Spherium Finance - Vesting Contract

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 23th, 2021 - August 28th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/25/2021	Juned Ansari
0.9	Document Creation	08/26/2021	Juned Ansari
1.0	Final Review	08/31/2021	Gabi Urrutia
1.1	Remediation Plan	10/27/2021	Alessandro Cara
1.2	Remediation Plan Review	10/28/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Juned Ansari	Halborn	Juned.Ansari@halborn.com
Alessandro Cara	Halborn	alessandro.cara@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Spherium Finance offers a complete suite of financial services comprising a universal wallet, token swap platform, money markets, and inter-blockchain liquidity transfer.

Spherium engaged Halborn to conduct a security assessment on their Vesting smart contracts beginning on August 23th, 2021 and ending August 28th, 2021. This security assessment was scoped to the Vesting smart contracts code in Solidity.

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 development.

1.2 AUDIT SUMMARY

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

The purpose of this audit to achieve the following:

- Ensure that all Vesting Contract functions are intended.
- Identify potential security issues with the assets in scope.

In summary, Halborn identified several security risk that were addressed and accepted by Spherium team.

1.3 TEST APPROACH & METHODOLOGY

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

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

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. 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 : Spherium Vesting-Contract

The security assessment was scoped to the following smart contract:

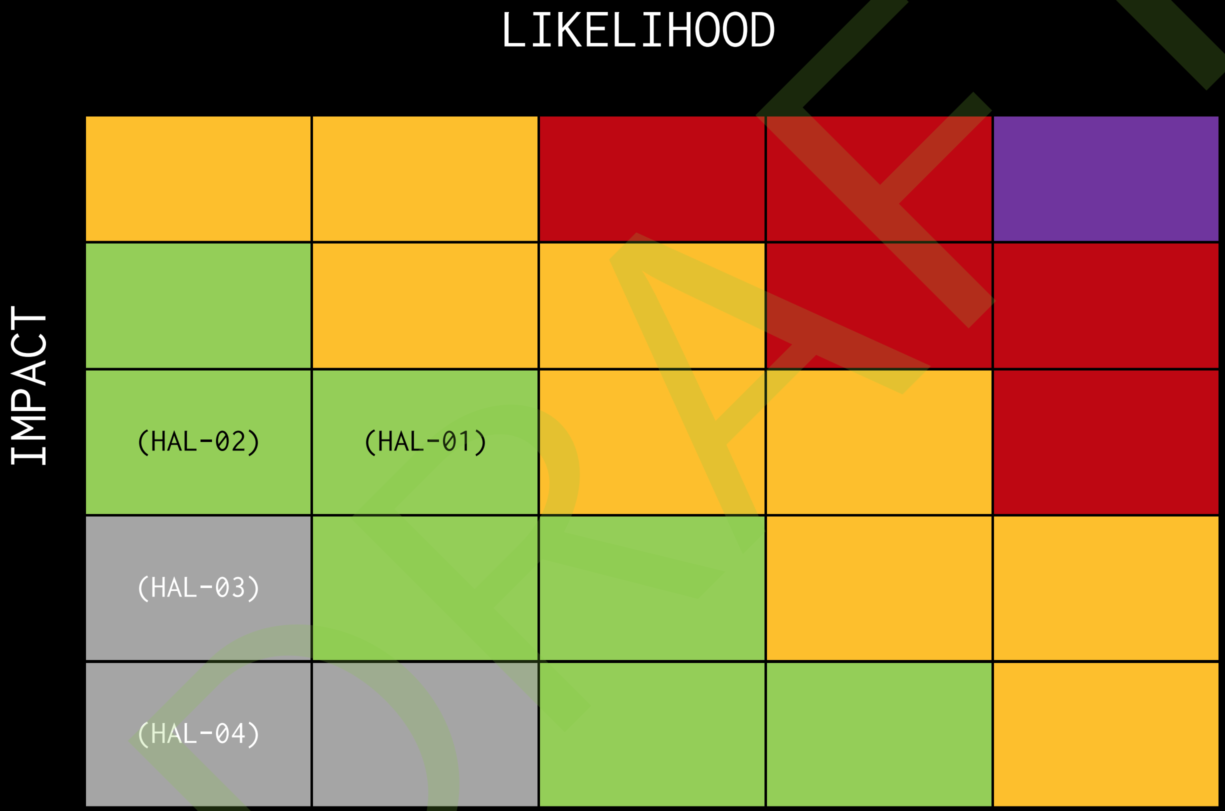
Listing 1: Vesting Contract

```
/vesting-contract/contracts/SphrVestingStatic.sol
/vesting-contract/contracts/SphrVesting.sol
/vesting-contract/contracts/interfaces/ITimelock.sol
/vesting-contract/contracts/interfaces/ISph.sol
```

OUT-OF-SCOPE : External libraries and economics attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - USAGE OF BLOCK-TIMESTAMP	Low	RISK ACCEPTED
HAL02 - FLOATING PRAGMA	Low	SOLVED 10/27/2021
HAL03 - MISSING EVENTS EMITTING	Informational	ACKNOWLEDGED
HAL04 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) USAGE OF BLOCK-TIMESTAMP - LOW

Description:

During a manual review, usage of block.timestamp in SphrVestingStatic.sol and SphrVesting.sol were observed. The contract developers should be aware that this does not mean current time. now is an alias for block.timestamp. The value of block.timestamp can be influenced by miners to a certain degree, so the testers should be warned that this may have some risk if miners collude on time manipulation to influence the price oracles. Miners can influence the timestamp by a tolerance of 900 seconds.

Code Location:

SphrVestingStatic.sol

Listing 2: SphrVestingStatic.sol (Lines 45)

```
function vestedAmount() public view virtual returns (uint256) {
        VestingSchedule[] memory vestingSchedules_ =
            _vestingSchedules[msg.sender];

        uint256 vestedAmount_;
        for(uint256 i=0; i<vestingSchedules_.length; i++) {
            if (vestingSchedules_[i].schedule < block.timestamp) {
                vestedAmount_ = vestingSchedules_[i].amount.add(
                    vestedAmount_);
            }


        }
        vestedAmount_ = vestedAmount_.sub(_releaseAmount[msg.
            sender]);
        return vestedAmount_;
    }
```

SphrVestingStatic.sol

Listing 3: SphrVestingStatic.sol (Lines 59)

```
54     function claim() public returns (bool) {
55         VestingSchedule[] memory vestingSchedules_ =
               _vestingSchedules[msg.sender];
56
57         uint256 vestedAmount_;
58         for(uint256 i=0; i<vestingSchedules_.length; i++) {
59             if (vestingSchedules_[i].schedule < block.timestamp) {
60                 vestedAmount_ = vestingSchedules_[i].amount.add(
                       vestedAmount_);
61                 delete vestingSchedules_[i];
62             }
63         }
64         vestedAmount_ = vestedAmount_.sub(_releaseAmount[msg.
               sender]);
65         require(vestedAmount_ > 0, "SphrVestingStatic: vested
               amount must be greater then 0");
66         _token.safeTransfer(msg.sender, vestedAmount_);
67         _releaseAmount[msg.sender] =  _releaseAmount[msg.sender].
               add(vestedAmount_);
68         return true;
69     }
```

SphrVesting.sol

Listing 4: SphrVesting.sol (Lines 90,92)

```
86     function _vestedAmount() private view returns (uint256) {
87         // uint256 currentBalance = _token.balanceOf(address(this)
               );
88         // uint256 totalBalance = currentBalance.add(_released[
               address(token)]);
89
90         if (block.timestamp < _cliff) {
91             return 0;
92         } else if (block.timestamp >= _start.add(_duration)) {
93             return _share[msg.sender];
94         } else {
95             return _share[msg.sender].mul(block.timestamp.sub(
                   _start)).div(_duration);
96         }
97     }
```

Risk Level:

Likelihood - 2
Impact - 3

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:

RISK ACCEPTED: Spherium team accepts this risk.

3.2 (HAL-02) FLOATING PRAGMA - LOW

Description:

Vesting interfaces contract ISph.sol and ITimelock.sol uses the floating pragma ˆ0.6.12. Contract should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too new which has not been extensively tested.

Code Location:

Listing 5: (Lines 1)

```
pragma solidity ^0.6.12;
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Consider locking the pragma version with known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

SOLVED: Spherium team locked the pragma version in commit c2394c97f6a29858bebf316663cf393f9400593b.

3.3 (HAL-03) MISSING EVENTS EMITTING - INFORMATIONAL

Description:

It has been observed that important functionality is missing emitting event for some functions on the SphrVestingStatic.sol contract. These functions should emit events.

Code Location:

Listing 6: Missing Events

```
function claim() public returns (bool)
function vestedAmount() public view virtual returns (uint256)
function addVestingSchedule(address benificiary, uint256 amount,
    uint256 schedule) public onlyOwner returns (bool)

```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Consider emitting an event when calling related functions on the list above.

Remediation Plan:

ACKNOWLEDGED: Spherium team acknowledges this issue.

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.
Also, methods do not necessarily have to be public if they are only called within the contract-in such case they should be marked internal.

Code Location:

Below are smart contracts and their corresponding functions affected:

SphrVestingStatic:
tokenContract, addVestingSchedule, vestedAmount, claim, getBlockTimestamp

SphrVesting:
getReleased, getShare, getBlockTimestamp, release

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, all can access to public functions, external functions only can be accessed externally and internal functions can only be called within the contract.

Remediation Plan:

ACKNOWLEDGED: Spherium team acknowledges this issue.

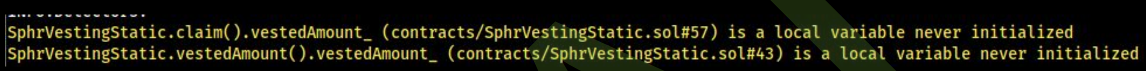
AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

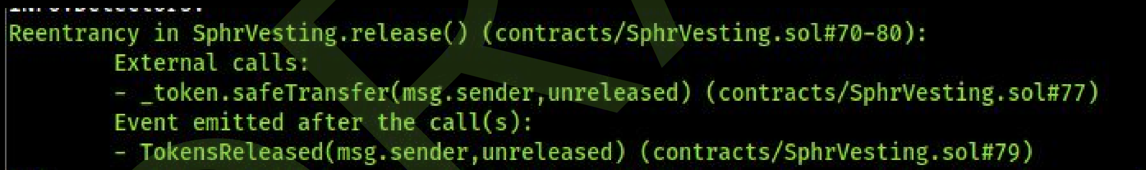
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats. 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:

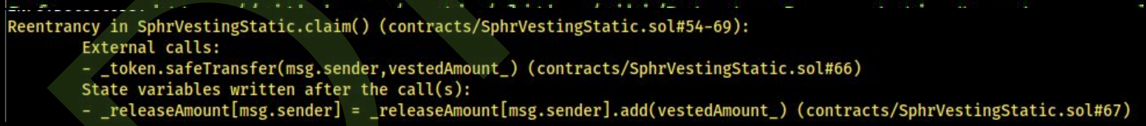


```
SphrVestingStatic.claim().vestedAmount_ (contracts/SphrVestingStatic.sol#57) is a local variable never initialized
SphrVestingStatic.vestedAmount().vestedAmount_ (contracts/SphrVestingStatic.sol#43) is a local variable never initialized
```

-Issue regarding uninitialized local variable is a false positive as uint256 by default initializes variable value zero.

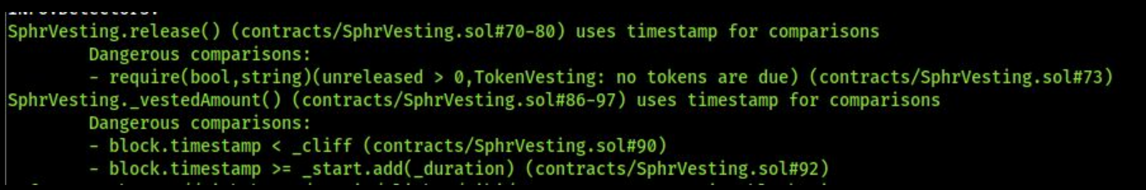


```
Reentrancy in SphrVesting.release() (contracts/SphrVesting.sol#70-80):
        External calls:
        - _token.safeTransfer(msg.sender,unreleased) (contracts/SphrVesting.sol#77)
        Event emitted after the call(s):
        - TokensReleased(msg.sender,unreleased) (contracts/SphrVesting.sol#79)
```

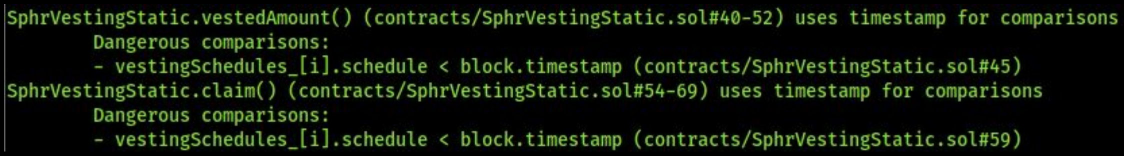


```
Reentrancy in SphrVestingStatic.claim() (contracts/SphrVestingStatic.sol#54-69):
        External calls:
        - _token.safeTransfer(msg.sender,vestedAmount_) (contracts/SphrVestingStatic.sol#66)
        State variables written after the call(s):
        - _releaseAmount[msg.sender] = _releaseAmount[msg.sender].add(vestedAmount_) (contracts/SphrVestingStatic.sol#67)
```

- Issues regarding 'Reentrancy' is a false-positive since the contract making an external call to safeTransfer openzeppelin functions which are considered to be standard and secure to use, calling these functions doesn't lead to fallback before emit events and adding operation.

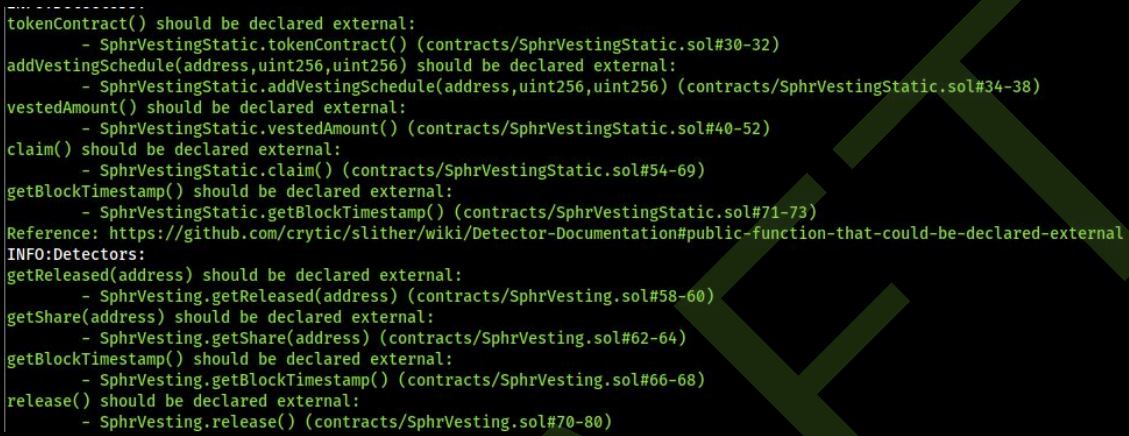


```
SphrVesting.release() (contracts/SphrVesting.sol#70-80) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(unreleased > 0,TokenVesting: no tokens are due) (contracts/SphrVesting.sol#73)
SphrVesting._vestedAmount() (contracts/SphrVesting.sol#86-97) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < _cliff (contracts/SphrVesting.sol#90)
        - block.timestamp >= _start.add(_duration) (contracts/SphrVesting.sol#92)
```



```
SphrVestingStatic.vestedAmount() (contracts/SphrVestingStatic.sol#40-52) uses timestamp for comparisons
        Dangerous comparisons:
        - vestingSchedules_[i].schedule < block.timestamp (contracts/SphrVestingStatic.sol#45)
SphrVestingStatic.claim() (contracts/SphrVestingStatic.sol#54-69) uses timestamp for comparisons
        Dangerous comparisons:
        - vestingSchedules_[i].schedule < block.timestamp (contracts/SphrVestingStatic.sol#59)
```

- Issue regarding usage of block-timestamp has been already mentioned in the above report.



```
tokenContract() should be declared external:
        - SphrVestingStatic.tokenContract() (contracts/SphrVestingStatic.sol#30-32)
addVestingSchedule(address,uint256,uint256) should be declared external:
        - SphrVestingStatic.addVestingSchedule(address,uint256,uint256) (contracts/SphrVestingStatic.sol#34-38)
vestedAmount() should be declared external:
        - SphrVestingStatic.vestedAmount() (contracts/SphrVestingStatic.sol#40-52)
claim() should be declared external:
        - SphrVestingStatic.claim() (contracts/SphrVestingStatic.sol#54-69)
getBlockTimestamp() should be declared external:
        - SphrVestingStatic.getBlockTimestamp() (contracts/SphrVestingStatic.sol#71-73)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Detectors:
getReleased(address) should be declared external:
        - SphrVesting.getReleased(address) (contracts/SphrVesting.sol#58-60)
getShare(address) should be declared external:
        - SphrVesting.getShare(address) (contracts/SphrVesting.sol#62-64)
getBlockTimestamp() should be declared external:
        - SphrVesting.getBlockTimestamp() (contracts/SphrVesting.sol#66-68)
release() should be declared external:
        - SphrVesting.release() (contracts/SphrVesting.sol#70-80)
```

- Issue regarding possible misuse of public functions has been already mentioned in the above report.

According to the test results, some of the findings found by these tools were considered as false positives while some of these findings were real security concerns. All relevant findings were reviewed by the auditors and relevant findings addressed on the report as security concerns.

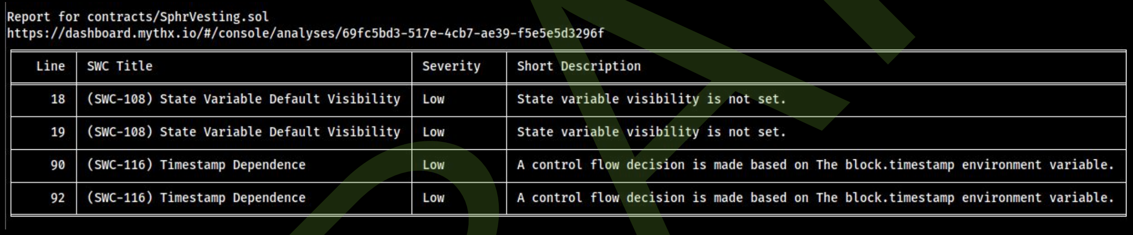
4.2 AUTOMATED SECURITY SCAN

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:

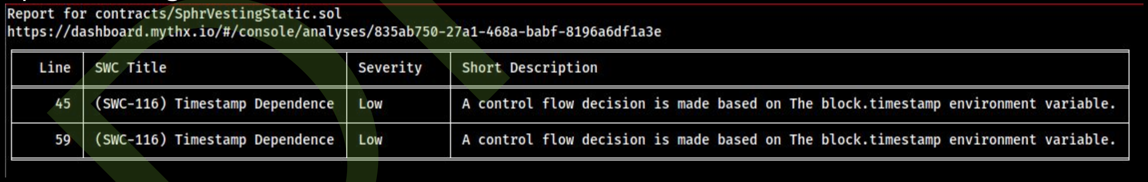
SphrVesting.sol



```
Report for contracts/SphrVesting.sol
https://dashboard.mythx.io/#/console/analyses/69fc5bd3-517e-4cb7-ae39-f5e5e5d3296f
```

Line	SWC Title	Severity	Short Description
18	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
19	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
90	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.
92	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

SphrVestingStatic.sol



```
Report for contracts/SphrVestingStatic.sol
https://dashboard.mythx.io/#/console/analyses/835ab750-27a1-468a-babf-8196a6df1a3e
```

Line	SWC Title	Severity	Short Description
45	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.
59	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

All relevant valid findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN