Casino Nite Coin Pusher

Smart Contract Security Audit

Prepared by Nexarias

July 29th, 2024 - July 31st, 2024

Nexarias.com

contact@nexarias.com

Document Properties

Client	Nitefeeder
Version	0.1
Classification	Private

Scope

The Casino Nite Coin Pusher Contract in the Casino Nite Coin Pusher Repository

Link	Address
https://etherscan.io/address/0x33d1F380BdF07AEe6D1Fc7cB3e01E543bcaE47f1#code	402dd47685585894ced9184ad080a9c2

Contacts

COMPANY	CONTACT
Nexarias	contact@nexarias.com

Contents

1 Introduction

Nitefeeder engaged Nexarias to conduct a security assessment on the Casino Nite Coin Pusher beginning on July 29th, 2024 and ending July 31st, 2024. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Casino Nite Coin Pusher

Issuer	Nitefeeder
Website	Nitefeeder.life
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

Nexarias used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by Nexarias are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

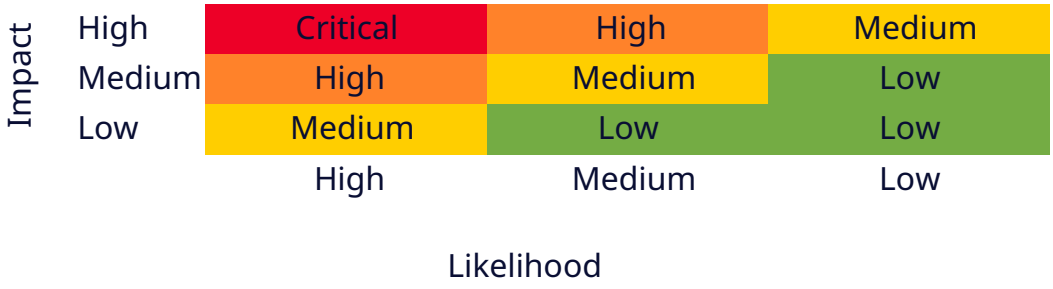
Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.

- Impact quantifies the technical and economic costs of a successful attack.

- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Casino Nite Coin Pusher implementation.During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool.Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 4 high-severity, 5 medium-severity, 2 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Balance Management in Withdraw Function	HIGH	Fixed
Centralization in emergency Withdraw functions	HIGH	Fixed
Lack of Balance Traceability	HIGH	Fixed
Incorrect ETH Value Check in withdraw	HIGH	Fixed
Potential for Token Address Misconfiguration	MEDIUM	Fixed
Incorrect Order of Division and Multiplication	MEDIUM	Fixed
Use of transfer Instead of Safe Transfer Method	MEDIUM	Fixed
Improper Use of int256 for Gas Fee	MEDIUM	Fixed
Lack of Input Validation in setGasFee	MEDIUM	Fixed
Missing Events for State Changes	LOW	Fixed
Unnecessary Declaration of smart Contract Address	LOW	Fixed

3 Finding Details

A P2EGame.sol

A.1 Balance Management in Withdraw Function [HIGH]

Description:

The withdraw function sets playerToken[msg.sender] to 0 after a withdrawal, which is inappropriate as it does not account for partial withdrawals. This could lead to issues where a player loses the remaining balance if they do not withdraw the full amount in one transaction.

Listing 1: P2EGame.sol

```
147     function withdraw(uint256 amount) public payable {
148         require(
149                         playerToken[msg.sender] >= amount,
150                         "Cannot Withdraw more then your Balance!!"
151                     );
152       require(msg.value == uint256(backPrice) * 1000000000, "Sent value
              ,→ not equals ETH gas price");
154         IERC20(TokenAdr).transfer(msg.sender, amount);
156          playerToken[msg.sender] = 0;
157     }
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

Adjust the function to decrement the player's balance by the withdrawn amount instead of setting it to 0. This ensures that players can make multiple withdrawals until their balance is fully depleted.

Status - Fixed

A.2 Centralization in emergency Withdraw functions [HIGH]

Description:

The emergencyWithdrawETH and emergencyWithdrawToken functions allow the CEO to withdraw all ETH and tokens from the contract, respectively. This introduces a significant centralization risk, as it grants a single address (the CEO) full control over the contract's funds. In the event of a compromised CEO account, all funds in the contract could be drained, leading to potential financial loss for players.

Listing 2: P2EGame.sol

```
129     function emergencyWithdrawETH() public onlyCeo() {
130         (bool os, ) = payable(msg.sender).call{value: address(this).
                ,→ balance}('');
131         require(os);
132     }


135     function emergencyWithdrawToken(address _adr) public onlyCeo() {
136         uint256 bal = IERC20(_adr). balanceOf(address(this));
137         IERC20(_adr).transfer(msg.sender, bal);
138     }
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

Implement multi-signature authorization for these emergency functions. Require multiple trusted parties to approve any emergency withdrawals to mitigate the centralization risk. Additionally, consider setting withdrawal limits and establishing a time delay mechanism to allow for community review and potential veto of large withdrawals.

Status - Fixed

A.3 Lack of Balance Traceability [HIGH]

Description:

The depositETH and depositToken functions do not track the deposited balances of users, leading to a lack of transparency and potential issues in verifying individual deposits.

Listing 3: P2EGame.sol

```
121     function depositETH() public payable {}



124     function depositToken(uint256 amount) public {
125       IERC20(TokenAdr).transferFrom(msg.sender, address(this), amount);
126     }
```

Risk Level:

Likelihood – 4
Impact - 3

Recommendation:

Implement a mapping to track the ETH and token balances for each user. Emit events during deposit and withdrawal actions to enhance transparency and allow users to query their balances.

Status - Fixed

A.4 Incorrect ETH Value Check in withdraw [HIGH]

Description:

In the withdraw function, the check for the sent value against the calculated backPrice is incorrect. The calculation of backPrice depends on the gasFee and the price, which may not be updated correctly before the withdrawal.

Listing 4: P2EGame.sol

```
4 require(msg.value == uint256(backPrice) * 1000000000, "Sent value not
      ,→ equals ETH gas price");
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

Ensure that backPrice is calculated correctly and updated prior to this check. Additionally, consider whether the ETH value check is necessary for the withdrawal process.

Status - Fixed

A.5 Potential for Token Address Misconfiguration [MEDIUM]

Description:

The setTokenAdr function allows the CEO to set a new token address without any checks. If the new address is invalid or points to a malicious contract, it could lead to loss of funds.

Listing 5: P2EGame.sol

```
144 function setTokenAdr(address newAddress) public onlyCeo() {
145     TokenAdr = newAddress;
146 }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

Implement checks to ensure the new token address is valid and adheres to the ERC20 interface.

Status - Fixed

The team removed setTokenAdr function

A.6 Incorrect Order of Division and Multiplication [MEDIUM]

Description:

The setETH function performs division before multiplication, which can lead to precision loss or incorrect calculations. Specifically, the calculation int256 ethPrice = (gasFee / price); followed by backPrice = (ethPrice * 100000);may yield incorrect results due to integer division.

Listing 6: P2EGame.sol

```
103     function setETH(address _adr, uint256 amount) public onlyCeo() {
104   (
105         ,// uint80 roundID,
106           int256 price,
107       , // uint startedAt,
108     , // uint timeStamp,
109     //uintt updatedAt
110         // uint80 answeredInRound
111      ) = priceFeed.latestRoundData();

113     int256 ethPrice = (gasFee /price);
114       backPrice = (ethPrice * 100000);
```

```
116         playerToken[_adr] = amount;

118       }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Reorder the operations to multiply before dividing to maintain precision.

Status - Fixed

The team removed setETH function

A.7 Use of transfer Instead of Safe Transfer Method [MEDIUM]

Description:

The withdraw function uses the transfer method of the ERC20 token interface to send tokens to the user. This method can fail silently if the token contract does not implement the expected behavior, leading to potential loss of funds. It is recommended to use a safe transfer method that checks for success.

Listing 7: P2EGame.sol

```
158 IERC20(TokenAdr).transfer(msg.sender, amount);
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Use a safe transfer method, such as safeTransfer, from the OpenZeppelin library, which ensures that the transfer was successful and reverts the transaction if it fails. This will enhance the reliability of token transfers.

Status - Fixed

A.8 Improper Use of int256 for Gas Fee [MEDIUM]

Description:

The gasFee variable is defined as an int256, which can lead to unexpected behavior if a negative value is assigned. Gas fees should always be non-negative. Using an unsigned integer type like uint256 would be more appropriate.

Listing 8: P2EGame.sol

```
76 int256 public gasFee;
```

Risk Level:

Likelihood – 3
Impact - 2

Recommendation:

Change the type of gasFee to uint256 to prevent negative values from being assigned.

Status - Fixed

The team removed gasFee variable

A.9 Lack of Input Validation in setGasFee [MEDIUM]

Description:

The setGasFee function does not validate the input value. This could allow the CEO to set an excessively high gas fee, leading to unintended consequences during the withdrawal process.

Listing 9: P2EGame.sol

```
2 function setGasFee(int256 amount) public onlyCeo() {
3         gasFee = amount;
4 }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Implement a validation check to ensure that the amount is a reasonable value before setting gasFee.

Status - Fixed

The team removed setGasFee function

A.10 Missing Events for State Changes [LOW]

Description:

The contract lacks events for critical state changes such as setting the gas fee, changing the CEO, and withdrawing funds. This can hinder transparency and tracking of contract activities.

Listing 10: P2EGame.sol

```
100 function setGasFee(int256 amount) public onlyCeo() {
101         gasFee = amount;
102 }
```

Listing 11: P2EGame.sol

```
92   function setPriceFeed(address newAggregator) public onlyCeo() {



95                       priceFeed = AggregatorV3Interface(newAggregator);
96   }
```

Listing 12: P2EGame.sol

```
171         function changeCEO(address _adr) public onlyCeo() {

173             ceoAddress = _adr;
174         }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Emit events for all state-changing functions to improve transparency and facilitate monitoring.

Status - Fixed

A.11 Unnecessary Declaration of smart Contract Address [LOW]

Description:

The contract contains an unnecessary declaration of the smartContractAddress variable in the constructor.The address of the contract can be directly accessed using address(this) whenever needed, making this declaration redundant.

Listing 13: P2EGame.sol

```
84  constructor(){
85      ceoAddress = msg.sender;
86      smartContractAddress = address(this);
87      priceFeed = AggregatorV3Interface(0
            ,→ x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419);
88      gasFee = 20000000000000;
89  }
```

Risk Level:

Likelihood – 1
Impact - 1

Recommendation:

Remove the smartContractAddress declaration and use address(this) directly whenever needed to access the contract address. This will reduce the contract's deployment size and gas costs.

Status - Fixed

4 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies.Slither, a Solidity static analysis framework,was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats.This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
P2EGameContract.depositToken(uint256) (P2EGameContract.sol#124-126)
    ,→ ignores return value by IERC20(TokenAdr).transferFrom(msg.sender,
    ,→ address(this),amount) (P2EGameContract.sol#125)
P2EGameContract.emergencyWithdrawToken(address) (P2EGameContract.sol
    ,→ #135-138) ignores return value by IERC20(_adr).transfer(msg.
    ,→ sender,bal) (P2EGameContract.sol#137)
P2EGameContract.withdraw(uint256) (P2EGameContract.sol#147-164) ignores
    ,→ return value by IERC20(TokenAdr).transfer(msg.sender,amount) (
    ,→ P2EGameContract.sol#158)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #unchecked-transfer
INFO:Detectors:
P2EGameContract.setETH(address,uint256) (P2EGameContract.sol#103-118)
    ,→ performs a multiplication on the result of a division:
        - ethPrice = (gasFee / price) (P2EGameContract.sol#113)
        - backPrice = (ethPrice * 100000) (P2EGameContract.sol#114)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #divide-before-multiply
INFO:Detectors:
Reentrancy in P2EGameContract.withdraw(uint256) (P2EGameContract.sol
    ,→ #147-164):
```

```
        External calls:
        - IERC20(TokenAdr).transfer(msg.sender,amount) (P2EGameContract.
            ,→ sol#158)
        State variables written after the call(s):
        - playerToken[msg.sender] = 0 (P2EGameContract.sol#160)
        P2EGameContract.playerToken (P2EGameContract.sol#68) can be used
            ,→ in cross function reentrancies:
        - P2EGameContract.playerToken (P2EGameContract.sol#68)
        - P2EGameContract.setETH(address,uint256) (P2EGameContract.sol
            ,→ #103-118)
        - P2EGameContract.withdraw(uint256) (P2EGameContract.sol#147-164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #reentrancy-vulnerabilities-1
INFO:Detectors:
P2EGameContract.setETH(address,uint256) (P2EGameContract.sol#103-118)
    ,→ ignores return value by (price) = priceFeed.latestRoundData() (
    ,→ P2EGameContract.sol#104-111)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #unused-return
INFO:Detectors:
P2EGameContract.changeCEO(address) (P2EGameContract.sol#167-170) should
    ,→ emit an event for:
        - ceoAddress = _adr (P2EGameContract.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #missing-events-access-control
INFO:Detectors:
P2EGameContract.setTokenAdr(address).newAddress (P2EGameContract.sol
    ,→ #140) lacks a zero-check on :
                - TokenAdr = newAddress (P2EGameContract.sol#142)
P2EGameContract.changeCEO(address)._adr (P2EGameContract.sol#167) lacks
    ,→ a zero-check on :
                - ceoAddress = _adr (P2EGameContract.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #missing-zero-address-validation
```

```
INFO:Detectors:
Pragma version^0.8.0 (P2EGameContract.sol#9) allows old versions
Pragma version^0.8.0 (P2EGameContract.sol#39) allows old versions
solc-0.8.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #incorrect-versions-of-solidity
INFO:Detectors:
Low level call in P2EGameContract.emergencyWithdrawETH() (
    ,→ P2EGameContract.sol#129-132):
        - (os) = address(msg.sender).call{value: address(this).balance}()
            ,→ (P2EGameContract.sol#130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #low-level-calls
INFO:Detectors:
Parameter P2EGameContract.setETH(address,uint256)._adr (P2EGameContract.
    ,→ sol#103) is not in mixedCase
Parameter P2EGameContract.emergencyWithdrawToken(address)._adr (
    ,→ P2EGameContract.sol#135) is not in mixedCase
Parameter P2EGameContract.changeCEO(address)._adr (P2EGameContract.sol
    ,→ #167) is not in mixedCase
Variable P2EGameContract.TokenAdr (P2EGameContract.sol#71) is not in
    ,→ mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #conformance-to-solidity-naming-conventions
INFO:Detectors:
P2EGameContract.constructor() (P2EGameContract.sol#80-86) uses literals
    ,→ with too many digits:
        - gasFee = 20000000000000 (P2EGameContract.sol#84)
P2EGameContract.setETH(address,uint256) (P2EGameContract.sol#103-118)
    ,→ uses literals with too many digits:
        - backPrice = (ethPrice * 100000) (P2EGameContract.sol#114)
P2EGameContract.withdraw(uint256) (P2EGameContract.sol#147-164) uses
    ,→ literals with too many digits:
```

```
            - require(bool,string)(msg.value == uint256(backPrice) *
                ,→ 1000000000,Sent value not equals ETH gas price) (
                ,→ P2EGameContract.sol#154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #too-many-digits
INFO:Detectors:
P2EGameContract.smartContractAddress (P2EGameContract.sol#67) should be
    ,→ immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #state-variables-that-could-be-declared-immutable
INFO:Slither:. analyzed (3 contracts with 94 detectors), 21 result(s)
    ,→ found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

5 Conclusion

In this audit, we examined the design and implementation of Casino Nite Coin Pusher contract and discovered several issues of varying severity. Nitefeeder team addressed all the issues raised in the initial report and implemented the necessary fixes.

The present code base is well-structured and ready for the mainnet.

For a Smart Contract Audit, contact us at contact@nexarias.com