APY.FINANCE GOVERNANCE TOKEN & REWARD DISTRIBUTOR

Smart Contract Security Audit

Prepared by: **Halborn**
Date of Engagement: October 12th, 2020
Visit: Halborn.com

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/12/2020	Gabi Urrutia
0.2	Document Edits	10/12/2020	Steven Walbroehl
1.0	Draft Version	10/30/2020	Steven Walbroehl

CONTACTS

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

1.1 INTRODUCTION

APY.Finance engaged Halborn to conduct a security assessment on their Governance Token smart contract beginning on October 12th, 2020 and ending October 30th, 2020. The security assessment was scoped to the contracts APYGovernanceToken.sol and APYRewardDistributor.sol an audit of the security risk and implications regarding the changes introduced by the development team at APY.Finance prior to its production release shortly following the assessments deadline.

Overall, the smart contracts code is extremely well documented, follows a high-quality software development standard, contain 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 in both contracts is regarding an experimental version ABIEncoderV2 that is enabled. It is best practice does not use experimental features in production and is recommended to be excluded from the final version of the Governance Token and Reward Distributor.

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 APY.Finance Token Contracts 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 & 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 2/8 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 Governance Token.
- 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 (Localhost)
- Smart Contract analysis and automatic exploitation (limited-time)
- Symbolic Execution / EVM bytecode security assessment (limited-time)

1.3 SCOPE

IN-SCOPE:
Code related to the APYGovernanceToken and APYRewardDistributor smart contracts. Specific commit of contract: commit 75a636fae40e3304671e990c5fb31b811d18d66e

OUT-OF-SCOPE:

External contracts, External Oracles, other smart contracts in the repository or imported by APYGovernanceToken and APYRewardDistributor, economic attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	1

SECURITY ANALYSIS	RISK LEVEL
EXPERIMENTAL FEATURES ENABLED	Low
USE OF INLINE ASSEMBLY	Informational
STATIC ANALYSIS REPORT	Informational
ERC CONFORMAL CHECKER	Informational

FINDINGS & TECH 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 publicly released while using this feature. However, the bug only manifests itself when all 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:

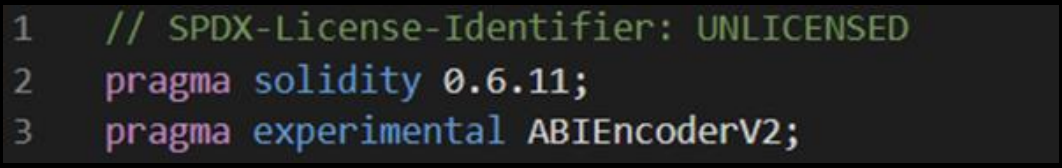
- All the structs or arrays only use uint256 or int256 types.
- If you only use integer types (that may be shorter) and only encode at most one array at a time.
- If you only return such data and do not use it in abi.encode, external calls or event data.

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

Code Location

APYGovernanceToken.sol Line #3

APYRewardDistributor.sol Line #3



```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.6.11;
pragma experimental ABIEncoderV2;
```

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)

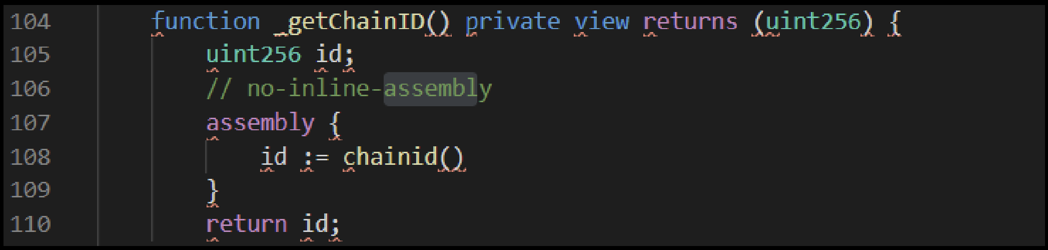
3.2. USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Etehreum Virtual Machine at a alow level. This discards several important safety features in Solidity.

Code Location:

APYRewardDistributor.sol Line #107



```
104     function _getChainID() private view returns (uint256) {
105         uint256 id;
106         // no-inline-assembly
107         assembly {
108             id := chainid()
109         }
110         return id;
```

Recommendation:

When possible, do not use inline assembly because it is a manner to access to the EVM (Ethereum Virtual Machine) at a low level. An attacker could bypass many important safety features of Solidity.

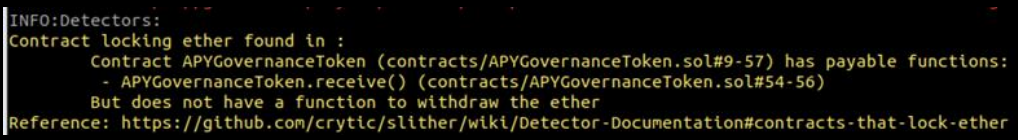
3.3 STATIC ANALYSIS REPORT - INFORMATIONAL

Description:

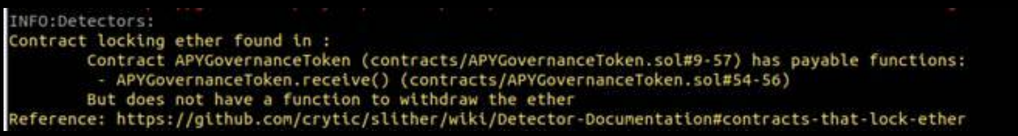
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the Governance Token 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 the APYGovernaceToken contract has a payable function, but it does not have a function to withdraw the ether.

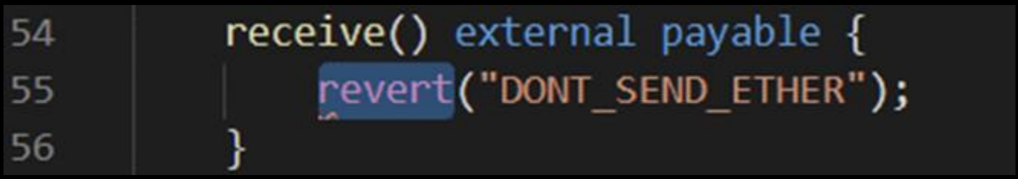


```
INFO:Detectors:
Contract locking ether found in :
        Contract APYGovernanceToken (contracts/APYGovernanceToken.sol#9-57) has payable functions:
         - APYGovernanceToken.receive() (contracts/APYGovernanceToken.sol#54-56)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```



```
INFO:Detectors:
Contract locking ether found in :
        Contract APYGovernanceToken (contracts/APYGovernanceToken.sol#9-57) has payable functions:
         - APYGovernanceToken.receive() (contracts/APYGovernanceToken.sol#54-56)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

The finding is a false positive, because the contract uses revert() in the payable function:



```
54        receive() external payable {
55            revert("DONT_SEND_ETHER");
56        }
```

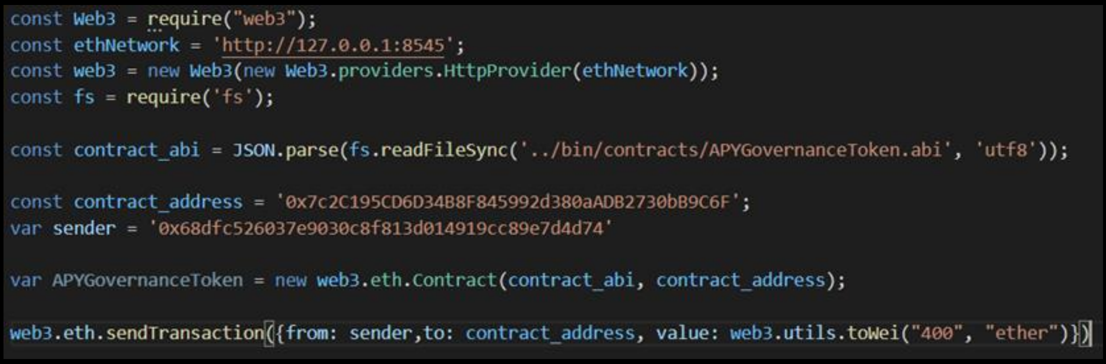
Thus, it is not possible to send ether to the contract address. Code syntax detections. Not security related.

3.3.1 PAYABLE FUNCTION TESTING - INFORMATIONAL

Description:

To check the false positive, a test case was performed over APYGovernanceToken.sol on the function receive(). localhost (JSON-RPC) has been used to correctly perform the test case, after setting up network, compiling the contract and deploying it.

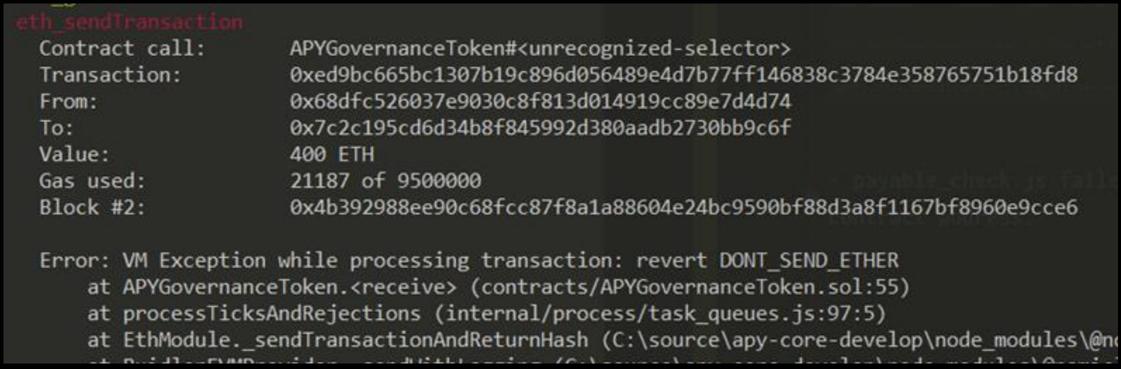
The test was performed by the following JavaScript script (payable_check.js):



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

const contract_abi = JSON.parse(fs.readFileSync('../bin/contracts/APYGovernanceToken.abi', 'utf8'));

const contract_address = '0x7c2C195CD6D34B8F845992d380aADB2730bB9C6F';
var sender = '0x68dfc526037e9030c8f813d014919cc89e7d4d74'

var APYGovernanceToken = new web3.eth.Contract(contract_abi, contract_address);

web3.eth.sendTransaction({from: sender,to: contract_address, value: web3.utils.toWei("400", "ether")})
```

Results:

payable_check.js failed since it was not possible to send ether to the contract address. So, revert() in 7/9 the payable function works well. The false positive is confirmed.

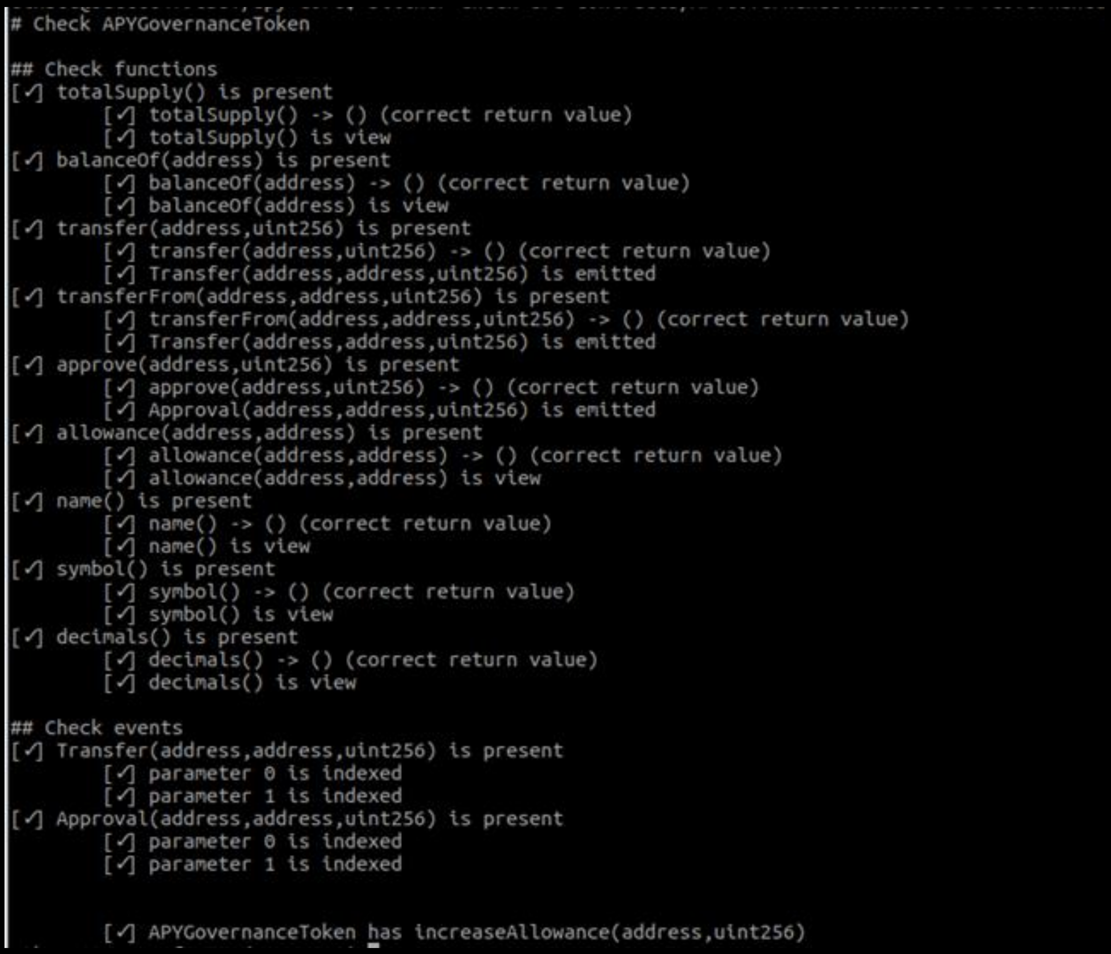


```
eth_sendTransaction
  Contract call:       APYGovernanceToken#<unrecognized-selector>
  Transaction:         0xed9bc665bc1307b19c896d056489e4d7b77ff146838c3784e358765751b18fd8
  From:                0x68dfc526037e9030c8f813d014919cc89e7d4d74
  To:                  0x7c2c195cd6d34b8f845992d380aadb2730bb9c6f
  Value:               400 ETH
  Gas used:            21187 of 9500000
  Block #2:            0x4b392988ee90c68fcc87f8a1a88604e24bc9590bf88d3a8f1167bf8960e9cce6

  Error: VM Exception while processing transaction: revert DONT_SEND_ETHER
      at APYGovernanceToken.<receive> (contracts/APYGovernanceToken.sol:55)
      at processTicksAndRejections (internal/process/task_queues.js:97:5)
      at EthModule._sendTransactionAndReturnHash (C:\source\apy-core-develop\node_modules\@n
```

3.3.2 ERC CONFORMAL CHECKER- INFORMATIONAL

Description:

Another tool by Slither can test ERC Token functions. Thus, slither-check-erc20 was performed over APYGovernanceToken:



```
# Check APYGovernanceToken

## 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


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

Results:

All tests were successfully passed.

3.4 AUTOMATED SECURITY SCAN - INFORMATIONAL

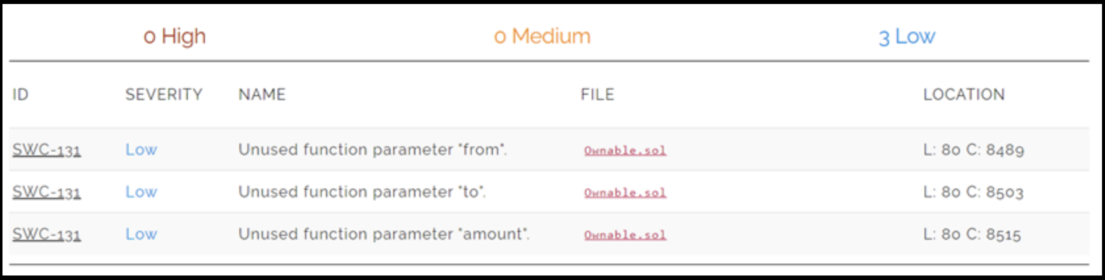
Description:

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 9/9 Detections are only in scope, and the analysis was pointed towards issues with APYGovernanceToken.sol and APYRewardDistributor.

Results:

APYGovernanceToken Results

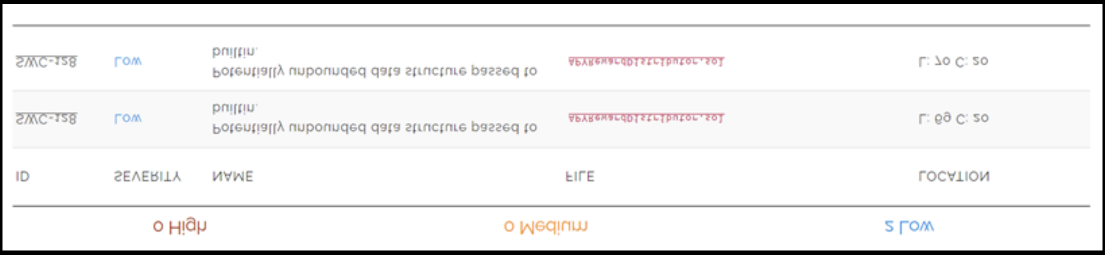
MythX detected 0 High findings, 0 Medium, and 3 Low.



0 High		0 Medium		3 Low
ID	SEVERITY	NAME	FILE	LOCATION
SWC-131	Low	Unused function parameter "from".	Ownable.sol	L: 80 C: 8489
SWC-131	Low	Unused function parameter "to".	Ownable.sol	L: 80 C: 8503
SWC-131	Low	Unused function parameter "amount".	Ownable.sol	L: 80 C: 8515

APYRewardDistributor Results

MythX detected 0 High findings, 0 Medium, and 2 Low.



0 High		0 Medium		2 Low
ID	SEVERITY	NAME	FILE	LOCATION
SWC-128	Low	Potentially unbounded data structure passed to builtin.	APYRewardDistributor.sol	L: 69 C: 20
SWC-128	Low	Potentially unbounded data structure passed to builtin.	APYRewardDistributor.sol	L: 70 C: 20

THANK YOU FOR CHOOSING

// HALBORN