CreoEngine

Smart Contract Audit Report

SYSFIXED

11 Mar 2022

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
CreoEngine	CREO	Binance Smart Chain

Addresses

Contract address	0x9521728bf66a867bc65a93ece4a543d817871eb7
Contract deployer address	0x939d640675e2f0Be82B7b85815F898f23C6A8fae

Project Website

https://creoengine.com/

Codebase

https://bscscan.com/address/0x9521728bf66a867bc65a93ece4a543d817871eb7#code

SUMMARY

Creo Engine is a gaming platform where developers can deploy their games to the Creo Engine user base. Creo Engine supports developers to create and earn through game development while providing interesting earning possibilities for players through high-quality mobile games. Creo Engine also established an in-house game studio and assembled a group of experts and artists to develop games. The first four games are Evermore Knights, Slime Haven, Merchant Marvels, and Peony Ranch.

Contract Summary

Documentation Quality

CreoEngine provides a very poor documentation with standard of solidity base code.

- The technical description is provided unclear and disorganized.

Code Quality

The Overall quality of the basecode is poor.

- Solidity basecode and rules are unclear and disorganized by CreoEngine.

Test Coverage

Test coverage of the project is 100% (Through Codebase)

Audit Findings Summary

- SWC-101 | It is recommended to use vetted safe math libraries for arithmetic operations consistently on lines 491.
- SWC-107 | It is recommended to use a reentrancy lock, reentrancy weaknesses detected on lines 481.
- SWC-110 SWC-123 | It is recommended to use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM on lines 481.
- SWC-113 SWC-128 | It is recommended to implement the contract logic to handle failed calls and block gas limit on lines 481.
- SWC-120 | It is recommended to use external sources of randomness via oracles on lines 398 and 431.

CONCLUSION

We have audited the CreoEngine project released on March 2022 to find issues and identify potential security vulnerabilities in the CreoEngine project. This process is used to find technical issues and security loopholes that may be found in smart contracts.

The security audit report yielded unsatisfactory results, discovering high-risk and low-risk issues.

Writing a contract that does not follow the Solidity style guide can pose a significant risk. The serious and low problems we found in the smart contract are The arithmetic operation can overflow. It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow. Low-risk issues are a call to a user-supplied address being executed, multiple calls being executed in the same transaction, the potential use of "block.number" as a source of randomness, and requirement violation. An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Re-entering the contract in an intermediate state may lead to unexpected behavior. Ensure no state modifications are executed after this call, and reentrancy guards are in place. This call is executed following another call within the same transaction. The call may never get executed if an initial call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or ensure that all callees can be trusted (i.e. they’re part of your codebase). A requirement was violated in a nested call, and the call was reverted. Ensure valid inputs are provided to the nested call (for instance, via passed arguments).

We were recommended to keep being aware of investing in this risky smart contract.

AUDIT RESULT

Article	Category	Description	Result
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	PASS
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	ISSUE FOUND
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	PASS
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	PASS
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	PASS
Unprotected Ether Withdrawal	SWC-105	Due to missing or insufficient access controls, malicious parties can withdraw from the contract.	PASS
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	PASS
Reentrancy	SWC-107	Check effect interaction pattern should be followed if the code performs recursive call.	ISSUE FOUND
Uninitialized Storage Pointer	SWC-109	Uninitialized local storage variables can point to unexpected storage locations in the contract.	PASS
Assert Violation	SWC-110 SWC-123	Properly functioning code should never reach a failing assert statement.	ISSUE FOUND
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	PASS
Delegate call to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	PASS

DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	ISSUE FOUND
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	PASS
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	PASS
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	PASS
Signature Unique ID	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	PASS
Incorrect Constructor Name	SWC-118	Constructors are special functions that are called only once during the contract creation.	PASS
Shadowing State Variable	SWC-119	State variables should not be shadowed.	PASS
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	ISSUE FOUND
Write to Arbitrary Storage Location	SWC-124	The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations.	PASS
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/.	PASS
Insufficient Gas Griefing	SWC-126	Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract.	PASS
Arbitrary Jump Function	SWC-127	As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value.	PASS

Typographical Error	SWC-129	A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable.	PASS
Override control character	SWC-130	Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract.	PASS
Unused variables	SWC-131 SWC-135	Unused variables are allowed in Solidity and they do not pose a direct security issue.	PASS
Unexpected Ether balance	SWC-132	Contracts can behave erroneously when they strictly assume a specific Ether balance.	PASS
Hash Collisions Variable	SWC-133	Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision.	PASS
Hardcoded gas amount	SWC-134	The transfer() and send() functions forward a fixed amount of 2300 gas.	PASS
Unencrypted Private Data	SWC-136	It is a common misconception that private type variables cannot be read.	PASS

SMART CONTRACT ANALYSIS

Started	Thursday Mar 10 2022 02:53:34 GMT+0000 (Coordinated Universal Time)
Finished	Friday Mar 11 2022 09:29:06 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	CreoEngine.sol

Detected Issues

ID	Title	Severity	Status
SWC-101	THE ARITHMETIC OPERATION CAN OVERFLOW.	high	acknowledged
SWC-107	A CALL TO A USER-SUPPLIED ADDRESS IS EXECUTED.	low	acknowledged
SWC-113	MULTIPLE CALLS ARE EXECUTED IN THE SAME TRANSACTION.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-123	REQUIREMENT VIOLATION.	low	acknowledged

SWC-101 | THE ARITHMETIC OPERATION CAN OVERFLOW.

LINE 491

high SEVERITY

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source File

- CreoEngine.sol

Locations

```
490
491   }
492
```

SWC-107 | A CALL TO A USER-SUPPLIED ADDRESS IS EXECUTED.

LINE 481

low SEVERITY

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source File

- CreoEngine.sol

Locations

```
480   function manageMaxWalletExempt(address account, bool excluded) external onlyOwner {
481   require(_isMaxWalletExempt[account] != excluded, "Account is already the value of
'excluded'");
482   _isMaxWalletExempt[account] = excluded;
483   emit maxWalletExemptChanged(account, excluded);
484
485
```

SWC-113 | MULTIPLE CALLS ARE EXECUTED IN THE SAME TRANSACTION.

LINE 481

low SEVERITY

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source File

- CreoEngine.sol

Locations

```
480   function manageMaxWalletExempt(address account, bool excluded) external onlyOwner {
481   require(_isMaxWalletExempt[account] != excluded, "Account is already the value of
'excluded'");
482   _isMaxWalletExempt[account] = excluded;
483   emit maxWalletExemptChanged(account, excluded);
484
485
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 398

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- CreoEngine.sol

Locations

```
397   function _transfer(address sender, address recipient, uint256 amount) internal {
398   require(sender != address(0), "BEP20: transfer from the zero address");
399   require(recipient != address(0), "BEP20: transfer to the zero address");
400   require(!_isSniper[recipient], "You have no power here!");
401   require(!_isSniper[sender], "You have no power here!");
402
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 431

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- CreoEngine.sol

Locations

```
430   _approve(_msgSender(), spender, amount);
431   return true;
432   }
433
434   function transferFrom(address sender, address recipient, uint256 amount) external
returns (bool) {
435
```

SWC-123 | REQUIREMENT VIOLATION.

LINE 481

low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source File

- CreoEngine.sol

Locations

```
480   function manageMaxWalletExempt(address account, bool excluded) external onlyOwner {
481   require(_isMaxWalletExempt[account] != excluded, "Account is already the value of
'excluded'");
482   _isMaxWalletExempt[account] = excluded;
483   emit maxWalletExemptChanged(account, excluded);
484
485
```

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to, or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Sysfixed's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 Sysfixed to perform a security assessment. This report 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.

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.