Whiterock Casino
ERC20 Token Audit

The BlockChain Auditor

Prepared by: Jorge Martinez

Version: 1
Date: Jan. 30, 2020

Summary of Findings

This document expresses all security concerns of the Whiterock Casino Smart Contract as expressed by Jorge Martinez. I took care to attempt to find as many ways to improve the security, code efficiency, best practices, and overall function of the smart contracts.

Contract Status: Safe to Use

Contract is Deployed at the following address:
0xA3D93c0616dbC31FEf1e112C7665A4bA4dDBf0be

0 Critical Issue(s) found were found.
0 Medium Issue(s) were found.
0 Low Issue(s) were found.
2 Informational Issue(s) were found.

Solidity Code Coverage

Jorge's Test Suite	Whiterock Casino	Industry Standard
95.65%	0%	95%

For this audit, I wasn't provided with a testing suite but as part of my audit methodology I developed a test suite to verify the functionality of the ERC20 contract, check its security, and to help reveal any underlying issues.

This audit should be seen as one step in the development process with the intent of raising awareness on the meticulous work involved in secure development and making no material statements or guarantees to the operational state of the smart contract(s) once they are deployed. This document is not an endorsement of the reliability or effectiveness of the smart contracts. This is an assessment of the smart contract logic, implementation, and best practices. I cannot take responsibility for any potential consequences of the deployment or use of the smart contract(s) related to the audit.

Test Suite Results

Jorge's Test Suite

```
PWC Token Test Suite
  Token Functionality
    deployment
      ✓ set name to PrimeWhiteRockCompany (915ms)
      ✓ sets symbol to PWC (38ms)
      ✓ has 18 decimals
      ✓ owner is target user
      ✓ total supply should be 50 million tokens
    allowance
      ✓ it returns an address' allowance (190ms)
    approve
      ✓ approve allows an address to approve another address to transfer their tokens (84ms)
    decreaseApproval
      ✓ can decrease another address' allowance (78ms)
      ✓ if decreasing an address' allowance will overflow it is set to zero
    increaseApproval
      ✓ increasing an address' allowance can overflow but will never decrease below its current value (51ms)
      ✓ an address can increase another address' allowance
    transfer
      ✓ cannot transfer to the zero address
      ✓ reverts if more tokens are transferred than owned
      ✓ holders can transfer tokens (55ms)
      ✓ if paused transfer reverts (38ms)
    transferFrom
      ✓ cannot transfer tokens to the zero address (44ms)
      ✓ cannot transfer more tokens out of address than it owns
      ✓ cannot transfer more tokens out of an address than allowed
      ✓ if paused transferFrom reverts (56ms)
    freezing and pausing functionality
      ✓ accounts with frozen balances should not be able to transfer tokens
      ✓ accounts have a frozen balance in addition to the regular token balance
      ✓ owner can pause and unpause (49ms)
      ✓ freezingCount() returns how many batches of frozen tokens an address has
      ✓ getFreezing() tells you when a frozen balance will be released
      ✓ users can freeze their tokens at a specified address until a given time (112ms)
      ✓ after freezing time has passed, account can claim their frozen tokens
      ✓ releaseOnce() doesn't release tokens if timelock is not satisfied (83ms)
      ✓ releaseOnce() reverts if there are no frozen tokens to release
      ✓ releaseAll() doesn't release tokens if timelock is not satisfied (85ms)
      ✓ freeze() won't let you freeze tokens into the past (38ms)
      ✓ accounts can freeze tokens at different times and addresses (298ms)
      ✓ cannot call freezeTo() on the zero address
      ✓ cannot freeze more tokens than are owned
    mintability
      ✓ the owner cannot mint more tokens after initialization
      ✓ no one can mint tokens after initialization
    burning
      ✓ anyone can burn their own tokens
      ✓ no one can burn more tokens than they own
    ownership
      ✓ only the owner can call transferOwnership()
      ✓ transferOwnership() cannot transfer ownership to the zero address
      ✓ it can transfer ownership to the zero address by calling renounceOwnership()


  40 passing (3s)

----------------|----------|----------|----------|----------|----------------|
File            |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------|----------|----------|----------|----------|----------------|
 contracts/     |    95.71 |    91.67 |    95.65 |    95.95 |                |
  Dummy.sol     |      100 |      100 |      100 |      100 |                |
  MainToken.sol |    95.65 |    91.67 |    95.56 |    95.91 |... 9,60,70,727 |
----------------|----------|----------|----------|----------|----------------|
All files       |    95.71 |    91.67 |    95.65 |    95.95 |                |
----------------|----------|----------|----------|----------|----------------|
```

Table of Contents

Audit Methodology & Techniques

The BlockChain Auditor has the following auditing process:

1. Our audits include
   a. Review of the specifications, source code, and instructions provided to the TheBlockchainAuditor to clearly identify the desired functionality of the smart contract(s).
   b. Manual line by line review of contract code to spot potential vulnerabilities.
   c. Identification of deviations between desired functionality expressed to the TheBlockchainAuditor and what the smart contract(s) are doing.
2. Automated static and symbolic analysis, as well as verifying testing coverage using the provided test suite.
   a. Automated static and symbolic analysis help determine what inputs cause each part of the smart contract to execute. Analysis of how much of the code base is tested and comparison to industry standard.
3. Examination of smart contracts and development process as a whole, ensuring best practices are followed, allowing improved efficiency and security based on established industry and academic practices.
4. Specific, itemized, and actionable recommendations to assist in securing the smart contract(s) in question.

Contract Checklist

MainToken.sol

Contract Vulnerability	
Integer Overflow	Pass
Race Condition	Pass
Denial of Service	Pass
Logical Vulnerability	Pass
Hardcoded Address	Pass
Function Input Parameter Check	Pass
Function Access Control Check	Pass
Random Number Generation	N/A
Random Number Use	N/A
Contract Specification	
Solidity Compiler Version	Pass
Event Use	Pass
Fallback Function Use	Pass
Constructor Use	Pass
Function Visibility Declaration	Pass
Variable Storage Declaration	Pass
Deprecated Keyword Use	Pass
ERC20/223 Standard	Pass
ERC721 Standard	N/A
Business Risk	
Able to Arbitratrily Create Token	N/A
Able to Arbitrarily Destroy Token	N/A
Can Suspend Transactions	Pass
Short Address Attack	Pass
Gas Optimization	
assert()/require()/revert() misused	Pass
Loop Optimization	Pass
Storage Optimization	Pass

Issue Classification

Critical

These issues in the smart contract can have catastrophic implications that could ruin your reputation, disrupt the contract's functionality, or impact the client and your users' sensitive information.

Informational

This issue relates to style and security best practices but does not pose an immediate risk.

Medium

An issue classified as medium has relatively small risk and isn't exploitable to circumvent desired functionality and could not have financial consequences but could put user's sensitive information at risk.

Acknowledged

The issue remains in the code but is a result of an intentional business or design decision.

Low

An issue classified as informational does not pose an immediate threat to disruption of functionality and could not be exploited on a recurring basis, however, it should be considered for security best practices or code integrity.

Unresolved

Although the client has been informed of the risk, it was decided to accept it because it was not relevant in the functionality of the smart contract.

Undetermined

The impact of the issue is uncertain and more investigation is required to understand the repercussions of the issue.

Mitigated

Actions were taken to minimize the impact or likelihood of the risk.

5.1 contract Consts

MainToken.sol 677

Invalid Address Checksum

Explanation:

On line 677, the address literal stored in TARGET_USER, the address that becomes the owner of PWCToken, has an invalid checksum.

Recommendation:

Since the contract is already deployed I was able to confirm that the ownership was changed to the correct address. If it the contract wasn't deployed the recommendation would be to use the correct checksum but it ended up working as intended.

5.2 init()

MainToken.sol 730

Invalid Address Checksum

Explanation:

On line 730, all the addresses stored in an address aray have an invalid checksum.

Recommendation:

Since the contract is already deployed I was able to confirm that the addresses were minted the tokens specified. If it the contract wasn't deployed the recommendation would be to use the correct checksum but it ended up working as intended.

Executive Summary

Overall Thoughts

After developing an extensive test suite, I was able to verify the functionality of the token. This token is EIP20 compliant so users will be able to transfer the token as expected. This token is also freezable, pausable, burnable, and mintable but only during deployment. Users with a token balance are able to freeze their tokens at whatever address they want for as long as they want which is something you don’t see as much in most tokens. If an exploit is found the owner of the token is able to halt transactions while they figure out what is going on. What is also interesting is that any user is able to burn their own tokens and only their own. Due to the way that the contract was written, after the function init() is called upon the deployment, minting new tokens shouldn’t be possible anymore so users don’t have to worry about being dumped on in that way. However, keep in mind that some tokens are locked away until 1614207602, February 24th, and 1623708002, June 14th. The freezable mechanism was well implemented, you can store as many tokens in as many different addresses as you want for as long as you want as many times as you want and the contract will keep track of it.

Appendix A

File Fingerprints

MainToken.sol	f91a74718eb727a18b36b41779eb9639

The Blockchain Auditor is honored to have the opportunity to help verify enhanced security and efficiency for the Whiterock Casino platform and provide security for users of the token.

I look forward to seeing how Whiterock Casino will utilize blockchain techonology in their casino infrastructure.

The BlockChain Auditor

- Jorge Martinez