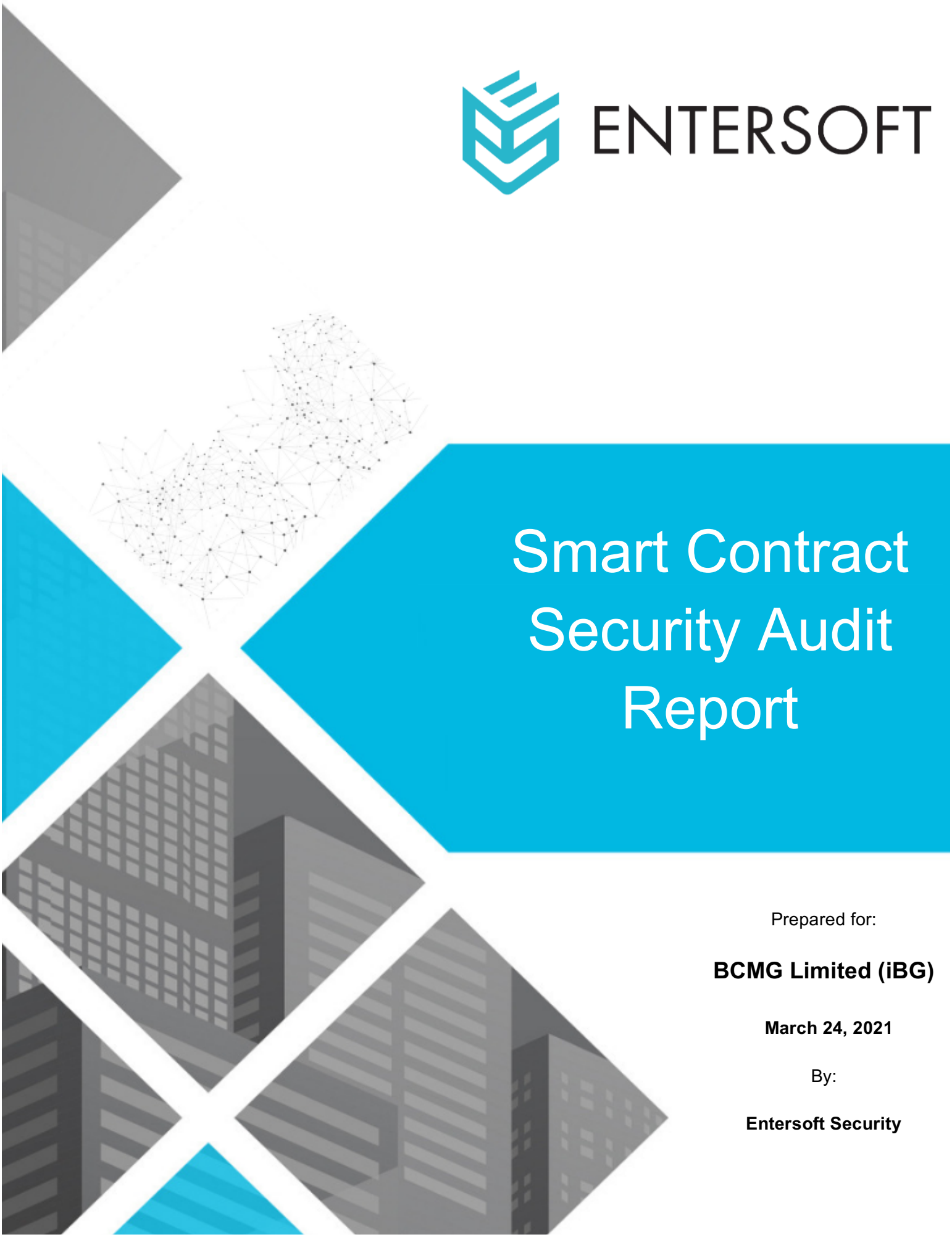

ENTERSOFT

Smart Contract Security Audit Report

Prepared for:

BCMG Limited (iBG)

March 24, 2021

By:

Entersoft Security

Contents

Version Control

Version Date	Created/Modified by	Description/Pages Modified
24/03/2021	Jake Lemke	Author
24/03/2021	Paul Kang	Review

1 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to: (i) smart contract best coding practices and issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). 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.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any all copies of this report downloaded and/or printed by you. 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 Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2 Background

Entersoft was commissioned by BCMG Limited to perform source code review on their solidity smart contract. The review was conducted between 19th March 2021 to 23rd March 2021.

The report is organized into the following sections.

- **Executive Summary:** A high-level overview of the security audit findings.
- **Technical analysis:** Our detailed analysis of the Smart Contract code

The information in this report should be used to understand overall code quality, security, correctness, and meaning that code will work as BCMG Limited describe in the smart contract. The analysis is static and entirely limited to Smart contract code.

3 Executive Summary

In the audit, we have reviewed the smart contract's code that implements the token mechanism. The following files are considered predominantly under review:

CONTRACT DETAILS	
TOKEN NAME	iBG
TOKEN SYMBOL	IBG
CONTRACT NAME	AmazingERC20
CONTRACT ADDRESS:	0xF16CD087e1C2C747b2bDF6f9A5498AA400D99C24
CONTRACT VERIFIED	Yes
CONTRACT SECURITY AUDITED	Yes
VULNERABILITIES	None

Scope

We have provided an independent technical audit to remove smart contract uncertainties and to keep it safe from any serious hacks. Smart contract audit keeps clients protected from any fraudulent activity. The audit was performed both manually and using automated tools. We have analyzed the smart contract code line by line and reported any suspicious code. After manual analysis, we have utilized automated tools to make sure the coverage is thorough.

We have considered the following Standards for the Smart contract code review:

1. ERC20 [Token Contract best practices]

We worked with the following tools to perform Automated tests:

1. **Automated Testing tools**
   - 1.1 Slither
   - 1.2 Manticore
   - 1.3 Mythril
   - 1.4 Echninda
2. **Manual Testing tools**
   - 2.1 Truffle
   - 2.2 Ganache
3. **Framework**
   - 3.1 Remix Ethereum

Auditing Approach and Methodologies applied

The Entersoft team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured securely with the safe use of third-party smart contracts and libraries.

Our team then performed a formal line by line inspection of the code of the Smart Contract to find any potential issues such as race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code audits included:

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
2. Analyzing the complexity of the code by thorough line-by-line manual review of the code.
3. Deploying the code on test net and using multiple clients to run live tests
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest versions.
6. Analyzing the security of the on-chain data.

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications that has been provided to the Auditing team. The audit activities can be grouped in the following three categories:

- **Security:** Identifying security related issues within each contract and the system of contracts.
- **Sound Architecture:** Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
- **Code Correctness and Quality:** A full review of the contract source code. The primary areas of focus include:
    - Correctness
    - Readability
    - Sections of code with high complexity
    - Quantity and quality of test coverage

Result

Audit was conducted on the provided one contract file. The following table provides an overall picture of the security posture. ✓ means no bugs are identified.

#	Smart Contract Penetration Test Objectives	Audit Subclass	Result
1	Overflow	-	✓
2	Race Condition	-	✓
3	permissions	Permission vulnerability Audit Excessive Auditing Authority	✓
4	Safety Design	Zeppelin safe math	✓
5	DDOS Attack	Call function security	✓
6	Gas Optimization	-	✓
7	Design Logic	-	✓
8	Know Attacks	-	✓
Overall Security Posture			**Secure**

Recommendations

Use case of the smart contract is very well designed and Implemented. Overall, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. BCMG Limited development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.

All the bugs, suggestions and recommends has been considered by BCMG Limited team and some of the issues they will handle to their own as those issues or calls have been handle by the only owner.

Code quality

The code quality is of high standard, and we believe it is professionally written along with high quality neat code. There is enough documentation for each function.

Code security

A static analysis of the code is performed to identify any loopholes in the Smart contract code. Also, to verify whether the contracts adhere to the Solidity Best Practices.

Security Level references

Every issue in this report was assigned a severity level from the following:

High severity issues:

The issue puts the vast majority of, or large numbers of, users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.

Medium severity issues:

The issue puts a subset of individual users' sensitive information is at risk, exploitation would be detrimental for the client's reputation, or is reasonably likely to lead to moderate financial impact.

Low severity issues:

The risk is relatively small and could not be exploited on a recurring basis or is a risk the client has indicated is not important or impactful in view of the client's business circumstances.

Informational:

The issue does not pose an immediate threat to continued operation or usage, but is relevant for security best practices, software engineering best practices, or defensive redundancy.

Number of Vulnerabilities by Severity

HIGH	MEDIUM	LOW	INFO
0	0	0	2

High severity Vulnerabilities:

- No High Severity Vulnerabilities.

Medium Severity Vulnerabilities:

- No Medium Severity Vulnerabilities.

Low Severity Vulnerabilities:

- No Medium Severity Vulnerabilities.

Informational:

- 2 Informational Quality recommendations

Severity	**Informational**
Description	**Local Variable Shadowing**
Code	ERC1363.constructor(string,string).name (IBG.sol#969) shadows: - ERC20.name() (IBG.sol#327-329) (function) ERC1363.constructor(string,string).symbol (IBG.sol#969) shadows: - ERC20.symbol() (IBG.sol#335-337) (function) IBG.constructor(string,string,uint8,uint256,address).name (IBG.sol#1216) shadows: - ERC20.name() (IBG.sol#327-329) (function) IBG.constructor(string,string,uint8,uint256,address).symbol (IBG.sol#1217) shadows: - ERC20.symbol() (auditContract.sol#335-337) (function) IBG.constructor(string,string,uint8,uint256,address).decimals (IBG.sol#1218) shadows: - ERC20.decimals() (IBG.sol#352-354) (function)
Recommendation	Rename the local variables that shadow another component.

Severity	Informational
Description	Public Function that could be declared external • Check: external-function • Severity: Optimization • Confidence: High
Code	finishMinting() should be declared external: - ERC20Mintable.finishMinting() (IBG.sol#1061-1063) renounceOwnership() should be declared external: - Ownable.renounceOwnership() (IBG.sol#1125-1128) transferOwnership(address) should be declared external: - Ownable.transferOwnership(address) (IBG.sol#1134-1138) recoverERC20(address,uint256) should be declared external: - TokenRecover.recoverERC20(address,uint256) (IBG.sol#1156-1158)
Recommendation	Use the external attribute for functions never called from the contract.

4. Technical Analysis

The following is our technical automated and manual analysis of the Smart contract code created and used by BGTBC

Checked Vulnerabilities

We have checked BGTBC smart contract for commonly known as well as specific business logic vulnerabilities. Following are the list of vulnerabilities tested in the smart contract code

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DOS with (Unexpected) revert
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility

Unit Testing - Test Suite

Contract: BCMG LIMITED Token Contracts

✓Should correctly deploy IBG contract
✓Should check a name of a token of IBG
✓Should check a symbol of a token IBG
✓Should check a owner of a token (61ms)
✓Should check a balance of a token contract
✓Should check a balance of a owner (73ms)
✓Should check the total supply of a token contract
✓should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4
✓should Approve accounts[1] to spend specific tokens of accounts[4]
✓should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (42ms)
✓should increase Approve accounts[4] to spend specific tokens of accounts[1]
✓should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (38ms)
✓should decrease Approve accounts[4] to spend specific tokens of accounts[1]
✓should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (58ms)
✓should decrease Approve accounts[4] to spend specific tokens of accounts[1]
✓should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (50ms)
✓should Approve accounts[1] to spend specific tokens of accounts[4] agin
✓should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (46ms)
✓should be able to transferfrom accounts[4] to accounts[1]
✓should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (50ms)
✓Should check a balance of a beneficiary of accounts[4] after sending all tokens
✓Should check a balance of a receiver accounts[1]
✓Should check a owner of a token before transferring ownership
✓Should not be able to transfer ownership before
✓Should be able to transfer ownership before
✓Should be able to accept transfer ownership before (39ms)
✓Should check a owner of a token after transferring ownership
✓Should be able to transfer ownership again to accounts[0] (39ms)
✓Should be able to accept transfer ownership before (41ms)
✓Should check a owner of a token after transferring ownership
✓Should check a balance of a accounts[6]

```
✓should check approval by accounts 5 to accounts 6 to spend locked tokens on the behalf of accounts 5
✓should Approve accounts[1] to spend specific tokens of accounts[4]
✓should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (40ms)
```

Result of Test:

64 PASSED.

0 FAILED.

Automation Tool testing: Slither, Mythril, Echidna, Manticore

```
TokenRecover.recoverERC20(address,uint256) (AmazingERC20.sol#1156-1158) ignores return value by IERC20(tokenAddress).transfer(owner(),tokenAmo
unt) (AmazingERC20.sol#1157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
```

```
Contract locking ether found in :
        Contract ServiceReceiver (AmazingERC20.sol#1169-1193) has payable functions:
         - ServiceReceiver.pay(string) (AmazingERC20.sol#1175-1179)
        But does not have a function to withdraw the ether
Contract locking ether found in :
        Contract AmazingERC20 (AmazingERC20.sol#1213-1250) has payable functions:
         - AmazingERC20.constructor(string,string,uint8,uint256,address) (AmazingERC20.sol#1215-1228)
        But does not have a function to withdraw the ether
```

```
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (AmazingERC20.sol#327-329)
symbol() should be declared external:
        - ERC20.symbol() (AmazingERC20.sol#335-337)
decimals() should be declared external:
        - ERC20.decimals() (AmazingERC20.sol#352-354)
totalSupply() should be declared external:
        - ERC20.totalSupply() (AmazingERC20.sol#359-361)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (AmazingERC20.sol#366-368)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (AmazingERC20.sol#378-381)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (AmazingERC20.sol#397-400)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (AmazingERC20.sol#415-419)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (AmazingERC20.sol#433-436)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (AmazingERC20.sol#452-455)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (AmazingERC20.sol#589-591)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (AmazingERC20.sol#604-609)
mintingFinished() should be declared external:
        - ERC20Mintable.mintingFinished() (AmazingERC20.sol#1040-1042)
mint(address,uint256) should be declared external:
        - ERC20Mintable.mint(address,uint256) (AmazingERC20.sol#1052-1054)
finishMinting() should be declared external:
        - ERC20Mintable.finishMinting() (AmazingERC20.sol#1061-1063)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (AmazingERC20.sol#1125-1128)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (AmazingERC20.sol#1134-1138)
recoverERC20(address,uint256) should be declared external:
```

```
INFO:Detectors:
Address.isContract(address) (AmazingERC20.sol#809-818) uses assembly
        - INLINE ASM None (AmazingERC20.sol#816)
Address._verifyCallResult(bool,bytes,string) (AmazingERC20.sol#846-863) uses assembly
        - INLINE ASM None (AmazingERC20.sol#855-858)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
```

```
INFO:Detectors:
ERC1363.constructor(string,string).name (AmazingERC20.sol#969) shadows:
        - ERC20.name() (AmazingERC20.sol#327-329) (function)
ERC1363.constructor(string,string).symbol (AmazingERC20.sol#969) shadows:
        - ERC20.symbol() (AmazingERC20.sol#335-337) (function)
AmazingERC20.constructor(string,string,uint8,uint256,address).name (AmazingERC20.sol#1216) shadows:
        - ERC20.name() (AmazingERC20.sol#327-329) (function)
AmazingERC20.constructor(string,string,uint8,uint256,address).symbol (AmazingERC20.sol#1217) shadows:
        - ERC20.symbol() (AmazingERC20.sol#335-337) (function)
AmazingERC20.constructor(string,string,uint8,uint256,address).decimals (AmazingERC20.sol#1218) shadows:
        - ERC20.decimals() (AmazingERC20.sol#352-354) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
```

Implementation Recommendations:

name() should be declared external:
- ERC20.name() (IBG.sol#327-329)
symbol() should be declared external:
- ERC20.symbol() (IBG.sol#335-337)
decimals() should be declared external:
- ERC20.decimals() (IBG.sol#352-354)
totalSupply() should be declared external:
- ERC20.totalSupply() (IBG.sol#359-361)
balanceOf(address) should be declared external:
- ERC20.balanceOf(address) (IBG.sol#366-368)
transfer(address,uint256) should be declared external:
- ERC20.transfer(address,uint256) (IBG.sol#378-381)
approve(address,uint256) should be declared external:
- ERC20.approve(address,uint256) (IBG.sol#397-400)
transferFrom(address,address,uint256) should be declared external:
- ERC20.transferFrom(address,address,uint256) (IBG.sol#415-419)
increaseAllowance(address,uint256) should be declared external:
- ERC20.increaseAllowance(address,uint256) (IBG.sol#433-436)
decreaseAllowance(address,uint256) should be declared external:
- ERC20.decreaseAllowance(address,uint256) (IBG.sol#452-455)
burn(uint256) should be declared external:
- ERC20Burnable.burn(uint256) (IBG.sol#589-591)
burnFrom(address,uint256) should be declared external:
- ERC20Burnable.burnFrom(address,uint256) (IBG.sol#604-609)
mintingFinished() should be declared external:
- ERC20Mintable.mintingFinished() (IBG.sol#1040-1042)
mint(address,uint256) should be declared external:
- ERC20Mintable.mint(address,uint256) (IBG.sol#1052-1054)
finishMinting() should be declared external:
- ERC20Mintable.finishMinting() (IBG.sol#1061-1063)
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (IBG.sol#1125-1128)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (IBG.sol#1134-1138)
recoverERC20(address,uint256) should be declared external:
- TokenRecover.recoverERC20(address,uint256) (IBG.sol#1156-1158)
pay(string) should be declared external:
- ServiceReceiver.pay(string) (IBG.sol#1175-1179)
getPrice(string) should be declared external:
- ServiceReceiver.getPrice(string) (IBG.sol#1181-1183)

setPrice(string,uint256) should be declared external:
- ServiceReceiver.setPrice(string,uint256) (IBG.sol#1185-1187)

5 Limitations on Disclosure and Use of this Report.

This report contains information concerning potential details of BCMG LIMITED and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein.

Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures.

In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the BCMG LIMITED Smart Contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis.

This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended.

This report was prepared by Entersoft for the exclusive benefit of BCMG LIMITED and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and BCMG Limited govern the disclosure of this report to all other parties including product vendors and suppliers.