Horizen - ERC20

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 24th, 2022 - January 28th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/24/2022	Roberto Reigada
0.2	Document Updates	01/28/2022	Roberto Reigada
0.3	Draft Review	01/28/2022	Gabi Urrutia
1.0	Remediation Plan	02/01/2022	Roberto Reigada
1.1	Remediation Plan Review	02/01/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Horizen Labs engaged Halborn to conduct a security audit on their ERC20 smart contract beginning on January 24th, 2022 and ending on January 28th, 2022. The security assessment was scoped to the smart contract provided in the GitHub repository HorizenLabs/grapes-erc20.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn did not find any security risk in this contract.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose

- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.

2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - **8** - HIGH
7 - **6** - MEDIUM
5 - **4** - LOW
3 - **1** - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contract:

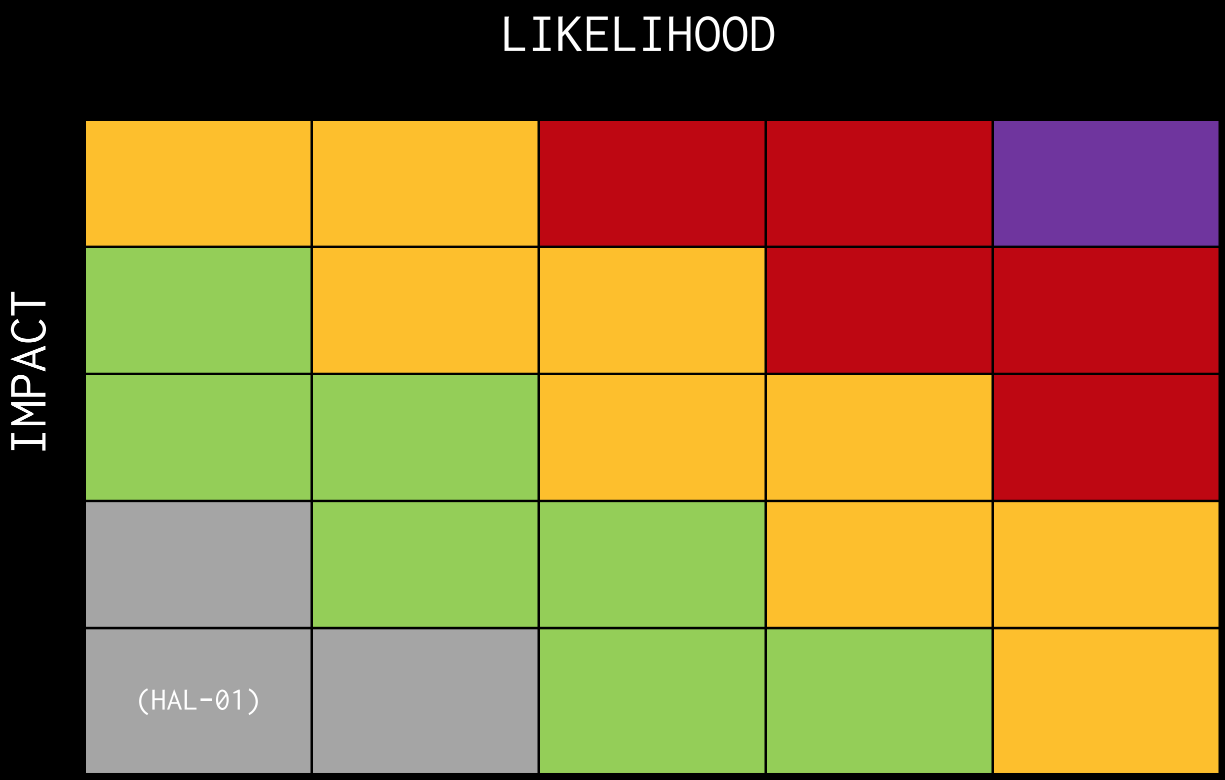
- SimpleToken.sol

Commit ID: 24445588a51ad9580d77b2938626a45bb4d99800

Fixed Commit ID: 12516d06959c5eb289863982ffcbd44f021cfec4

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - UNNECESSARY PAYABLE MODIFIER IN THE CONSTRUCTOR	Informational	SOLVED - 02/01/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNNECESSARY PAYABLE MODIFIER IN THE CONSTRUCTOR - INFORMATIONAL

Description:

The constructor of the contract SimpleToken contains the payable modifier. This modifier can be removed, as this ERC20 token should never receive any Ether. All the Ether sent to this contract would be lost, as the contract lacks a function to withdraw the Ether.

Code Location:

Listing 1: SimpleToken.sol (Lines 13)

```
 9 constructor(
10     string memory name,
11     string memory symbol,
12     uint256 totalSupply_
13 ) payable ERC20(name, symbol) {
14     _mint(msg.sender, totalSupply_);
15 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove the payable modifier from the constructor.

Remediation Plan:

SOLVED: The issue was solved in the commit ID: 12516d06959c5eb289863982ffcbd44f021cfec4

The Horizen Labs team removed the payable modifier from the constructor.

AUTOMATED TESTING

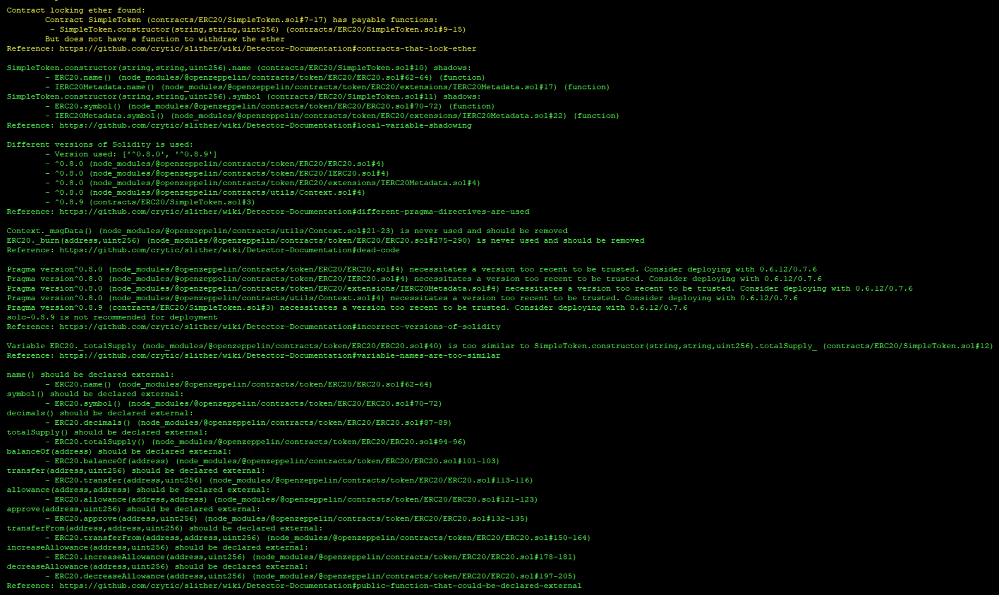
4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

SimpleToken.sol



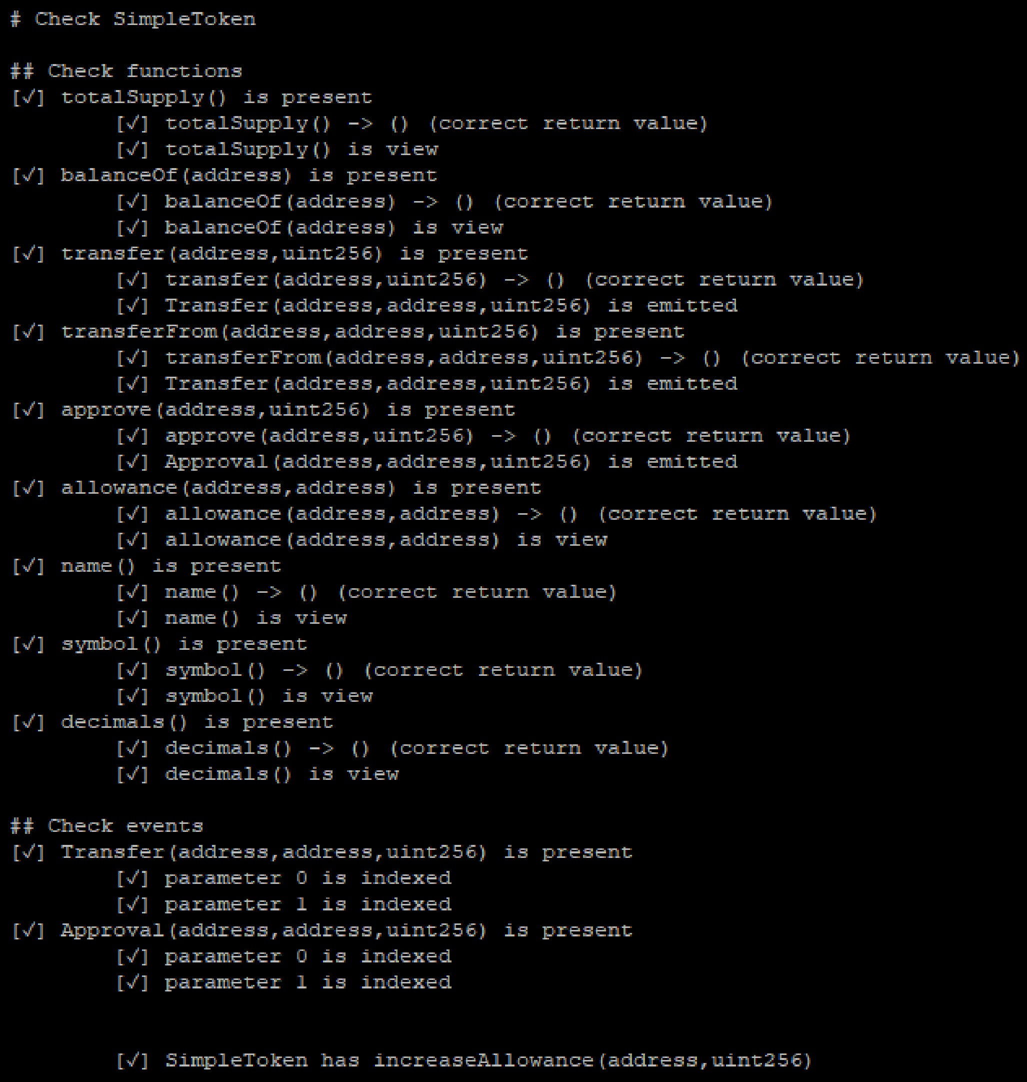
```
Contract locking ether found:
        Contract SimpleToken (contracts/ERC20/SimpleToken.sol#7-17) has payable functions:
         - SimpleToken.constructor(string,string,uint256) (contracts/ERC20/SimpleToken.sol#9-15)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

SimpleToken.constructor(string,string,uint256).name (contracts/ERC20/SimpleToken.sol#10) shadows:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64) (function)
        - IERC20Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#17) (function)
SimpleToken.constructor(string,string,uint256).symbol (contracts/ERC20/SimpleToken.sol#11) shadows:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72) (function)
        - IERC20Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#22) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Different versions of Solidity is used:
        - Version used: ['^0.8.0', '^0.8.9']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.9 (contracts/ERC20/SimpleToken.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ERC20._burn(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#275-290) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.9 (contracts/ERC20/SimpleToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable ERC20._totalSupply (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#40) is too similar to SimpleToken.constructor(string,string,uint256).totalSupply_ (contracts/ERC20/SimpleToken.sol#12)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#87-89)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#94-96)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#113-116)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#121-123)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#132-135)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#150-164)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#178-181)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#197-205)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- No major issues found by Slither. Although, Slither correctly flagged that because of the payable modifier in the constructor, Ether would get locked in the contract forever if it was sent during the deployment.

ERC20 checks:

SimpleToken.sol

The SimpleToken contract is a standard ERC20 token. The whole totalSupply of this token will be minted during the deployment to the deployer address. The totalSupply amount will be based on the constructor parameter totalSupply_:



```
# Check SimpleToken

## 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


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

As we can see above, after the deployment, the token cannot be minted or burnt. Tokens can only be transferred between different accounts. The totalSupply will never change.

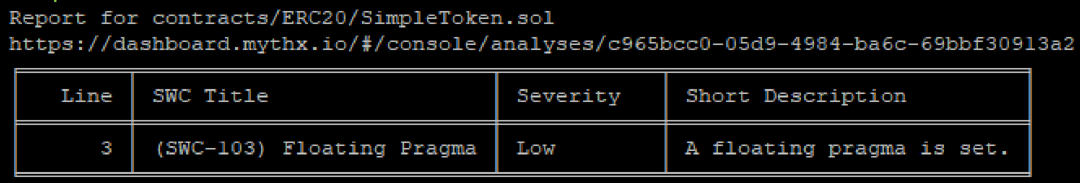
4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

SimpleToken.sol



```
Report for contracts/ERC20/SimpleToken.sol
https://dashboard.mythx.io/#/console/analyses/c965bcc0-05d9-4984-ba6c-69bbf30913a2
╒════════╤════════════════════════════════════╤════════════╤═══════════════════════════════════╕
│   Line │ SWC Title                          │ Severity   │ Short Description                 │
╞════════╪════════════════════════════════════╪════════════╪═══════════════════════════════════╡
│      3 │ (SWC-103) Floating Pragma          │ Low        │ A floating pragma is set.         │
╘════════╧════════════════════════════════════╧════════════╧═══════════════════════════════════╛
```

- The floating pragma flagged by MythX is a false positive, as the pragma is set in the `truffle-config.js` file to the `0.8.10` version.

THANK YOU FOR CHOOSING

// HALBORN