



Jarvis – AerariumMilitare

Smart Contract Security Audit

Prepared by: Halborn

Date of Engagement: April 30th-May 7th, 2021

Visit: [Halborn.com](https://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	12
Recommendation	12
Remediation Plan	13
3.2 (HAL-02) PRAGMA VERSION - INFORMATIONAL	14
Description	14
Code Location	14
Risk Level	14
Recommendation	14
Remediation Plan	15
3.3 (HAL-03) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL	16
Description	16

	Code Location	16
	Risk Level	16
	Recommendation	16
3.4	KOVAN TESTING	18
	Description	18
	Result	18
3.5	STATIC ANALYSIS REPORT	19
	Description	19
	Results	19
3.6	AUTOMATED SECURITY SCAN	20
	Description	20
	Results	20

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/03/2021	Gabi Urrutia
0.2	Document Edits	05/06/2021	Gabi Urrutia
1.0	Final Version	05/07/2021	Gabi Urrutia
1.1	Remediation plan	05/12/2021	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



EXECUTIVE OVERVIEW



1.1 INTRODUCTION

JARVIS engaged Halborn to conduct a security assessment on a smart contract beginning on April 30th, 2021 and ending May 7th, 2021. The security assessment was scoped to the smart contract provided in the Gitlab repository [JARVIS Smart Contract Repository](#) Halborn conducted this audit to measure security risk and identify any new vulnerabilities introduced during the final stages of development before the JARVIS production release. The security assessment was scoped to the smart contract [AerariumMilitare.sol](#).

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 four weeks for the engagement and assigned three full time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts 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 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 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 ([Kovan](#))

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:

- `AerariumMilitare.sol`

Commit ID: `4657416fa5e6cfe6111fe2b4e7b83b0a318aac76`

Fixed Commit ID: `8646d2c01a02175ee001f64f1a921fea23600d65`

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	1	2

LIKELIHOOD

IMPACT

	(HAL-01)			
(HAL-02)				
(HAL-03)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
FLOATING PRAGMA	Low	RISK ACCEPTED: 05/11/2021
PRAGMA VERSION	Informational	SOLVED: 05/11/2021
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED: 05/11/2021
KOVAN TESTING	-	-
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN RESULTS	-	-



FINDINGS & TECH DETAILS



3.1 (HAL-01) FLOATING PRAGMA - LOW

Description:

JARVIS contract uses the floating pragma `^8.0.0`. 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:

`AerariumMilitare.sol` Line #3

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
4
```

Reference: [ConsenSys Diligence - Lock pragmas](#)

Risk Level:

Likelihood - 2

Impact - 3

Recommendation:

Consider locking the pragma version. It is not recommended to use floating pragma in production. Apart from just locking the pragma version in the code, the sign (^) need to be removed. 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:

RISK ACCEPTED: Jarvis fixed the pragma version in `truffle-config`.

3.2 (HAL-02) PRAGMA VERSION - INFORMATIONAL

Description:

The JARVIS contract uses one of the latest pragma version (0.8.0) which was released Dec 16, 2020. The latest pragma version is (0.8.4) and it was released April 2021. Many pragma versions have been released, going from version 0.6.x to the recently released version 0.8.x. in just 6 months.

Reference: <https://github.com/ethereum/solidity/releases>

Code Location:

AerariumMilitare.sol Line #3

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
4
```

Risk Level:

Likelihood - 1

Impact - 2

Recommendation:

In the Solitidy Github repository, there is a json file listing the bugs reported for each compiler version. No bugs have been found in > 0.7.3 versions and very few in 0.7.0 -- 0.7.3. The latest stable version is pragma 0.6.12. Furthermore, pragma 0.6.12 is widely used by Solidity developers and has been extensively tested in many security audits. We recommend using at minimum the latest stable version.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Remediation Plan:

SOLVED: Jarvis team changed the pragma version to 0.7.3.

3.3 (HAL-03) 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:

AerariumMilitare.sol Lines #76, 87

```

76     function claim() public {
77         require(block.timestamp < endTime, "The end time has passed so liquidate should be called to withdraw");
78         uint256 totalAmount = investorInfo[msg.sender];
79         uint256 timePassed = block.timestamp.sub(startTime);
80         uint256 amount = timePassed.mul(totalAmount).div(lockTime).sub(claimedAmount[msg.sender]);
81         claimedAmount[msg.sender] = claimedAmount[msg.sender].add(amount);
82         token.safeTransfer(msg.sender, amount);
83     }

87     function liquidate(address[] calldata _investors) public {      Iliyan Iliev, 2 weeks ago • [investors-token-lock
88         require(block.timestamp >= endTime, "The end time has passed, so we can automatically withdraw all");
89         for(uint256 i=0; i<_investors.length; i++){
90             _liquidate(_investors[i]);
91         }
92     }

```

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

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.

SOLVED: Jarvis team marked `claim()` and `liquidate()` functions as external.

3.4 KOVAN TESTING




Description:

Testing Smart Contract in testnets such as Kovan and Rinkeby is very useful for development team and auditors. Testnet deployments allow checking how, approximately, the behavior of smart contracts on the mainnet will be.

JARVIS team funded a Kovan account (`0xA7CA1409C5306F77914108271Af95CcC0C84722f`) with 100000 JRT tokens. Then, `AerariumMilitare.sol` contract was deployed in Kovan and one-day time distribution was assigned.

First, an attempt was made to withdraw the tokens from another account and it was not possible. Transaction failed because there was not enough balance.

Then, JRT tokens were claimed by the funded account and it worked. Otherwise, a minor issue happened when operation were tried too fast. Apparently, it was a user error.

 <code>0x56abe51680eae5a394...</code>	Claim	24654194	1 min ago	<code>0xa7ca1409c5306f7791...</code>	OUT	<code>0xf29ef982e2eededf2dc...</code>	0 Ether	0.000069
 <code>0xaaade0ba845df3a9c6...</code>	Claim	24654187	1 min ago	<code>0xa7ca1409c5306f7791...</code>	OUT	<code>0xf29ef982e2eededf2dc...</code>	0 Ether	0.000151497
 <code>0x339a6834f5c115bd5cf...</code>	Claim	24654182	2 mins ago	<code>0xa7ca1409c5306f7791...</code>	OUT	<code>0xf29ef982e2eededf2dc...</code>	0 Ether	0.130998

Finally, JRT tokens were claimed after the `endTime`, and operation was reverted.

Result:

It was verified that the contract works correctly in testnet.

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:

AerariumMilitare.sol

```
INFO:Detectors:
AerariumMilitare.constructor(IERC20,uint256,uint256) (AerariumMilitare.sol#38-46) uses timestamp for comparisons
  Dangerous comparisons:
    - require(bool,string)(block.timestamp < startTime,Start time must be after actual time) (AerariumMilitare.sol#40)
AerariumMilitare.addInvestors(address[],uint256[]) (AerariumMilitare.sol#53-66) uses timestamp for comparisons
  Dangerous comparisons:
    - require(bool,string)(block.timestamp < startTime,Current time should be before the start of the distribution) (AerariumMilitare.sol#54)
AerariumMilitare.claim() (AerariumMilitare.sol#76-83) uses timestamp for comparisons
  Dangerous comparisons:
    - require(bool,string)(block.timestamp < endTime,The end time has passed so liquidate should be called to withdraw) (AerariumMilitare.sol#77)
AerariumMilitare.liquidate(address[]) (AerariumMilitare.sol#87-92) uses timestamp for comparisons
  Dangerous comparisons:
    - require(bool,string)(block.timestamp >= endTime,The end time has passed, so we can automatically withdraw all) (AerariumMilitare.sol#88)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (Address.sol#26-35) uses assembly
  - INLINE ASM (Address.sol#33)
Address._verifyCallResult(bool,bytes,string) (Address.sol#171-188) uses assembly
  - INLINE ASM (Address.sol#180-183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version^0.7.4 (Address.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version^0.7.4 (AerariumMilitare.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version^0.7.4 (Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version^0.7.4 (IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version^0.7.4 (Ownable.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version^0.7.4 (SafeERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version^0.7.4 (SafeMath.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
solc-0.7.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (Address.sol#53-59):
  - (success) = recipient.call{value: amount}() (Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Address.sol#114-121):
  - (success,returndata) = target.call{value: value}(data) (Address.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (Address.sol#139-145):
  - (success,returndata) = target.staticcall(data) (Address.sol#143)
Low level call in Address.functionDelegateCall(address,bytes,string) (Address.sol#163-169):
  - (success,returndata) = target.delegatecall(data) (Address.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter AerariumMilitare.addInvestors(address[],uint256[]).addresses (AerariumMilitare.sol#53) is not in mixedCase
Parameter AerariumMilitare.addInvestors(address[],uint256[]).tokens (AerariumMilitare.sol#53) is not in mixedCase
Parameter AerariumMilitare.liquidate(address[]).investors (AerariumMilitare.sol#87) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
addInvestors(address[],uint256[]) should be declared external:
  - AerariumMilitare.addInvestors(address[],uint256[]) (AerariumMilitare.sol#53-66)
claim() should be declared external:
  - AerariumMilitare.claim() (AerariumMilitare.sol#76-83)
liquidate(address[]) should be declared external:
  - AerariumMilitare.liquidate(address[]) (AerariumMilitare.sol#87-92)
renounceOwnership() should be declared external:
  - Ownable.renounceOwnership() (Ownable.sol#54-57)
transferOwnership(address) should be declared external:
  - Ownable.transferOwnership(address) (Ownable.sol#63-67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:AerariumMilitare.sol analyzed (7 contracts with 45 detectors), 26 result(s) found
```

No new issues were found by Slither.

3.6 AUTOMATED SECURITY SCAN

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. In addition, security detections are only in scope.

Results:

AerariumMilitare.sol

```
analyze AerariumMilitare.sol
Found 1 job(s). Submit? [y/N]: y
[#####] 100%
Report for AerariumMilitare.sol
https://dashboard.mythx.io/#/console/analyses/5cee8584-3cdf-433b-b6ba-b213d7a49c81
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
38	(SWC-100) Function Default Visibility	Low	Function visibility is not set.
76	(SWC-000) Unknown	Medium	Function could be marked as external.
87	(SWC-000) Unknown	Medium	Function could be marked as external.



THANK YOU FOR CHOOSING

// HALBORN

