation/

REPORT Appendix

FINDING CATEGORIES

The assessment process will utilize a mixture of static analysis, dynamic analysis, in-depth manual review and/or other security techniques.

This report has been prepared for ProxyMeta Labs project using the above techniques to examine and discover vulnerabilities and safe coding practices in ProxyMeta Labs's smart contract including the libraries used by the contract that are not officially recognized.

A comprehensive static and dynamic analysis has been performed on the solidity code in order to find vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds.

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The testing methods find and flag issues related to gas optimizations that help in reducing the overall gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

AUDIT SCORES

D3ploy Audit Score is not a live dynamic score. It is a fixed value determined at the time of the report issuance date.

D3ploy Audit Score is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports and scores are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts d3ploy to perform a security review.

d3ploy

WEBSITE d3ploy.co

@d3ploy_ TWITTER