SALUS SECURITY

FEB 2023

CODE SECURITY ASSESSMENT

THRUPENNY

Overview

Project Summary

- Name: Thrupenny
- Version: Commit b58cedd
- Platform: Ethereum
- Language: Solidity
- Repository: https://github.com/0xBuooy/erc20-smart-vesting-contract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Thrupenny
Version	v2
Type	Solidity
Dates	Feb 17 2023
Logs	Feb 03 2023; Feb 17 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	4
Total	6

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Unnecessary receive and fallback function	Low	Business Logic	Resolved
2	Centralization risk	Low	Centralization	Acknowledged
3	Single-step ownership transfer pattern risk	Informational	Authentication	Acknowledged
4	Redundant code	Informational	Redundancy	Acknowledged
5	SafeMath library not needed since Solidity 0.8.0	Informational	Redundancy	Acknowledged
6	Gas optimization suggestions	Informational	gas optimization	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Unnecessary receive and fallback function	
Severity: Low	Category: Business Logic
Target: - contracts/TokenVesting.sol	

Description

contracts/TokenVesting.sol:L67-L69

```
receive() external payable {}

fallback() external payable {}
```

There is a payable **fallback** function and a payable **receive** function in the TokenVesting contract that allows the contract to receive funds from other addresses. However, there is no withdrawal logic in the Token Vesting contract.
As a result, if one accidentally sends Ether to the TokenVesting contract, the funds are locked in the contract and there is no way to move the funds out.

Recommendation

Consider removing the payable fallback function and payable receive function. Receiving ether is an unexpected function of the TokenVesting contract.

Status

This issue has been resolved by the team by removing the payable fallback function and payable receive function in commit d7d26b4.

2. Centralization risk

Severity: Low	Category: Centralization
Target: - contracts/TokenVesting.sol	

Description

contracts/TokenVesting.sol:L140-L153

```
/**
 * @notice Revokes the vesting schedule for given identifier.
 * @param vestingScheduleId the vesting schedule identifier
 */
function revoke(bytes32 vestingScheduleId) public onlyOwner
onlyIfVestingScheduleNotRevoked(vestingScheduleId) {
    VestingSchedule storage vestingSchedule = vestingSchedules[vestingScheduleId];
    uint256 vestedAmount = _computeReleasableAmount(vestingSchedule);
    if(vestedAmount > 0){
        release(vestingScheduleId, vestedAmount);
    }
    uint256 unreleased = vestingSchedule.amountTotal.sub(vestingSchedule.released);
    vestingSchedulesTotalAmount = vestingSchedulesTotalAmount.sub(unreleased);
    vestingSchedule.revoked = true;
}
```

The owner in TokenVesting can call **revoke**() to revoke the vesting schedule for a given identifier. If the owner's private key is compromised and obtained by a malicious person who then calls this function. The vesting schedule for the given identifier will be canceled.
If the privileged owner account is a plain EOA account, this can be worrisome and pose a risk to the users.

Recommendation

It is recommended to use a Multisig wallet for the owner address.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Single-step ownership transfer pattern risk	
Severity: Informational	Category: Authentication
Target: - contracts/TokenVesting.sol	

Description

contracts/TokenVesting.sol:L8

```
import "@openzeppelin/contracts/access/Ownable.sol";
```

Inheriting from OpenZeppelin's **Ownable** contract means that you are using a single-step ownership transfer pattern. If an admin provides an incorrect address for the new owner this will result in none of the methods modified by onlyOwner being callable.
There is an Ownable2Step.sol contract in the OpenZeppelin library that is designed for two-step ownership transfer which is more secure than the single-step ownership transfer used by Ownable.sol.

Recommendation

Consider using OpenZeppelin's Ownable2Step.sol instead of Ownable.sol.

Status

This issue has been acknowledged by the team.

4. Redundant code	
Severity: Informational	Category: Redundancy
Target: - contracts/TokenVesting.sol	

Description

contracts/TokenVesting.sol:L4

```
import "@openzeppelin/contracts/finance/VestingWallet.sol";
```

contracts/TokenVesting.sol:L9-L10

```
import "@openzeppelin/contracts/utils/math/Math.sol";
```

The **VestingWallet.sol** and **Math.sol** are unused imports.

contracts/TokenVesting.sol:L120

```
require(this.getWithdrawableAmount() >= vestingAmount, "TokenVesting: not enough
withdrawable funds");
```

contracts/TokenVesting.sol:L123

```
bytes32 vestingScheduleId =
this.computeNextVestingScheduleIdForHolder(beneficiaryAddress);
```

contracts/TokenVesting.sol:L223

```
require(this.getWithdrawableAmount() >= burnAmount, "TokenBurning: not enough
withdrawable funds");
```

The function visibility of **getWithdrawableAmount** and **computeNextVestingScheduleIdForHolder** is public, so **this** keyword is not required for contract internal.

contracts/TokenVesting.sol:L225-L226

```
_token.approve(address(0x000000000000000000000000000000000000dEaD), 0);
_token.approve(address(0x000000000000000000000000000000000000dEaD), burnAmount);
```

The above lines are unnecessary.

contracts/TokenVesting.sol:L182

```
address payable beneficiaryPayable = payable(vestingSchedule.beneficiary);
```

The address does not need to be declared as **payable** since there is no ether sending to it.

Recommendation

Consider removing the redundant code.

Status

This issue has been acknowledged by the team. The redundant **this** keyword has been removed in commit d7d26b4.

5. SafeMath library not needed since Solidity 0.8.0	
Severity: Informational	Category: Redundancy
Target: - contracts/TokenVesting.sol	

Description

contracts/TokenVesting.sol:L6

```
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
```

SafeMath is used to check underflow and overflow for arithmetic operations. However, since Solidity version 0.8.0, arithmetic operations revert on underflow and overflow by default. Since the bounce project uses a Solidity version no less than 0.8.0, it is unnecessary to use the **SafeMath** library.

Recommendation

It is recommended to remove the **SafeMath** library.

Status

This issue has been acknowledged by the team.

6. Gas optimization suggestions	
Severity: Informational	Category: gas optimization
Target: - contracts/TokenVesting.sol	

Description

contracts/TokenVesting.sol:L33

```
uint256 private burntTotalAmount = 0;
```

The default value for uint256 type is 0. Removing redundant variable initialization can reduce gas cost.

contracts/TokenVesting.sol:L119

```
function createVestingSchedule(address beneficiaryAddress, uint256 vestingAmount, uint8
vestingType) public onlyOwner nonReentrant
```

contracts/TokenVesting.sol:L144

```
function revoke(bytes32 vestingScheduleId) public onlyOwner
onlyIfVestingScheduleNotRevoked(vestingScheduleId)
```

contracts/TokenVesting.sol:L199-L200

```
function computeReleasableAmount(bytes32 vestingScheduleId) public
```

contracts/TokenVesting.sol:L264-L265

```
function getLastVestingScheduleForHolder(address holder) public
```

The **createVestingSchedule(), revoke(), computeReleasableAmount(), getLastVestingScheduleForHolder()** functions are not called inside the contract; thus, the visibility can be changed to external to save gas.

Recommendation

It is recommended to remove redundant variable initialization and change the function visibility to external for functions that are not called within the contract.

Status

This issue has been resolved by the team in commit d7d26b4.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file in Commit b58cedd :

File	SHA-1 hash
TokenVesting.sol	255114301927d883fd4148b2f884c0fe4692b938