Published in SmartDec Cybersecurity Blog

Artyom Orlov Follow
Apr 28, 2018 · 7 min read · Listen

Save

MinerOne Smart Contracts Security Analysis

In this report we consider the security of the MinerOne project. Our task is to find and describe security issues in the smart contracts of the platform.

Procedure

In our audit, we consider the following crucial features of the smart contract code:

1. Whether the code is secure.

We perform our audit according to the **shortened procedure**:

Automated analysis

- we scan project's smart contracts with our own Solidity static code analyzer SmartCheck
- we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Remix, Oyente and Securify (beta version since full version was unavailable at the moment this report was made)
- we manually verify (reject or confirm) all the issues found by tools

Manual audit

- we manually analyze smart contracts for security vulnerabilities
- we check smart contracts logic and compare it with the one described in the whitepaper

Report

- we reflect all the gathered information in the report

Disclaimer

The audit does not give any warranties on the security of the code. One audit can not be considered enough. We always recommend proceeding to several independent audits and a public bug bounty program to ensure the security of the smart contracts. Besides, security audit is not an investment advice.

Checked vulnerabilities

We have scanned MinerOne smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded

- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

About The Project

Project Description

In our analysis we consider MinerOne whitepaper (minerone_whitepaper.pdf, sha1sum: 2c772af9fba7ab82d7ba2fa3475e0ad35701be54) and smart contracts code (minerone.zip, sha1sum: 2f50d0ce61001b16fde7d49033ca1c7f65ec37ad).

Project Architecture

For the audit, we have been provided with the following set of files:

- MinerOneToken.sol — inherits the MintableToken contract from the OpenZeppelin library version 1.6.0
- MinerOneCrowdsale.sol — inherits the Ownable contract from the OpenZeppelin library version 1.6.0
- MinerOneCrowdsaleFlat.sol — flat file combined of MinerOneToken.sol and

Provided file set is a truffle project and an npm package.

The project compiles successfully with the `truffle compile` command.

Automated Analysis

We used several publicly available automated Solidity analysis tools.

Securify does not support 0.4.18 compiler version; the specified version was changed in the code to 0.4.16 for this tool.

Here are the combined results of SmartCheck, Solhint, Securify, and Remix.

All the issues found by tools were manually checked (rejected or confirmed).

Securify* — beta version, full version is unavailable.

Manual Analysis

The contracts were manually analyzed in a shortened procedure, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified. All confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence smart contracts operation in current implementation. We highly recommend addressing them.

Discrepancies with the whitepaper

Our audit brought out the following discrepancies with the whitepaper:

1. Crowdsale start time in the code and in the whitepaper differ.

According to the whitepaper, page 29:

```
“The Crowdsale will begin at 12:00 GMT on Thursday, February 15,
2018.”
```

MinerOneCrowdsale.sol, line 26:

2. According to the whitepaper, page 29:

> “The minimum amount required to participate is 0.01 ETH.”

However, minimum amount is 100 tokens, see MinerOneCrowdsale.sol, line 31:

```
uint256 public constant MIN_TOKEN_AMOUNT = 100e18;
```

3. There is no distribution of pre-ICO tokens in the contract. However, according to the whitepaper:

> “A pre-ICO was held in late 2017 to raise initial funds, during which time 330,000 MIO Tokens were assigned at a preferred rate of 50% off.”.

4. According to the whitepaper:

> “Date of crowdsale end: +90 days or date Hard Cap is reached, if sooner”.

However, the owner can set end time at any moment with setIcoEndTime function.

5. According to the whitepaper:

> “If the Crowdsale campaign does not reach its Soft Cap, all funds will be returned automatically to contributors via Ethereum-based smart contracts, excluding pre-ICO contributors.”

However, there is no automatic refund in the contract: users have to call `claimRefund` or

However, there is a bonus calculation in MinerOneCrowdsale.sol, lines 225–226:

```
bonus = bonus.add(tokens >= LARGE_PURCHASE ? LARGE_PURCHASE_BONUS :
0);

bonus = bonus.add(msg.sender == tokenDeskProxy ? TOKEN_DESK_BONUS :
0);
```

7. According to the whitepaper:

```
"Upon commencement of mining operations, 77 percent of monthly net
output (total output minus electricity, facility, and maintenance
costs, including maintenance staff) will be converted to Ether (ETH)
and automatically distributed to MIO Token Holders via Ethereum-
based "smart contracts."
```

However, the contracts do not distribute net output automatically: someone has to send ETH to the token contract manually.

We highly recommend fixing the whitepaper or/and the code, so that they will fully correspond to each other.

Overpowered contract owner

The contract owner has the following entitlements:

1. Only owner can finalize crowdsale, including initiating refund process, minting token to the team, RnD and bounty, finishing minting.
2. Owner can mint tokens while crowdsale is in process, not sending ETH to the contract (for free) — `mintTokens` function. This way the owner can imitate soft cap. Owner can not mint more than hard cap using this function.
3. Owner can change important crowdsale parameters at any moment using following functions: `setTokenMinter`, `setTokenDeskProxy`, `setIcoEndTime`.

Contrary to a popular misconception, the private modifier does not make a variable invisible. We recommend taking into account that private variables will be visible:

- MinerOneCrowdsale.sol, line 48

```
address private tokenMinter;
```

- MinerOneCrowdsale.sol, line 49

```
address private tokenDeskProxy;
```

Low severity issues

Low severity issues can influence smart contracts operation in future versions of code. We recommend to take them into account.

ERC20 approve issue

There is ERC20 approve issue (MinerOneCrowdsaleFlat.sol, line 199). We recommend instructing users not to use approve directly and to use `increaseApproval/decreaseApproval` functions instead.

Another option is to change the approved amount to 0, wait for the transaction to be mined, and then to change the approved amount to the desired value — link.

Changing the approved amount from a nonzero value to another nonzero value allows a double spending with a front-running attack.

Ineffective use of gas

There is a calculation of remainder in MinerOneToken.sol, line 65:

The remainder is usually about some weis. However, gas usage for this calculation may cost more than the remainder itself.

We recommend to consider omitting the remainder.

Gas limit and loops

The contract includes traversing through an array of variable length. If there are too many items in the array, the execution will fail due to an out-of-gas exception. The issue was found in the following lines:

- MinerOneCrowdsale.sol, line 101
- MinerOneToken.sol, line 120

We recommend paying attention to the sizes of arrays in transactions.

Revert vs require

MinerOneCrowdsale.sol, line 132:

```
if (tokens < MIN_TOKEN_AMOUNT) revert();
```

We recommend using `require(condition);` instead of `if (condition) {revert();}` to reduce gas spending and for better code readability.

Fallback function requires too much gas

The following fallback functions are large:

- MinerOneCrowdsale.sol, line 89
- MinerOneToken.sol, line 76

If the fallback function requires more than 2300 gas, the contract cannot receive ETH by some of standard ways. We recommend to avoid creating big fallback functions. If it is impossible, we recommend instructing users to explicitly call the function with more gas.

- MinerOneCrowdsale.sol, line 11
- MinerOneCrowdsale.sol, line 13
- MinerOneCrowdsale.sol, line 15
- MinerOneCrowdsale.sol, line 17

These addresses might be used for some malicious activity. Thus, we highly recommend to check these addresses after the deployment.

Misspelling

There is a typo in the code in MinerOneCrowdsale.sol, line 206

`_tokekDeskProxy` should be changed to `_tokenDeskProxy`

Conclusion

In this report we have considered the security of MinerOne smart contracts. We performed our audit according to the procedure described above.

The audit showed medium code quality and no critical issues. However, several medium and low severity issues were found. We highly recommend addressing them.

> *This audit was performed by SmartDec, a security team specialized in static code analysis, decompilation and secure development.*
>
> *Feel free to use SmartCheck, our smart contract security tool for Solidity language, and follow us on Medium. We are also available for smart contract development and auditing work.*

SmartDec

More from SmartDec Cybersecurity Blog

Follow

Security tutorials, tools, and ideas

Artyom Orlov · Apr 19, 2018

Lendingblock Smart Contracts Security Audit

Ethereum 9 min read

Artyom Orlov · Apr 17, 2018

Rate3 Smart Contracts Security Audit

Ethereum 7 min read

Artyom Orlov · Apr 13, 2018

NaviToken Smart Contracts Security Audit

Ethereum 7 min read

Artyom Orlov · Apr 9, 2018

MEDIA Protocol Contracts Security Audit

Ethereum 12 min read

Artyom Orlov · Apr 3, 2018

Viola Smart Contracts Security Audit

Ethereum 16 min read

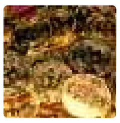
Recommended from Medium

deadeyes

Ecology and Bitcoin : match made in heaven

RickyPolitics

Politics and Cryptocurrency



Coinbase in The Coinbase Blog

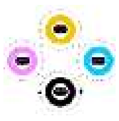
Coinbase plans to add 2,000 employees across Product, Engineering and Design in 2022

LiveArtX in LiveArtX

Welcome to LiveArtX

Limitless Insights

Coinbase Fetch.ai Answers Learn and Earn $3 FET



CryptoSensors

Guide indicators in cryptocurrency trading or the Truman effect in action.

Crabada - An Exciting Undersea Adventure Awaits🦀⚔

Chapter 2: The Clawsome Gift of Giving (Part 1)🎅🦀

Arkadiko Protocol

Gearing up for Testnet ⚙️