Aztec-Kingdom

ATC Token

Smart Contract Audit
Final Report

May 28, 2022

Introduction

1. About Aztec-Kindom

According to legend, the ancestors of the Aztecs came from a place called the Aztecs in the north. and they traveled south to Lake Texcoco in Anahuac valley according to the instructions of the Sun God Huizilopochetri. When they came to the island in the middle of the lake, they saw an eagle with a snake perched on a cactus. This phenomenon told them that they should build a city here. Therefore, in 1325, the Aztecs built Tenochtitlán, a huge artificial island in this place.

Visit https://aztec-kingdom.com/ to know more about it.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provides professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-ups with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The Aztec team has provided no document for the purpose of the audit.

Audit Process & Methodology

ImmuneBytes 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 and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet 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 version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Aztec-Kingdom
- Contracts Name: ATCTOKEN
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Address for audit: 0x45B8ddD1dE34eD2C41Dd529b6FEF4052bF335c97
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

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

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

Security Level Reference

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

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	2	3
Closed	-	-	-

High Severity Issues

No issues were found.

Medium Severity Issues

1. public functions that are never called by the contract should be declared external to save gas.

   Functions:
   ERC20.totalSupply() (ATCToken.sol#130-132)
   ERC20.allowance(address,address)
   ERC20.approve(address,uint256) (ATCToken.sol#150-153)
   ERC20.transferFrom(address,address,uint256)
   ERC20.increaseAllowance(address,uint256)
   ERC20.decreaseAllowance(address,uint256)
   MinterRole.addMinter(address)
   MinterRole.renounceMinter()
   ERC20Mintable.mint
   ATCToken.burn(uint256)

   Recommendation
   Use the external attribute for functions never called from the contract.

2. Every require need error message for revert statement, require statements are used however the revert strings are not provided in these require statements. It is a standard practice to always provide revert reasons in require statements.

   Recommendation
   Consider adding revert strings in all the require statements.

Low Severity Issues

1. In the ATC Token contract there are several numbers with too many digits, making them hard to read and error-prone. We recommend replacing them with their scientific notation or ether suffix equivalents.
   In line number 290: *ERC20Mintable(1000000000 * 1e18)*

2. Remove comment from ATC Token constructor:
   Line number 291: *//allow[msg.sender] = true;*

3. The contract should not use floating pragma, e.g. (^0.8.0 or >=0.6.0 *0.8.0), which allows a range of compiler versions. It is important to lock the pragma (for example, not using ^ in pragma solidity 0.8.0) to prevent contracts from being accidentally deployed using an older compiler with unfixed bugs.

Recommendation / Informational

Solidity contracts can have a special form of comments that form the basis of the Ethereum Natural Language Specification Format, also known as Natspec. Natspec is developed and promoted by Ethereum itself.

The Natspec commenting format follows the Doxygen notation style:

1. Single line comments, using three slashes ///
2. Multi-line comments, using double asterisk block /** ... */
3. Comments are included above function, contract, library, interface, function, and constructor.

Automated Audit

```
# Check ATCToken

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] ATCToken has increaseAllowance(address,uint256)
```

Mythril:

```
The analysis was completed successfully. No issues were detected.
```

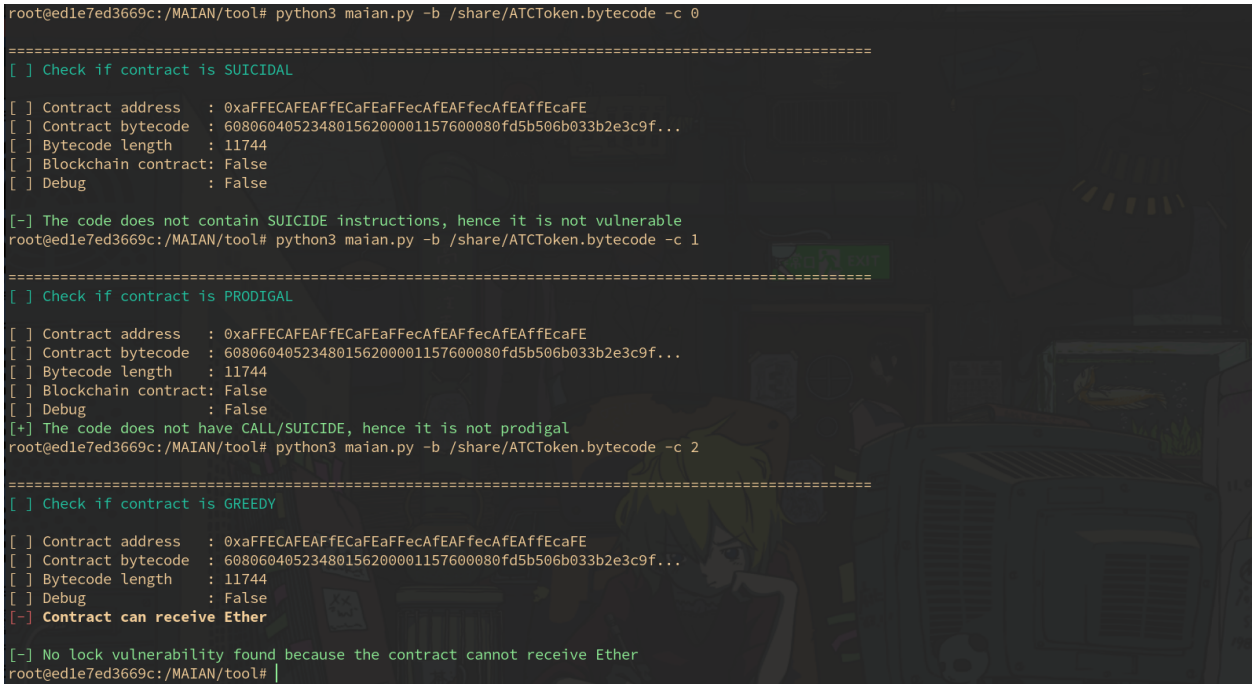
Mythx:

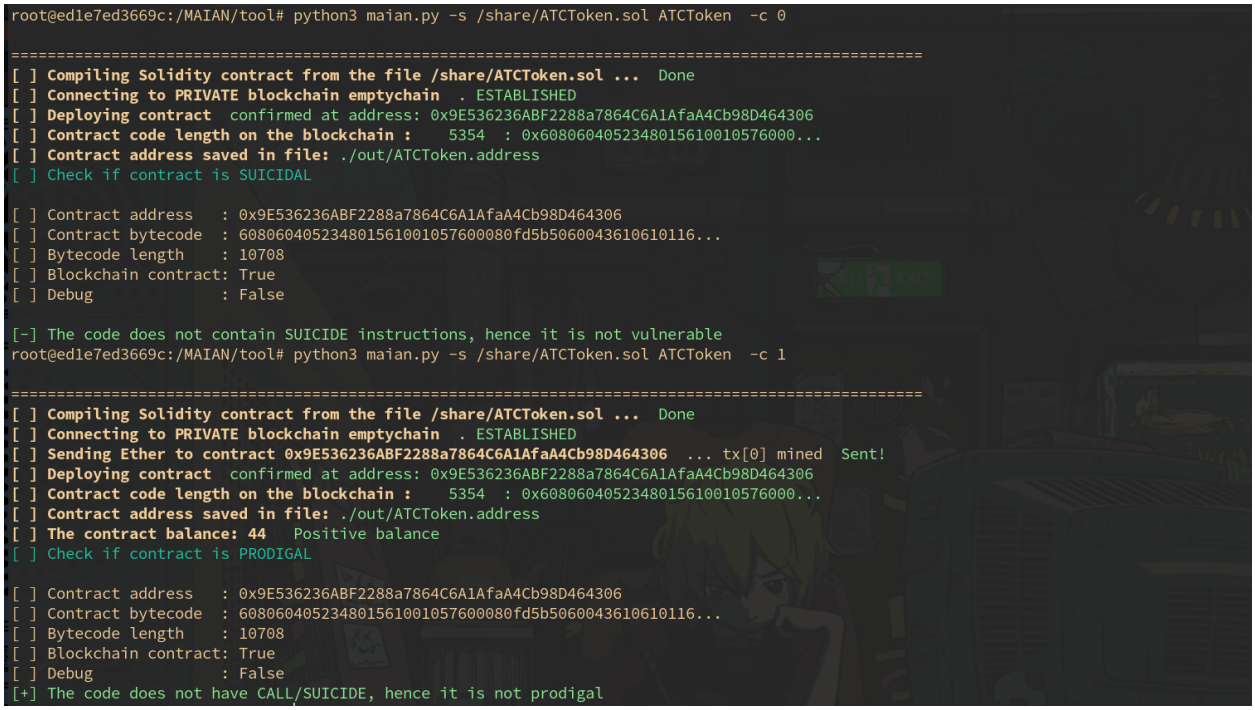
```
Report for contracts/ATCToken.sol
https://dashboard.mythx.io/#/console/analyses/7b8addad-029a-4b1d-8152-7060a1be0ea3
```

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Maian bytecode analysis:





```
root@ed1e7ed3669c:/MAIAN/tool# python3 maian.py -s /share/ATCToken.sol ATCToken  -c 2

================================================================================================
[ ] Compiling Solidity contract from the file /share/ATCToken.sol ...  Done
[ ] Connecting to PRIVATE blockchain emptychain  . ESTABLISHED
[ ] Deploying contract  confirmed at address: 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract code length on the blockchain :    5354  : 0x6080604052348015610010576000...
[ ] Contract address saved in file: ./out/ATCToken.address
[ ] Check if contract is GREEDY

[ ] Contract address   : 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract bytecode  : 6080604052348015610010576000080fd5b5060043610610116...
[ ] Bytecode length    : 10708
[ ] Debug              : False
[-] Contract can receive Ether

[-] No lock vulnerability found because the contract cannot receive Ether
```

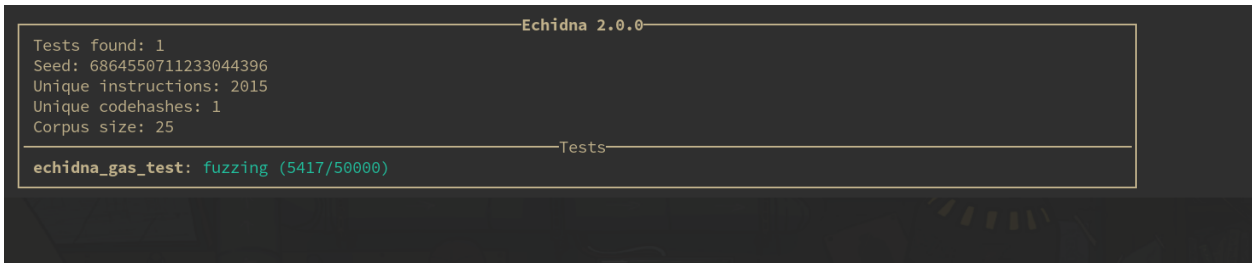
Code Coverage:

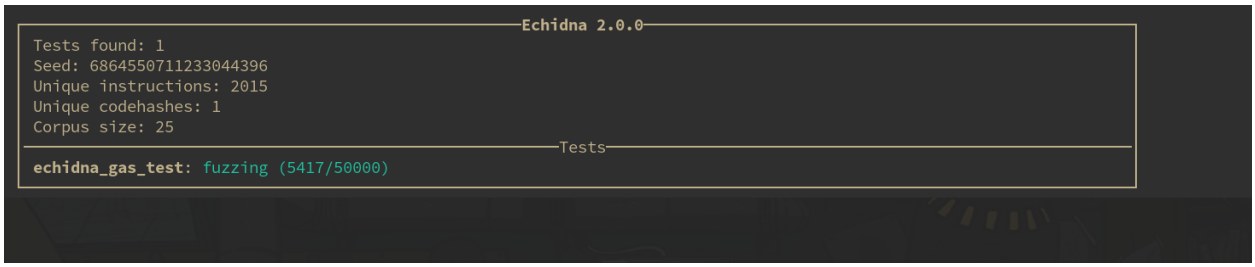
```
---------------|----------|----------|----------|----------|----------------|
File           |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
---------------|----------|----------|----------|----------|----------------|
 contracts/    |        0 |        0 |        0 |        0 |                |
  ATCToken.sol |        0 |        0 |        0 |        0 |... 302,303,308 |
---------------|----------|----------|----------|----------|----------------|
All files      |        0 |        0 |        0 |        0 |                |
---------------|----------|----------|----------|----------|----------------|
```

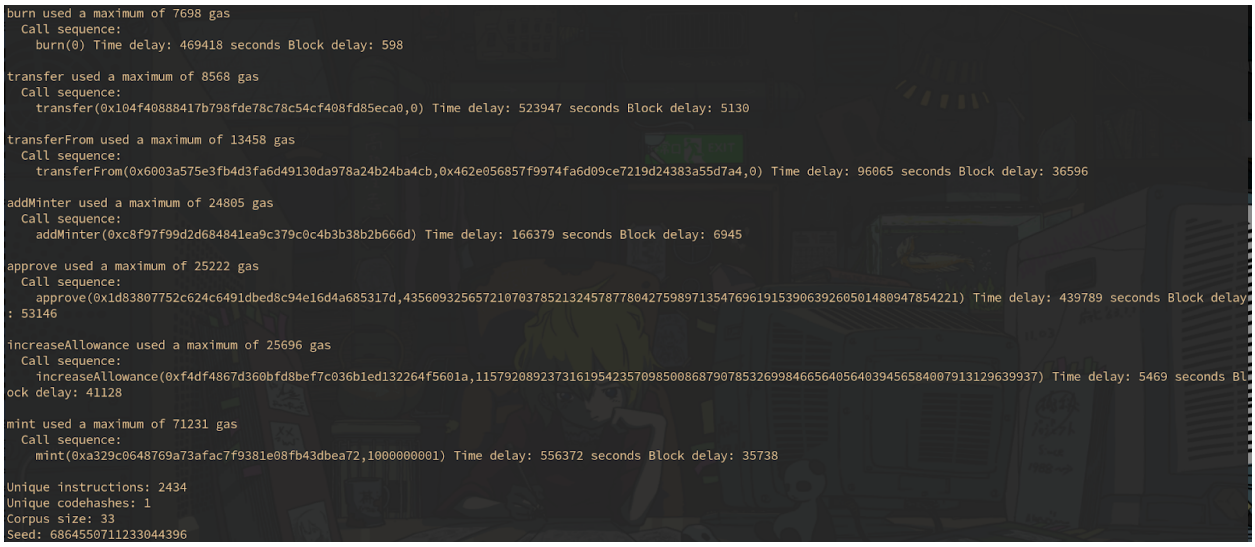
Automated Scanner:

```
Compiled with solc
Number of lines: 310 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 7 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 10
Number of informational issues: 7
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+----------+-------------+-------+--------------------+--------------+----------+
|   Name   | # functions |  ERCS |     ERC20 info     | Complex code | Features |
+----------+-------------+-------+--------------------+--------------+----------+
|  Roles   |      3      |       |                    |      No      |          |
| SafeMath |      5      |       |                    |      No      |          |
| ATCToken |      32     | ERC20 |     ∞ Minting      |      No      |          |
|          |             |       | Approve Race Cond. |              |          |
|          |             |       |                    |              |          |
+----------+-------------+-------+--------------------+--------------+----------+
ATCToken.sol analyzed (7 contracts)
```

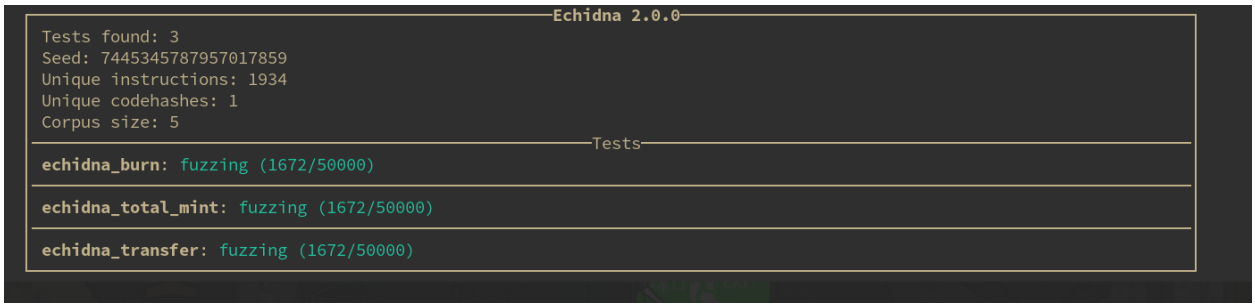
Gas test:



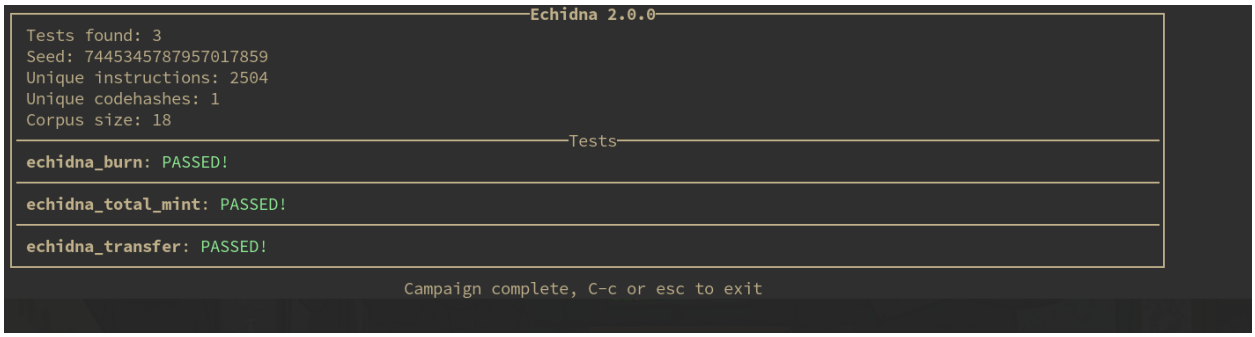




Fuzz test:



This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.



Concluding Remarks

While conducting the audits of the Aztec-Kingdom smart contract, it was observed that the contracts contain Medium and Low severity issues.

Our auditors suggest that the Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Aztec-Kingdom platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes