Polkaswitch

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 19 - April 23, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 3
CONTACTS 3
1 EXECUTIVE OVERVIEW 4
1.1 INTRODUCTION 5
1.2 AUDIT SUMMARY 5
1.3 TEST APPROACH & METHODOLOGY 6
RISK METHODOLOGY 6
1.4 SCOPE 8
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 9
3 FINDINGS & TECH DETAILS 10
3.1 (HAL-01) FLOATING PRAGMA - LOW 12
Description 12
Code Location 12
Risk Level 13
Recommendations 13
Remediation Plan 14
3.2 (HAL-02) MISSING RE-ENTRANCY PROTECTION - LOW 14
Description 14
Code Location 14
Risk Level 15
Recommendation 15
Remediation Plan 15
3.3 (HAL-03) USE OF BLOCK.TIMESTAMP - INFORMATIONAL 15
Description 15

Code Location 16

Recommendation 17

Remediation Plan 17

3.4 (HAL-04) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL 18

Description 18

Code Location 18

Risk Level 21

Recommendations 21

Remediation Plan 21

3.5 STATIC ANALYSIS REPORT 22

Description 22

Results 22

Recommendations 25

3.6 AUTOMATED SECURITY SCAN 25

MYTHX 25

Results 25

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/20/2021	Nishit Majithia
0.2	Document Edits	04/21/2021	Nishit Majithia
1.0	Final Version	04/23/2021	Gabi Urrutia
1.1	Remediation Plan	05/04/2021	Nishit Majithia

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
Nishit Majithia	Halborn	nishit.majithia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Polkaswitch is a decentralized, non-custodial cross-chain liquidity protocol built for the financial future. The security assessment was scoped to the smart contracts Greeter.sol, SwitchToken.sol, staking/Farming.sol , vesting/PrivateVesting.sol and vesting/Vesting.sol. Halborn conducted this audit to measure security risk and identify any new vulnerabilities introduced during the final stages of development before the Polkaswitch production release.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided a one week time frame for the engagement and assigned two full time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart contract security experts, with experience in advanced penetration testing, smart contract hacking, and have a deep knowledge in multiple blockchain protocols.

The purpose of this audit to achieve the following:
- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified few security risks, and 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.

1.3 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 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 and purpose
- Smart Contract manual code review and walk-through
- 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)

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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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 smart contracts:

- Greeter.sol
- SwitchToken.sol
- staking/Farming.sol
- vesting/PrivateVesting.sol
- vesting/Vesting.sol

commit ID: 3eadba5d4295e3073e5ea70d7feafa6b021cd112

Fix commit ID: 661db86b73c6b26ffd53938ed94b58fa0f909419

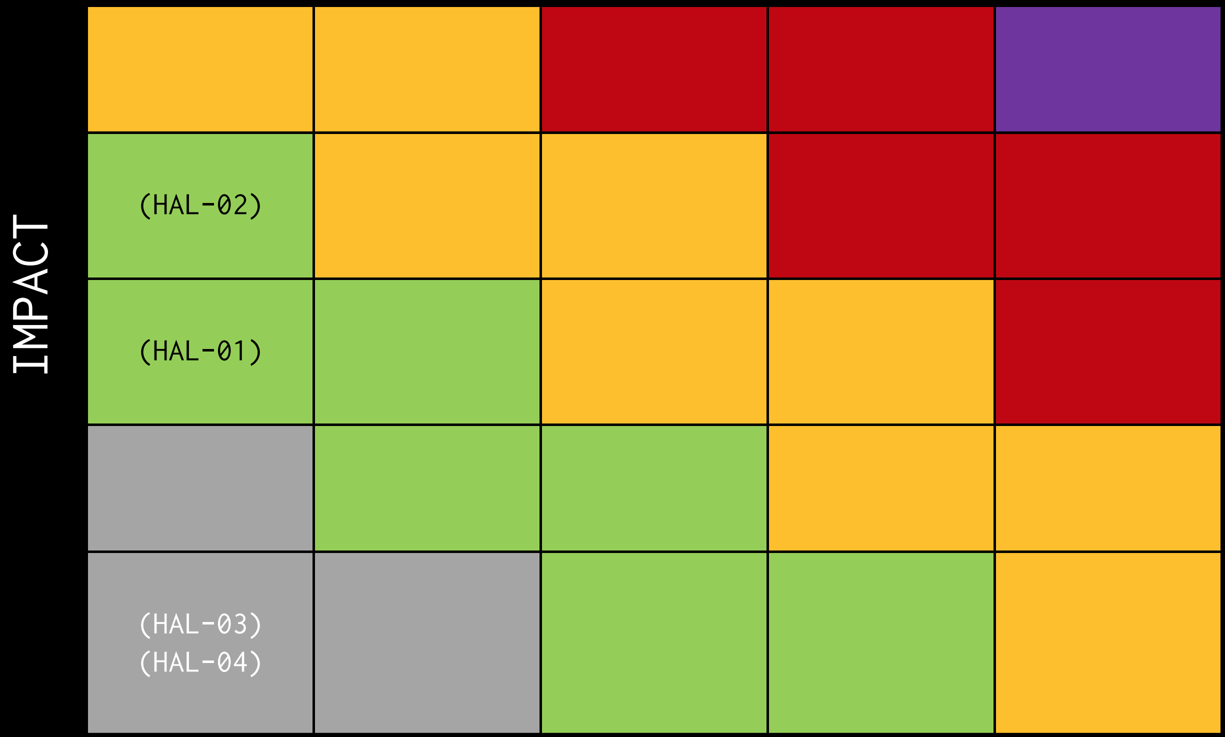
OUT-OF-SCOPE:

Other smart contracts in the repository, external libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	2

LIKELIHOOD



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
FLOATING PRAGMA	Low	SOLVED - 05/04/2021
MISSING RE-ENTRANCY PROTECTION	Low	SOLVED - 05/04/2021
USE OF BLOCK.TIMESTAMP	Informational	NOT APPLICABLE
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED: 05/04/2021
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) FLOATING PRAGMA - LOW

Description:

All Smart Contracts use the floating pragma ^0.6.12. Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the **pragma** helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively or recently released pragma versions may have unknown security vulnerabilities.

Reference: ConsenSys Diligence - Lock pragmas

Code Location:

Greeter.sol: [Line #2]

```
1  //SPDX-License-Identifier: Unlicense
2  pragma solidity ^0.6.12;
3
```

SwitchToken.sol: [Line #23]

```
22
23  pragma solidity ^0.6.12;
24
25  // SwitchToken with Governance and adds a cap to the supply of tokens..
26  contract Switch is IERC20, Ownable {
27      using SafeMath for uint256;
```

staking/Farming.sol: [Line #1]

```
/*

 .______      ______    __       __  ___      ___           _______.
 |   _  \    /  __  \  |  |     |  |/  /     /   \         /       |
 |  |_)  |  |  |  |  | |  |     |  '  /     /  ^  \       |   (----`
 |   ___/   |  |  |  | |  |     |    <     /  /_\  \       \   \
 |  |       |  `--'  | |  `----.|  .  \   /  _____  \  .----)   |
 | _|        \______/  |_______||__|\__\ /__/     \__\ |_______/

description: Cross-chain liquidity built for traders.
website: https://polkaswitch.com/
telegram: https://t.me/polkaswitchANN
*/

pragma solidity ^0.6.12;

```

vesting/PrivateVesting.sol: [Line #1]

```
1  pragma solidity ^0.6.12;
2
```

vesting/Vesting.sol: [Line #1]

```
1  pragma solidity ^0.6.12;
2
```

Reference: ConsenSys Diligence - Lock pragmas

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Consider locking the pragma version. It is not recommended to use a floating pragma in production. It is possible to lock the pragma by fixing the version both in truffle-config.js for Truffle framework or in hardhat-config.js for HardHat framework.

Remediation Plan:

SOLVED: Polkaswitch team locked the pragma version to 0.6.12 in all contracts.

3.2 (HAL-02) MISSING RE-ENTRANCY PROTECTION - LOW

Description:

To protect against cross-function reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdraw function with a recursive call. OpenZeppelin has it's own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against reentrancy attacks.

Code Location:

Vesting.sol Lines #134-142

```
    function claim() public {
        require(!_upfrontReleased[msg.sender], "Vesting: token already claimed");

        uint256 upfront = _beneficiaries[msg.sender].upfront;
        Switch.safeTransfer(msg.sender, upfront);
        _upfrontReleased[msg.sender] = true;

        emit TokenClaimed(msg.sender, upfront);
    }
```

PrivateVesting.sol Lines #102-110

```
    function claim() public {
        require(!_upfrontReleased[msg.sender], "Vesting: token already claimed");

        uint256 upfront = _beneficiaries[msg.sender].upfront;
        Switch.safeTransfer(msg.sender, upfront);
        _upfrontReleased[msg.sender] = true;

        emit TokenClaimed(msg.sender, upfront);
    }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

The Vesting.sol and PrivateVesting.sol smart contracts are missing a nonReentrant guard. Use the nonReentrant modifier to avoid introducing future vulnerabilities.

Remediation Plan:

SOLVED: nonReentrant modifier was added to all the above reported methods.

3.3 (HAL-03) USE OF BLOCK.TIMESTAMP - INFORMATIONAL

Description:

During a manual review, we noticed the use of block.timestamp. The contract developers should be aware that this does not mean current time. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. The use of now creates a risk that time manipulation can be performed to manipulate price oracles. Miners can modify the timestamp by up to 900 seconds.

Code Location:

SwitchToken.sol: [Line #251]

```
249             require(signatory != address(0), "SWITCH::delegateBySig: invalid signature");
250             require(nonce == nonces[signatory]++, "SWITCH::delegateBySig: invalid nonce");
251             require(block.timestamp <= expiry, "SWITCH::delegateBySig: signature expired");
252             return _delegate(signatory, delegatee);
253         }
254
```

staking/Farming.sol: [Line #126]

```
123     }
124
125     function lastTimeRewardApplicable(uint i) public view returns (uint256) {
126         return Math.min(block.timestamp, tokenRewards[i].periodFinish);
127     }
128
129     function rewardPerToken(uint i) public view returns (uint256) {
130         TokenRewards storage tr = tokenRewards[i];
```

[Line #192], [Line #196], [Line #205], [Line #206]

```
190         uint256 duration = tr.duration;
191
192         if (block.timestamp >= tr.periodFinish) {
193             require(reward >= duration, "Reward is too small");
194             tr.rewardRate = reward.div(duration);
195         } else {
196             uint256 remaining = tr.periodFinish.sub(block.timestamp);
197             uint256 leftover = remaining.mul(tr.rewardRate);
198             require(reward.add(leftover) >= duration, "Reward is too small");
199             tr.rewardRate = reward.add(leftover).div(duration);
200         }
201
202         uint balance = tr.gift.balanceOf(address(this));
203         require(tr.rewardRate <= balance.div(duration), "Reward is too big");
204
205         tr.lastUpdateTime = block.timestamp;
206         tr.periodFinish = block.timestamp.add(duration);
207         emit RewardAdded(i, reward);
208     }
209
210     function setRewardDistribution(uint i, address _rewardDistribution) external onlyOwne
```

vesting/PrivateVesting.sol: [Line #75]

```
71      }
72
73      function addBeneficiary(address beneficiary, uint256 start, uint256 amount) public onlyOwner {
74          require(beneficiary != address(0), "Vesting: beneficiary is the zero address");
75          require(start.add(WAVE_3) > block.timestamp, "Vesting: final time is before current time");
76
77          VestingInfo storage vesting = _beneficiaries[beneficiary];
78          vesting.start = start;
```

[Line #158], [Line #160], [Line #162], [Line #164]

```
157
158         if (block.timestamp < _beneficiaries[beneficiary].start) {
159             return 0;
160         } else if (block.timestamp >= _beneficiaries[beneficiary].start.add(WAVE_1) && block.timestamp < _beneficiaries[beneficiary].start.add(WAVE_2)) {
161             return totalBalance.div(5);
162         } else if (block.timestamp >= _beneficiaries[beneficiary].start.add(WAVE_2) && block.timestamp < _beneficiaries[beneficiary].start.add(WAVE_3)) {
163             return totalBalance.div(5).mul(2);
164         } else if (block.timestamp >= _beneficiaries[beneficiary].start.add(WAVE_3) || _revoked[beneficiary]) {
165             return totalBalance;
166         } else {
167             return 0;
168         }
169     }
```

vesting/Vesting.sol: [Line #105]

```
97      }
98
99      function addBeneficiary(address beneficiary, uint256 start, uint256 cliffDuration, uint256 duration, uin
100         require(beneficiary != address(0), "Vesting: beneficiary is the zero address");
101         // solhint-disable-next-line max-line-length
102         require(duration > 0, "Vesting: duration is 0");
103         require(cliffDuration <= duration, "Vesting: cliff is longer than duration");
104         // solhint-disable-next-line max-line-length
105         require(start.add(duration) > block.timestamp, "Vesting: final time is before current time");
106
107         VestingInfo storage vesting = _beneficiaries[beneficiary];
108         vesting.duration = duration;
```

[Line #190], [Line #192], [Line #197]

```
189
190         if (block.timestamp < _beneficiaries[beneficiary].cliff) {
191             return 0;
192         } else if (block.timestamp >= _beneficiaries[beneficiary].start.add(_beneficiaries[beneficiary].duration) || _revoked[beneficiary]) {
193             return totalBalance;
194         } else {
195             uint256 vestingDuration = _beneficiaries[beneficiary].start.add(_beneficiaries[beneficiary].duration).sub(_beneficiaries[beneficiary].cl
196             uint256 totalNumWave = vestingDuration.div(WAVE);
197             uint256 waveNum = block.timestamp.sub(_beneficiaries[beneficiary].cliff).div(WAVE);
198             return totalBalance.mul(waveNum).div(totalNumWave);
199         }
200     }
```

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

NOT APPLICABLE: Development team confirmed that their timescale is larger than 900 seconds when using block.timestamp. So this issue will not affect in this case.

3.4 (HAL-04) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

Code Location:

SwitchToken.sol: [Line #86], [Line #93]

```
84
85      /// @notice Creates `_amount` token to `_to`. Must only be called by the owner.
86      function mint(address _to, uint _rawAmount) public onlyOwner {
87          uint96 _amount = safe96(_rawAmount, "SWITCH::mint: amount exceeds 96 bits");
88          require(_totalSupply + _amount <= _cap, "SWITCH::mint: cap exceeded");
89          _mint(_to, _amount);
90      }
91
92      /// @notice Destroys `_amount` token from `_account`. Must only be called by the owner.
93      function burn(address _from, uint _rawAmount) public onlyOwner {
94          uint96 _amount = safe96(_rawAmount, "SWITCH::burn: amount exceeds 96 bits");
95          _burn(_from, _amount);
96      }
97
```

[Line #136]

```
136     function transfer(address dst, uint rawAmount) public override returns (bool) {
137         uint96 amount = safe96(rawAmount, "SWITCH::transfer: amount exceeds 96 bits");
138         _transferTokens(msg.sender, dst, amount);
139         return true;
140     }
```

vesting/PrivateVesting.sol: [Line #48], [Line #55], [Line #62], [Line #69]

```
     */
    function start(address beneficiary) public view returns (uint256) {
        return _beneficiaries[beneficiary].start;
    }

    /**
     * @return true if the vesting is revocable.
     */
    function revocable() public view returns (bool) {
        return _revocable;
    }

    /**
     * @return the amount of the token released.
     */
    function released(address beneficiary) public view returns (uint256) {
        return _released[beneficiary];
    }

    /**
     * @return true if the token is revoked.
     */
    function revoked(address beneficiary) public view returns (bool) {
        return _revoked[beneficiary];
    }
```

[Line #73], [Line #86], [Line #102]

```
72
73      function addBeneficiary(address beneficiary, uint256 start, uint256 amount) public onlyOwner {
74          require(beneficiary != address(0), "Vesting: beneficiary is the zero address");
75          require(start.add(WAVE_3) > block.timestamp, "Vesting: final time is before current time");
76
77          VestingInfo storage vesting = _beneficiaries[beneficiary];
78          vesting.start = start;
79          vesting.amount = amount;
80          vesting.upfront = amount.mul(3).div(20);
81      }
82
83      /**
84       * @notice Transfers vested tokens to beneficiary.
85       */
86      function release() public {
87          address beneficiary = msg.sender;
88          uint256 unreleased = _releasableAmount(beneficiary);
89
90          require(unreleased > 0, "Vesting: no tokens are due");
91
92          _released[beneficiary] = _released[beneficiary].add(unreleased);
93
94          Switch.safeTransfer(beneficiary, unreleased);
95
96          emit TokensReleased(beneficiary, unreleased);
97      }
98
99      /**
100      * @notice Transfers upfront tokens to beneficiary.
101      */
102     function claim() public {
103         require(!_upfrontReleased[msg.sender], "Vesting: token already claimed");
104
105         uint256 upfront = _beneficiaries[msg.sender].upfront;
106         Switch.safeTransfer(msg.sender, upfront);
```

[Line #116], [Line #139]

```
    /**
     * @notice Allows the owner to revoke the vesting. Tokens already vested
     * remain in the contract, the rest are returned to the owner.
     */
    function revoke(address beneficiary) public onlyOwner {
        require(_revocable, "Vesting: cannot revoke");
        require(!_revoked[beneficiary], "Vesting: token already revoked");

        uint256 balance = _beneficiaries[beneficiary].amount;

        uint256 unreleased = _releasableAmount(beneficiary);
        uint256 refund = balance.sub(unreleased);

        if (_upfrontReleased[beneficiary]) {
            refund = refund.sub(_beneficiaries[beneficiary].upfront);
        }

        _revoked[beneficiary] = true;

        Switch.safeTransfer(owner(), refund);

        emit TokenVestingRevoked(beneficiary);
    }

    /**
     * @notice Make contract non-revocable.
     */
    function finalizeContract() public onlyOwner {
        _revocable = false;
    }
```

vesting/Vesting.sol: [Line #99], [Line #118], [Line #134]

```
98
99      function addBeneficiary(address beneficiary, uint256 start, uint256 cliffDuration, uint256 duration, uint256 amount, uint256 upfront) public onlyOwner {
100         require(beneficiary != address(0), "Vesting: beneficiary is the zero address");
101         // solhint-disable-next-line max-line-length
102         require(duration > 0, "Vesting: duration is 0");
103         require(cliffDuration <= duration, "Vesting: cliff is longer than duration");
104         // solhint-disable-next-line max-line-length
105         require(start.add(duration) > block.timestamp, "Vesting: final time is before current time");
106
107         VestingInfo storage vesting = _beneficiaries[beneficiary];
108         vesting.duration = duration;
109         vesting.cliff = start.add(cliffDuration);
110         vesting.start = start;
111         vesting.amount = amount;
112         vesting.upfront = upfront;
113     }
114
115     /**
116      * @notice Transfers vested tokens to beneficiary.
117      */
118     function release() public {
119         address beneficiary = msg.sender;
120         uint256 unreleased = _releasableAmount(beneficiary);
121
122         require(unreleased > 0, "Vesting: no tokens are due");
123
124         _released[beneficiary] = _released[beneficiary].add(unreleased);
125
126         Switch.safeTransfer(beneficiary, unreleased);
127
128         emit TokensReleased(beneficiary, unreleased);
129     }
130
131     /**
132      * @notice Transfers upfront tokens to beneficiary.
133      */
134     function claim() public {
135         require(!_upfrontReleased[msg.sender], "Vesting: token already claimed");
136
137         uint256 upfront = _beneficiaries[msg.sender].upfront;
138         Switch.safeTransfer(msg.sender, upfront);
139         _upfrontReleased[msg.sender] = true;
140
```

[Line #148], [Line #171]

```
     * remain in the contract, the rest are returned to the owner.
     */
    function revoke(address beneficiary) public onlyOwner {
        require(_revocable, "Vesting: cannot revoke");
        require(!_revoked[beneficiary], "Vesting: token already revoked");

        uint256 balance = _beneficiaries[beneficiary].amount;

        uint256 unreleased = _releasableAmount(beneficiary);
        uint256 refund = balance.sub(unreleased);

        if (_upfrontReleased[beneficiary]) {
            refund = refund.sub(_beneficiaries[beneficiary].upfront);
        }

        _revoked[beneficiary] = true;

        Switch.safeTransfer(owner(), refund);

        emit TokenVestingRevoked(beneficiary);
    }

    /**
     * @notice Make contract non-revocable.
     */
    function finalizeContract() public onlyOwner {
        _revocable = false;
    }

    /**
     * @dev Calculates the amount that has already vested but hasn't been relea
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Consider declaring external variables instead of public variables. A best practice is to use external if expecting a function to only be called externally and public if called internally. Public functions are always accessible, but external functions are only available to external callers.

Remediation Plan:

SOLVED: Modifiers has been changed to external for all the methods above reported

3.5 STATIC ANALYSIS REPORT

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. 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.

Results:

```
INFO:Detectors:
Reentrancy in Farming.exit() (staking/Farming.sol#164-167):
        External calls:
        - withdraw(balanceOf(msg.sender)) (staking/Farming.sol#165)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (openzeppelin/contracts/token/ERC20/SafeERC20.sol#69

                - underlying.safeTransfer(msg.sender,amount) (staking/Farming.sol#61)
                - (success,returndata) = target.call{value: value}(data) (openzeppelin/contracts/utils/Address.sol#119)
        - getAllRewards() (staking/Farming.sol#166)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (openzeppelin/contracts/token/ERC20/SafeERC20.sol#69

                - (success,returndata) = target.call{value: value}(data) (openzeppelin/contracts/utils/Address.sol#119)
                - tr.gift.safeTransfer(msg.sender,reward) (staking/Farming.sol#174)
        External calls sending eth:
        - withdraw(balanceOf(msg.sender)) (staking/Farming.sol#165)
                - (success,returndata) = target.call{value: value}(data) (openzeppelin/contracts/utils/Address.sol#119)
        - getAllRewards() (staking/Farming.sol#166)
                - (success,returndata) = target.call{value: value}(data) (openzeppelin/contracts/utils/Address.sol#119)
        State variables written after the call(s):
        - getAllRewards() (staking/Farming.sol#166)
                - tr.rewards[msg.sender] = 0 (staking/Farming.sol#173)
                - tr.rewardPerTokenStored = rewardPerToken(i) (staking/Farming.sol#115)
                - tr.lastUpdateTime = lastTimeRewardApplicable(i) (staking/Farming.sol#116)
                - tr.rewards[account] = earned(i,account) (staking/Farming.sol#118)
                - tr.userRewardPerTokenPaid[account] = tr.rewardPerTokenStored (staking/Farming.sol#119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
```

Issue showing that an uninitialized state variable can cause an exception and abrupt crash of the contract.

```
INFO:Detectors:
Farming.onlyRewardDistribution(uint256) (staking/Farming.sol#88-91) uses a dangerous strict equality:
        - require(bool,string)(msg.sender == tokenRewards[i].rewardDistribution,Access denied: Caller is not reward distribution) (staking/Farming.sol
#89)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
```

```
INFO:Detectors:
PrivateVesting._vestedAmount(address) (vesting/PrivateVesting.sol#155-169) performs a multiplication on the result of a division:
        -totalBalance.div(5).mul(2) (vesting/PrivateVesting.sol#163)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Switch._writeCheckpoint(address,uint32,uint96,uint96) (SwitchToken.sol#357-375) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (SwitchToken.sol#367)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in PrivateVesting.claim() (vesting/PrivateVesting.sol#102-110):
        External calls:
        - Switch.safeTransfer(msg.sender,upfront) (vesting/PrivateVesting.sol#106)
        State variables written after the call(s):
        - _upfrontReleased[msg.sender] = true (vesting/PrivateVesting.sol#107)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

```
INFO:Detectors:
Vesting._vestedAmount(address) (vesting/Vesting.sol#187-200) performs a multiplication on the result of a division:
        -waveNum = block.timestamp.sub(_beneficiaries[beneficiary].cliff).div(WAVE) (vesting/Vesting.sol#197)
        -totalBalance.mul(waveNum).div(totalNumWave) (vesting/Vesting.sol#198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Switch._writeCheckpoint(address,uint32,uint96,uint96) (SwitchToken.sol#357-375) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (SwitchToken.sol#367)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in Vesting.claim() (vesting/Vesting.sol#134-142):
        External calls:
        - Switch.safeTransfer(msg.sender,upfront) (vesting/Vesting.sol#138)
        State variables written after the call(s):
        - _upfrontReleased[msg.sender] = true (vesting/Vesting.sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

The divide before multiply finding indicates this calculation may lose integer precision because of the order of operations.

The issue showing dangerous strict equality is not applicable.

```
INFO:Detectors:
Reentrancy in Vesting.claim() (vesting/Vesting.sol#134-142):
        External calls:
        - Switch.safeTransfer(msg.sender,upfront) (vesting/Vesting.sol#138)
        Event emitted after the call(s):
        - TokenClaimed(msg.sender,upfront) (vesting/Vesting.sol#141)
Reentrancy in Vesting.release() (vesting/Vesting.sol#118-129):
        External calls:
        - Switch.safeTransfer(beneficiary,unreleased) (vesting/Vesting.sol#126)
        Event emitted after the call(s):
        - TokensReleased(beneficiary,unreleased) (vesting/Vesting.sol#128)
Reentrancy in Vesting.revoke(address) (vesting/Vesting.sol#148-166):
        External calls:
        - Switch.safeTransfer(owner(),refund) (vesting/Vesting.sol#163)
        Event emitted after the call(s):
        - TokenVestingRevoked(beneficiary) (vesting/Vesting.sol#165)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
INFO:Detectors:
Reentrancy in PrivateVesting.claim() (vesting/PrivateVesting.sol#102-110):
        External calls:
        - Switch.safeTransfer(msg.sender,upfront) (vesting/PrivateVesting.sol#106)
        Event emitted after the call(s):
        - TokenClaimed(msg.sender,upfront) (vesting/PrivateVesting.sol#109)
Reentrancy in PrivateVesting.release() (vesting/PrivateVesting.sol#86-97):
        External calls:
        - Switch.safeTransfer(beneficiary,unreleased) (vesting/PrivateVesting.sol#94)
        Event emitted after the call(s):
        - TokensReleased(beneficiary,unreleased) (vesting/PrivateVesting.sol#96)
Reentrancy in PrivateVesting.revoke(address) (vesting/PrivateVesting.sol#116-134):
        External calls:
        - Switch.safeTransfer(owner(),refund) (vesting/PrivateVesting.sol#131)
        Event emitted after the call(s):
        - TokenVestingRevoked(beneficiary) (vesting/PrivateVesting.sol#133)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

The issue regarding ReentrancyGuard use has already been documented above.

```
INFO:Detectors:
Farming.getAllRewards() (staking/Farming.sol#179-184) uses timestamp for comparisons
        Dangerous comparisons:
        - i < len (staking/Farming.sol#181)
Farming.notifyRewardAmount(uint256,uint256) (staking/Farming.sol#186-208) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= tr.periodFinish (staking/Farming.sol#192)
        - require(bool,string)(reward.add(leftover) >= duration,Reward is too small) (staking/Farming.sol#198)
        - require(bool,string)(tr.rewardRate <= balance.div(duration),Reward is too big) (staking/Farming.sol#203)
Farming.setDuration(uint256,uint256) (staking/Farming.sol#216-221) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= tr.periodFinish,Not finished yet) (staking/Farming.sol#218)
Farming.addGift(IERC20,uint256,address) (staking/Farming.sol#223-237) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(gift != tokenRewards[i].gift,Gift is already added) (staking/Farming.sol#226)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

FINDINGS & TECH DETAILS

```
INFO:Detectors:
Switch.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (SwitchToken.sol#212-253) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,SWITCH::delegateBySig: signature expired) (SwitchToken.sol#251)
PrivateVesting.addBeneficiary(address,uint256,uint256) (vesting/PrivateVesting.sol#73-81) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(start.add(WAVE_3) > block.timestamp,Vesting: final time is before current time) (vesting/PrivateVesting.sol#75)
PrivateVesting._vestedAmount(address) (vesting/PrivateVesting.sol#155-169) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < _beneficiaries[beneficiary].start (vesting/PrivateVesting.sol#158)
        - block.timestamp >= _beneficiaries[beneficiary].start.add(WAVE_1) && block.timestamp < _beneficiaries[beneficiary].start.add(WAVE_2) (vesting
/PrivateVesting.sol#160)
        - block.timestamp >= _beneficiaries[beneficiary].start.add(WAVE_2) && block.timestamp < _beneficiaries[beneficiary].start.add(WAVE_3) (vesting
/PrivateVesting.sol#162)
        - block.timestamp >= _beneficiaries[beneficiary].start.add(WAVE_3) || _revoked[beneficiary] (vesting/PrivateVesting.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
Switch.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (SwitchToken.sol#212-253) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,SWITCH::delegateBySig: signature expired) (SwitchToken.sol#251)
Vesting.addBeneficiary(address,uint256,uint256,uint256,uint256,uint256) (vesting/Vesting.sol#99-113) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(start.add(duration) > block.timestamp,Vesting: final time is before current time) (vesting/Vesting.sol#105)
Vesting.release() (vesting/Vesting.sol#118-129) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(unreleased > 0,Vesting: no tokens are due) (vesting/Vesting.sol#122)
Vesting._vestedAmount(address) (vesting/Vesting.sol#187-200) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < _beneficiaries[beneficiary].cliff (vesting/Vesting.sol#190)
        - block.timestamp >= _beneficiaries[beneficiary].start.add(_beneficiaries[beneficiary].duration) || _revoked[beneficiary] (vesting/Vesting.so
#192)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

The issue regarding block.timestamp has already been mentioned above.

```
INFO:Detectors:
greet() should be declared external:
        - Greeter.greet() (Greeter.sol#15-17)
setGreeting(string) should be declared external:
        - Greeter.setGreeting(string) (Greeter.sol#19-22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

```
INFO:Detectors:
mint(address,uint256) should be declared external:
        - Switch.mint(address,uint256) (SwitchToken.sol#86-90)
burn(address,uint256) should be declared external:
        - Switch.burn(address,uint256) (SwitchToken.sol#93-96)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (openzeppelin/contracts/token/ERC20/ERC20.sol#115-118)
        - Switch.transfer(address,uint256) (SwitchToken.sol#136-140)
```

```
cliff(address) should be declared external:
        - Vesting.cliff(address) (vesting/Vesting.sol#60-62)
start(address) should be declared external:
        - Vesting.start(address) (vesting/Vesting.sol#67-69)
duration(address) should be declared external:
        - Vesting.duration(address) (vesting/Vesting.sol#74-76)
revocable() should be declared external:
        - Vesting.revocable() (vesting/Vesting.sol#81-83)
released(address) should be declared external:
        - Vesting.released(address) (vesting/Vesting.sol#88-90)
revoked(address) should be declared external:
        - Vesting.revoked(address) (vesting/Vesting.sol#95-97)
addBeneficiary(address,uint256,uint256,uint256,uint256,uint256) should be declared external:
        - Vesting.addBeneficiary(address,uint256,uint256,uint256,uint256,uint256) (vesting/Vesting.sol#99-113)
release() should be declared external:
        - Vesting.release() (vesting/Vesting.sol#118-129)
claim() should be declared external:
        - Vesting.claim() (vesting/Vesting.sol#134-142)
revoke(address) should be declared external:
        - Vesting.revoke(address) (vesting/Vesting.sol#148-166)
finalizeContract() should be declared external:
        - Vesting.finalizeContract() (vesting/Vesting.sol#171-173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

```
start(address) should be declared external:
        - PrivateVesting.start(address) (vesting/PrivateVesting.sol#48-50)
revocable() should be declared external:
        - PrivateVesting.revocable() (vesting/PrivateVesting.sol#55-57)
released(address) should be declared external:
        - PrivateVesting.released(address) (vesting/PrivateVesting.sol#62-64)
revoked(address) should be declared external:
        - PrivateVesting.revoked(address) (vesting/PrivateVesting.sol#69-71)
addBeneficiary(address,uint256,uint256) should be declared external:
        - PrivateVesting.addBeneficiary(address,uint256,uint256) (vesting/PrivateVesting.sol#73-81)
release() should be declared external:
        - PrivateVesting.release() (vesting/PrivateVesting.sol#86-97)
claim() should be declared external:
        - PrivateVesting.claim() (vesting/PrivateVesting.sol#102-110)
revoke(address) should be declared external:
        - PrivateVesting.revoke(address) (vesting/PrivateVesting.sol#116-134)
finalizeContract() should be declared external:
        - PrivateVesting.finalizeContract() (vesting/PrivateVesting.sol#139-141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Issues regarding public functions have been already documented in the

Findings section.

Recommendations:

Use the nonReentrant modifier to prevent reentrancy attacks.

3.6 AUTOMATED SECURITY SCAN

MYTHX:

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. Only security-related findings are shown below.

Results:

Greeter.sol

Report for Greeter.sol
https://dashboard.mythx.io/#/console/analyses/90e14070-0404-406b-b28f-8c18cf83adea

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
8	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
15	(SWC-000) Unknown	Medium	Function could be marked as external.
19	(SWC-000) Unknown	Medium	Function could be marked as external.

SwitchToken.sol

Report for SwitchToken.sol
https://dashboard.mythx.io/#/console/analyses/6348196f-b8ba-4606-86e1-1cdf79d25dec

Line	SWC Title	Severity	Short Description
86	(SWC-000) Unknown	Medium	Function could be marked as external.
93	(SWC-000) Unknown	Medium	Function could be marked as external.
136	(SWC-000) Unknown	Medium	Function could be marked as external.
281	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
365	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

staking/Farming.sol

Report for staking/Farming.sol
https://dashboard.mythx.io/#/console/analyses/f32d2ca6-b81d-443a-ac54-b0ca9f711e92

Line	SWC Title	Severity	Short Description
15	(SWC-103) Floating Pragma	Low	A floating pragma is set.
27	(SWC-000) Unknown	Medium	Incorrect ERC20 implementation
66	(SWC-000) Unknown	Medium	Incorrect ERC20 implementation
100	(SWC-128) DoS With Block Gas Limit	Low	Potentially unbounded data structure passed to builtin.
104	(SWC-128) DoS With Block Gas Limit	Low	Potentially unbounded data structure passed to builtin.
113	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
152	(SWC-000) Unknown	Medium	Function could be marked as external.
181	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.

vesting/PrivateVesting.sol

Report for vesting/PrivateVesting.sol
https://dashboard.mythx.io/#/console/analyses/5e09a293-5149-494d-aa93-46770c40a1c3

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
48	(SWC-000) Unknown	Medium	Function could be marked as external.
55	(SWC-000) Unknown	Medium	Function could be marked as external.
62	(SWC-000) Unknown	Medium	Function could be marked as external.
69	(SWC-000) Unknown	Medium	Function could be marked as external.
73	(SWC-000) Unknown	Medium	Function could be marked as external.
86	(SWC-000) Unknown	Medium	Function could be marked as external.
102	(SWC-000) Unknown	Medium	Function could be marked as external.
116	(SWC-000) Unknown	Medium	Function could be marked as external.
139	(SWC-000) Unknown	Medium	Function could be marked as external.

vesting/Vesting.sol

Report for vesting/Vesting.sol
https://dashboard.mythx.io/#/console/analyses/6348196f-b8ba-4606-86e1-1cdf79d25dec

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
60	(SWC-000) Unknown	Medium	Function could be marked as external.
67	(SWC-000) Unknown	Medium	Function could be marked as external.
74	(SWC-000) Unknown	Medium	Function could be marked as external.
81	(SWC-000) Unknown	Medium	Function could be marked as external.
88	(SWC-000) Unknown	Medium	Function could be marked as external.
95	(SWC-000) Unknown	Medium	Function could be marked as external.
99	(SWC-000) Unknown	Medium	Function could be marked as external.
118	(SWC-000) Unknown	Medium	Function could be marked as external.
134	(SWC-000) Unknown	Medium	Function could be marked as external.
148	(SWC-000) Unknown	Medium	Function could be marked as external.
171	(SWC-000) Unknown	Medium	Function could be marked as external.

THANK YOU FOR CHOOSING

HALBORN