hack·a·chain

SECURITY AUDIT REPORT

in favor of META FIGHTER

Summary

This report has been prepared for MetaFighters in the source code of the project as well as in project dependencies that are not part of an officially recognized library. The audit has been conducted by combining static and dynamic code analysis with a manual review of the source code by Hack-a-Chain's research team.

The audit process analyses:

1) Adherence to widely recognized best practices and industry standards;
2) Vulnerability to most common attack vectors;
3) Thorough line-by-line review of the code base;
4) Ensuring that contract logic meets the specifications of the project's whitepaper.

The security audit result is composed of different findings, whose vulnerabilities are classified from critical to informational, according to the following impact versus likelihood matrix:

Impact	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low / Informational

Likelihood

After presenting the findings to the client, they are granted a 7 days period to fix the vulnerabilities. This report will specify all vulnerabilities found and whether they were fixed by the team.

Overview

Project Summary

Project Name	Meta Fighter
Description	Vesting contract for the project
Platform	BSC (Binance Smart Chain protocol, EVM)
Language	Solidity
Codebase	https://gitlab.com/fruktorum/backend/meta-fighter/contracts/test
Commit	3cfbb4251d2ceb4ae8754bc089195d42bb50f37e

Audit Summary

First delivery date	03/29/2022
Final delivery date	04/11/2022
Audit Methodology	Manual review combined with static/dynamic analysis

Audit Scope

ID	File	SHA256 Checksum
VES	Vesting.sol	90f3fde26187801b88dafe65e660f48d1b9e8c05e151ca7664b1f80d0d714bf3

Findings

ID	Title	Category	Severity	Status
VES-1	External vs. Public functions	Gas optimization	Informational	Alleviated
VES-2	Unchecked release	Logical issue	Low	Acknowledged
VES-3	Emitting events	Best practice	Informational	Alleviated

VES-1 - External vs. Public functions

Category	Severity	Location	Status
Gas optimization	Informational	Vesting.sol:210, 244, 274, 298, 360, 377, 416, 483	-

Description

Best practices dictate the use of `external` functions instead of `public` ones when they'll only ever be called from outside the contract. The recommendation arises from the fact that `external` functions provide gas cost savings in comparison to `public` functions.

Recommendation

We advise the use of `external` functions wherever possible, that is, when the function is only meant to be called from outside the contract and not from other functions in the contract.

Alleviation

Team changed the function implementations to utilize external keyword

VES-2 - Unchecked release

Category	Severity	Location	Status
Logical issue	Low	Vesting.sol: 421	

Description

The `_computeReleasableAmount` function is used inside a `require()` statement in the `release()` function to assert that the beneficiary is not trying to withdraw more tokens than they are entitled to.

```
function release(bytes32 investorDataId, uint256 amount)
        public
        nonReentrant
        onlyIfNotRevoked(investorDataId)
    {
        InvestorData storage investorData = investorsData[investorDataId];
        require(
            msg.sender == investorData.investor ||
                hasRole(DEFAULT_ADMIN_ROLE, msg.sender),
            "Vesting: only investor and admin can release vested tokens"
        );
        uint256 vestedAmount = _computeReleasableAmount(investorData);
        require(
            vestedAmount >= amount,
            "Vesting: cannot release tokens, not enough vested tokens"
        );
        if (amount != 0) {
            if (investorData.cliffPaid) {
                investorData.released = investorData.released + amount;
                vestingTotalAmount = vestingTotalAmount - amount;
            } else {
                investorData.released =
                    (investorData.released + amount) -
                    investorData.amountAfterCliff;
                vestingTotalAmount =
                    (vestingTotalAmount + investorData.amountAfterCliff) -
                    amount;
            }
            investorData.cliffPaid = true;
            address payable investorPayable = payable(investorData.investor);

            _token.safeTransfer(investorPayable, amount);
        }
    }
```

However, the `_computeReleasableAmount` function will not consider all previous withdrawals by the beneficiary if `investorData.cliffPaid` evaluates to false.

```
function _computeReleasableAmount(InvestorData memory investorData)
        internal
        view
        returns (uint256)
    {
        uint256 currentTime = getCurrentTime();
        VestingPhase memory vestingPhase =
vestingPhases[investorData.phaseID];

        if (
            (currentTime < vestingPhase.cliff) || (investorData.amount == 0)
// If cliff not finished or total amount = 0 (schedule was canceled)
        ) {
            return 0;
        } else if (currentTime >= vestingPhase.cliff + vestingPhase.duration)
{
            // If vesting period finished
            return investorData.amount - investorData.released;
        } else {
            uint256 timeFromStart = currentTime - vestingPhase.cliff;
            uint256 secondsPerSlice = vestingPhase.slicePeriodSeconds;
            uint256 vestedSlicePeriods = timeFromStart / secondsPerSlice;
            uint256 vestedSeconds = vestedSlicePeriods * secondsPerSlice;
            uint256 vestedAmount = (investorData.amount * vestedSeconds) /
                vestingPhase.duration;
            if (investorData.cliffPaid) {
                vestedAmount = vestedAmount - investorData.released;
            } else {
                vestedAmount = vestedAmount + investorData.amountAfterCliff;
            }
            return vestedAmount;
        }
    }
```

At first, this should not be a problem, because at the first time that the user has funds released from the `release()` function it reassigns the `cliffPaid` parameter of `investorData` to `true`.
However, an issue might eventually arise when the `changeInvestorSchedule()` function is called. This function requires the caller to select a new value for `cliffPaid`, which might cause one of the following problems:

(1) If cliff had not yet been paid and is changed to paid, the investor will no longer be able to withdraw the `amountAfterCliff` unless the `changeInvestorSchedule()` function is called again to fix the issue.
(2) If cliff has been paid and is changed back to false, the investor will be able to withdraw again not only the `amountAfterCliff` but all of the funds previously withdrawn.

Both of this exploit scenarios require a mistake by the part of the contract administrator, which is why the severity of the issue is marked as low, but such a

scenario is likely in a context of a fast moving web3 startup and should therefore be taken into account at a security audit.

Recommendation

First of all, it would be advisable to reduce the `investorData.released` amount even if cliffPaid is false inside the `_computeReleasableAmount` function. In normal instances `investorData.released` will be 0 and will not change the result of the function, but might protect the contract in the case of exploit (2) described above.

```
else {
            uint256 timeFromStart = currentTime -
vestingPhase.cliff;
            uint256 secondsPerSlice =
vestingPhase.slicePeriodSeconds;
            uint256 vestedSlicePeriods = timeFromStart /
secondsPerSlice;
            uint256 vestedSeconds = vestedSlicePeriods *
secondsPerSlice;
            uint256 vestedAmount = (investorData.amount *
vestedSeconds) /
                vestingPhase.duration;
            vestedAmount = vestedAmount - investorData.released;
            if (!investorData.cliffPaid) {
                vestedAmount = vestedAmount +
investorData.amountAfterCliff;
            }
            return vestedAmount;
        }
```

In second place, it would be advisable to change the contract logic to also track the release of `amountAfterCliff`. Otherwise, the contract will always be exposed to flaws in the changing `investorData.cliffPaid` caused by the calling of the `changeInvestorSchedule()` function.

Alleviation

In face of time constraints, team chose not to implement the recommendation, but assured that they are aware of the dangers of calling `changeInvestorSchedule()` and will only do so in extraordinary conditions with the uttermost caution to ensure safety of the assets.

VES-4 - Emitting events

Category	Severity	Location	Status
Best practices	Informational	-	-

Description

Best practices dictate the use of events to track relevant actions performed in the code. In the case of this contract those might be the creation, alteration, deletion of vesting phases, investors and schedules, and release/withdrawal of funds from the contract.
Although not necessary for contract security it is considered a best practice for the fetching of contract information and ease of debugging in case of future contract errors/attacks.

Recommendation

We advise the addition of events to the contract logic, so that users and the team can better keep track of important actions being performed

Alleviation

Team implemented event emissions for all relevant functions.

Appendix

Finding Categories

Code Style

Code style issues refer to the implementation of the contract code deviating from industry standards and best practices in commenting, documenting, naming, formatting and other style components.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Checksum

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specific commit.
The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Issue Checking Status

N°	Issue Description	Checking Status
1	Compiler Warnings.	Passed
2	Race conditions and Reentrancy. Cross-function race conditions.	Passed
3	Possible delays in data delivery.	Passed
4	Oracle calls.	Passed
5	Front running.	Passed
6	Timestamp dependence.	Passed
7	Integer Overflow and Underflow.	Passed
8	DoS with Revert.	Passed
9	DoS with block gas limit.	Passed
10	Methods execution permissions.	Passed
11	Economy model.	Passed
12	The impact of the exchange rate on logic.	Passed
13	Private user data leaks.	Passed
14	Malicious Event log.	Passed
15	Scoping and Declarations.	Passed
16	Uninitialized storage pointers.	Passed
17	Arithmetic accuracy.	Passed
18	Design Logic.	Passed
19	Cross-function race conditions.	Passed
20	Safe Zeppelin Module.	Passed
21	Fallback function security.	Passed
22		
23		
24		
25		

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Hack-a-Chain's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Hack-a-Chain to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intended to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Hack-a-Chain's position is that each company and individual are responsible for their own due diligence and continuous security.

Hack-a-Chain's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Hack-a-Chain are subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are

emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS,OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, HACK-A-CHAIN HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, HACK-A-CHAIN SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, HACK-A-CHAIN MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, HACK-A-CHAIN PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED. WITHOUT LIMITING THE FOREGOING, NEITHER HACK-A-CHAIN NOR ANY OF HACK-A-CHAIN’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. HACK-A-CHAIN WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY

OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT HACK-A-CHAIN'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST HACK-A-CHAIN WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF HACK-A-CHAIN CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST HACK-A-CHAIN WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

SECURITY AUDIT CERTIFICATE

MetaFighter

We, Hack-a-Chain, Blockchain Specialist Software Development and Audit Company, in this act represented by our Chief Technology Officer, João Antônio Schmidt da Veiga, grant this **Security Audit Certificate** in favor of **MetaFighter**, recognizing that they have passed throw the security audit process and corrected all the issues that have been found in the following smart contracts:

1. **Vesting**

The full security audit report and it's disclaimer can be found in the following link:

https://github.com/hack-a-chain/security-audits

Devoted to enhancing security in the Blockchain Ecosystem and to provide the best quality service for our clients and the community, we sign this Certificate:

João Antônio Schmidt da Veiga
Chief Technology Officer