Smart Electrum

Token

Smart Contract Audit
Final Report

IMMUNEBYTES

Audits

SMARTELECTRUM

October 07, 2021

Introduction

1. About Smart Electrum

Smart Electrum would like to create a wide and unique ecosystem that allows our users to freely use cryptocurrencies in their daily life.
Smart Electrum Ecosystem will allow users to easily exchange cryptocurrencies, together with a decentralized market where products and services. Smart Electrum is aimed at businesses and individuals involved in virtually any type of transaction.

Hundreds of millions of people have attained modern energy access over the last two decades through distribution networks.
This means that more people on Earth than ever before are now connected to ever-growing and interconnected electricity networks. This creates an enormous appetite for innovative new peer-to-peer (P2P) energy transaction platforms.

Smart Electrum would like to change the market by introducing elements of real value to it so that each token holder can experience the feeling of having something more — a digital record, a special solution that will be able to change the world.

Visit https://smartelectrum.io/ to learn more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide 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, dydx.

The team has been able to secure 75+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up 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 Smart Electrum team has provided the following doc for the purpose of audit:

1. https://smartelectrum.io/assets/Whitepaper.pdf

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: Smart Electrum
- Contracts Name: SelectToken.sol, Ownable.sol
- Languages: Solidity(Smart contract)
- Github link for initial audit: https://gist.github.com/WojcikMM/1f87777a8e5027809ee13e6029c9b3ee
- Platforms and Tools: Remix IDE, Truffle, Ganache, Solhint, 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 References

Every issue in this report was 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	-	-	-
Closed	-	-	5

Contract SelectToken.sol

High Severity Issues

No issues were found.

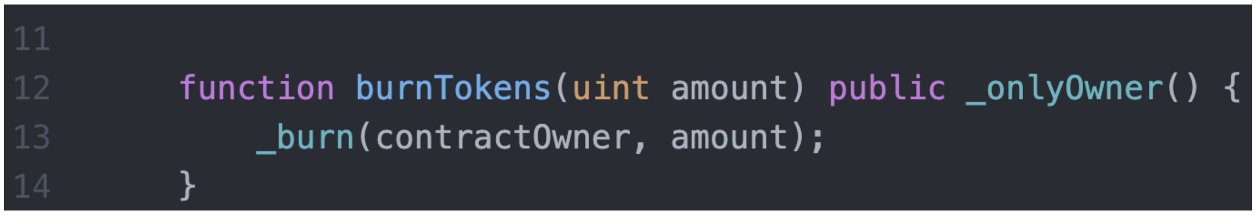
Medium severity issues

No issues were found.

Low Severity Issues

1. **External Visibility should be preferred**

   Explanation:
   Functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
   This will effectively result in Gas Optimization as well.



```
11
12      function burnTokens(uint amount) public _onlyOwner() {
13          _burn(contractOwner, amount);
14      }
```

   Therefore, the following function must be marked as external within the contract:
   - **burnTokens**() function at Line 12

   Recommendation:
   If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

2. **NatSpec Annotations must be included**

   Explanation:
   The smart contracts do not include the NatSpec annotations adequately.

   Recommendation:
   Cover by NatSpec all Contract methods.

Contract Ownable.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

1. **Internal function never used throughout the contract**
   Line no - 17-20

   Explanation:
   The **Ownable** contract includes a **isOwner**() function in the contract which has been assigned **internal** visibility.

   However, the function is never called throughout the contract. Moreover, since the function is marked internal, it cannot be accessed from outside the contract as well.
   This makes the function unusable and reduces its significance in the contract.

   Recommendation:
   It is recommended to either assign an adequate visibility specifier to the above-mentioned function or ensure that the function is used somewhere in the contract.
   However, if the function is not supposed to be used anywhere in the given contracts, the function can be removed.

2. **Ownable Contract doesn't include ownership transfer features**

   Explanation:
   The Ownable contract doesn't include any function that allows the transfer of Ownership of the protocol. Keeping in mind the immutable nature of smart contracts, it's quite imperative to implement such functions to ensure that the contract doesn't result in unwanted behavior in the future.

   Recommendation:
   It's strongly recommended to use Openzeppelin's Ownable contracts to avoid such scenarios.

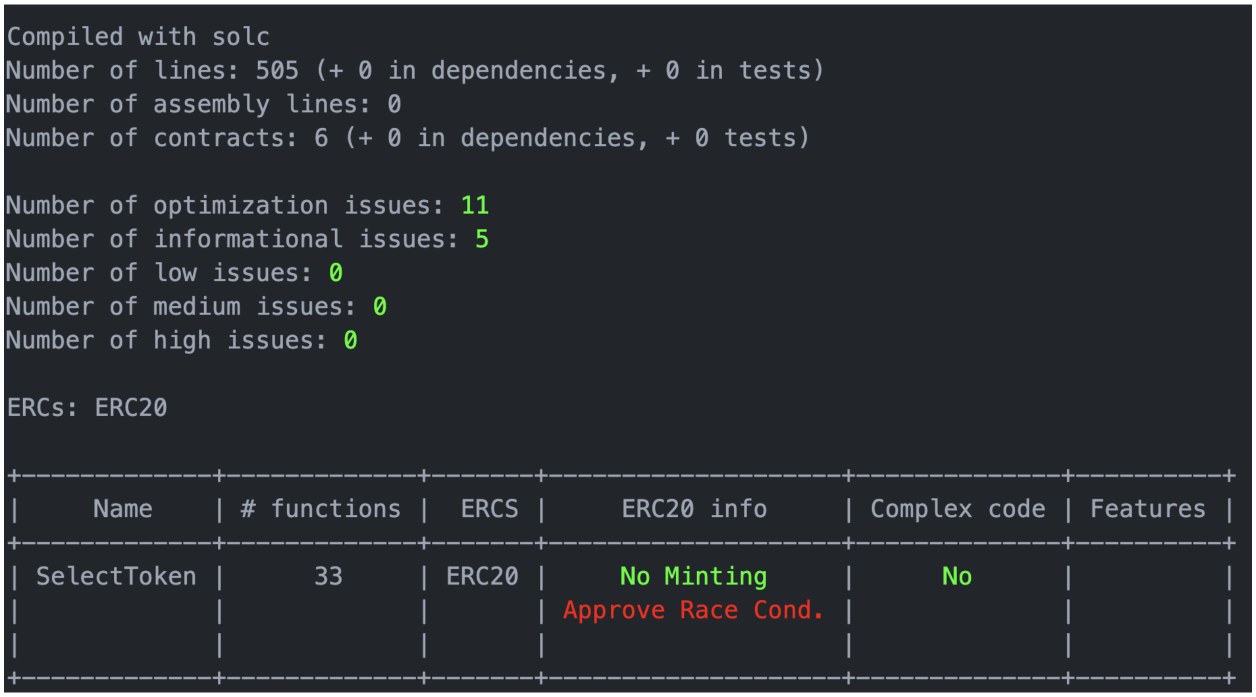
3. **NatSpec Annotations must be included**

   Explanation:
   The smart contracts do not include the NatSpec annotations adequately.

   Recommendation:
   Cover by NatSpec all Contract methods.

Automated Test Results

1. SelectToken.sol



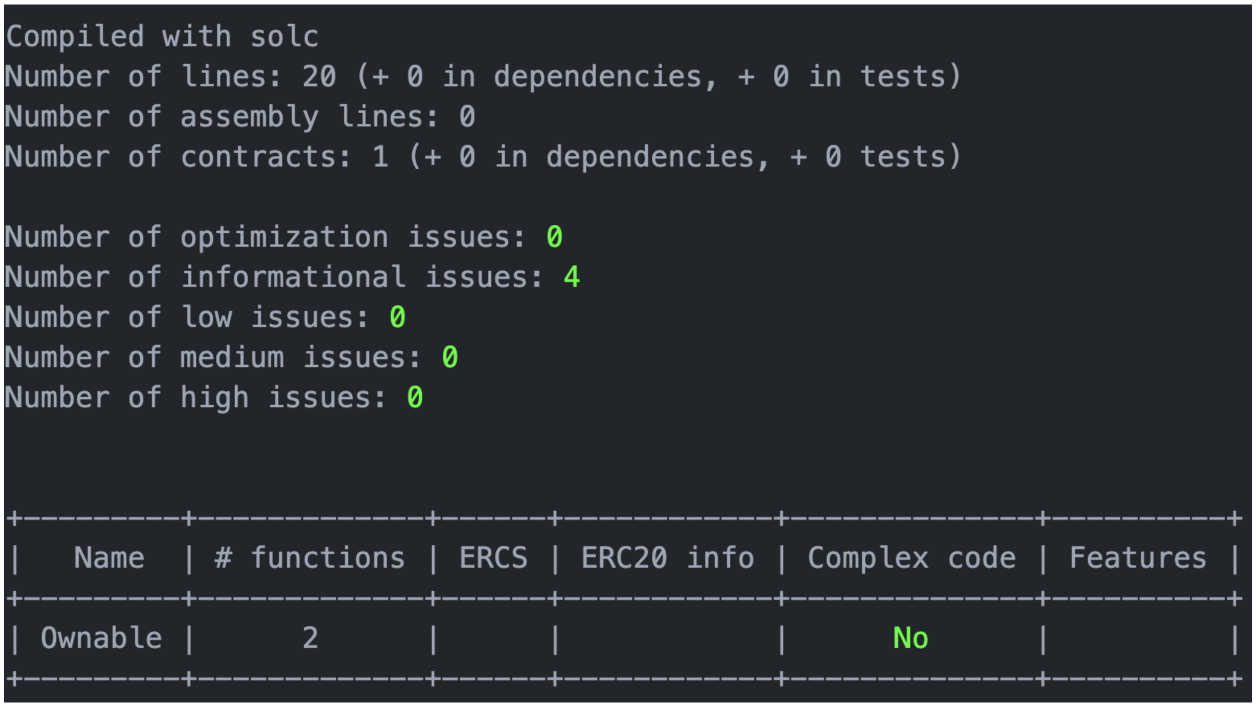
```
Compiled with solc
Number of lines: 505 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 11
Number of informational issues: 5
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 |
+-------------+-------------+-------+--------------------+--------------+----------+
| SelectToken |      33     | ERC20 |     No Minting     |      No      |          |
|             |             |       | Approve Race Cond. |              |          |
|             |             |       |                    |              |          |
+-------------+-------------+-------+--------------------+--------------+----------+
```

2. Ownable.sol



```
Compiled with solc
Number of lines: 20 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 1 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 4
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0


+---------+-------------+------+------------+--------------+----------+
|   Name  | # functions | ERCS | ERC20 info | Complex code | Features |
+---------+-------------+------+------------+--------------+----------+
| Ownable |      2      |      |            |      No      |          |
+---------+-------------+------+------------+--------------+----------+
```

Concluding Remarks

While conducting the audits of the Smart Electrum smart contract, it was observed that the contracts contained only Low severity issues with a few areas of recommendations. No High or Medium severity was found.

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

- ***Smart Electrum team has fixed the Low issue based on the auditor's recommendation.***

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 Smart Electrum 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