SMART CONTRACT SECURITY REPORT.

Personally audited and created for:
HeadDAO.

CRUCIAL

29th of October 2021

TABLE OF CONTENTS

Introduction

This report has been prepared for HeadDAO. An extensive analysis has been performed by manual review, Static Analysis and symbolic execution.

In our Audits, we focus on the following areas:

- Analyzing that the smart contracts' logic meet the intentions of the client.
- Examining the smart contracts against conventional and unconventional attack vectors
- Verifying the codebase meets the current Industry standard and best practices
- Cross-referencing the Project against the implementation, contract, and structure of similar Industry-leading Projects
- Elaborate line by line manual review of the entire Codebase.

The findings of the security evaluation resulted ranged from high to informational. We advise you to address these findings as soon as possible to assure a foremost level of security for your project and community.

- Increase good coding practices for a better structure in the source code
- Increase the number of unit test to cover all possible angles of use cases
- Add more comments per function for readability.

Overview

Project Summary

Project name	HeadDAO
Platform	Ethereum
Language	Solidity
Codebase	https://etherscan.io/token/0xf62c6a8e7bcdc96cda11bd765b40afa9ffc19ab9
Commit	

Audit Summary

QRUCIAL's team contacted HeadDAO to audit the Logic, Design as well as the implementation of the HeadDAO NFT smart contract. QRUCIAL security experts audited the HeadDAO smart contracts against possible vulnerabilities and logic bugs that can lead to financial or reputational loss.

The goal of this audit was to uncover potential security vulnerabilities to inspect its general architecture and design of the solidity implementation of the business model, as well as finding bugs which could imperil the software in production.

The findings of the initial Audit were sent to the Project's team and the source code is expected to be re-assessed before a second round of auditing is being carried out.

Vulnerability Summary

Level	Total	Pending	Rejected	Accepted	Partially fixed	Fixed
Critical	0	-	-	-	-	-
High	1	-	-	-	-	-
Medium	2	-	-	-	-	-
Low	1	-	-	-	-	-
Informational	2	-	-	-	-	-

Scope

Audited Code:	The HeadDAO smart contract
Blockchain Explorer Link:	https://etherscan.io/token/0xf62c6a8e7bcdc96cda11bd765b40afa9ffc19ab9
Compiler:	^0.8.0
Number of files:	19
Scope (list of files)	Address.sol ContentMixin.sol Context.sol EIP712Base.sol ERC165.sol ERC721Enumerable.sol ERC721.sol ERC721Tradable.sol IERC165.sol IERC721Enumerable.sol IERC721Metadata.sol IERC721Receiver.sol IERC721.sol Initializable.sol MetaCoin.sol NativeMetaTransaction.sol Ownable.sol SafeMath.sol Strings.sol

Risk Classifications

Critical:
Vulnerabilities that can lead to a loss of funds, impairment, or control over the system or its function.
We recommend that findings of this classification are fixed immediately.

High:
Findings of this classification can impact the flow of logic and can cause direct disruption in the system and the project's organization.
We recommend that issues of this classification are fixed as soon as possible.

Medium:
Vulnerabilities of this class have impact on the flow of logic, but does not cause any disturbance that would halt the system or organizational continuity.
We recommend that findings of this class are fixed nonetheless.

Low:
Bugs, or vulnerability that have minimal impact and do not pose a significant threat to the project or its users.
We recommend that issues of this class are fixed nonetheless because they increase the attack surface when your project is targeted by malicious actors.

Informational:
Findings of this class have a negligible risk factor but refer to best practices in syntax, style or general security.

Findings

To be disclosed.

HIGH: [redacted]

Description:

[redacted]

Impact:

[redacted]

Recommendations:

[redacted]

References:

[redacted]

Technical Details:

To be disclosed.

[redacted]

MEDIUM: Visibility of functions should be stricter

Description:
If a function does not require to be called internally, it is possible to save gas costs and improve security by changing their visibility from public to external.
Note also that the previous high vulnerability might be easier to exploit if the visibility of these functions are public and are callable internally.

Impact:
Function calls will cost less gas (both deploy and call times) and security will be slightly improved.

Recommendations:
Replace “public“ to “external“ in all functions listed above.

References:
https://ezcook.de/2018/01/29/Gas-Used-by-Public-and-External-Function-in-Solidity/
https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Technical Details:

```
name() should be declared external:
    - ERC721.name() (ERC721.sol#78-80)
symbol() should be declared external:
    - ERC721.symbol() (ERC721.sol#85-87)
tokenURI(uint256) should be declared external:
    - ERC721.tokenURI(uint256) (ERC721.sol#92-97)
approve(address,uint256) should be declared external:
    - ERC721.approve(address,uint256) (ERC721.sol#111-121)
setApprovalForAll(address,bool) should be declared external:
    - ERC721.setApprovalForAll(address,bool) (ERC721.sol#135-140)
transferFrom(address,address,uint256) should be declared external:
    - ERC721.transferFrom(address,address,uint256) (ERC721.sol#152-161)
safeTransferFrom(address,address,uint256) should be declared external:
    - ERC721.safeTransferFrom(address,address,uint256) (ERC721.sol#166-172)
tokenOfOwnerByIndex(address,uint256) should be declared external:
    - ERC721Enumerable.tokenOfOwnerByIndex(address,uint256)
(ERC721Enumerable.sol#36-39)
tokenByIndex(uint256) should be declared external:
    - ERC721Enumerable.tokenByIndex(uint256) (ERC721Enumerable.sol#51-54)
contractURI() should be declared external:
    - HeadDao.contractURI() (MetaCoin.sol#51-53)
setBaseURI(string) should be declared external:
    - HeadDao.setBaseURI(string) (MetaCoin.sol#59-61)
setSaleStatus(bool) should be declared external:
    - HeadDao.setSaleStatus(bool) (MetaCoin.sol#67-69)
changePrice(uint256) should be declared external:
    - HeadDao.changePrice(uint256) (MetaCoin.sol#71-73)
changeBatchSize(uint256) should be declared external:
    - HeadDao.changeBatchSize(uint256) (MetaCoin.sol#76-78)
mintHead(uint256) should be declared external:
    - HeadDao.mintHead(uint256) (MetaCoin.sol#83-98)
devMint(uint256) should be declared external:
    - HeadDao.devMint(uint256) (MetaCoin.sol#100-112)
withdraw() should be declared external:
    - HeadDao.withdraw() (MetaCoin.sol#114-134)
withdraw_emerg() should be declared external:
    - HeadDao.withdraw_emerg() (MetaCoin.sol#136-142)
get_all() should be declared external:
    - HeadDao.get_all() (MetaCoin.sol#147-149)
executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) should be declared
external:
    -
NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8
) (NativeMetaTransaction.sol#33-67)
getNonce(address) should be declared external:
    - NativeMetaTransaction.getNonce(address) (NativeMetaTransaction.sol#85-87)
renounceOwnership() should be declared external:
    - Ownable.renounceOwnership() (Ownable.sol#53-55)
transferOwnership(address) should be declared external:
    - Ownable.transferOwnership(address) (Ownable.sol#61-64)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-
function-that-could-be-declared-external
```

MEDIUM: Unused code in the contract

Description:
The more functions are available, the larger the attack surface is. Also, the code is more complex by that, hence the chance for bugs, errors are increased.

Impact:
Increased complexity and attack surface.

Recommendations:
Remove the unused code by upgrading the contract. We recommend making a contract upgrade that fixes all vulnerabilities together.

References:
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Technical Details:

```
Address.functionCall(address,bytes) (Address.sol#79-81)
Address.functionCall(address,bytes,string) (Address.sol#89-95)
Address.functionCallWithValue(address,bytes,uint256) (Address.sol#108-114)
Address.functionCallWithValue(address,bytes,uint256,string)
(Address.sol#122-133)
Address.functionDelegateCall(address,bytes) (Address.sol#168-170)
Address.functionDelegateCall(address,bytes,string) (Address.sol#178-187)
Address.functionStaticCall(address,bytes) (Address.sol#141-143)
Address.functionStaticCall(address,bytes,string) (Address.sol#151-160)
Address.sendValue(address,uint256) (Address.sol#54-59)
Address.verifyCallResult(bool,bytes,string) (Address.sol#195-215)
Context._msgData() (Context.sol#20-22)
Context._msgSender() (Context.sol#16-18)
ERC721._burn(uint256) (ERC721.sol#304-316)
ERC721._safeMint(address,uint256) (ERC721.sol#250-252)
ERC721._safeMint(address,uint256,bytes) (ERC721.sol#258-268)
SafeMath.div(uint256,uint256,string) (SafeMath.sol#190-199)
SafeMath.mod(uint256,uint256) (SafeMath.sol#150-152)
SafeMath.mod(uint256,uint256,string) (SafeMath.sol#216-225)
SafeMath.sub(uint256,uint256) (SafeMath.sol#106-108)
SafeMath.sub(uint256,uint256,string) (SafeMath.sol#167-176)
SafeMath.tryAdd(uint256,uint256) (SafeMath.sol#21-27)
SafeMath.tryDiv(uint256,uint256) (SafeMath.sol#63-68)
SafeMath.tryMod(uint256,uint256) (SafeMath.sol#75-80)
SafeMath.tryMul(uint256,uint256) (SafeMath.sol#46-56)
SafeMath.trySub(uint256,uint256) (SafeMath.sol#34-39)
Strings.toHexString(uint256) (Strings.sol#39-50)
Strings.toHexString(uint256,uint256) (Strings.sol#55-65)
```

LOW: totalHead and totalCount are not constant

Description:
Improve security by making variables constant. Also, the project says the total count of Heads are limited, so a constant would be more accurate business-wise.

Impact:
Security is improved by making possible vulnerabilities harder to exploit.

Recommendations:
Make totalCount and totalHead constant. We recommend making a contract upgrade that fixes all vulnerabilities together.

References:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

Technical Details:

```
HeadDao.totalCount (MetaCoin.sol#26) should be constant
HeadDao.totalHead (MetaCoin.sol#25) should be constant
```

INFORMATIONAL: Use a standard naming logic

Description:
Making the code more readable is important for both developers and auditors. Using a fixed logic for naming, e.g., “mixed case“ helps to make better code.

Impact:
Code is less readable and looks less professional.

Recommendations:
Define a naming logic standard.

References:
https://docs.soliditylang.org/en/v0.8.9/style-guide.html#naming-conventions

Technical Details:

```
Parameter ERC721.safeTransferFrom(address,address,uint256,bytes)._data
(ERC721.sol#181)
Parameter HeadDao.setContractURI(string)._cURI (MetaCoin.sol#55)
Parameter HeadDao.setBaseURI(string)._newURI (MetaCoin.sol#59)
Parameter HeadDao.setSaleStatus(bool)._start (MetaCoin.sol#67)
Parameter HeadDao.changePrice(uint256)._newPrice (MetaCoin.sol#71)
Parameter HeadDao.changeBatchSize(uint256)._newBatch (MetaCoin.sol#76)
Parameter HeadDao.mintHead(uint256)._count (MetaCoin.sol#83)
Parameter HeadDao.devMint(uint256)._count (MetaCoin.sol#100)
Function HeadDao.withdraw_emerg() (MetaCoin.sol#136-142)
Function HeadDao.get_all() (MetaCoin.sol#147-149)
Variable HeadDao.sale_active (MetaCoin.sol#34)
Variable HeadDao.community_wallet (MetaCoin.sol#37)
Variable HeadDao.RZ_wallet (MetaCoin.sol#38)
Variable HeadDao.AR_wallet (MetaCoin.sol#39)
Variable HeadDao.shamdoo_wallet (MetaCoin.sol#40)
```

INFORMATIONAL: Use of low-level calls and assembly are error-prone

Description:
Using low-level calls and assembly in smart contracts highly increase project complexity and possibilities for error, hence they are better to be avoided if not required.

Impact:
Even if not directly a vulnerability, low-level calls might open up the surface for high-impact attacks.

Recommendations:
Avoid low-level calls whenever possible.

References:
https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
https://docs.soliditylang.org/en/v0.8.9/control-structures.html#error-handling-assert-require-revert-and-exceptions

Technical Details:

```
Low level call in Address.sendValue(address,uint256) (Address.sol#54-59):
    - (success) = recipient.call{value: amount}() (Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string)
(Address.sol#122-133):
    - (success,returndata) = target.call{value: value}(data) (Address.sol#131)
Low level call in Address.functionStaticCall(address,bytes,string)
(Address.sol#151-160):
    - (success,returndata) = target.staticcall(data) (Address.sol#158)
Low level call in Address.functionDelegateCall(address,bytes,string)
(Address.sol#178-187):
    - (success,returndata) = target.delegatecall(data) (Address.sol#185)
Low level call in
NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8
) (NativeMetaTransaction.sol#33-67):
    - (success,returnData) =
address(this).call(abi.encodePacked(functionSignature,userAddress))
(NativeMetaTransaction.sol#61-63)

Address.isContract(address) (Address.sol#26-36) uses assembly
    - INLINE ASM (Address.sol#32-34)
Address.verifyCallResult(bool,bytes,string) (Address.sol#195-215) uses assembly
    - INLINE ASM (Address.sol#207-210)
ContextMixin.msgSender() (ContentMixin.sol#6-25) uses assembly
    - INLINE ASM (ContentMixin.sol#14-20)
EIP712Base.getChainId() (EIP712Base.sol#52-58) uses assembly
    - INLINE ASM (EIP712Base.sol#54-56)
ERC721._checkOnERC721Received(address,address,uint256,bytes)
(ERC721.sol#369-390) uses assembly
    - INLINE ASM (ERC721.sol#382-384)
```

DISCLAIMER

THANK YOU.

Our team gave their full commitment to craft this audit with precision and focus on details with the goal to support you improving your work.

With this audit we want to provide you a guidance to make your project more
secure and for presenting your
community a product they can trust in.

We make security our priority, so you don‘t have to.

CRUCIAL

----Polkadot{.js} account validation address----

5EHagRYLNsCJUx5bA5Y8MZWLqPVzqQbFMC76V68Wzv7GHHXD

----Polkadot{.js} account validation address----

QRUCIAL OÜ c/o Gate to Baltics
Narva mnt 7-652, 10117 | Tallinn, Estonia