

# WeTrust ROSCA contract code audit

FEBRUARY 28, 2017 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY



The WeTrust team asked us to review and audit their new [ROSCA](#) contract code. We looked at their contracts and now publish our results.

The audited contracts are at their [rosca-contracts GitHub repo](#). The version used for this report is commit `2af29be97d529488f5488fe0592f9e6b3585254f`. The main contract file is [ROSCA.sol](#).

Here's our assessment and recommendations, in order of importance:

## Severe

We haven't found any severe security problems with the code.

## Potential problems

## Use safe math

There are many unchecked math operations in the code. It's always better to be safe and perform checked operations. Consider [using a safe math library](#), or performing pre-condition checks on any math operation.

## Be careful with integer division

Several parts of the code use integer division. See [1,2, 3, 4, 5](#). Integer division result will discard the remainder and keep the quotient. This could bring unexpected results in some cases.

For example, in [line 288](#), if **currentRoundTotalDiscounts** is not a multiple of **membersAddresses.length**, **totalDiscounts** will be incremented by the quotient, and the remainder of the division will not be considered.

We haven't detected any actual attacks or inconsistencies in the contract due to this fact, but we recommend being extra careful. In this case, a solution could be to make **totalDiscounts** have the total amount of discounts instead of the discounts per member.

## Timestamp usage

There's a problem with using timestamps and **now** (alias for **block.timestamp**) for contract logic, based on the fact that miners can perform some manipulation. In general, [it's better not to rely on timestamps for contract logic](#). The solution is to use **block.number** instead, and approximate dates with expected block heights and time periods with expected block amounts.

The ROSCA.sol contract uses timestamps at [several this comment notes](#), this won't affect the functioning of the contract, but the miner of the **cleanUpPreviousRound** call transaction will have absolute control on who the

next winner is. We recommend the team to consider the potential risk of this manipulation and switch to **block.number** if necessary.

For more info on this topic, see [this stack exchange question](#)

## Warnings

### Use of send

Use of **send** is always risky and should be analyzed in detail. Three occurrences found in [line 427](#), [line 496](#) of ROSCA.sol.

- Always check send return value: OK.
- Consider calling send at the end of the function: OK.
- Favor pull payments over push payments: Warning. All 3 occurrences of send are push payments. Although we couldn't find any attack vectors on this contract, consider using [OpenZeppelin's PullPayment contract](#) to implement pull payments in ROSCA.sol.

For more info on this problem, see [this note](#).

### Usage of magic constants

There are several magic constants

- <https://github.com/WeTrustPlatform/rosca-contracts/blob/2af29be97d529488f5488fe0592f9e6b3585254f/contracts/ROSCA.sol#L341>
- <https://github.com/WeTrustPlatform/rosca-contracts/blob/2af29be97d529488f5488fe0592f9e6b3585254f/contracts/ROSCA.sol#L389>

Use of magic constants reduces code readability and makes it harder to understand code intention. We recommend extracting magic constants into contract constants.

## Bug Bounty

Formal security audits are not enough to be safe. We recommend implementing an automated contract-based bug bounty[see this guide](#)

`totalFees` will always be `0` on `LogFundsWithdrawal`

In [line 502 of ROSCA.sol](#), a `LogFundsWithdrawal` event is emitted if the send does not fail. The problem is that the variable `totalFees` will always be 0, as it is set to 0 in [line 495](#)

## Avoid duplicated code

Duplicate code makes it harder to understand the code's intention and thus, auditing the code correctly. It also increases the risk of introducing hidden bugs when modifying one of the copies of some code and not the others.

The logic in [getParticipantBalance\(\)](#) and [the start of withdraw\(\)](#) is very similar and could be refactored to avoid repetition. Consider using `getParticipantBalance` in `withdraw`.

## Naming suggestions

- The `members_` constructor parameter could be confused as the full set of members, and it is in fact all members but the contract creator. Consider calling it `otherMembers_` or something like so, to avoid this confusion.

- The **grossTotalFees** uint256 variable in [recalculateTotalFees](#) is not really the gross total fees, because it's never multiplied by the service fee per mille. Consider renaming it to **grossTotal** or something like so, to avoid confusion.
- The **totalDiscounts** uint256 contract variable does not represent the total discounts, but the discounts per member. Consider renaming to **discountPerMember** to avoid confusion.

## Use latest version of Solidity

[Current code](#) is written for an old version of solc (0.4.4). We recommend changing the solidity version pragma for the latest version ( `pragma solidity ^0.4.10;` ) to enforce latest compiler version to be used.

## Additional Information and Notes

- Good naming for logs, all starting with `Log`, as recommended in [our article about smart contract security best practices](#)
- Good work with using a fail-early-and-loudly programming style. Most exceptional conditions are handled with `throws`, as recommended in our guide. For example, all modifiers do this.
- Good work using the EscapeHatch pattern to make the contract safer.
- [FEE\\_ADDRESS](#) uses a different naming convention to other constants. Consider regularizing notations.
- Bad indentation at [line 243](#).
- Consider checking against `0x0` instead of `0` when comparing addresses. E.g:[here](#) and [here](#)
- Use `!winnerSelectedThroughBid` in the conditional in [line 253](#)

- Consider changing [these complex condition checks](#) into multiple ifs and throws.
- Interesting error handling technique [in line 394](#)
- No need to throw [in line 430](#), but still simpler and equivalent to do so.
- Comment in [line 444](#) contains a typo, should read: "if ROSCA *has* ended".

## Conclusions

No severe security issues were found. Some changes were recommended to follow best practices and reduce potential attack surface.

Overall, code quality is good, it's well commented, and most well-known security good practices were followed. This was one of the most well written contracts we had to audit. 👍

If you're interested in discussing smart contract security, [follow us on Medium](#) or [join our slack channel](#). We're also available for smart contract security development and auditing work.

*Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the ROSCA contract. We have not reviewed the related WeTrust project. The above should not be construed as investment advice or an offering of tokens. For general information about smart contract security, check out our thoughts [here](#).*

---

## Security Audits

- If you are interested in smart contract security, you can continue the discussion in our [forum](#), or even better, [join the team](#) 🚀
- If you are building a project of your own and would like to request a security audit, please do so [here](#).

## RELATED POSTS



< ether.camp >

**ProjectKudos Audit**

**Z** OpenZeppelin | security

SECURITY AUDITS

### EtherCamp's ProjectKudos public code audit

Following our audit of the HackerGold HKG token, we've been asked by the EtherCamp team to review...

[READ MORE](#)

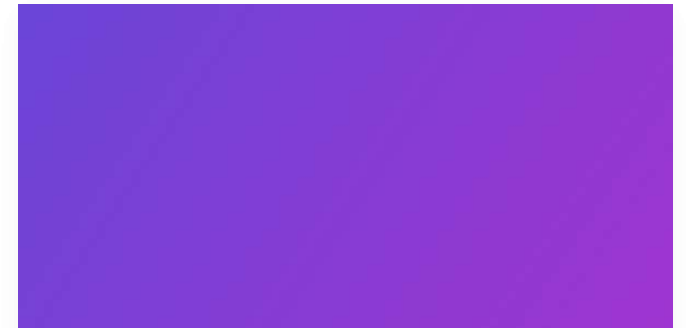


GUIDES

### Smart Contract Upgradeability using Eternal Storage

As mentioned in our Development Roadmap Part Two, the Kernel is the foundational layer of...

[READ MORE](#)



GUIDES

### Deconstructing a Solidity Contract —Part I: Introduction

You're on the road, driving fast in your rare, fully restored 1969 Mustang Mach 1. The sunlight...

[READ MORE](#)

ADD COMMENT

Name \*

Email \*

Website

☐ Save my name, email, and website in this browser for the next time I comment.

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.



POST COMMENT

Email\*

Get our monthly news roundup

由 reCAPTCHA 提供保护  
隐私权 - 使用条款

SIGN UP

## Products

Contracts

Defender

## Security

Security Audits

## Learn

Docs

Forum

Ethernaut

## Company

Website

About

Jobs

Logo Kit