// HALBORN

POWER TRADE VESTING CONTRACTS

Smart Contract Security Audit

Prepared by: **Halborn**
Date of Engagement: **09.21-23.2020**
Visit: Halborn.com

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	9/23/2020	Gabi Urrutia
0.2	Document Edits	9/23/2020	Steven Walbroehl
1.0	Draft Version	9/23/2020	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

Power Trade engaged Halborn to conduct a security assessment on their Vesting smart contracts beginning on September 20th, 2020 and ending September 23rd 2020. The security assessment was scoped to Vesting contracts 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.

Contracts scoped in this assessment are Vesting Contracts: VestingContract.sol, VestingContractWithoutDelegation.sol and VestingDepositAccount.sol. Since the token acquisition process is becoming more complex, vesting is increasingly important. Therefore, PTF token acquisition process is supported by Vesting Contracts to correctly work the acquisition and vesting 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.

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 Vesting 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 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)
- 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 Vesting set of smart contracts.

OUT-OF-SCOPE:

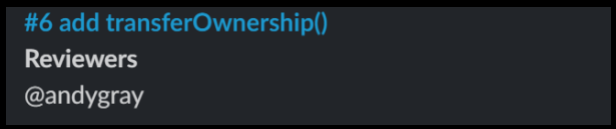
External contracts, External Oracles, other smart contracts in the repository or imported by FuelToken, economic attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	1

SECURITY ANALYSIS	RISK LEVEL
DIVIDE BEFORE MULTIPLY	Very Low
STATIC ANALYSIS REPORT	Informational
AUTOMATED SECURITY SCAN RESULTS	Informational

Note: - One of the previous issues identified was regarding the transferring of the “ownership” role of the vesting contracts. The PowerTrade Development team addressed this issue by implementing fixes via the transferOwnership(). The change was added to PR https://github.com/Power-Trade/fuel-dao/pull/6 and subsequently tested during the logical execution of the contracts by Halborn successfully.



FINDINGS & TECH DETAILS

3.1 DIVIDE BEFORE MULTIPLY - VERY LOW

Description

Solidity integer division might truncate. As a result, performing multiplication before division might reduce precision. Due to the sensitivity of precision, and the amount of detail the development team is putting on the dynamic balancing mechanics involved in PTF, this may be a factor in accuracy of weights/rates.

Code Location:

VestingContract.sol Line #297-298

```
297         uint256 drawDownRate = schedule.amount.div(end.sub(start));
298         uint256 amount = timePassedSinceLastInvocation.mul(drawDownRate);
```

Recommendation

Consider ordering multiplication before division to ensure balances, vesting, amounts, and underlying math stay accurate over large numbers of transactions over time.

3.2 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 Vesting contracts. 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:

Static analysis results show warnings from the Reentrancy Detectors around for functions ‘createVestingSchedule’ and ‘updateScheduleBeneficiary’. This is due to the fact that there are external calls where the state variables get written afterwards. However, these findings are false positives due to the fact that Open Zeppelin library of mitigation contracts are in pace. In particular “ReentrancyGuard.sol” protects the PTF contracts, and prevents exploitation.

```
INFO:Detectors:
Reentrancy in VestingContract.createVestingSchedule(address,uint256) (VestingContract.sol#90-121):
        External calls:
        - depositAccount.init(address(token),address(this),_beneficiary) (VestingContract.sol#104)
        State variables written after the call(s):
        - vestingSchedule[_beneficiary] = Schedule(_amount,depositAccount) (VestingContract.sol#107-110)
Reentrancy in VestingContract.updateScheduleBeneficiary(address,address) (VestingContract.sol#138-159):
        External calls:
        - require(bool,string)(_drawDown(_currentBeneficiary),VestingContract::_updateScheduleBeneficiary: Unable to d
rawn down) (VestingContract.sol#147)
                - require(bool,string)(schedule.depositAccount.transferToBeneficiary(amount),VestingContract::_drawDow
n: Unable to transfer tokens) (VestingContract.sol#244-247)
        State variables written after the call(s):
        - vestingSchedule[_newBeneficiary] = Schedule(schedule.amount.sub(totalDrawn[_currentBeneficiary]),schedule.de
positAccount) (VestingContract.sol#153-156)
        - voided[_currentBeneficiary] = true (VestingContract.sol#150)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

Two false positive detections of reentrancy.

```
INFO:Detectors:
VestingContractWithoutDelegation._availableDrawDownAmount(address) (VestingContractWithoutDelegation.sol#221-247) perf
orms a multiplication on the result of a division:
        -drawDownRate = vestedAmount[_beneficiary].div(end.sub(start)) (VestingContractWithoutDelegation.sol#243)
        -amount = timePassedSinceLastInvocation.mul(drawDownRate) (VestingContractWithoutDelegation.sol#244)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

Detection of multiplication after division, which was the informational finding mentioned.

Code syntax detections. Not security related.

3.3 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 VestingContract.sol, VestingContractWithoutDelegation.sol and VestingDepositAccount.sol.

Results:

MythX detected 0 High findings, 0 Medium, and 1 Low.

0 High		0 Medium		1 Low
ID	SEVERITY	NAME	FILE	LOCATION
SWC-103	Low	A floating pragma is set.	vestingcontract.sol	L: 1 C: 0

MythX detected 0 High findings, 0 Medium, and 1 Low.

0 High		0 Medium		1 Low
ID	SEVERITY	NAME	FILE	LOCATION
SWC-103	Low	A floating pragma is set.	vestingcontractwithoutdelegation.sol	L: 1 C: 0

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

0 High		0 Medium		6 Low
ID	SEVERITY	NAME	FILE	LOCATION
SWC-103	Low	A floating pragma is set.	vestingdepositaccount.sol	L: 1 C: 0
SWC-107	Low	A call to a user-supplied address is executed.	vestingdepositaccount.sol	L: 29 C: 8
SWC-107	Low	A call to a user-supplied address is executed.	vestingdepositaccount.sol	L: 52 C: 8
SWC-107	Low	A call to a user-supplied address is executed.	vestingdepositaccount.sol	L: 39 C: 15
SWC-123	Low	Requirement violation.	vestingdepositaccount.sol	L: 29 C: 8
SWC-123	Low	Requirement violation.	vestingdepositaccount.sol	L: 6 C: 0

THANK YOU FOR CHOOSING

// HALBORN