

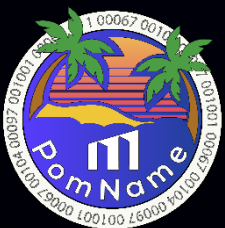


Request your audit at coinsult.net

Advanced Manual

Smart Contract Audit

March 1, 2023



Audit requested by

PomName

0xBc8fB5e2BD2D2eEdf51F42D893efB9739cF9C54D

Table of Contents

Table of Contents	2
Audit Summary	3
Audit Scope	4
Source Code	4
Audit Method	4
Automated Vulnerability Check	4
Manual Code Review	4
Used Tools	4
Static Analysis	5
Audit Results	7
Risk Classification	7
Manual Code Review	7
Foundry	8
Centralization Privileges	11
Notes	12
Constructor Snapshot	13
Disclaimer	14

Audit Summary

Project Name	PomName
Website	http://pomname.org
Blockchain	Proof of Memes
Smart Contract Language	Solidity
Contract Address	0xBc8fB5e2BD2D2eEdf51F42D893efB9739cF9C54D
Audit Method	Static Analysis, Manual Review
Start date of audit	March 1, 2023

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Audit Scope

Source Code

Coinsult was commissioned by PomName to perform an audit based on the following code:

<https://memescan.io/address/0xBc8fB5e2BD2D2eEdf51F42D893efB9739cF9C54D/contracts#address-tabs>

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used Tools

- ✓ Slither: Solidity static analysis framework
- ✓ Remix: IDE Developer Tool
- ✓ CWE: Common Weakness Enumeration
- ✓ SWC: Smart Contract Weakness Classification and Test Cases
- ✓ DEX: Testnet Blockchains

Static Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.





ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	Passed

SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed
SWC-134	Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

Audit Results





Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
 Informational	Does not compromise the functionality of the contract in any way
 Low-Risk	Won't cause any problems, but can be adjusted for improvement
 Medium-Risk	Will likely cause problems and it is recommended to adjust
 High-Risk	Will definitely cause problems, this needs to be adjusted

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
 Informational	0	0	0	0
 Low-Risk	0	0	0	0
 Medium-Risk	0	0	0	0
 High-Risk	2	0	0	2

Error Code	Description	Severity
FND-001	Foundry Test	● Passed

Foundry

Foundry is a smart contract development toolchain.

Foundry manages your dependencies, compiles your project, runs tests, deploys, and lets you interact with the chain from the command-line and via Solidity scripts.

```
Running 12 tests for test/Test.t.sol:nameTest
[PASS] testFail can not buy if not payed enough() (gas: 187600)
[PASS] testFail can not register if is not expired() (gas: 149794)
[PASS] testFail can not set primary if is not holder() (gas: 120170)
[PASS] testFail can not transfer if is not holder() (gas: 102031)
[PASS] testFail primary name will be reset after transfer() (gas: 279960)
[PASS] test can extend name() (gas: 131775)
[PASS] test can increaseFees() (gas: 690458)
[PASS] test can list name for sale if is holder() (gas: 155579)
[PASS] test can list name for sale if is holder and receives ether if sold() (gas: 199059)
[PASS] test can register a name() (gas: 136465)
[PASS] test can set primary name() (gas: 180170)
[PASS] test can tranfser if is holder() (gas: 142833)
Test result: ok. 12 passed; 0 failed; finished in 4.36ms
```


Error Code	Description	Severity
HIGH-001	Everyone is able to extend a name's duration	● Resolved

Everyone is able to extend a name's duration

The fact that someone else is able to extend the duration of another person's name without their consent may introduce the potential for abuse, as someone may maliciously extend the duration of a name to prevent others from using it, or to cause harm to the actual name holder.

```
function extendNameExpiry(uint256 nameToExtend) payable public {
    uint256 extendFee;

    checkNameFormat(nameToExtend);
    // Check if name is expired
    require(isNameExpired(nameToExtend)==false, "Can't extend an expired name.");

    (, uint256 extendLetters) = countLetters(nameToExtend);
    // You don't have to be the holder to extend a name
    if (extendLetters<=20 && extendLetters > 0) {
        extendFee = extensionFees[extendLetters];
        require(msg.value == extendFee, "Incorrect amount of POM sent to extend name
expiry.");
        nameExpiry[nameToExtend] += 126144000; // 4 years
    }
}
```

Recommendation

it is suggested to ensure that appropriate access controls and security measures are in place to prevent unauthorized or malicious use of this function.

Resolved

PomName team added a line where they check the creator of the name.

Error Code	Description	Severity
HIGH-002	Low level call	● Resolved

Low level calls

buyName function is using low level call to send ether to the name seller, through checking the contract.

```
function buyName(uint256 nameToBuy) payable public {

    checkNameFormat(nameToBuy);
    // Check to ensure the name isn't expired
    require(isNameExpired(nameToBuy)==false, "That name is expired. Register it
instead.");
    require(isNameForSale[nameToBuy] == true, "That name is not for sale.");
    require(msg.value == salePrice[nameToBuy], "POM sent must be equal to the sale
price.");

    address payable pomReceiver = payable(nameHolder[nameToBuy]);
    uint256 transactionPrice = salePrice[nameToBuy];

    // Toggle to not for sale and reset sale price.
    isNameForSale[nameToBuy] = false;
    salePrice[nameToBuy] = 0;

    // Reset the primary name of the seller if their primary name is being sold.
    if (primaryName[pomReceiver] == nameToBuy) {
        primaryName[pomReceiver] = 0;
    }

    // Map the new wallet to the name
    nameHolder[nameToBuy] = msg.sender;

    (bool sent,) = pomReceiver.call{value: transactionPrice}("");
    require(sent, "Failed to send POM.");
}
```

Recommendation

Use a reentrancy guard to prevent potential reentrancy attacks.

Resolved

PomName team added a reentrancy guard variable.

Centralization Privileges

The owner of the contract is allowed to interact with the following functions:

```
setFees  
changeOwner  
activateNameRegistration  
activateMigration  
adminRegister
```

Notes

- redundant use of checkNameFormat at some view functions, like getPrice and holderOf, the idea is that if this names are listed, they passed the format checking hence we don't need to check this names again.

● Resolved

- classifying public functions that have not been used internally as "external" in order to potentially save on gas expenses. ● Resolved

Security – buyName function is using low level call to send ether to the name seller, through checking the contract.

Suggestion:

use a reentrancy guard to prevent potential reentrancy attacks. ● Resolved

Constructor Snapshot

```
contract dotMeme {

    address public owner = 0x3C6B0615d70Fc018d91d76d93a1e5e311C9fc52b;

    mapping(uint256 => uint256) public nameExpiry;
    mapping(uint256 => bool) public migrationComplete;
    mapping(uint256 => address) internal nameHolder;
    mapping(address => uint256) internal primaryName;
    mapping(uint256 => uint256) internal registryFees;
    mapping(uint256 => uint256) internal extensionFees;
    mapping(uint256 => bool) internal isNameForSale;
    mapping(uint256 => uint256) internal salePrice;

    PomName internal pomNameV1;

    bool public registrationActive;
    bool public migrationActive;
    bool private enterLock = false;
    uint256 internal migrateTempName;

    constructor() {
        for(uint k=1; k<=20; k++){
            registryFees[k] = 10000000000000000000000000000; // Set initial mint fee for 1-20
characters
            extensionFees[k] = 10000000000000000000000000000; // Set initial renewal fee for 1-
20 characters
        }
        registrationActive = false;
        migrationActive = true;
        pomNameV1 = PomName(0x6FE0234B5Cae49B27D960f358F0825CA1D6CeC26);
    }
}
```

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.