PUBLIC REPORT

Vault12 Smart Contract Security Audit

peppersec

Foreword

Status of a vulnerability or security issue

Clarity is a rare commodity. That is why for the convenience of both the client and the reader, we have introduced a system of marking vulnerabilities and security issues we discover during our security audits.

No issue

Let's start with an ideal case. If an identified security imperfection bears no impact on the security of our client, we mark it with the label.

✓ Fixed

The fixed security issues get the label that informs those reading our public report that the flaws in question should no longer be worried about.

Addressed

In case a client addresses an issue in another way (e.g., by updating the information in the technical papers and specification) we put a nice tag right in front of it.

Acknowledged

If an issue is planned to be addressed in the future, it gets the tag, and a client clearly sees what is yet to be done.

Although the issues marked "Fixed" and "Acknowledged" are no threat, we still list them to provide the most detailed and up-to-date information for the client and the reader.

Severity levels

We also rank the magnitude of the risk a vulnerability or security issue pose. For this purpose, we use 4 "severity levels" namely:

1. Minor
2. Medium
3. Major
4. Critical

More details about the ranking system as well as the description of the severity levels can be found in **Appendix 1. Terminology**.

TABLE OF CONTENTS

15. Vault12LockedTokens.addTokenGrant accepts _recipient, which should be only v12MutiSig — Acknowledged

16. SafeMath doesn't work for uint16 — ✓ Fixed

17.17ю 3_distro.js and smart contracts assume different number of seconds in a year — ✓ Fixed

18. Vault12LockedTokens locks team tokens after 24 years — ✓ Fixed

APPENDIX 1. TERMINOLOGY

1. Severity

01. Introduction

1 Source code

Object	Location
Vault12	Source code was given personally

2 Security assessment methodology

The code of a smart contract has been automatically and manually scanned for known vulnerabilities and logic errors that may cause security threats. The conformity of requirements (e.g., White Paper) and practical implementation has been reviewed as well. More information on the used methodology can be found here.

3 Auditors

1. Alexey Pertsev
2. Katerina Belotskaya

02. Summary

Below, you can find a table with all the discovered bugs and security issues listed.

Vulnerability description	Severity
Team won't recieve 20M per year for Advisors	Critical
Anyone can call addTokenGrant function	Major
addTokenGrant doesn't check that _vestingDuration <= 25 years	
addTokenGrant doesn't check that _vestingCliff <= 10 years	
Team can rewrite team vesting	
Redundant approve	
removeTokenGrant doesn't clear activeGrants	
Vault12LockedTokens locks team tokens after 24 years	
Unnecessary functionality	Medium
addTokenGrant function doesn't check recipient address	
Vault12LockedTokens doesn't take into account Leap year	
changeMultiSig function doesn't validate address	
Solidity version should be locked at final project stage	
Vault12LockedTokens.addTokenGrant accepts _recipient, which should be only v12MutiSig	
SafeMath doesn't work for uint16	
3_distro.js and smart contracts assume different number of seconds in a year	
multisend	Minor
Redundant Ownable for VaultGuardianToken	

03. Requirements

- Name: Vault Guardian Token
- Symbol: VGT
- Total: 1 Billion == 1E^9 == 1,000,000,000

1 Investors

- First 18,940,497 tokens should be locked till May 15 2019
- The rest of tokens can be distributed directly to their owners' addresses

2 Advisors

- Each year starting from May 15, 2018, grants Vault12 20M tokens to grant to advisors.
- First 20M grant for 2018 can be just transferred to our advisory vesting contract, and we will grant them to current advisors based on their vesting schedule.
- The rest is unlocked as 4x grants from the supply every year going forward.

3 Locked tokens

647,737,363 locked tokens vesting per the white paper formula.

The white paper uses a recursive formula that grants the Vault12 tokens at a rate of 10% of remaining locked tokens per year from July 1, 2018. So first year (7/1/2019) would be ~64.7M, while next year would be ~58.29M, etc.

4 Vaul12 reserve

50M tokens reserved for Vault12's activities.

04. General issues

1 Unnecessary functionality

Severity: MEDIUM

A new version of the openzeppelin `Ownable` contract has the `renounceOwnership` function. For more information, see here.

This function is inherited by your `VaultGuardianToken` unnoticeably. The `renounceOwnership` function seems superfluous. It is worth considering whether the function is necessary for the project?

Recommendations:

1. Consider rewriting `renounceOwnership` to empty the implementation.

Status:

✓ Fixed VaultGuardianToken.sol#L166

2 multisend

Severity: MINOR

The `multisend` function of the `VaultGuardianToken` smart contract requires the count of `_recipients` less than `multiSendLimit`, which is `150` by default. The current implementation allows increasing the value up to at least 220.

Recomendations:

1. Consider increasing default value to maximum.

Status:

✓ Fixed VaultGuardianToken.sol#L18

3 Anyone can call addTokenGrant

Severity: MAJOR

The `addTokenGrant` function of `AdvisorsVesting` smart contract does not check that only multisender can call it. If `AdvisorsVesting` contract still has the tokens approved for the transfer, anyone can create a Grand for themselves and take the tokens.

Recomendations:

1. Consider using only the V12MultiSig modifier in the function.

Status:

✓ Fixed AdvisorsVesting.sol#L59

4 addTokenGrant doesn't check recipient address

Severity: MEDIUM

The `addTokenGrant` function of `AdvisorsVesting` smart contract does not validate `recipient` address.

Recomendations:

1. Consider adding a following check:

```
require(_recipient != address(0) && _recipient != this && _recipient != address(token));
```

Status:

✓ Fixed AdvisorsVesting.sol#L60

4 Vault12LockedTokens doesn't take into account Leap year

Severity: MEDIUM

Vault12LockedTokensr define a year as 31540000 seconds, which is not always true.

Recomendations:

1. Consider adding 21600 (6 hours) to `SECONDS_PER_YEAR` constant to make calculations more accurate.

Status:

✓ Fixed Vault12LockedTokens.sol#L9

6 changeMultiSig doesn't validate address

Severity: **MEDIUM**

The `changeMultiSig` function of the AdvisorsVesting and Vault12LockedTokens smart contracts does not validate `_newMultisig` address.

Recomendations:

1. Consider adding address validation.

Status:

✓ Fixed AdvisorsVesting.sol#L195

✓ Fixed Vault12LockedTokens.sol#L128

7 addTokenGrant doesn't check that _vestingDuration <= 25 years

Severity: **MAJOR**

The `addTokenGrant` function of the `AdvisorsVesting` and `Vault12LockedTokens` smart contracts does not check that `_vestingDuration` <= 25 yrs, which is a special client requirement.

Recomendations:

1. Consider adding the validation of _vestingDuration.

Status:

✓ Fixed AdvisorsVesting.sol#L195

✓ Fixed Vault12LockedTokens.sol#L128

8 addTokenGrant doesn't check that _vestingCliff <= 10 years

Severity: MAJOR

The addTokenGrant function of AdvisorsVesting smart contrac does not check that _ vestingCliff <= 10 yrs, which is a special client requirement.

Recomendations:

1. Consider adding the validation of _vestingCliff.

Subsection:

✓ Fixed AdvisorsVesting.sol#L62

9 Solidity version should be locked at final project stage

Severity: MEDIUM

Contracts should be deployed with the same compiler version and flags that they have been tested most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may pose higher risks of undiscovered bugs. Contracts may also be deployed by others and pragma indicates the compiler version intended by the original authors.

Recommendations:

1. Consider locking specific version of pragma.

Status

✓ Fixed AdvisorsVesting.sol#L1

✓ Fixed Vault12LockedTokens.sol#L1

✓ Fixed VaultGuardianToken.sol#L1

10 Redundant Ownable for VaultGuardianToken

Severity: MINOR

The `VaultGuardianToken` contract is inherited from `Ownable` and `Claimable`, which is inherited from `Ownable` as well. However, the compiler resolves this graph correctly, so there is no security impact here for now.Recommendations:

1. Consider removing explicit Ownable inheritance

Status:

✓ Fixed VaultGuardianToken.sol#L10

11 Team can rewrite team vesting

Severity: MAJOR

The team can use the `addTokenGrant` function of `Vault12LockedTokens` smart contract to rewrite their Vesting Grand. The contract does not check Grand already exists. That action cannot be used to cheat actually, but it can cause a DoS.

Recomendations:

1. Add the validation of `_recipient`.

Status:

✓ Fixed Vault12LockedTokens.sol#L54

12 Team won't recieve 20M per year for Advisors

Severity: CRITICAL

TokenDistro.md has the following requirements:

> #2: Advisors: 100,000,000 tokens total over 5 years
>
> - Each year starting from May 15, 2018 grants Vault12 20M tokens to grant to advisors.
> - [x] First 20M grant for 2018 can be just transferred to our advisory vesting contract and we will grant to current advisors based on their vesting schedule.
> - [ ] Rest unlock as 4x grants from supply every year going forward.

But `3_distro.js` migration satisfies the second requirement only. For additional 4 grands, the script utilizes the `Vault12LockedTokens` contract. It means the team will get 10% per year, which is not 20M per year (as it has been required in `TokenDistro.md`).

Recomendations:

1. Use the `AdvisorsVesting` contract for creating 4 additional vestings that start from 2018, 2019, 2020 and 2021 year respectively with 1 year Cliff.

Status:

✓ Fixed `3_distro.js#L124-L144`

13 Redundant approve

Severity: MAJOR

The `3_distro.js` script has redundant `approve` to `AdvisorsVesting` for `20M` tokens, which is unused.

Recomendations:

1. Consider removing the unused `approve` to avoid attacks.

Status:

✓ Fixed `3_distro.js`

14 removeTokenGrant doen't clear activeGrants

Severity: MAJOR

The removeTokenGrant function of AdvisorsVesting smart contract does not clear activeGrants mapping. So, after removing vesting the getActiveGrants still returns it.

Recomendations:

1. Consider removing vesting from activeGrants in the removeTokenGrant function.

Status:

✓ Fixed AdvisorsVesting.sol#L178. Now Grand has isActive variable which is checks in getActiveGrants

15 Vault12LockedTokens.addTokenGrant accepts _recipient, which should be only v12MutiSig

Severity: MEDIUM

The addTokenGrant function of VaultGuardianToken accepts _recipient, which is supposed to be only one address - address of Vault12's team MultiSig. The team may desire to create cleaner implementation with _recepient value hardcoded into smart contract.

Recomendations:

1. Consider removing _recipient argument.

Status:

Acknowledged

16 SafeMath doesn't work for uint16

Severity: **MEDIUM**

The `AdvisorsVesting` and `Vault12LockedTokens.sol` smart contracts use SafeMath for `uint16`, which works for `uint256` only. There is no security impact for the current implementation, but it can happen in case of some smart contract changes.

Recomendations:

1. Consider using `uint256` only.

Status:

✓ Fixed

17 3_distro.js and smart contracts assume different number of seconds in a year

Severity: **MEDIUM**

The `3_distro.js` uses `31556926` as the number of seconds in year, but `Vault12LockedTokens` uses `31561600`. This desynchronization can lead to problems with the token claim in future.

Recomendations:

1. Consider using `31561600` as the number of seconds in a year in `3_distro.js`.

Status:

✓ Fixed 3_distro.js#L131

18 Vault12LockedTokens locks team tokens after 24 years

Severity: MAJOR

The `claimVestedTokens` function of `Vault12LockedTokens` smart contract accumulate `yearsClaimed` per call instead of rewriting. That leads to locking rest amount of tokens after 24 years.

Recomendations:

1. Consider implementing rewriting.

Status:

✓ Fixed Vault12LockedTokens.sol#L114

Appendix 1. Terminology

1 Severity

Severity is the category that described the magnitude of an issue.

		Severity		
Impact	Major	Medium	Major	Critical
	Medium	Minor	Medium	Major
	Minor	None	Minor	Medium
		Minor	Medium	Major
		Likelihood		

MINOR

Minor issues are generally subjective in their nature or potentially associated with the topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code. The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

MEDIUM

Medium issues are generally objective in their nature but do not represent any actual bugs or security problems. These issues should be addressed unless there is an apparent reason not to.

MAJOR

Major issues are things like bugs or vulnerabilities. These issues may be unexploitable directly or may require a certain condition to arise to be exploited. If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations which make the system exploitable.

CRITICAL

Critical issues are directly exploitable bugs or security vulnerabilities. If unaddressed, these issues are likely or guaranteed to cause major problems and ultimately a full failure in the operations of the contract.

About Us

Worried about the security of your project? You're on the right way! The second step is to find a team of seasoned cybersecurity experts who will make it impenetrable. And you've just come to the right place.

PepperSec is a group of whitehat hackers seasoned by many-year experience and have a deep understanding of the modern Internet technologies. We're ready to battle for the security of your project.

LET'S KEEP IN TOUCH


peppersec.com


hello@peppersec.com


Github