Published in New Alchemy

New Alchemy Follow
Apr 17, 2018 · 10 min read

Right Mesh Smart Contract Audit

Right Mesh engaged New Alchemy to audit the smart contracts for their "RMESH" token. We focused on identifying security flaws in the design and implementation of the contracts and on finding differences between the contracts' implementation and their behaviour as described in public documentation.

The audit was performed over four days in February and March of 2018. This document describes the issues discovered in the audit. An initial version of this document was provided to RightMesh, who made various changes to their contracts based on New Alchemy's findings; this document was subsequently updated in March 2018 to reflect the changes.

Files Audited

RightMesh made additional fixes whose commit hash was not shared with New Alchemy.

New Alchemy's audit was additionally guided by the following documents:

- RightMesh Whitepaper, version 4.0 (February 14 2018)
- RightMesh Technical Whitepaper, version 3.1 (December 17 2017)
- RightMesh Frequently Asked Questions

The review identified one critical finding, which allowed the crowd sale owner to issue large quantities of tokens to addresses that it controls by abusing a flaw in the mechanism for minting "predefined tokens". Three additional minor flaws were identified, all of which are best-practice violations of limited practical exploitability: lack of two-phase ownership transfer and of mitigations for the short-address attack, and token allocation configuration that is less than ideally transparent. An additional minor flaw was documented in some earlier versions of this report but was determined to be a false positive.

After reviewing an initial version of this report, RightMesh made changes to their contracts to prevent predefined tokens from being minted multiple times and to mitigate short-address attacks. No changes were made to ownership transfers or to the configuration of predefined token allocations.

General Discussion

These contracts implement a fairly simple token and crowdsale, drawing heavily on base contracts from the OpenZeppelin project[1]. The code is well commented. However, the RightMesh white papers and other documentation provide very little detail about the operation of the crowd sale or token. It was not clear to New Alchemy who receives pre-defined token allocations; from `MeshCrowdsale`, how large these allocations are, or why they receive them. Likewise, it was not clear how `Timelock` fits into the token ecosystem. RightMesh later clarified that the pre-defined token allocations are for the

Some of the OpenZeppelin base contracts inherited by the RightMesh contracts have changed substantially since the RightMesh contracts were written. Consequently, the RightMesh contracts cannot be built against the head of OpenZeppelin. RightMesh should either copy a fork of the relevant OpenZeppelin contracts into their repository or document the OpenZeppelin release or commit that should be used to build their contracts.

Critical Issues

Fixed: Predefined tokens can be minted multiple times

As its name implies, the function `MeshCrowdsale.mintPredefinedTokens` mints tokens according to an allocation set during deployment. This function does not check that it has not previously been called, so it can be called multiple times. Despite comments to the contrary, this function is tagged `onlyOwner`, so this function will only ever be called more than once if an owner makes a mistake or deliberately misbehaves. Further, `MeshToken` gets deployed in a default state of `paused`, which prevents any token transfers, and `mintPredefinedTokens` *does* check that the balance of each beneficiary is zero, so if `mintPredefinedTokens` has already been called, subsequent calls *should* have no effect. However, there are still possible conditions under which a beneficiary could transfer tokens prior to an extra call to `mintPredefinedTokens`:

- An owner could call `MeshToken.unpause`, which would allow all token holders to transfer tokens. `MeshToken` cannot be re-paused once unpaused, so any call to `mintPredefinedTokens` after `MeshToken` has been unpaused may mint additional tokens.
- An owner could use `MeshToken.updateAllowedTransfers` to flag a beneficiary as being allowed to make transfers despite `MeshToken` being paused.

In the worst case, a rogue owner deploys `MeshCrowdsale` with a beneficiary address that it controls, flags that address to permit transfers despite `MeshToken` being paused, waits for some tokens to be sold, then alternates calls to `MeshCrowdsale.mintPredefinedTokens` and `MeshToken.transfer` to allocate up to the


of control to ensure that `MeshToken` remains paused until the crowdsale completes may also be useful. Further, the comment or the declaration of `mintPredefinedTokens` should be amended so that they agree on what users are allowed to call this function.

Re-test results: RightMesh added logic to prevent `mintPredefinedTokens` from being called twice and corrected the function comment to indicate that it can only be called by the owner.

Minor Issues

Not Fixed: Lack of two-phase ownership transfer

In contracts that inherit the common `Ownable` contract from the OpenZeppelin project^2 (including `MeshToken`, `MeshCrowdsale`, and `Timelock`), a contract has a single owner. That owner can unilaterally transfer ownership to a different address. However, if the owner of a contract makes a mistake in entering the address of an intended new owner, then the contract can become irrecoverably unowned.

In order to preclude this, New Alchemy recommends implementing two-phase ownership transfer. In this model, the original owner designates a new owner, but does not actually transfer ownership. The new owner then accepts ownership and completes the transfer. This can be implemented as follows:

```
contract Ownable {
    address public owner;
    address public newOwner

    event OwnershipTransferred(address indexed previousOwner,
address indexed newOwner);

    function Ownable() public {
        owner = msg.sender;
        newOwner = address(0);
    }

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
```

 

```
    function acceptOwnership() public {
        require(msg.sender == newOwner);
        OwnershipTransferred(owner, msg.sender);
        owner = msg.sender;
        newOwner = address(0);
    }
}
```

Re-test results: RightMesh opted to preserve the current ownership transfer mechanism.

Fixed: Lack of short-address attack protections

Some Ethereum clients may create malformed messages if a user is persuaded to call a method on a contract with an address that is not a full 20 bytes long. In such a "short-address attack", an attacker generates an address whose last byte is 0x00, then sends the first 19 bytes of that address to a victim. When the victim makes a contract method call, it appends the 19-byte address to `msg.data` followed by a value. Since the high-order byte of the value is almost certainly 0x00, reading 20 bytes from the expected location of the address in `msg.data` will result in the correct address. However, the value is then left-shifted by one byte, effectively multiplying it by 256 and potentially causing the victim to transfer a much larger number of tokens than intended. `msg.data` will be one byte shorter than expected, but due to how the EVM works, reads past its end will just return 0x00.

This attack effects methods that transfer tokens to destination addresses, where the method parameters include a destination address followed immediately by a value. In the RightMesh contracts, such methods include `MeshToken.mint`, `MeshToken.transfer`, `MeshToken.transferFrom`, `MeshToken.approve`, `MeshToken.increaseApproval`, `MeshToken.decreaseApproval`, (all inherited from OpenZeppelin base contracts), and `Timelock.allocateTokens`.

While the root cause of this flaw is buggy serializers and how the EVM works, it can be easily mitigated in contracts. When called externally, an affected method should verify that `msg.data.length` is *at least* the minimum length of the method's expected arguments (for instance, `msg.data.length` for an external call to


padding to the end). This can be implemented in a modifier. External calls can be detected in the following ways:

- Compare the first four bytes of `msg.data` against the method hash. If they don't match, then the call is internal and no short-address check is necessary.
- Avoid creating `public` methods that may be subject to short-address attacks; instead create only `external` methods that check for short addresses as described above. `public` methods can be simulated by having the external methods call `private` or `internal` methods that perform the actual operations and that do not check for short-address attacks.

Whether or not it is appropriate for contracts to mitigate the short-address attack is a contentious issue among smart-contract developers. Many, including those behind the OpenZeppelin project, have explicitly chosen not to do so. While it is New Alchemy's position that there is value in protecting users by incorporating low-cost mitigations into likely target functions, RightMesh would not stand out from the community if they also choose not to do so.

Re-test results: RightMesh overrode the listed functions to require that `msg.data.length` is at least 68. All are public, so they may not work properly if called internally from something with a shorter argument list.

Not Fixed: Predefined token allocations are not hard-coded

According to the RightMesh FAQ, tokens are allocated as follows:

- 30%: Public distribution (crowdsale)
- 30%: RightMesh GmbH & Community
- 20%: Left & team
- 10%: Advisors & TGE costs
- 10%: Airdrop to community

These last five allocations are controlled at deployment by the [illegible] and

interested parties, the state of the contract is not as easily located or reviewed as its source code.

The current predefined token allocation in `config/predefined-minting-config.js` appears to try five times to assign 100 tokens to the address `0x5D51E3558757Bfdfc527867d046260fD5137Fc0F` (this should only succeed once due to the balance check), though this may be test data.

For optimal transparency, RightMesh should instead hard-code the allocation percentages or token counts so that anyone reviewing the contract source code can easily verify that tokens were issued as documented.

Re-test results: RightMesh opted to preserve the current allocation configuration mechanism.

Line by line comments

This section lists comments on design decisions and code quality made by New Alchemy during the review. They are not known to represent security flaws.

MeshCrowdsale.sol

Lines 12, 52–53

OpenZeppelin has radically refactored their crowdsale contracts as of late February 2018. Among other things, `CappedCrowdsale` has been moved, the functionality for starting and ending times has been moved to `TimedCrowdsale`, and `Crowdsale.validPurchase` no longer exists. In order to ensure that a version of OpenZeppelin compatible with these contracts can be easily identified, RightMesh should copy a fork of the relevant contracts into their repository or at least document the commit that should be used.

Re-test results: RightMesh added a comment to their code indicating that the version of OpenZeppelin at commit hash `4d7c3cca7590e554b76f6d91cbaaae87a6a2e2e3` should be


“og” should be “of”.

Re-test results: This issue has been fixed as recommended.

Lines 96, 116, 132, 149, 159

There is no need for these functions to return `bool` : they all unconditionally return `true` and throw on failure. Removing the return values would make code that calls these functions simpler, as it would not need to check return values. It would also make them marginally cheaper in gas to execute.

Re-test results: This issue has been fixed as recommended.

Line 167

This function is declared as returning `bool` , but never returns anything. As above, there is no need for it to return anything.

Re-test results: This issue has been fixed as recommended.

MeshToken.sol

Lines 60

The function should be tagged `public` or `external` rather than relying on the default visibility.

Re-test results: RightMesh reports fixing this issue as recommended.

Timelock.sol

Line 91

If `cliffReleasePercentage` and `slopeReleasePercentage` ever sum to less than 100, then the remaining fraction of tokens will become available all at once once the slope duration expires, essentially creating a second cliff at the bottom of the slope. If this is not intended behaviour, then the check should be amended to require that the sum is 100%.

There is no need for these functions to return `bool` : they all unconditionally return `true` and throw on failure. Removing the return values would make code that calls these functions simpler, as it would not need to check return values. It would also make them marginally cheaper in gas to execute.

Re-test results: RightMesh reports fixing this issue as recommended.

Line 157

Consider checking `withdrawalPaused` in `availableForWithdrawal` instead of in `withdraw` . As currently implemented, `availableForWithdrawal` may report a non-zero quantity available for a paused address, but withdrawal will fail. It would be more intuitive if `availableForWithdrawal` reported 0 for a paused address.

Re-test results: RightMesh reports that this behaviour is by design: it allows employees to see unlocked tokens even if withdrawal is paused.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bugfree status. The audit documentation is for discussion purposes only.

New Alchemy *is a strategy and technology advisory group specializing in tokenization. One of the only companies to offer a full spectrum of guidance from tactical technical execution to high-level theoretical modeling, New Alchemy provides blockchain technology, token game theory, smart contracts, security audits, and ICO advisory to the most innovative startups worldwide.* ***Get in touch with us at Hello@NewAlchemy.io***