SMART CONTRACT AUDIT – ZAMZAM TOKEN

DECEMBER 31, 2021

Author: ATEITIS

Eng Carlos Adolfo Alejandro

CTO

ATEITIS INFORMATION TECHNOLOGY LLC

ATEITIS INFORMATION TECHNOLOGY

Table of Contents

Confidentiality Clause

The document is property of ZAMZAM TOKEN and includes information that is privileged, confidential and/or cannot be disclosed. If you are not an authorized recipient, please return this document to the owner mentioned above.

This is a security audit report document and may contain information that is confidential. It includes any potential vulnerabilities and malicious codes that can be used to exploit the software. This must be referred internally and should only be available to the public once the issues have been resolved.

The disclosure, distribution, copying or use of all or part of this document to any person other than the corresponding recipient is forbidden without prior written permission of ZAMZAM TOKEN.

Limitations and Disclaimer

ZAMZAM TOKEN is a standard BEP20 token smart contract. This audit only considers ZAMZAM TOKEN smart contract.

The observations or conclusions expressed in this document are the result of the analysis of the information obtained from the execution of technical tests performed on the set of devices identified as part of the selected sample, therefore:

- They are only applicable to the reviewed environment under the conditions discussed above, i.e. they are not applicable to other environments;
- There may be other vulnerabilities in addition to those currently detected related to the modification of the environment under analysis.

Throughout the evaluation process described in this document, the tests have been carried out only on the addressing defined in the aforementioned SCOPE, excluding intermediate elements not included in it.

This report presents all the findings regarding the audit performed on December 31, 2021.

The purpose of this audit was to cover the following:

- Ensure that all claimed functions exist and function correctly.

- Identify any security vulnerabilities that may be present in the smart contract.

ATEITIS INFORMATION TECHNOLOGY LLC team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. It is important to consider that this report should not be relied upon alone. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Therefore, the audit cannot guarantee explicit security of the audited smart contracts.

Investors Advice: the technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Scope and Methodology

Name	**ZAMZAM TOKEN**
Platform	**BSC / Solidity**
File	New ZAMZAM Token.sol
Smart Contract BSC	https://bscscan.com/token/0xa5e279e14efd60a8f29e5ac3b464e3de0c6bb6b8
Smart Contract Hash MD5	747038FFD4F03C80D7819A2689AF9A94
Audit Date	December 31, 2021

The goals of our security audits are to improve the quality of systems we review We use the following methodology in our security audit process:

Manual Code Review

We look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors and random number generators. We also look for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

We look at the project's website to get a high level understanding of what functionality the software under review provides. After that, we meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies and generally investigate details other than the implementation.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful solution. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not verified the feasibility and impact of the issue yet. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. After this we analyze the feasibility of an attack in a live system.

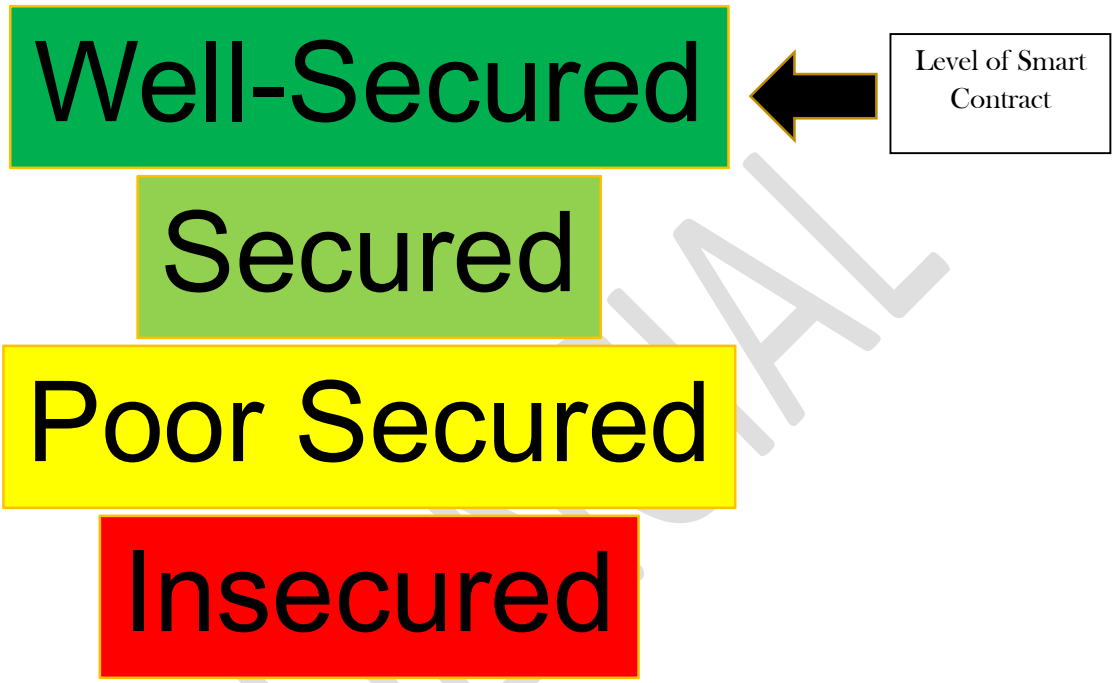
Suggested Solutions

We search for immediate mitigations that live deployments can take and finally we suggest the requirements for solution engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Executive Summary

Overall Result

According to the standard audit assessment, ZAMZAM TOKEN (smart contracts details in scope) are "Well-Secured".



All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 2 low.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Insecure Compiler Version	Failed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Function Default Visibility	Failed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Passed
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Code Quality

The audit scope has 1 (one) smart contract file. Smart contract contains libraries, smart contracts, inherits and interfaces. This is a compact and well written smart contract.

The libraries in ZAMZAM TOKEN are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the ZAMZAM TOKEN.

The ZAMZAM TOKEN team has provided scenario and unit test scripts, which helped to determine the integrity of the code in an automated way.

Code parts are well commented on smart contracts.

CONCLUSION: PASSED

Documentation

ZAMZAM TOKEN has given us a smart contract code in the form of code. The hash of that code is mentioned above in the table.

As code parts are well commented, it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

CONCLUSION: PASSED

Use of Dependencies

According to our observation, this smart contract infrastructure uses libraries that are based on well-known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

CONCLUSION: PASSED

AS-IS overview

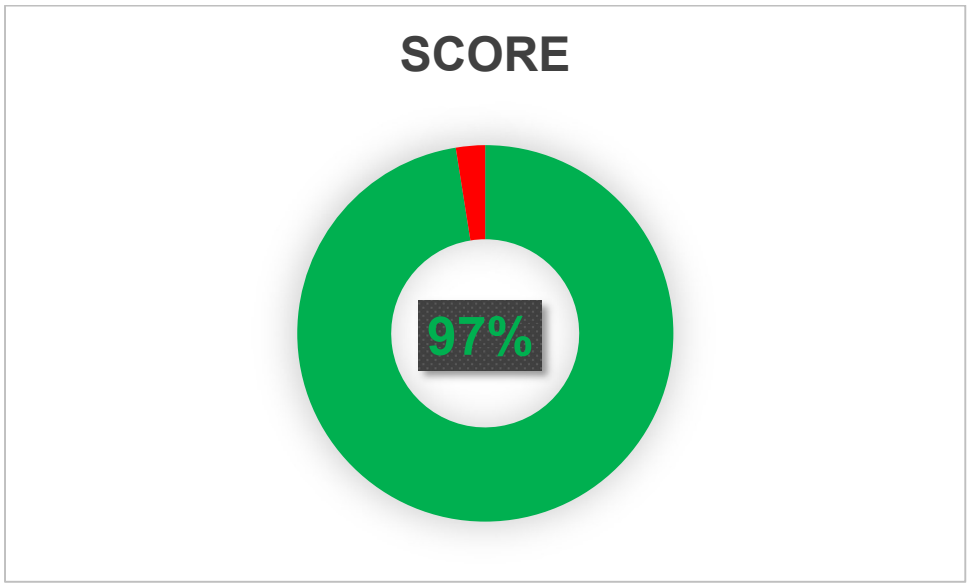
ID	Name	Context	Modifiers	Audit Level Risk
1	owner	Public	NO	Well-Secured
2	renounceOwnership	Public	Only Owner	Low Risk
3	transferOwnership	Public	Only Owner	Low Risk
4	<Constructor>	Public	NO	Well-Secured
5	getOwner	External	NO	Well-Secured
6	decimals	External	NO	Well-Secured

7	symbol	External	NO	Well-Secured
8	name	External	NO	Well-Secured
9	totalSupply	External	NO	Well-Secured
10	balanceOf	External	NO	Well-Secured
11	transfer	External	NO	Well-Secured
12	allowance	External	NO	Well-Secured
13	approve	External	NO	Well-Secured
14	transferFrom	External	NO	Well-Secured
15	increaseAllowance	Public	NO	Low Risk
16	decreaseAllowance	Public	NO	Low Risk
17	mint	Public	Only Owner	Low Risk

CONCLUSION: LOW RISK

Conclusion

Taking into account the information provided by the client including the investigations and each of the analyses reported in this report, ATEITIS INFORMATION TECHNOLOGY LLC is of the opinion that the overall security status of the smart contract is "Well-Secured". It should be noted that this opinion is based on each of the aspects specified in this report.ATEITIS INFORMATION TECHNOLOGY LLC recommends mitigating the low criticality issues specified in the report, in order to increase the performance and efficiency of the report, as well as to increase the level of security in the audited environment.



Based on the result obtained from the Audit, ATEITIS INFORMATION TECHNOLOGY LLC is of the opinion that the present token complies with good security practices in general and can be deployed in different CEX.

OVERALL RESULT : "WELL SECURED".

Report of Vulnerabilities

Severity definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually easy to exploit and can lead to loss of tokens, among others. Impact in the loss of funds for contract owner or users.
High	High-level vulnerabilities are difficult to exploit. However, they also have significant impact on smart contract execution, e.g. public access to crucial.
Medium	Medium-level vulnerabilities are important to fix. However, they can't lead to the loss of tokens.
Low	Low-level vulnerabilities are mostly related to outdated,unused code snippets, among others,that can't have significant impact on execution.

Audit Findings

CRITICAL SEVERITY

No Critical severity vulnerabilities were found.

HIGH SEVERITY

No High severity vulnerabilities were found.

MEDIUM SEVERITY

No Medium severity vulnerabilities were found.

LOW SEVERITY

Insecure Compiler Version

Line 5 in ZAMZAM TOKEN TOKEN Contract

```
pragma solidity 0.5.16;
```

Vulnerability detail

Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.

Solution

It is recommended to use a recent version of the Solidity compiler.

SWCREGISTRY

- https://swcregistry.io/docs/SWC-102

Function Default Visibility

Line 320 in ZAMZAM TOKEN TOKEN Contract

```
function renounceOwnership() public onlyOwner {
  emit OwnershipTransferred(_owner, address(0));
  _owner = address(0);
}
```

Line 329 in ZAMZAM TOKEN TOKEN Contract

```
function transferOwnership(address newOwner) public onlyOwner {
  _transferOwnership(newOwner);
}
```

Line 469 in ZAMZAM TOKEN TOKEN Contract

```
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
  _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
  return true;
}
```

Line 488 in ZAMZAM TOKEN TOKEN Contract

```
function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
  _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "BEP20: decreased allowance below zero"));
  return true;
}
```

Line 501 in ZAMZAM TOKEN TOKEN Contract

```
function mint(uint256 amount) public onlyOwner returns (bool) {
  _mint(_msgSender(), amount);
  return true;
}
```

Vulnerability detail

Functions that do not have a function visibility type specified are public by default. This can lead to a vulnerability if a developer forgot to set the visibility and a malicious user is able to make unauthorized or unintended state changes. About the vulnerability, its recommended to review and implement "external" functions because are more efficients.

Solution

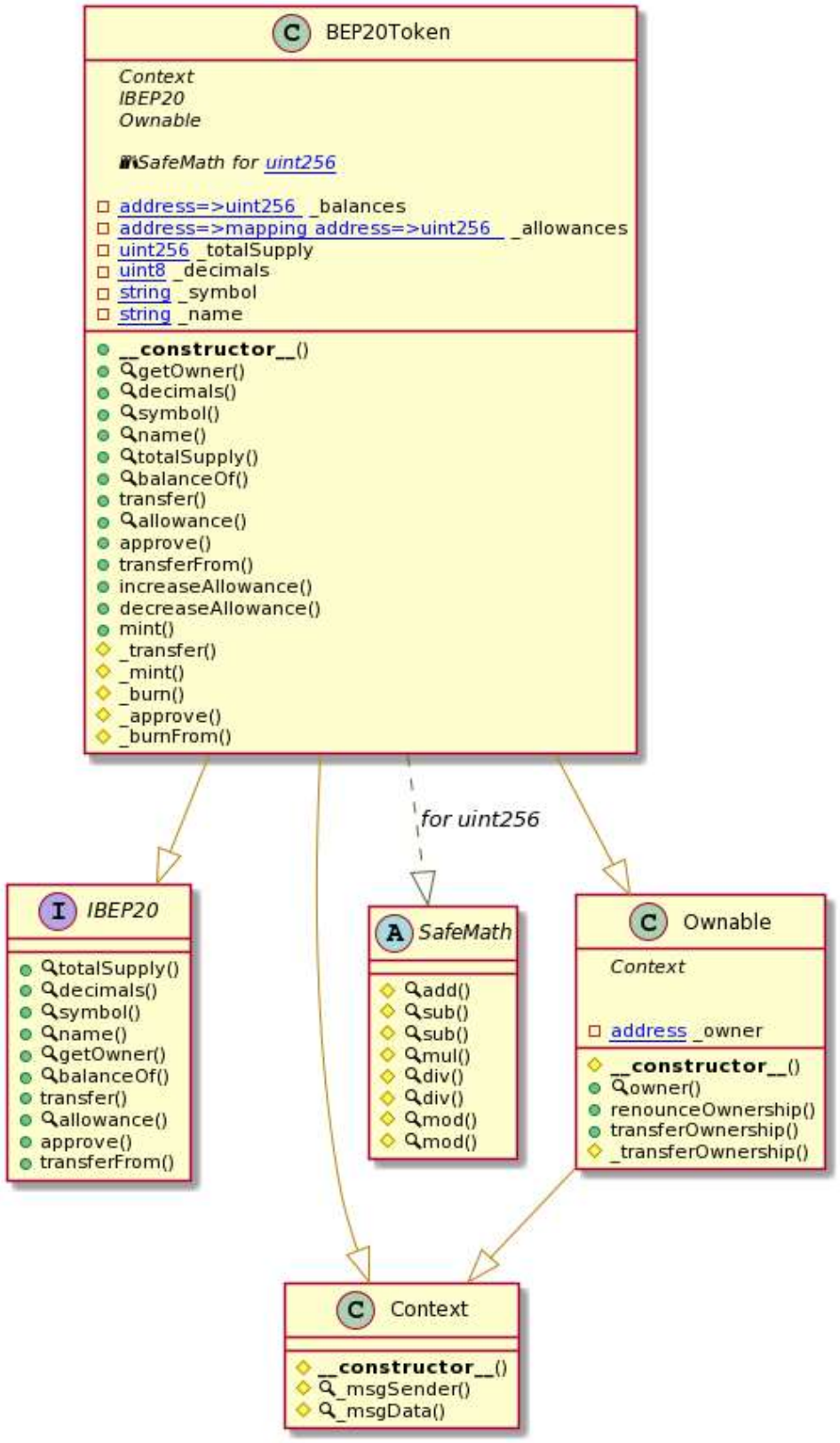
Considering to declare "public" functions as "external". "public" functions that are never called by the contract could be declared "external".

SWCREGISTRY

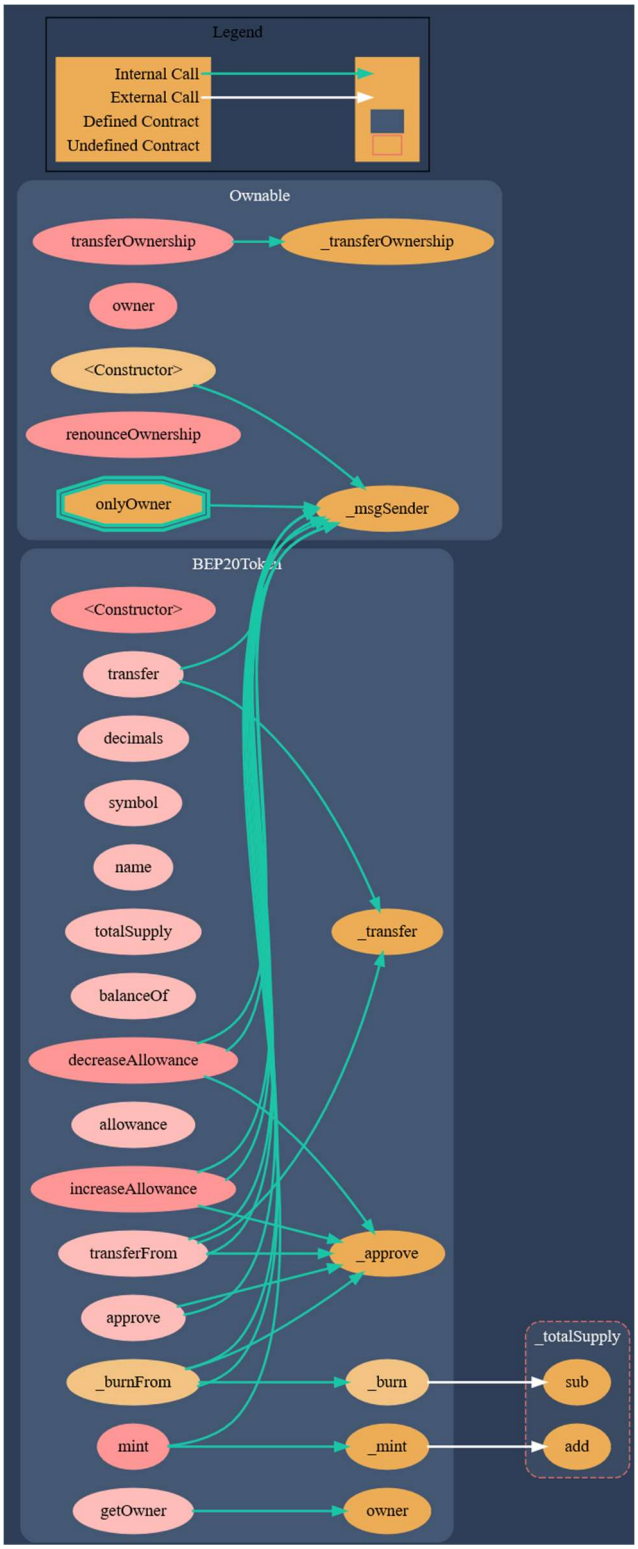
- https://swcregistry.io/docs/SWC-100

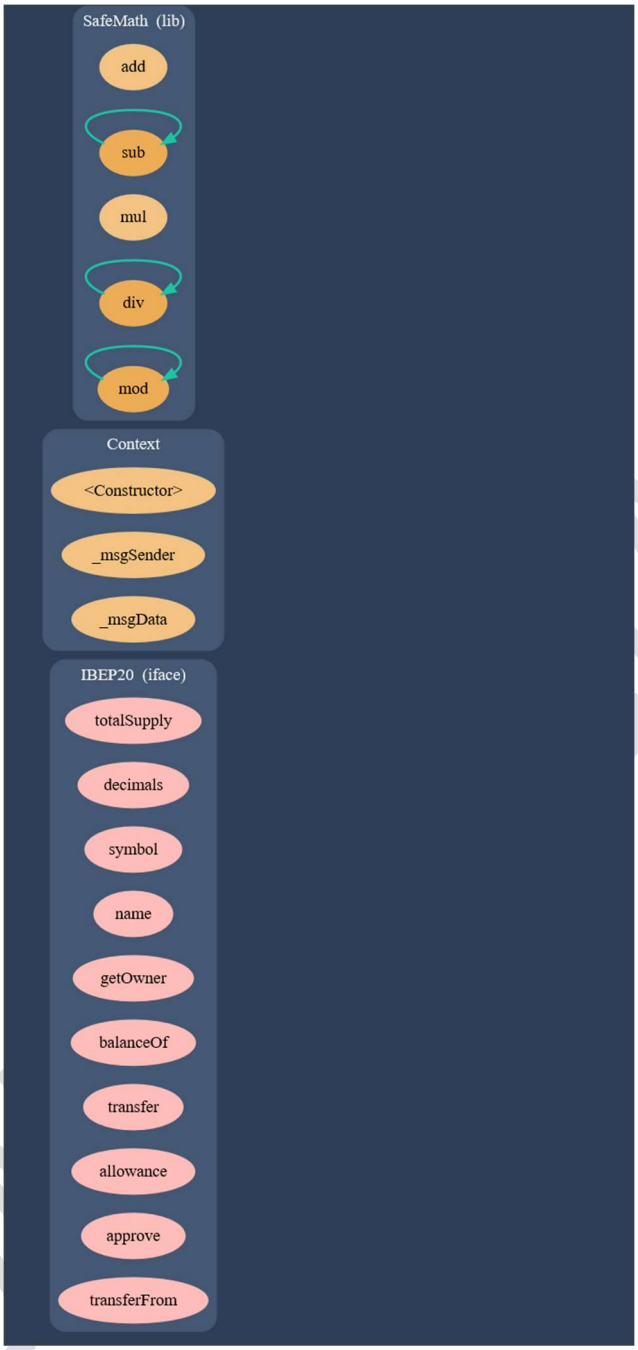
Appendix

UML Diagram



UML Diagram





Slither Results Log

```
INFO:Detectors:
BEP20Token.allowance(address,address).owner (ZAMZAM Token.sol#419) shadows:
        - Ownable.owner() (ZAMZAM Token.sol#297-299) (function)
BEP20Token._approve(address,address,uint256).owner (ZAMZAM Token.sol#574) shadows:
        - Ownable.owner() (ZAMZAM Token.sol#297-299) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Pragma version>=0.4.22<0.8.0 (Migrations.sol#2) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (ZAMZAM Token.sol#316-319)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (ZAMZAM Token.sol#325-327)
increaseAllowance(address,uint256) should be declared external:
        - BEP20Token.increaseAllowance(address,uint256) (ZAMZAM Token.sol#465-468)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20Token.decreaseAllowance(address,uint256) (ZAMZAM Token.sol#484-487)
mint(uint256) should be declared external:
        - BEP20Token.mint(uint256) (ZAMZAM Token.sol#497-500)
setCompleted(uint256) should be declared external:
```

MythX Results

Without Issues

HoneyPot checker