BU-ERC20 Contract Audit - Ankit

Introduction

This audit report highlights the overall security of the BU-ERC20 smart contract. With this report, I have tried to ensure the reliability of the smart contract by completing the assessment of their system's architecture and smart contract codebase.

Auditing approach and Methodologies applied

In this audit, I consider the following crucial features of the code.

- Whether the implementation of ERC 20 standards.
- Whether the code is secure.
- Whether the code meets the best coding practices.
- **Whether the code meets the SWC Registry issue.**

The audit has been performed according to the following procedure:

• **Manual audit**

1. Inspecting the code line by line and revert the initial algorithms of the protocol and then
compare them with the specification
2. Manually analyzing the code for security vulnerabilities.
3. Assessing the overall project structure, complexity & quality.
4. Checking SWC Registry issues in the code.
5. Unit testing by writing custom unit testing for each function.
6. Checking whether all the libraries used in the code of the latest version.
7. Analysis of security on-chain data.
8. Analysis of the failure preparations to check how the smart contract performs in case of bugs and vulnerability.

• **Automated analysis**

1. Scanning the project's code base with Mythril, Slither, Echidna , Manticore , SmartCheck
2. Manually verifying (reject or confirm) all the issues found by tools.
3. Performing Unit testing.
4. Manual Security Testing (SWC-Registry, Overflow)
5. Running the tests and checking their coverage.

Report: All the gathered information is described in this report.

Audit details

Project Name: BUMO
Token symbol: BU (BUMO)

Language: Solidity
Platform and tools: Remix, VScode, securify and other tools mentioned in the automated analysis section.

Audit Goals

The focus of this audit was to verify whether the smart contract is secure, resilient, and working focus includes.

- Correctness.
- Section of code with high complexity.
- Readability.
- Quantity and quality of test coverage.

Security

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

High severity issues
Issues mentioned here are critical to smart contract performance and functionality and should be fixed before moving to mainnet.

Medium severity issues
This could potentially bring the problem in the future and should be fixed.

Low severity issues
These are minor details and warnings that can remain unfixed but would be better if it got fixed in the future.

No. of issue per severity

Severity	High	Medium	Low
Open	**0**	**0**	**2**

Manual audit

Following are the report from our manual analysis

SWC Registry test

We have tested some known SWC registry issues. Out of all tests only SWC 102 and 103. Both are low priority. We have about it above already.

Serial No.	Description	Comments
SWC-132:	Unexpected Ether balance	Pass: Avoided strict equality checks for the Ether balance in a contract
SWC-131:	Presence of unused variables	Pass: No unused variables
SWC-128:	DoS With Block Gas Limit	Pass: Properly handled
SWC-122:	Lack of Proper Signature Verification	Pass
SWC-120:	Weak Sources of Randomness from Chain Attributes	Pass: No random value used insufficiently

SWC-115:	Authorization through tx.origin	Pass: No tx.origin found
SWC-114:	Transaction Order Dependence	Pass
SWC-113:	DoS with Failed Call	Pas: No failed call
SWC-112	Delegatecall to Untrusted Callee	Pass
SWC-111:	Use of Deprecated Solidity Functions	Pass : No deprecated function used
SWC-108	State Variable Default Visibility	Pass: Explicitly defined visibility for all state variables
SWC-107:	Reentrancy	Pass: Properly used
SWC-106:	Unprotected SELF-DESTRUCT Instruction	Pass: Not found any such vulnerability
SWC-104:	Unchecked Call Return Value	Pass: Not found any such vulnerability
SWC-103	Floating Pragma	Pass
SWC-102:	Outdated Compiler Version	**Found:** Latest version is Version 0.7.4. In code 0.7.2 is used
SWC-101	Integer Overflow and Underflow	Found:: safe math is used

High severity issues

No High Severity Issue found.

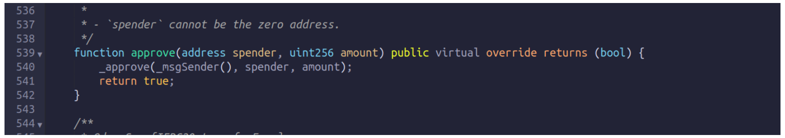
Medium severity issues

No Medium Severity Issue found.

There were 2 low severity issues found.

1. Using the approve function of the ERC-20 token standard [Line 539-542]

The approve function of ERC-20 is vulnerable. Using a front-running attack one can spend approved tokens before the change of allowance value.



```
536      *
537      * - `spender` cannot be the zero address.
538      */
539     function approve(address spender, uint256 amount) public virtual override returns (bool) {
540         _approve(_msgSender(), spender, amount);
541         return true;
542     }
543
544     /**
```

To prevent attack vectors described above, clients should make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. Though the contract itself shouldn't enforce it, to allow backward compatibility with contracts deployed before.

Detailed reading around it can be found at EIP 20

2. Prefer external to public visibility level [line 220-222 , 248-252, 469-471, 477-479, 501-503, 508-510, 520-523, 528-530, 539-542, 557-561, 575-578, 594-597, 604-609, 622-629, 636-639] Link

```
function owner() public view returns (address) {
        return _owner;
}
```

A function with a **public** visibility modifier that is not called internally. Changing the visibility level to **external** increases code readability. Moreover, in many cases, functions with **external** visibility modifiers spend less gas compared to functions with **public** visibility modifiers.

Recommendations: Use the **external** visibility modifier for functions never called from the contract via internal call.

Note: Exact same issue was found while using automated testing by smartcheck.

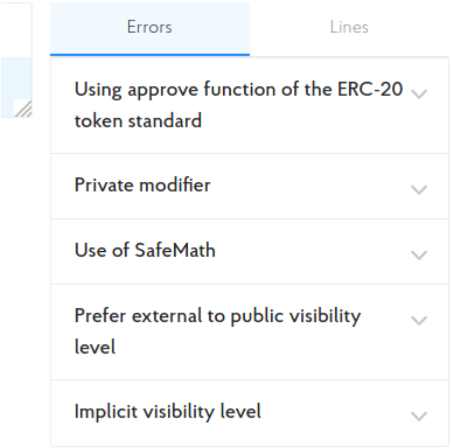
Automated test :

We have used multiple automated testing frameworks. This makes code more secure common attacks. The results are below.

BU-ERC20 Contract Audit - Ankit

SmartCheck automatically checks Smart Contracts for vulnerabilities and bad practices. Automated tests have been conducted and got the following report. A total of five errors were found. Out of five, two have already been covered above.

https://tool.smartdec.net/scan/81bee80fdc7d49b6ba2ac2baff25fdf9



Using approve function of the ERC-20 token standard

Private modifier

Use of SafeMath

Prefer external to public visibility level

Implicit visibility level

Error 1: Using approve function of the ERC-20 token standard :
Already discussed in the manual testing section.

Error 2: Private modifier

8 times this error has been reported. [Line 448, 441, 275, 443, 449, 447, 445, 205]
We need to be careful while using a private modifier. The private modifier does not make a variable invisible. Miners have access to all contracts' code and data. Developers must account for the lack of privacy in Ethereum. Reference: Link

Error 3:Use of SafeMath

Smart Check doesn't recommend using SafeMath library for all arithmetic operations. According to them, It's good practice to use explicit checks where it is really needed and to avoid extra checks where overflow/underflow is impossible.

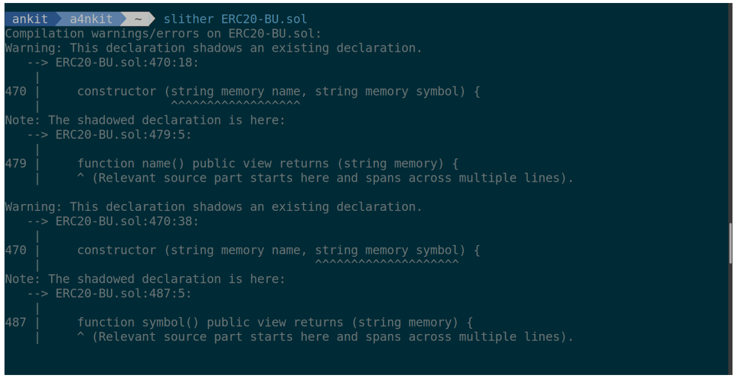
Error 4: Prefer external to public visibility level :
Already discussed in the manual testing section.

Error 5: Implicit visibility level

The default function visibility level in contracts is ***public***, in interfaces - ***external***, state variable default visibility level is ***internal***. In contracts, the fallback function can be ***external*** or ***public***. In interfaces, all the functions should be declared as ***external***. Explicitly define function visibility to prevent confusion.

Slither:

Slither is a Solidity static analysis framework which runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses. We got a report with a few warnings and errors.



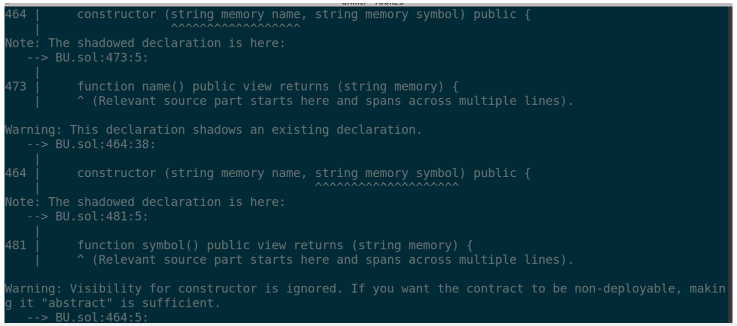
```
 ankit  a4nkit  ~  slither ERC20-BU.sol
Compilation warnings/errors on ERC20-BU.sol:
Warning: This declaration shadows an existing declaration.
   --> ERC20-BU.sol:470:18:
    |
470 |     constructor (string memory name, string memory symbol) {
    |                  ^^^^^^^^^^^^^^^^^^
Note: The shadowed declaration is here:
   --> ERC20-BU.sol:479:5:
    |
479 |     function name() public view returns (string memory) {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning: This declaration shadows an existing declaration.
   --> ERC20-BU.sol:470:38:
    |
470 |     constructor (string memory name, string memory symbol) {
    |                                      ^^^^^^^^^^^^^^^^^^^^
Note: The shadowed declaration is here:
   --> ERC20-BU.sol:487:5:
    |
487 |     function symbol() public view returns (string memory) {
    |     ^ (Relevant source part starts here and spans across multiple lines).
```

All the warnings coming there we have discussed in manual test methodology.

Manticore:

Manticore is a symbolic execution tool for the analysis of smart contracts and binaries. It executes a program with symbolic inputs and explores all the possible states it can reach. It also detects crashes and other failure cases in binaries and smart contracts.

Manticore results throw some warnings which are similar to Slither warning.

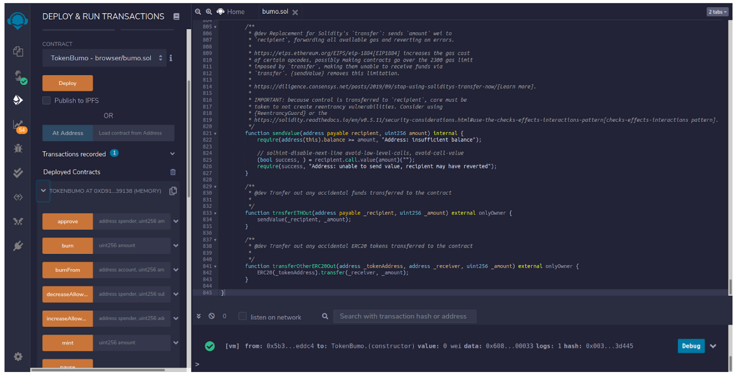


```
464 |     constructor (string memory name, string memory symbol) public {
    |                  ^^^^^^^^^^^^^^^^^^
Note: The shadowed declaration is here:
   --> BU.sol:473:5:
    |
473 |     function name() public view returns (string memory) {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning: This declaration shadows an existing declaration.
   --> BU.sol:464:38:
    |
464 |     constructor (string memory name, string memory symbol) public {
    |                                      ^^^^^^^^^^^^^^^^^^^^
Note: The shadowed declaration is here:
   --> BU.sol:481:5:
    |
481 |     function symbol() public view returns (string memory) {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, makin
g it "abstract" is sufficient.
   --> BU.sol:464:5:
```

declaration.

Remix IDE

Remix was able to compile code perfectly and was behaving according to the required property. Attaching the screenshot.



DEPLOY & RUN TRANSACTIONS
CONTRACT
TokenBumo - browser/bumo.sol
Deploy
□ Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 1
Deployed Contracts
TOKENBUMO AT 0XD91...39138 (MEMORY)
approve
address spender, uint256 am
burn
uint256 amount
burnFrom
address account, uint256 am
decreaseAllow...
address spender, uint256 sut
increaseAllow...
address spender, uint256 adc
mint
uint256 amount
pause
Home
bumo.sol
2 tabs
□ listen on network
Search with transaction hash or address
Debug

```
    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Tranfer out any accidental funds transferred to the contract
     *
     */
    function trnsferETHOut(address payable _recipient, uint256 _amount) external onlyOwner {
        sendValue(_recipient, _amount);
    }

    /**
     * @dev Tranfer out any accidental ERC20 tokens transferred to the contract
     *
     */
    function transferOtherERC20Out(address _tokenAddress, address _receiver, uint256 _amount) external onlyOwner {
        ERC20(_tokenAddress).transfer(_receiver, _amount);
    }

}
```

```
[vm] from: 0x5b3...eddc4 to: TokenBumo.(constructor) value: 0 wei data: 0x608...00033 logs: 1 hash: 0x003...3d445
>
```

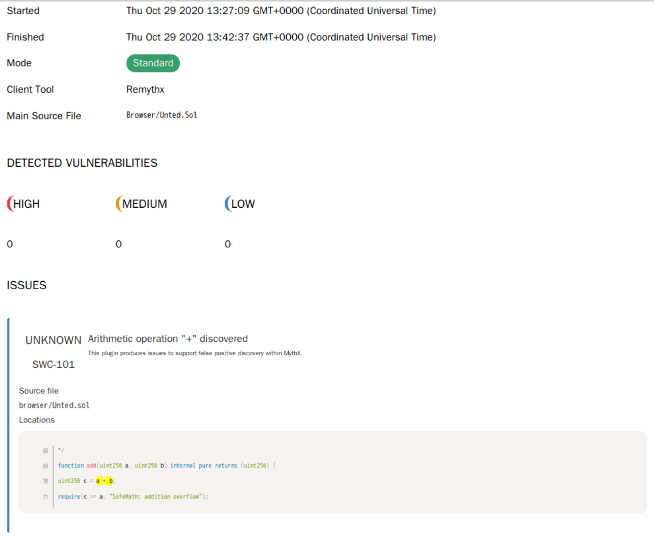
There was no error/warning at Remix IDE.

Mythx:

MythX is a security analysis tool and API that performs static analysis, dynamic analysis, symbolic execution, and fuzzing on Ethereum smart contracts. MythX checks for and reports on the common security vulnerabilities in open industry-standard SWC Registry.

There are many contracts within the whole file. I have separately put them for analysis. Below are the reports generated for each contract separately.

1. Contract Context: No vulnerability found. There were a few warnings around SWC-101.



Started	Thu Oct 29 2020 13:27:09 GMT+0000 (Coordinated Universal Time)
Finished	Thu Oct 29 2020 13:42:37 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Remythx
Main Source File	Browser/Unted.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

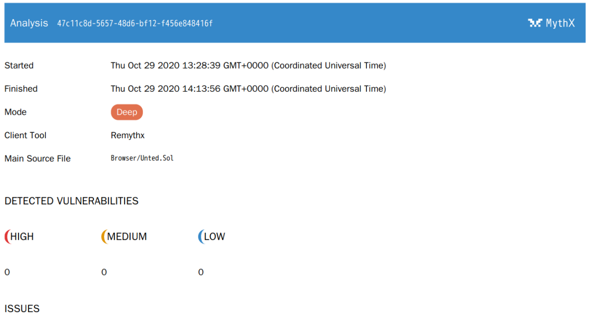
Source file

browser/Unted.sol

Locations

```
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
uint256 c = a + b;
require(c >= a, "SafeMath: addition overflow");
```

2. Contract TokenBumo: No vulnerability found.



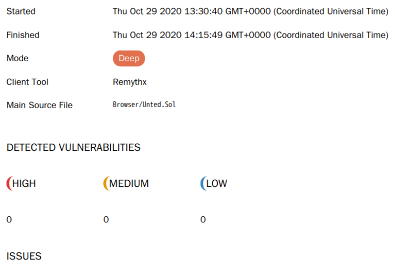
Analysis 47c11c8d-5657-48d6-bf12-f456e848416f

Started	Thu Oct 29 2020 13:28:39 GMT+0000 (Coordinated Universal Time)
Finished	Thu Oct 29 2020 14:13:56 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	Browser/Unted.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES



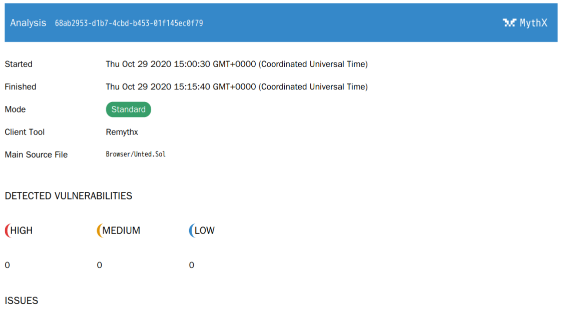
Started	Thu Oct 29 2020 13:30:40 GMT+0000 (Coordinated Universal Time)
Finished	Thu Oct 29 2020 14:15:49 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	Browser/Unted.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES

4. Contract SafeMath: No vulnerability found.



Analysis 68ab2953-d1b7-4cbd-b453-01f145ec0f79

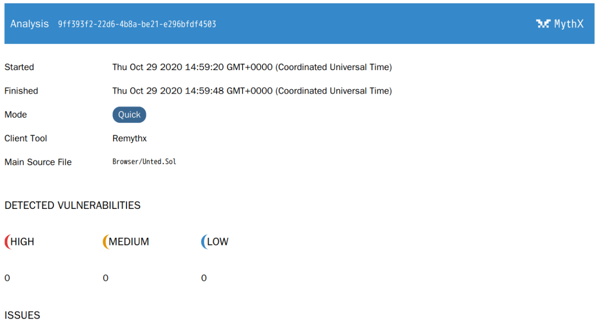
Started	Thu Oct 29 2020 15:00:30 GMT+0000 (Coordinated Universal Time)
Finished	Thu Oct 29 2020 15:15:40 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Remythx
Main Source File	Browser/Unted.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES

5. Contract Pausable: No vulnerability found.



Analysis 9ff393f2-22d6-4b8a-be21-e296bfdf4503

Started	Thu Oct 29 2020 14:59:20 GMT+0000 (Coordinated Universal Time)
Finished	Thu Oct 29 2020 14:59:48 GMT+0000 (Coordinated Universal Time)
Mode	Quick
Client Tool	Remythx
Main Source File	Browser/Unted.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code. Besides, a security audit, please don't consider this report as investment advice.

Summary

The use of smart contracts is simple and the code is relatively small. Altogether the code is written and demonstrates effective use of abstraction, separation of concern, and modularity. But there are a few issues/vulnerabilities to be tackled at various security levels, it is recommended to fix them before deploying the contract on the main network. Given the subjective nature of some assessments, it will be up to the Bumo team to decide whether any changes should be made.

About the Auditor: Ankit Raj

Ankit is a technology expert with many years of expert experience building, managing, and automating systems at scale for blockchain, distributed systems, and storage projects.

Started a career as a developer with Red Hat where he developed the DHT module for GlusterFS. GlusterFS is being used by Facebook and financial institutions for clustering large chunks of data, images, and videos. Later he got a grant from Ethereum Foundation to work on Solidity language. There he wrote solidity code as well as maintained docs for the solidity programming language which is used by the developer across the globe. Then he worked with various crypto startups like Ocean Protocol, Coss exchange leading a full-stack development team. At Ocean, he built the protocol for safe data transfer. Ankit also founded Blockvidhya, a document verification startup service relying on the blockchain, which was incubated at IIT Mandi and part of YC startup school. He was also part of Entrepreneur First in Singapore where he was leveraging his blockchain and Open Finance skills.

Currently, he is actively doing contributions in Ethereum, Polkadot & Near Protocol ecosystem, and doing research around Open Finance.

During his free time, he participates in hackathons. He has won more than 5 international hackathons across the globe organized by Ethglobal, Matic, Near Protocol and Barclays labs. He also actively writes around open finance and DeFi protocols.
Link: Linkedin, Twitter, Medium, website

Published by Google Drive – Report Abuse