RADICALLY OPEN SECURITY

Penetration Test Report

Gobalsky Labs Ltd.

V 1.0
Diemen, August 23rd, 2021
Confidential

Document Properties

Client	Gobalsky Labs Ltd.
Title	Penetration Test Report
Target	ERC20_Vesting.sol
Version	1.0
Pentester	Joachim de Koning
Authors	Joachim de Koning, Patricia Piolon
Reviewed by	Patricia Piolon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	April 17th, 2021	Joachim de Koning	Initial draft
1.0	August 23rd, 2021	Patricia Piolon	Review

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Overdiemerweg 28 1111 PP Diemen The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between July 12, 2021 and August 25, 2021, Radically Open Security B.V. carried out a penetration test for Gobalsky Labs Ltd.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following target(s):

- ERC20_Vesting.sol

The scoped services are broken down as follows:

- pentesting: 8.75 days
- **Total effort: 8.75 days**

1.3 Project objectives

ROS will perform a penetration test of an Ethereum vesting smart contract with Vega in order to assess the security of the smart contract implementation. To do so ROS will access ERC20_Vesting.sol and guide Vega in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The Security Audit took place between July 12, 2021 and August 25, 2021.

1.5 Results In A Nutshell

We did not find any serious issue during this grey-box penetration test.

We used a Truffle/Ganache testing environment to compile the smart contract and check for issues in doing so. Just like VEGA, we used the Solidity compiler version 0.8.1.

Finding vulnerabilities during this second test turned out to be much more of a challenge than during the first, as the vesting contract had been updated according to the recommendations offered in our first pentest report.

Overall this contract contains two low-threat vulnerabilities that have no impact on the security of client funds. The most important of these is the intentional withdrawal assist functionality, which enables the contract administrator to help in withdrawing funds from a vesting tranche to a client wallet. In this event, however, user funds remain secure, since the withdrawal target can be nothing other than the client's own wallet address.

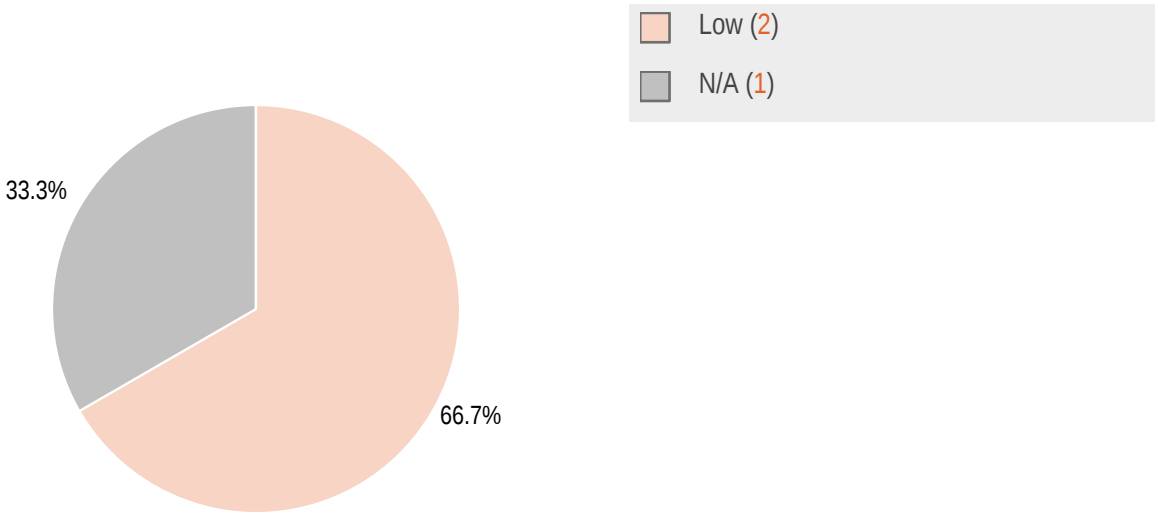
The following is a list of smart contract files involved, as well as their SHA256 hashes:

```
5f74e8d38147652ca9206ff72e05b575bcd4a14b1110d954be12b0320df87436  ERC20.sol
8c1af2536da7d70fe48d896f94a319cd76c1592dd66891d720122da35da73615  ERC20_Vesting.sol
c291d0b0bb1f0a727d1a9ce80d9acd530b94e5bdbcdf00c6a11b28958d9a7284  IERC20.sol
3ece681eb083c6d6b8f82c8a1932f2ff103279c09eaaffb56976e4f89c13f9c2  IStake.sol
ad89f894aa11a3cd78aef0898e04a2ad865f386fe19ff5f654dc44ec5801475c  Migrations.sol
7fb38f7cebe04c6ec052dc2e341325bb357811a17d907b45d681be3f9d492b9d
                                                  [ Vega_1_TEST_STANDIN_DO_NOT_DEPLOY.sol
6d7c1c19a22403a331aec92bb64e21d82973c7c6576bb6632e17284a48b9febe  Vega_2.sol
```

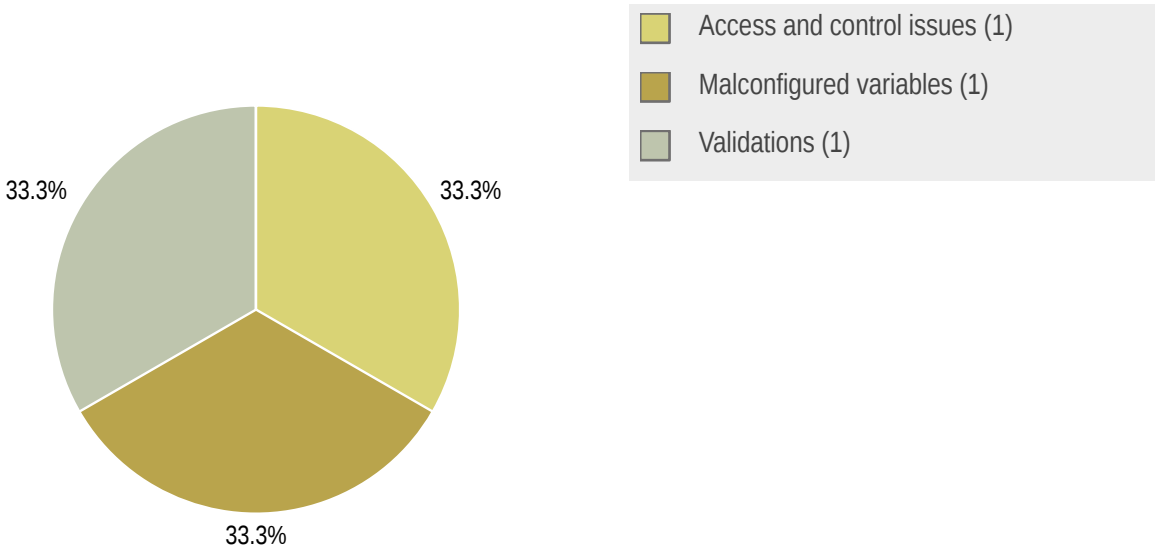
1.6 Summary of Findings

ID	Type	Description	Threat level
VEGA-002	Access and Control Issues	The assisted_withdraw_from_tranche function infringes on the user's control of their own assets. This has most likely been implemented intentionally. We do however deem it prudent to mention the security risk involved.	Low
VEGA-004	Malconfigured Variables	We observed the use of a so-called 'magic number' in the code. Magic numbers are values that should most often be replaced by variables for easier identification. In the issue_into_tranche function, magic number 0 (zero) is used.	Low
VEGA-003	Validations	Not determining the active state could potentially make it possible to abuse the stake functions.	N/A

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
VEGA-002	Access and Control Issues	Make sure a governance framework is in place for the financial processes surrounding the smart contract, to ensure the power over user funds is not used maliciously.
VEGA-004	Malconfigured Variables	Replace magic number by `default_tranche_id`.
VEGA-003	Validations	It would be prudent to make a developer's note detailing the motivation behind not checking the active state, and that the current economical implementation is in part handled by the well-defined boundaries built into the Ethereum Virtual Machine.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2017) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following tools.

- Truffle – https://www.trufflesuite.com
- Ganache – https://www.trufflesuite.com/ganache
- Etherscan – https://etherscan.io/

4 Findings

We have identified the following issues:

4.1 VEGA-002 — `assisted_withdraw_from_tranche` Infringes on User Control of Own Assets

Vulnerability ID: VEGA-002

Vulnerability type: Access and Control Issues

Threat level: Low

Description:

The `assisted_withdraw_from_tranche` function infringes on the user's control of their own assets. This has most likely been implemented intentionally. We do however deem it prudent to mention the security risk involved.

Technical description:

The contract controller is able to control user assets when requested, but also when not requested.

Impact:

Because this assisted withdrawal can do no more than move the target's funds from vesting to the target's own non-custodial wallet, this does not allow an administrator to steal or remove funds, only to pull them out of vesting autonomously. In this case the impact would be zero.

Recommendation:

Make sure a governance framework is in place for the financial processes surrounding the smart contract, to ensure the power over user funds is not used maliciously.

4.2 VEGA-004 — Legibility of contract: avoid magic numbers

Vulnerability ID: VEGA-004

Vulnerability type: Malconfigured Variables

Threat level: Low

Description:

We observed the use of a so-called 'magic number' in the code. Magic numbers are values that should most often be replaced by variables for easier identification. In the `issue_into_tranche` function, magic number `0` (zero) is used.

Technical description:

See line: 128

The term 'magic number' or 'magic constant' refers to the anti-pattern of using numbers directly in source code. This has been referred to as breaking one of the oldest rules of programming, dating back to the COBOL, FORTRAN and PL/1 manuals of the 1960s (source: Wikipedia). The use of unnamed magic numbers in code obscures the developers' intent in choosing that number, increases opportunities for subtle errors and makes it more difficult for the program to be adapted and extended in the future.

Impact:

The contract becomes more slightly difficult to read for developers, making it easier to miss programming mistakes when working on or upgrading the smart contract.

Recommendation:

Replace magic number by `default_tranche_id`.

4.3 VEGA-003 — `stake_tokens`, `remove_stake` lack state checks

Vulnerability ID: VEGA-003

Vulnerability type: Validations

Threat level: N/A

Description:

Not determining the active state could potentially make it possible to abuse the stake functions.

Technical description:

The functions `stake_tokens` and `remove_stake` seem to be missing a method to determine which states are active (wrong public key used, for instance).

Impact:

Likely no impact, as Ethereum EVM will make it impossible to add stake to empty accounts. Nevertheless it is good practice in languages other than Solidity to thoroughly check and catch errors of states related to money management.

Recommendation:

It would be prudent to make a developer's note detailing the motivation behind not checking the active state, and that the current economical implementation is in part handled by the well-defined boundaries built into the Ethereum Virtual Machine.

5 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

5.1 NF-001 — `withdraw_from_tranche`, `assisted_withdraw_from_tranche` duplicate code

The functions `withdraw_from_tranche` and `assisted_withdraw_from_tranche` on lines 222 and 243 seemed to contain duplicate code. This looked like a finding, yet turned out to be intentional and by design.

6 Future Work

- **Review compiler and its security measures on update**
  On every update or upgrade of the contract it must be ensured that only Solidity compiler 0.8.1 is used, and otherwise the compiler and security measures thereof must be reviewed.
- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

7 Conclusion

We discovered 2 Low and 1 N/A-severity issues during this penetration test.

Magic numbers in the contract code must be replaced by legible variable names. This will make it less likely that developers misunderstand the code and introduce new issues.

Control over the assets of tranche participants is an important issue. As currently written, the smart contract allows the contract administrator to pull user funds out of vesting and send them back to the target's wallet. This has intentionally been implemented and the associated risk has been acknowledged and is part of the agreement between the organization and their clients.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Joachim de Koning	Joachim is founder of hybrix.io - leveraging his expertise there as a tech maven, and who is also CEO of hy.company.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.