AUDITS

SECURITY ASSESSMENT

BRZ TOKEN

AUGUST 31 ST 2022

AUDITS

TABLE OF CONTENTS

LEGAL DISCLAIMER

MH Audits are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are 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 MH Audits to perform a security review.

MH Audits 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.

MH Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

The report is provided only for the contract(s) mentioned in the report and does not include any other potential additions and/or contracts deployed by Owner. The report does not provide a review for contract(s), applications and/or operations, that are out of this report scope.

MH Audits' goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

MH Audits represents an extensive auditing 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk. MH Audits' position is that each company and individual are responsible for their own due diligence and continuous security.

The security audit is not meant to replace functional testing done before a software release. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits and a public bug bounty program to ensure the security of the smart contracts.

INTRODUCTION

MH Audits is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Secure your project with MH Audits

We offer field-proven audits with in-depth reporting and a range of suggestions to improve and avoid contract vulnerabilities.

Industry-leading comprehensive and transparent smart contract auditing on all public and private blockchains.

Vunerability checking

A crucial manual inspection carried out to eliminate any code flaws and security loopholes. This is vital to avoid vulnerabilities and exposures incurring costly errors at a later stage.

Contract verification

A thorough and comprehensive review in order to verify the safety of a smart contract and ensure it is ready for launch and built to protect the end-user.

Risk assessment

Analyse the architecture of the blockchain system to evaluate, assess and eliminate probable security breaches. This includes a full assessment of risk and a list of expert suggestions.

In-depth reporting

A truly custom exhaustive report that is transparent and depicts details of any identified threats and vulnerabilities and classifies those by severity.

Fast turnaround

We know that your time is valuable and therefore provide you with the fastest turnaround times in the industry to ensure that both your project and community are at ease.

Best-of-class blockchain engineers

Our engineers combine both experience and knowledge stemming from a large pool of developers at our disposal. We work with some of the brightest minds that have audited countless smart contracts over the last 4 years.

PROJECT SUMMARY

PROJECT INTRODUCTION

The BRZ is an ERC-20 token built-up on top of the Ethereum blockchain designed to maintain a 1:1 peg to Brazilian Real. According to their website, price stability will be pursued by market agents based on the reserves held by the reserve managers and the issuance of new tokens with the underlying reserves will be audited by a reputable third-party firm.

BRZ is the biggest non-USD stablecoin in the world. By sending and receiving BRZs, brazilians are experiencing financial freedom at a lower cost and faster than any other option on the market.

Project Name *BRZ Token*

Contract Name *BRZ*

Contract Address *0x491a4eb4f1fc3bff8e1d2fc856a6a46663ad556f (MATIC)*

Contract Chain *Mainnet*

Contract Type *Smart Contract*

Platform *EVM*

Language *Solidity*

Codebase *Private Repository*

INFO & SOCIALS

Network *Multiple Blockchain Deployments - Polygon (MATIC); RSK; Solana; Algorand; Ethereum; BNB Chain; Stellar*

Total Supply *1,000,000,000*

Website *https://www.brztoken.io/*

Twitter *https://twitter.com/BrzToken*

Telegram Chat *https://t.me/brz_token*

Telegram Ann *https://t.me/brztoken*

Medium *https://medium.com/@BrzToken*

PolygonScan *https://polygonscan.com/token/0x491a4eb4f1fc3bff8e1d2fc856a6a46663ad556f*

AUDIT SCORES

Issues	5
Critical	0
Major	1
Medium	1
Minor	0
Informational	3
Discussion	0

All issues are described in further detail on the following pages.

AUDIT SCOPE

FILE	LOCATION
brz-v3.sol	*GitHub Repository*

REVIEW METHODOLOGY

TECHNIQUES

This report has been prepared for BRZ Token to discover issues and vulnerabilities in the source code of the BRZ Token project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic, Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective in the comments below.

TIMESTAMP

Version	v1.0
Date	2022/08/22
Description	Layout project Automated / Manual review / Static & dynamic security testing Summary
Version	v1.1
Date	2022/08/31
Description	Reaudit addressed issues Final summary

KEY FINDINGS

TITLE	SEVERITY	STATUS
Functions Should Be Declared External	**Gas**	*Fixed*
Missing Return Value Validation	**Medium**	*Fixed*
Redundant Code	**Informational**	*Fixed*
Use msg.sender **instead of** _msgSender()	**Gas**	*Fixed*
_approve() **Frontrunning Attack**	**Major**	*Fixed*

Description: *Public functions that are never called by a contract should be declared external in order to conserve gas.*

The following functions were declared as public but were not called anywhere in the contract, making the public visibility useless.

Affected Code: mint() - ***L1287***
burn() - ***L1292***
burnFrom() - ***L1297***
pause() - ***L1305***
unpause() - ***L1310***
tokenWithdraw() - ***L1323***

Impacts:

Smart Contracts are required to have effective Gas usage as they cost real money, and each function should be monitored for the amount of gas it costs to make it gas efficient.

*"*public*" functions cost more Gas than "*external*" functions.*

Issue: *Functions Should Be Declared* External

Type: *Gas Optimization*

Level: *Gas*

Recommendation: *Use the "*external*" state visibility for functions that are never called from inside the contract.*

Alleviation / Retest: *All the affected functions have been updated to* external.

Description: *The contract "Token.sol" is making an external transfer call on line <line_number> inside the function "*tokenWithdraw()*". Several tokens do not revert and return false. This may cause issues and failed assumptions when making token transfers.*

Affected Code:

```
   function tokenWithdraw(address addressToken) public {
          require(hasRole(ADMIN_ROLE, _msgSender()), "Token: must have
admin role to tokenWithdraw");
       IERC20 token = IERC20(addressToken);
       uint256 balance = token.balanceOf(address(this));
       token.transfer(msg.sender, balance);
   }
```

Impacts: *Missing error handling on transfer return value may cause issues if the call fails as it will create inconsistencies with failed function calls.*

Issue: *Missing Return Value Validation*

Type: *Unchecked Call Return Value - SWC-104*
https://swcregistry.io/docs/SWC-104

Level: *Medium*

Recommendation: *It is recommended to use "SafeERC20" or check the return values of transfer and handle the errors appropriately.*

Alleviation / Retest: *A require check has been added to handle the return value.*

Description: *The contract was declaring a function called "*decimals()*" which is returning a value of 4 but it is never used anywhere in the code. Even though the function is pure and won't cost gas, it is recommended to remove the function if it's not being used anywhere.*

Affected Code:

```
function decimals() public pure override returns (uint8) {
    return 4;
}
```

Impacts: *Having undeclared functions in the code creates confusion and introduces inconsistencies for code reviewers and auditors.*

Issue: *Redundant Code*

Type: *Gas Optimization*

Level: *Informational*

Recommendation: *It is recommended to remove the function declaration and use the value of 4 directly wherever required.*

Alleviation / Retest: *The redundant code has been removed.*

Description: *The contract is using* _msgSender() *function call to get the current* msg.sender*. This is being used at multiple places and is not recommended since Solidity already provides a* msg.sender *to get the address of the user who initiated a transaction.*

Replacing all the _msgSender() *function calls with* msg.sender *will save around 14000 gas.*

Impacts: *Using* _msgSender() *function is costing the contract deployment around 14000 more gas.*

Issue: *Use* msg.sender *instead of* _msgSender()

Type: *Gas Optimization*

Level: *Gas*

Recommendation: *It is recommended to replace all the* _msgSender() *calls with* msg.sender*.*

Alleviation / Retest: *The function* _msgSender() *has been replaces with* msg.sender.

Description: *The contract uses an "*_approve()*" function call inside the function "*burnFrom()*". ERC20 Approve is well known to be vulnerable to front-running attacks. This may be exploited in cases where in case the "account" decides to modify the spending amount in quick succession and the "*msg.sender*" sees the change and burns more tokens than required.*

Affected Code:

```
    function burnFrom(address account, uint256 amount) public {
            require(hasRole(ADMIN_ROLE, _msgSender()), "Token: must have
admin role to burnFrom");
        uint256 currentAllowance = allowance(account, _msgSender());
         require(currentAllowance >= amount, "ERC20: burn amount exceeds
allowance");
        _approve(account, _msgSender(), currentAllowance - amount);
        _burn(account, amount);
    }
```

Impacts: *This vulnerability allows the* msg.sender *to burn more tokens than allowed by frontrunning the function call to "*burnFrom()*".*

Issue: _approve() ***Frontrunning Attack***

Type: *Frontrunning*

Level: *Major*

Recommendation: *Instead of* _approve() *to change the allowance, it is recommended to use* increaseAllowance *and* decreaseAllowance *functions which are meant for this use case.*

It is also recommended to refer to the following documentation for more information:

https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit

Alleviation / Retest: decreaseAllowance() *has been added to mitigate the risk.*

Private GitHub Repository

FINDING CATEGORIES

The assessment process will utilize a mixture of static analysis, dynamic analysis, in-depth manual review and/or other security techniques.

This report has been prepared for BRZ Token project using the above techniques to examine and discover vulnerabilities and safe coding practices in BRZ Token's smart contract including the libraries used by the contract that are not officially recognized.

A comprehensive static and dynamic analysis has been performed on the solidity code in order to find vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds.

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The testing methods find and flag issues related to gas optimizations that help in reducing the overall gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

AUDIT SCORES

MH Audits AuditScores is not a live dynamic score. It is a fixed value determined at the time of the report issuance date.

MH Audits AuditScores are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports and scores are 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 MH Audits to perform a security review.

AUDITS

WEBSITE
MHAUDITS.IO

TWITTER
@MHAUDITS