hack·a·chain

SECURITY AUDIT REPORT

in favor of DELIQ FINANCE

Deliq FINANCE

Summary

This report has been prepared for Deliq Finance 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	Deliq Finance
Description	Vesting contract for project's native tokens
Platform	AVAX (Avalanch protocol, EVM)
Language	Solidity
Codebase	https://github.com/SmartDev84/delphi-vesting-contract.git
Commit	a424bcb3dc845cba0976352ee0570ea651661e47
Deployment Address	0x613bC531573520D8B7AaE71C5B812cf0Eb0778e7

Audit Summary

First delivery date	03/28/2022
Final delivery date	03/29/2022
Audit Methodology	Manual review combined with static/dynamic analysis

Audit Scope

ID	File	SHA256 Checksum
VES	TokenVesting.sol	e95b58301d6c82ef77e12bcf1b2d8113b26c01ea4aa8415af400ec143716205b

Findings

ID	Title	Category	Severity	Status
VES-1	Division before multiplication	Integer math	Low	Pending
VES-2	Unchecked release	Logical exploit	High	Pending
VES-3	Unnecessary use of SafeMath	External libraries	Informational	Pending

VES-1 - Division before multiplication

Category	Severity	Location	Status
Integer Math	Low	TokenVesting.sol: 428	Fixed by team

Description

The internal variable `vestedSeconds` in the `_computeReleasableAmount` function is the result of a multiplication on a previously divided integer. Best practices for integer math require all multiplications to be performed before divisions, to prevent rounding errors from altering the results substantially.

Moreover, the `vestedSeconds` variable is the result of `vestedSlicePeriods.mul(secondsPerSlice)`, whereas `vestedSlicePeriods` is the result of `timeFromStart.div(secondsPerSlice)`. In more objective terms, `vestedSeconds` could safely be substituted by `timeFromStart`.

Recommendation

We advise the use of the .`timeFromStart` variable instead of `vestedSeconds`, since they contain the same value and the use of `timeFromStart` prevents any math problem with the order of multiplication and division operations.

Alleviation

Team implemented the recommendations on the source code.

VES-2 - Unchecked release

Category	Severity	Location	Status
Logical Exploit	High	TokenVesting.sol: 421	Fixed by team

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 vestingScheduleId,
        uint256 amount
    )
        public
        nonReentrant
        onlyIfVestingScheduleNotRevoked(vestingScheduleId){
        VestingSchedule storage vestingSchedule =
vestingSchedules[vestingScheduleId];
        bool isBeneficiary = msg.sender == vestingSchedule.beneficiary;
        bool isOwner = msg.sender == owner();
        require(
            isBeneficiary || isOwner,
            "TokenVesting: only beneficiary and owner can release vested
tokens"
        );
        uint256 vestedAmount = _computeReleasableAmount(vestingSchedule);
        require(vestedAmount >= amount, "TokenVesting: cannot release tokens,
not enough vested tokens");
        vestingSchedule.released = vestingSchedule.released.add(amount);
        address payable beneficiaryPayable =
payable(vestingSchedule.beneficiary);
        vestingSchedulesTotalAmount =
vestingSchedulesTotalAmount.sub(amount);
        _token.safeTransfer(beneficiaryPayable, amount);
    }
```

However, the `_computeReleasableAmount` function will not consider all previous withdrawals by the beneficiary if the vesting schedule is still behind the cliff date:

```
if ((currentTime < vestingSchedule.cliff)) {
            return
vestingSchedule.amountTotal.mul(vestingSchedule.TGEPercentage).div(100);
        } else if (currentTime >=
vestingSchedule.start.add(vestingSchedule.duration)) {
            return vestingSchedule.amountTotal.sub(vestingSchedule.released);
```

Notice the difference between the if clause (behind cliff date) which does not subtract `vestingSchedule.released` before returning the value. This could essentially allow an ill intended vesting beneficiary to withdraw all funds from the contract using a simple client side exploit such as:

```
let vestingContract = new web3.eth.Contract(
    compiled.abi,
    address
);

while (true) {
    await vestingContract.methods.release(vestingId, amount).send({from:
accounts[0]});
}
```

Conditions for this exploit are: (1) that the exploiter is a vesting beneficiary or gains control of a beneficiary's wallet; (2) that the submitted amount is smaller than the pre cliff releasable amount for the vesting schedule

Recommendation

We advise a change to the contract's code in line 421 from:

```
return
vestingSchedule.amountTotal.mul(vestingSchedule.TGEPercentage).div(1
00);
```

to:

```
return
vestingSchedule.amountTotal.mul(vestingSchedule.TGEPercentage).div(1
00).sub(vestingSchedule.released);
```

Alleviation

Team implemented the recommendations on the source code.

VES-3 - Unnecessary use of SafeMath

Category	Severity	Location	Status
External libraries	Informational	multiple	Fixed by team

Description

Since solidity version 0.8 all underflowing/overflowing operations throw an error and revert the whole transaction. As such, there is no more need to use the SafeMath.sol module from OpenZeppelin contract suite.
The use of SafeMath will perform the overflow/underflow evaluation twice in solidity ^0.8.0 which will cause a slight increase in gas costs.

Recommendation

We advise the use of native solidity mathematical operations instead of the ones in the SafeMath module and the removal of the module from the contract's design altogether.

Alleviation

Team implemented the recommendations on the source code.

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.	VES-1
18	Design Logic.	VES-2
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

Deliq Finance

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 **Deliq Finance**, recognizing that they have passed throw the security audit process and corrected all the issues that have been found in the following smart contract:

1. **Vesting contract for project's native tokens**
   Deployment address: 0x613bC531573520D8B7AaE71C5B812cf0Eb0778e7

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