HALBORN

ELITE CYBERSECURITY FOR BLOCKCHAIN ENTERPRISES

POWER TRADE FUEL TOKEN (PTF) SMART CONTRACT SECURITY AUDIT

Report Prepared by: Halborn
Date of Engagement: September 17-22, 2020

Table of Contents

Document Revision History

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	9/21/2020	Gabi Urrutia
0.2	Document Edits	9/22/2020	Steven Walbroehl
1.0	Document DRAFT	9/22/2020	Steven Walbroehl

Contacts

CONTACT	COMPANY	EMAIL	PHONE
STEVEN WALBROEHL	Halborn	Steven.Walbroehl@halborn.com	--
ROB BEHNKE	Halborn	Rob.behnke@halborn.com	--
GABI URRUTIA	Halborn	Gabi.Urrutia@halborn.com	--
NISHIT MAJITHIA	Halborn	Nishit.Majithia@halborn.com	--

1 - Executive Summary

1.1 Introduction

Power Trade engaged Halborn to conduct a security assessment on their Fuel Token smart contract beginning on September 14th, 2020 and ending September 23rd 2020. The security assessment was scoped to the contract FuelToken.sol and an audit of the security risk and implications regarding the changes introduced by the development team at Power Trade prior to its production release shortly following the assessments deadline.

The contract scoped in this assessment is focused to the functions scoped in PTF token paper, primarily mint() and burn(). PTF is a DAO token which governs the treasury that covers PowerTrade traders in the unlikely event of a black swan, where margin collateral is not sufficient to cover open margin positions. The PTF token is a contract similar to the ones used for Compound DAO, which can be extended in the future to meet the functionality described in the token paper.

Overall, the smart contract code is extremely well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted.

The most significant observation made in the security assessment is in regard to an experimental version ABIEncoderV2 that is enabled. It best practice not use experimental features in production, and is recommended to be excluded from the final version of PTF.

Though the outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties related to the FuelToken was performed to achieve objectives and deliverables set in the scope. It is important to remark the use of the best practices for secure smart contract development. Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 Test Approach and Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts 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, purpose, and use of PTF based on Compound Protocol.**
- **Smart Contract manual code read 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.**

- **Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (*MythX*)**
- **Static Analysis of security for scoped contract, and imported functions. (*Slither*)**
- **Testnet deployment (*Truffle*, *Ganache, Infura)***
- **Smart Contract Fuzzing and dynamic state exploitation (*Echidna*)**
- **Symbolic Execution / EVM bytecode security assessment (limited-time)**

1.3 Scope

IN-SCOPE:

Code related to the FuelToken smart contract.
Specific commit of contract: 200947a7b3c283e87caae2dc02587883f46049ee

OUT-OF-SCOPE:

External contracts, External Oracles, other smart contracts in the repository or imported by FuelToken, economic attacks.

2 - Assessment Summary and Findings Overview

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	2

SECURITY ANALYSIS	RISK LEVEL
EXPERIMENTAL FEATURES ENABLED	LOW
STATIC ANALYSIS REPORT – (STRICT EQUALITY FINDING)	LOW
FUNCTION LOGIC TESTING AND ANALYSIS	INFORMATIONAL
ONLYMINTER FUNCTION TESTING	INFORMATIONAL
SECURITY FUZZING RESULTS	INFORMATIONAL
AUTOMATED SECURITY SCAN RESULTS	INFORMATIONAL

3 - Findings and Technical Details

3.1 EXPERIMENTAL FEATURES ENABLED – LOW

Description:

`ABIEncoderV2` is enabled to be able to pass struct type into a function both web3 and another contract. The use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to `bytesNN` types, `bool`, `enum` and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside `abi.encode(...)` as arguments in external function calls or in event data without prior assignment to a local variable. Using `return` does not trigger the bug. The types `bytesNN` and `bool` will result in corrupted data while `enum` might lead to an invalid revert.

Furthermore, arrays with elements shorter than 32 bytes may not be handled correctly even if the base type is an integer type. Encoding such arrays in the way described above can lead to other data in the encoding being overwritten if the number of elements encoded is not a multiple of the number of elements that fit a single slot. If nothing follows the array in the encoding (note that dynamically-sized arrays are always encoded after statically-sized arrays with statically-sized content), or if only a single array is encoded, no other data is overwritten. There are known bugs that are publically released while using this feature. However, the bug only manifests itself when all of the following conditions are met:

- Storage data involving arrays or structs is sent directly to an external function call, to abi.encode or to event data without prior assignment to a local (memory) variable
- There is an array that contains elements with size less than 32 bytes or a struct that has elements that share a storage slot or members of type bytesNN shorter than 32 bytes.

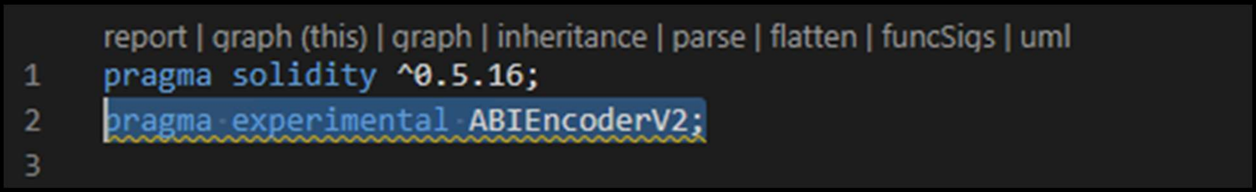
In addition to that, in the following situations, your code is NOT affected:

- All the structs or arrays only use uint256 or int256 types
- If you only use integer types (that may be shorter) and only encode at most one array at a time
- If you only return such data and do not use it in abi.encode, external calls or event data.

Reference: https://blog.ethereum.org/2019/03/26/solidity-optimizer-and-abiencoderv2-bug/

Code Location:

FuelToken.sol Line #2


report | graph (this) | graph | inheritance | parse | flatten | funcSigs | uml

```
1    pragma solidity ^0.5.16;
2    pragma experimental ABIEncoderV2;
3
```

ABIEncoderV2 is enabled to be able to pass struct type into a function both web3 and another contract. Naturally, any bug can have wildly varying consequences depending on the program control flow, but we expect that this is more likely to lead to malfunction than exploitability.

The bug, when triggered, will under certain circumstances send corrupt parameters on method invocations to other contracts.

Recommendation:

When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using uint256)

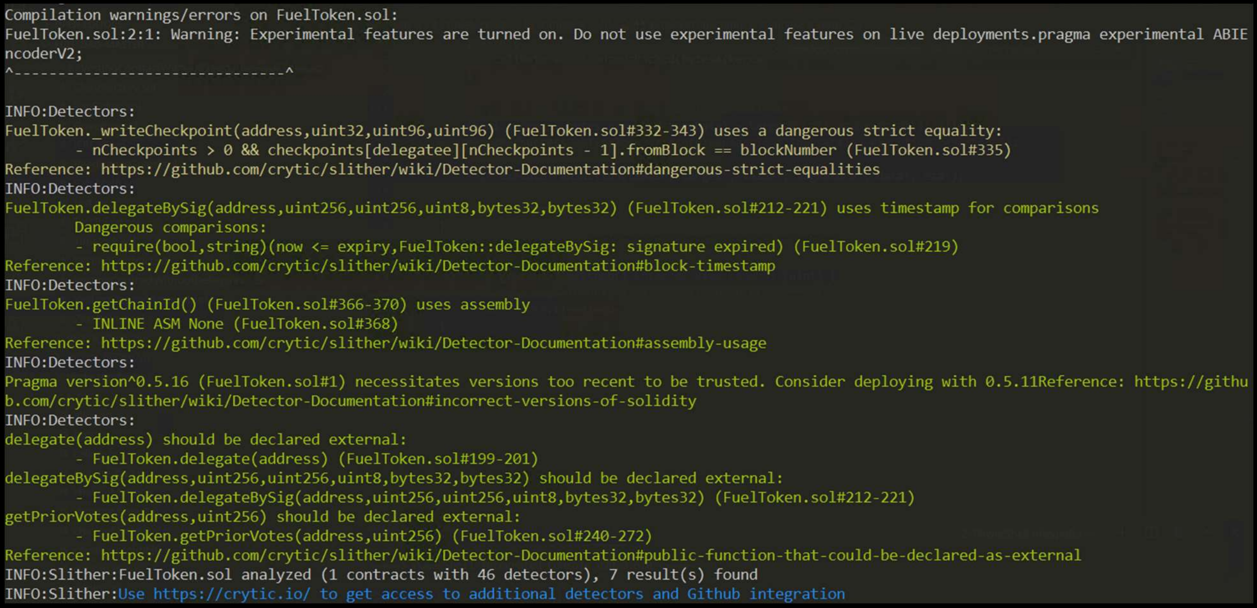
3.2 STATIC ANALYSIS REPORT - LOW

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. 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 FuelToken contract. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

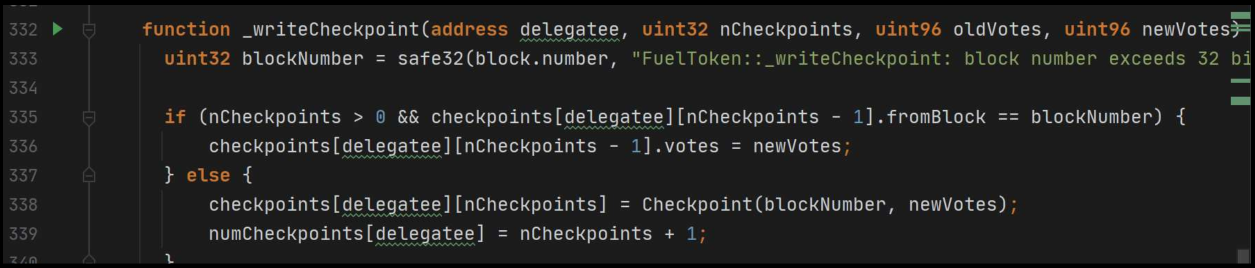
Results:

Slither shows that an experimental feature is turned on. As it is said in 3.1., experimental features should not be used on live deployments.



```
Compilation warnings/errors on FuelToken.sol:
FuelToken.sol:2:1: Warning: Experimental features are turned on. Do not use experimental features on live deployments.pragma experimental ABIE
ncoderV2;
^-------------------------------^

INFO:Detectors:
FuelToken._writeCheckpoint(address,uint32,uint96,uint96) (FuelToken.sol#332-343) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (FuelToken.sol#335)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
FuelToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (FuelToken.sol#212-221) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,FuelToken::delegateBySig: signature expired) (FuelToken.sol#219)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
FuelToken.getChainId() (FuelToken.sol#366-370) uses assembly
        - INLINE ASM None (FuelToken.sol#368)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version^0.5.16 (FuelToken.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11Reference: https://githu
b.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
delegate(address) should be declared external:
        - FuelToken.delegate(address) (FuelToken.sol#199-201)
delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
        - FuelToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (FuelToken.sol#212-221)
getPriorVotes(address,uint256) should be declared external:
        - FuelToken.getPriorVotes(address,uint256) (FuelToken.sol#240-272)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external
INFO:Slither:FuelToken.sol analyzed (1 contracts with 46 detectors), 7 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

The other finding is involved with using dangerous strict equalities, which can be manipulated by an attacker in some circumstances. In this case, we can see that the checkpoints are using this equality function for voting.



```
function _writeCheckpoint(address delegatee, uint32 nCheckpoints, uint96 oldVotes, uint96 newVotes)
  uint32 blockNumber = safe32(block.number, "FuelToken::_writeCheckpoint: block number exceeds 32 bi

  if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
      checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
  } else {
      checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
      numCheckpoints[delegatee] = nCheckpoints + 1;
  }
```

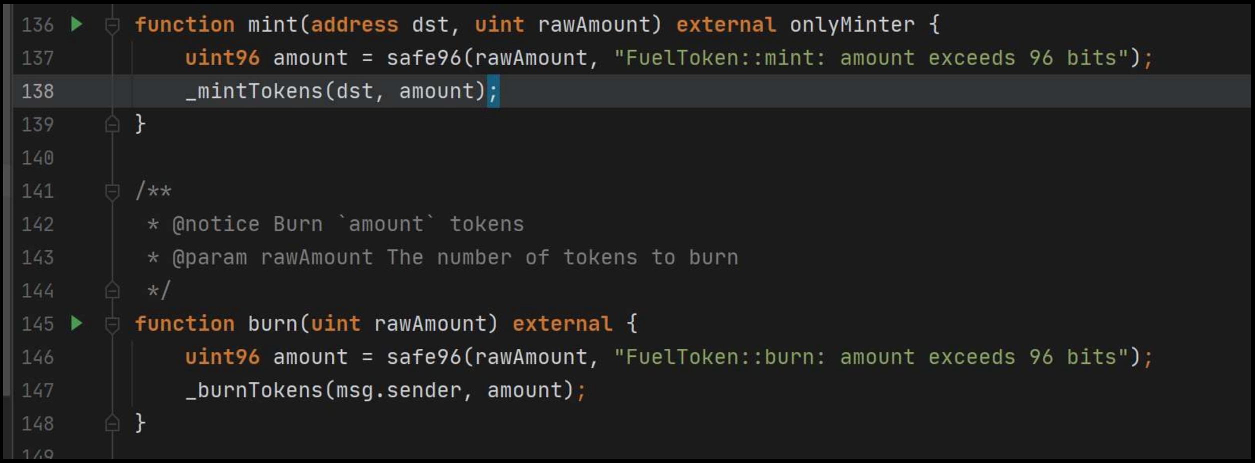
Recommendation:

Consider use of >= or <= rather than ==, to prevent an attacker from trapping the contract due to strict equalities.

3.3 FUNCTION TESTING AND ANALYSIS - INFORMATIONAL

Description:

Halborn performed test cases over FuelToken.sol focused on the two most important functions: mint() and burn().



```
function mint(address dst, uint rawAmount) external onlyMinter {
    uint96 amount = safe96(rawAmount, "FuelToken::mint: amount exceeds 96 bits");
    _mintTokens(dst, amount);
}

/**
 * @notice Burn `amount` tokens
 * @param rawAmount The number of tokens to burn
 */
function burn(uint rawAmount) external {
    uint96 amount = safe96(rawAmount, "FuelToken::burn: amount exceeds 96 bits");
    _burnTokens(msg.sender, amount);
}
```

Both localhost (JSON-RPC) and Infura have been used to correctly perform test cases, after setting up network, compiling the contract and deploying it.

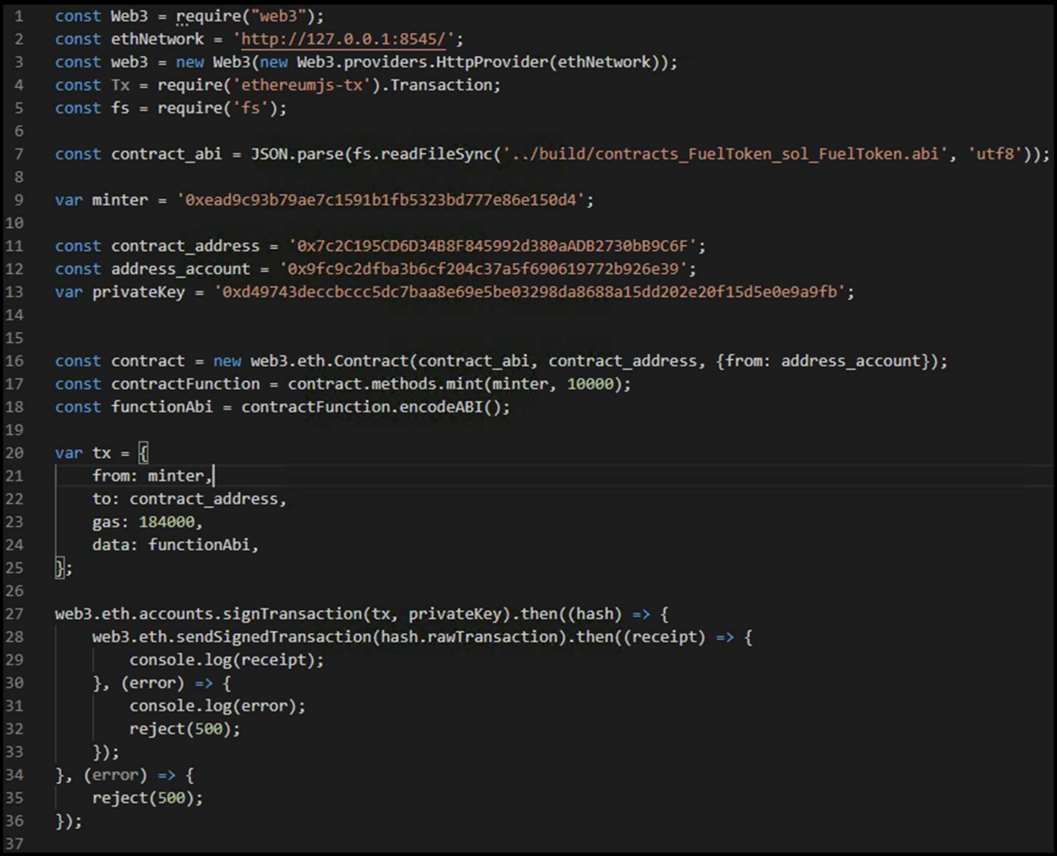
Results:

All logical validation passes for mint and burn functions in a dynamic test environment. An exploitable situation would be any condition that allows a user to burn tokens other than their own, or an address other than the minter to mint tokens. Neither of these conditions were detected.

3.3.1. ONLYMINTER FUNCTION TESTING - INFORMATIONAL

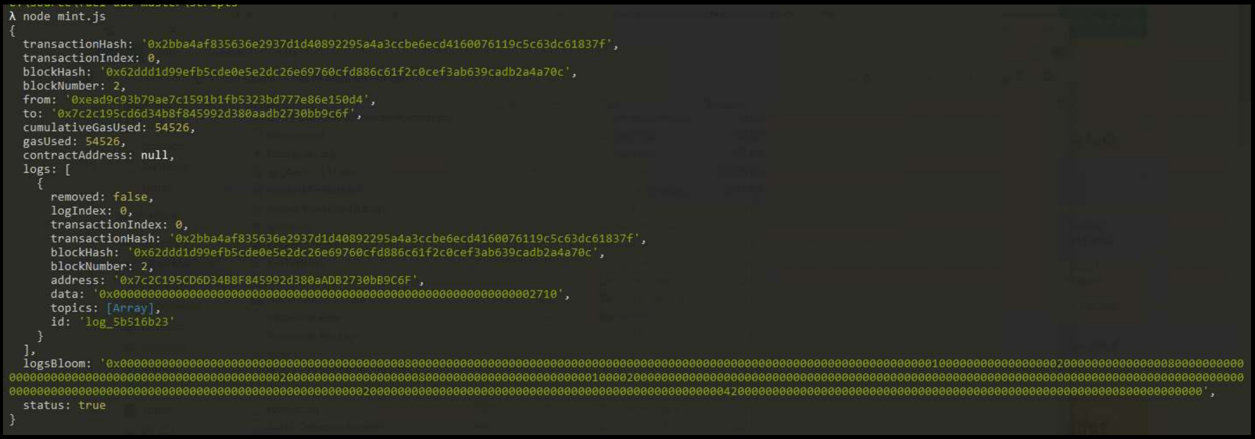
Description:

A test to check if only minter is able to execute onlyMinter functions such as mint() and changeMinter(). First test was performed by the following JavaScript script (mint.js):



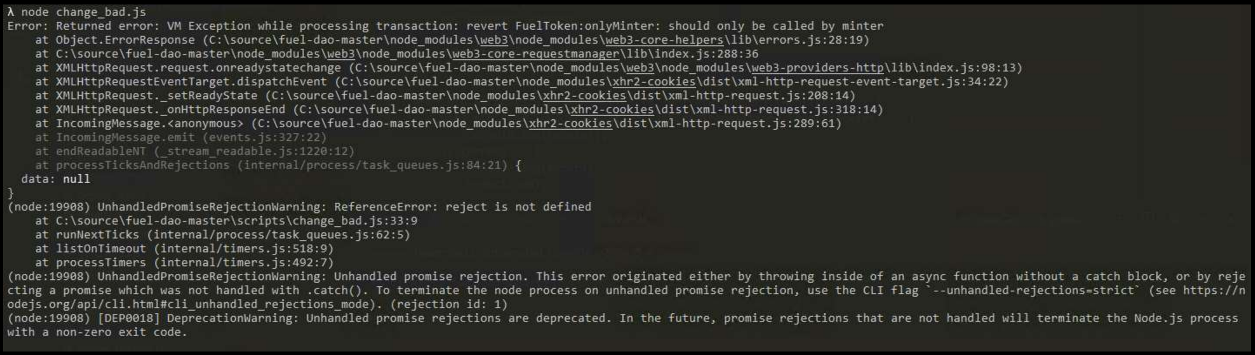
```
const Web3 = require("web3");
const ethNetwork = 'http://127.0.0.1:8545/';
const web3 = new Web3(new Web3.providers.HttpProvider(ethNetwork));
const Tx = require('ethereumjs-tx').Transaction;
const fs = require('fs');

const contract_abi = JSON.parse(fs.readFileSync('../build/contracts_FuelToken_sol_FuelToken.abi', 'utf8'));

var minter = '0xead9c93b79ae7c1591b1fb5323bd777e86e150d4';

const contract_address = '0x7c2C195CD6D34B8F845992d380aADB2730bB9C6F';
const address_account = '0x9fc9c2dfba3b6cf204c37a5f690619772b926e39';
var privateKey = '0xd49743deccbccc5dc7baa8e69e5be03298da8688a15dd202e20f15d5e0e9a9fb';


const contract = new web3.eth.Contract(contract_abi, contract_address, {from: address_account});
const contractFunction = contract.methods.mint(minter, 10000);
const functionAbi = contractFunction.encodeABI();

var tx = {
    from: minter,
    to: contract_address,
    gas: 184000,
    data: functionAbi,
};

web3.eth.accounts.signTransaction(tx, privateKey).then((hash) => {
    web3.eth.sendSignedTransaction(hash.rawTransaction).then((receipt) => {
        console.log(receipt);
    }, (error) => {
        console.log(error);
        reject(500);
    });
}, (error) => {
    reject(500);
});
```

- *Mint.js script was successfully executed by the Minter account signed by his private key.*



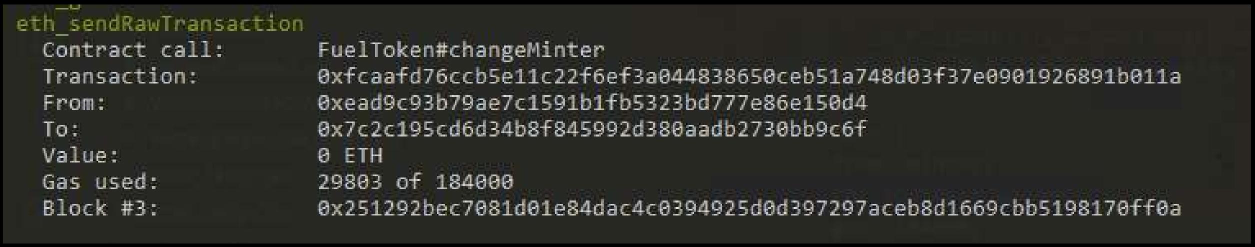
```
λ node mint.js
{
  transactionHash: '0x2bba4af835636e2937d1d40892295a4a3ccbe6ecd4160076119c5c63dc61837f',
  transactionIndex: 0,
  blockHash: '0x62ddd1d99efb5cde0e5e2dc26e69760cfd886c61f2c0cef3ab639cadb2a4a70c',
  blockNumber: 2,
  from: '0xead9c93b79ae7c1591b1fb5323bd777e86e150d4',
  to: '0x7c2c195cd6d34b8f845992d380aadb2730bb9c6f',
  cumulativeGasUsed: 54526,
  gasUsed: 54526,
  contractAddress: null,
  logs: [
    {
      removed: false,
      logIndex: 0,
      transactionIndex: 0,
      transactionHash: '0x2bba4af835636e2937d1d40892295a4a3ccbe6ecd4160076119c5c63dc61837f',
      blockHash: '0x62ddd1d99efb5cde0e5e2dc26e69760cfd886c61f2c0cef3ab639cadb2a4a70c',
      blockNumber: 2,
      address: '0x7c2C195CD6D34B8F845992d380aADB2730bB9C6F',
      data: '0x0000000000000000000000000000000000000000000000000000000000002710',
      topics: [Array],
      id: 'log_5b516b23'
    }
  ],
  logsBloom: '0x0000000000000000000000000000000000000000008000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000200000000000000080000000000
000000000000000000000000000000000000000200000000000000000008000000000000000000000001000020000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000004200000000000000000000000000000000000000000000000000000080000000000000',
  status: true
}
```

- *Nevertheless, when mint.js script was executed by a non-Minter account, script failed since only Minter can mint, as expected.*



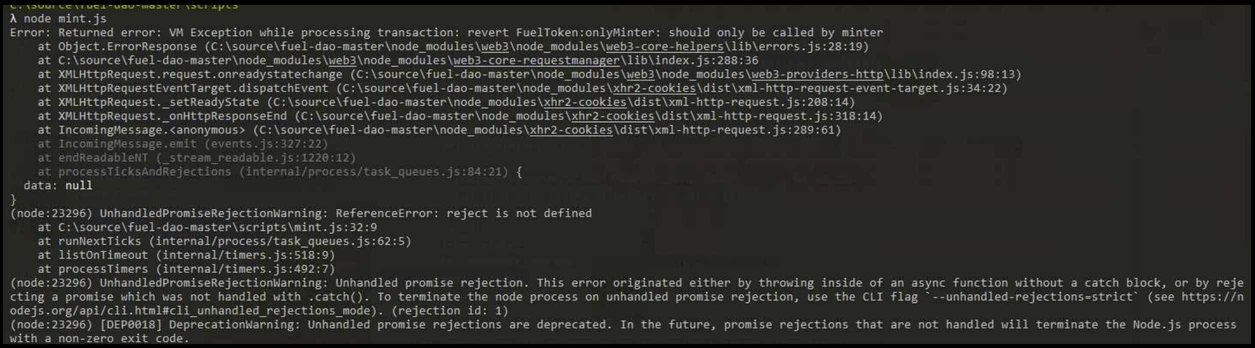
```
λ node change_bad.js
Error: Returned error: VM Exception while processing transaction: revert FuelToken:onlyMinter: should only be called by minter
    at Object.ErrorResponse (C:\source\fuel-dao-master\node_modules\web3\node_modules\web3-core-helpers\lib\errors.js:28:19)
    at C:\source\fuel-dao-master\node_modules\web3\node_modules\web3-core-requestmanager\lib\index.js:288:36
    at XMLHttpRequest.request.onreadystatechange (C:\source\fuel-dao-master\node_modules\web3\node_modules\web3-providers-http\lib\index.js:98:13)
    at XMLHttpRequestEventTarget.dispatchEvent (C:\source\fuel-dao-master\node_modules\xhr2-cookies\dist\xml-http-request-event-target.js:34:22)
    at XMLHttpRequest._setReadyState (C:\source\fuel-dao-master\node_modules\xhr2-cookies\dist\xml-http-request.js:208:14)
    at XMLHttpRequest._onHttpResponseEnd (C:\source\fuel-dao-master\node_modules\xhr2-cookies\dist\xml-http-request.js:318:14)
    at IncomingMessage.<anonymous> (C:\source\fuel-dao-master\node_modules\xhr2-cookies\dist\xml-http-request.js:289:61)
    at IncomingMessage.emit (events.js:327:22)
    at endReadableNT (_stream_readable.js:1220:12)
    at processTicksAndRejections (internal/process/task_queues.js:84:21) {
  data: null
}
(node:19908) UnhandledPromiseRejectionWarning: ReferenceError: reject is not defined
    at C:\source\fuel-dao-master\scripts\change_bad.js:33:9
    at runNextTicks (internal/process/task_queues.js:62:5)
    at listOnTimeout (internal/timers.js:518:9)
    at processTimers (internal/timers.js:492:7)
(node:19908) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block, or by reje
cting a promise which was not handled with .catch(). To terminate the node process on unhandled promise rejection, use the CLI flag `--unhandled-rejections=strict` (see https://n
odejs.org/api/cli.html#cli_unhandled_rejections_mode). (rejection id: 1)
(node:19908) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js process
with a non-zero exit code.
```

- *Only minter can also change the minter. To check this function, another script was built (changeMinter.js):*



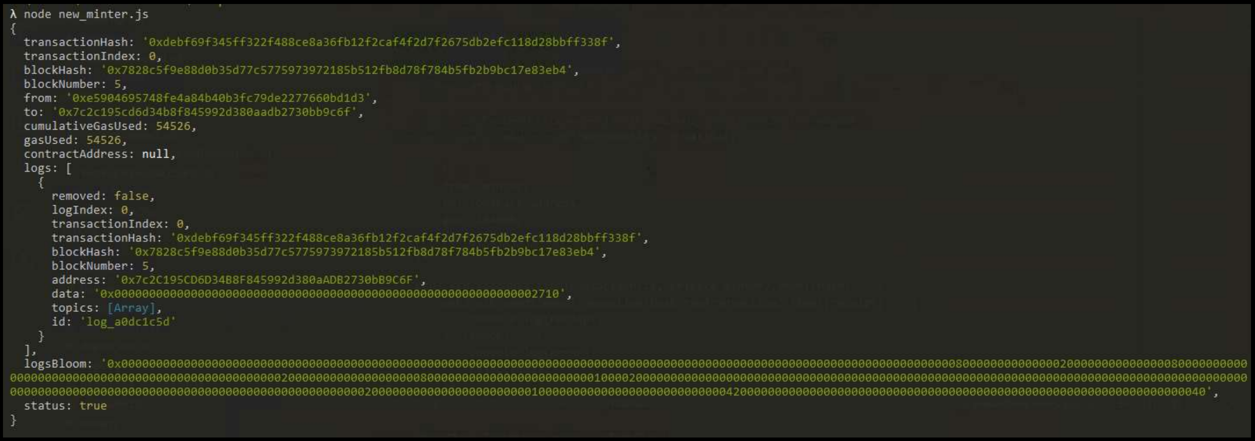
```
eth_sendRawTransaction
  Contract call:       FuelToken#changeMinter
  Transaction:         0xfcaafd76ccb5e11c22f6ef3a044838650ceb51a748d03f37e0901926891b011a
  From:                0xead9c93b79ae7c1591b1fb5323bd777e86e150d4
  To:                  0x7c2c195cd6d34b8f845992d380aadb2730bb9c6f
  Value:               0 ETH
  Gas used:            29803 of 184000
  Block #3:            0x251292bec7081d01e84dac4c0394925d0d397297aceb8d1669cbb5198170ff0a
```

- *changeMinter.js script is executed.*



```
C:\source\fuel-dao-master\scripts
λ node mint.js
Error: Returned error: VM Exception while processing transaction: revert FuelToken:onlyMinter: should only be called by minter
    at Object.ErrorResponse (C:\source\fuel-dao-master\node_modules\web3\node_modules\web3-core-helpers\lib\errors.js:28:19)
    at C:\source\fuel-dao-master\node_modules\web3\node_modules\web3-core-requestmanager\lib\index.js:288:36
    at XMLHttpRequest.request.onreadystatechange (C:\source\fuel-dao-master\node_modules\web3\node_modules\web3-providers-http\lib\index.js:98:13)
    at XMLHttpRequestEventTarget.dispatchEvent (C:\source\fuel-dao-master\node_modules\xhr2-cookies\dist\xml-http-request-event-target.js:34:22)
    at XMLHttpRequest._setReadyState (C:\source\fuel-dao-master\node_modules\xhr2-cookies\dist\xml-http-request.js:208:14)
    at XMLHttpRequest._onHttpResponseEnd (C:\source\fuel-dao-master\node_modules\xhr2-cookies\dist\xml-http-request.js:318:14)
    at IncomingMessage.<anonymous> (C:\source\fuel-dao-master\node_modules\xhr2-cookies\dist\xml-http-request.js:289:61)
    at IncomingMessage.emit (events.js:327:22)
    at endReadableNT (_stream_readable.js:1220:12)
    at processTicksAndRejections (internal/process/task_queues.js:84:21) {
  data: null
}
(node:23296) UnhandledPromiseRejectionWarning: ReferenceError: reject is not defined
    at C:\source\fuel-dao-master\scripts\mint.js:32:9
    at runNextTicks (internal/process/task_queues.js:62:5)
    at listOnTimeout (internal/timers.js:518:9)
    at processTimers (internal/timers.js:492:7)
(node:23296) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block, or by reje
cting a promise which was not handled with .catch(). To terminate the node process on unhandled promise rejection, use the CLI flag `--unhandled-rejections=strict` (see https://n
odejs.org/api/cli.html#cli_unhandled_rejections_mode). (rejection id: 1)
(node:23296) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js process
with a non-zero exit code.
```

- *It was then verified and checked if old Minter was able to mint, but an error was shown as expected.*



```
λ node new_minter.js
{
  transactionHash: '0xdebf69f345ff322f488ce8a36fb12f2caf4f2d7f2675db2efc118d28bbff338f',
  transactionIndex: 0,
  blockHash: '0x7828c5f9e88d0b35d77c5775973972185b512fb8d78f784b5fb2b9bc17e83eb4',
  blockNumber: 5,
  from: '0xe5904695748fe4a84b40b3fc79de2277660bd1d3',
  to: '0x7c2c195cd6d34b8f845992d380aadb2730bb9c6f',
  cumulativeGasUsed: 54526,
  gasUsed: 54526,
  contractAddress: null,
  logs: [
    {
      removed: false,
      logIndex: 0,
      transactionIndex: 0,
      transactionHash: '0xdebf69f345ff322f488ce8a36fb12f2caf4f2d7f2675db2efc118d28bbff338f',
      blockHash: '0x7828c5f9e88d0b35d77c5775973972185b512fb8d78f784b5fb2b9bc17e83eb4',
      blockNumber: 5,
      address: '0x7c2C195CD6D34B8F845992d380aADB2730bB9C6F',
      data: '0x0000000000000000000000000000000000000000000000000000000000002710',
      topics: [Array],
      id: 'log_a0dc1c5d'
    }
  ],
  logsBloom: '0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000800000000000000200000000000000800000000000
0000000000000000000000000000000000000002000000000000000000000080000000000000000000000100002000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000020000000000000000000000001000000000000000000000000004200000000000000000000000000000000000000000000000000000000000000000040',
  status: true
}
```

- *It was tried to mint by new minter and new minter successfully mint:*

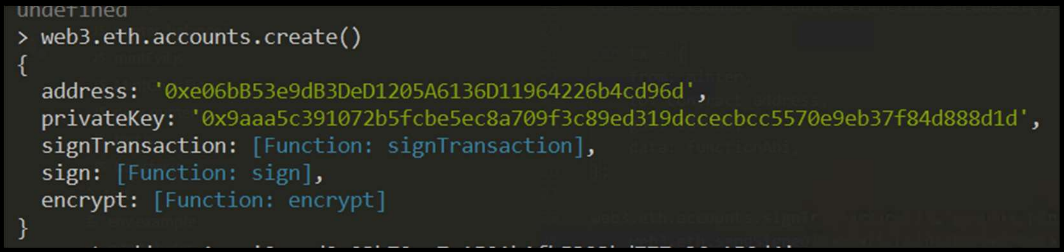
Results:

OnlyMinter conditions correctly work.

3.3.2. HIDDEN ACCOUNT CREATED- INFORMATIONAL

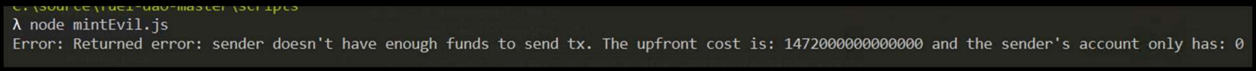
Description:

A new account was created within the network to test what kind of actions this account is able to carry out, simulating a regular user of the contract.



```
undefined
> web3.eth.accounts.create()
{
  address: '0xe06bB53e9dB3DeD1205A6136D11964226b4cd96d',
  privateKey: '0x9aaa5c391072b5fcbe5ec8a709f3c89ed319dccecbcc5570e9eb37f84d888d1d',
  signTransaction: [Function: signTransaction],
  sign: [Function: sign],
  encrypt: [Function: encrypt]
}
```

- *An account was created, and an address and private key was given in the test system:*



```
c:\source\fuel-dao-master\scripts
λ node mintEvil.js
Error: Returned error: sender doesn't have enough funds to send tx. The upfront cost is: 1472000000000000 and the sender's account only has: 0
```

- *It was tried to mint with the new account, but it was not possible because it was not the minter.*
- *It was tried new account became in minter by changeMinter function, and new account became the new minter, but it was not able to mint because of it didn't have funds, and was not idenfied a minter by a voting on the DAO to change the address.*

Results:

Although results of this test is out of the scope, it should be known by Power Trade that if a new account becomes a minter outside the normal logic and voting by the DAO, this would be a large security incident. The voting contracts that determine a change in the minter address are outside this scope, and external to the direct FuelToken contract, which include the functions determining the changing of a minter. Halborn recommends a test of these external contracts as well.

3.4 SECURITY FUZZING RESULT - INFORMATIONAL

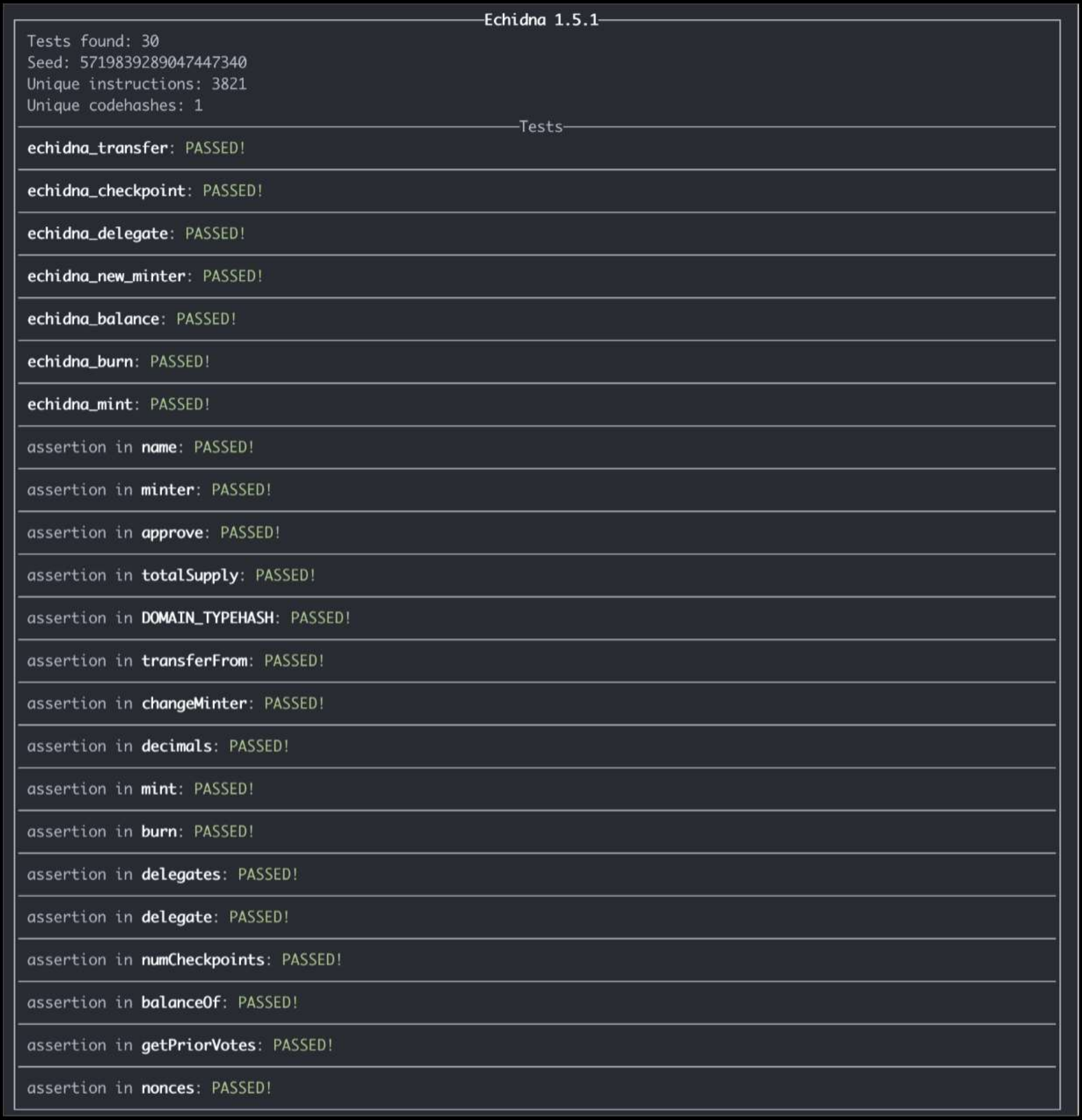
Description:

Fuzzing tests were performed using functions mint() and burn(). These tests were carry out by echidna, which takes as input to the contract a list of properties that should always remain true. Although contract FuelToken.sol does not contain any Boolean property(variable), some Boolean properties were manually added to perform the fuzzing. So, a new contract was created, called ZFuelToken.

```
pragma solidity ^0.5.16;
import './FuelToken.sol';

contract Zfueltoken is FuelToken(100, 0x00A329C0648769a73aFAc7F9381E08fB43DbeA50,
0x00a329c0648769A73afAc7F9381E08FB43dBEA72){

        constructor() public{ }

        function echidna_balance() public view returns(bool){
        return (totalSupply == 100);
        }

        function echidna_mint() public returns(bool){
        return !s1;
        }
        function echidna_burn() public returns(bool){
        return !s2;
        }
        function echidna_new_minter() public returns(bool){
        return !s3;
        }
        function echidna_transfer() public returns(bool){
        return !s4;
        }
        function echidna_delegate() public returns(bool){
        return !s5;
        }
        function echidna_checkpoint() public returns(bool){
        return !s6;
        }}
```

Results:

All tests were successfully passed.



```
Echidna 1.5.1
Tests found: 30
Seed: 5719839289047447340
Unique instructions: 3821
Unique codehashes: 1
Tests
echidna_transfer: PASSED!
echidna_checkpoint: PASSED!
echidna_delegate: PASSED!
echidna_new_minter: PASSED!
echidna_balance: PASSED!
echidna_burn: PASSED!
echidna_mint: PASSED!
assertion in name: PASSED!
assertion in minter: PASSED!
assertion in approve: PASSED!
assertion in totalSupply: PASSED!
assertion in DOMAIN_TYPEHASH: PASSED!
assertion in transferFrom: PASSED!
assertion in changeMinter: PASSED!
assertion in decimals: PASSED!
assertion in mint: PASSED!
assertion in burn: PASSED!
assertion in delegates: PASSED!
assertion in delegate: PASSED!
assertion in numCheckpoints: PASSED!
assertion in balanceOf: PASSED!
assertion in getPriorVotes: PASSED!
assertion in nonces: PASSED!
```

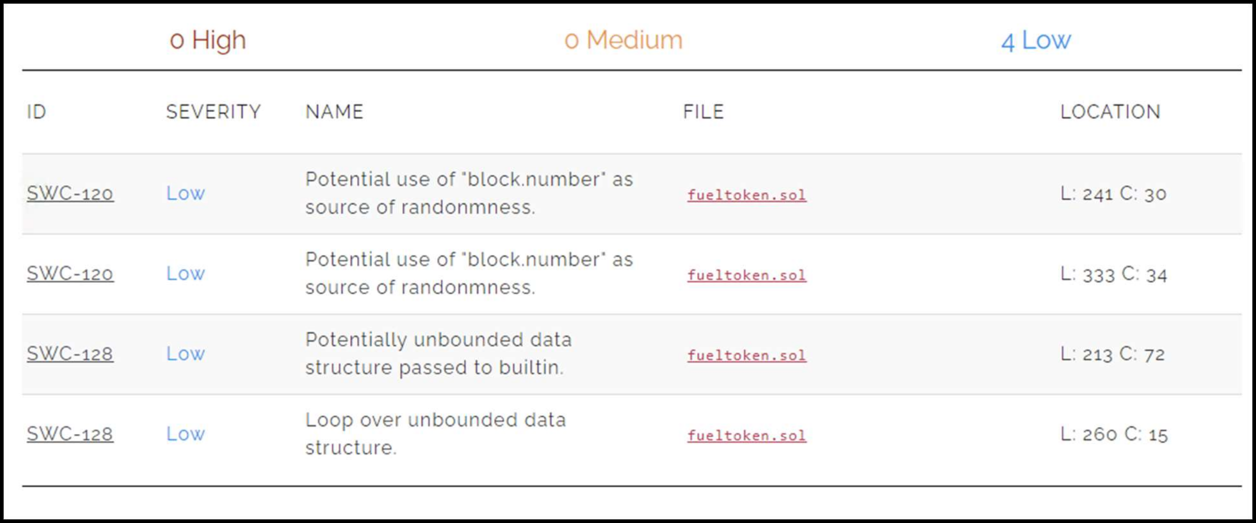
3.5 AUTOMATED SECURITY SCAN - INFORMATIONAL

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit 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 the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Security Detections are only in scope, and the analysis was pointed towards issues with the FuelToken.sol.

Results:

MythX detected 0 High findings, 0 Medium, and 4 Low.



0 High | 0 Medium | 4 Low

ID	SEVERITY	NAME	FILE	LOCATION
SWC-120	Low	Potential use of "block.number" as source of randonmness.	fueltoken.sol	L: 241 C: 30
SWC-120	Low	Potential use of "block.number" as source of randonmness.	fueltoken.sol	L: 333 C: 34
SWC-128	Low	Potentially unbounded data structure passed to builtin.	fueltoken.sol	L: 213 C: 72
SWC-128	Low	Loop over unbounded data structure.	fueltoken.sol	L: 260 C: 15