Smart Contract Audit

coinspect

MINDS

Minds Smart Contract Audit

Prepared for Minds. • July 2018

MINDSv2018

1. Executive Summary

From May 2018 to September 2019, Minds engaged Coinspect to perform a series of security audits of the Minds Platform which included:

- A gray-box penetration test of their web application.
- A source code review of their web application.
- A source code review of the Minds Token contracts.

The objective of the audits was to evaluate the security of the web application and the smart contracts.

The present report contains the results of the security audit of Minds smart contracts. During this phase of the engagement, Coinspect identified the following issues:

High Risk	Medium Risk	Low Risk	Zero Risk
0	**2**	**5**	**0**

2. Introduction

The contract “`MindsToken`” defines the mintable and ownable ERC-20 compatible token that will be used by the different users of the social network to generate Boosts and Wires. The “`approveAndCall`” function can be used to both approve the transfer of funds to another contract and generate a new Boost or Wire.

The contract “`MindsBoostStorage`” will be the container where all Boost records will be stored, and only the addresses defined by the contract Owner will be able to modify the existing Boosts or generate new ones. This will be done by the “`MindsBoost`” contract which will generate the Boosts and terminate them giving the users the opportunity to either accept, reject or revoke them.

The contract “`MindsWireStorage`” will be the container where all Wire records will be stored, and only the addresses defined by the contract Owner will be able to generate new Wire records. This will be done by the “`MindsWire`” contract which will generate the Wire records and transfer of tokens from one party to the other.

The contract “`MindsTokenSaleEvent`” is the one used for the crowdsale event where users will be able to get MINDS tokens in exchange for ETH. Users will transfer their ETH to this contract and will be awarded in exchange with a number of tokens which rate of conversion with the transferred ETH will be determined at deployment time. The contract will then forward the collected funds to an external wallet address.

The contract “`MindsWithdraw`” will be used by the user to request a transfer from their contribution rewards located in their off-chain Minds wallet. Then the request will be checked offline and if it’s correct, the funds will be transferred to the user on-chain.

A whitebox security audit was conducted on these smart contracts and the dependencies which in this case are the following 7 contracts from OpenZeppelin:

- `zeppelin-solidity/contracts/math/SafeMath.sol`
- `zeppelin-solidity/contracts/ownership/Ownable.sol`
- `zeppelin-solidity/contracts/token/ERC20Basic.sol`
- `zeppelin-solidity/contracts/token/ERC20.sol`
- `zeppelin-solidity/contracts/token/BasicToken.sol`
- `zeppelin-solidity/contracts/token/StandardToken.sol`
- `zeppelin-solidity/contracts/token/ERC20/MintableToken.sol`

The following checks, related to best practices, were performed:

- Confusion of the different method calling possibilities: send(), transfer(), and call.value()
- Missing error handling of external calls
- Erroneous control flow assumptions after external calls
- The use of push over pull for external calls
- Lack of enforcement of invariants with assert() and require()
- Rounding errors in integer division
- Fallback functions with higher gas limit than 2300
- Functions and state variables without explicitly visibility
- Missing pragmas to for compiler version
- Race conditions, such as contract Reentrancy
- Transaction front running
- Timestamp dependence
- Integer overflow and underflow
- Code blocks that consume a non-constant amount of gas, that grows over block gas limit
- Denial of Service attacks
- Suspicious code or underhanded code

3. Summary Of Findings

ID	Description	Risk	Fixed
MND-101	Users can create and complete Boosts on behalf of another user	Medium	✔
MND-102	Users can create and complete Wires on behalf of another user	Medium	✔
MND-103	Outdated Solidity version	Low	✔
MND-104	Use “emit” keyword to raise Events	Low	✔
MND-105	Use the “constructor” keyword to define constructor functions	Low	✔
MND-106	Missing visibility modifiers in several methods	Low	✔
MND-107	Use ‘view’ instead of deprecated ‘constant’	Low	✔

4. Findings

MND-101 Users can create and complete Boosts on behalf of another user

Total Risk	Impact	Location
Medium	Medium	MindsBoost.sol:84
Fixed ✔	Likelihood Low	

Description

As the "`boostFrom`" function can be called by anyone directly (and not through the "`receiveApproval`" and "boost") a user may generate a new boost using any sender address.

For example, a user A may generate a Boost request using a victim (user B) as a sender and himself (user A) as the receiver. As long as the victim has previously approved the transfer of funds from his account to the Minds Token contract, the Boost will be created and the funds transferred to the "`MindsBoost`" contract. In a second step, user A may accept the Boost and get the tokens transferred to his wallet.

Recommendations

Mark the "`boostFrom`" function as internal. This way it can only be called from the "`receiveApproval`" and "`boost`" functions, which specify the sender directly or indirectly as the message sender.

MND-102 Users can create and complete Wires on behalf of another user

Total Risk	Impact	Location
Medium	Medium	MindsWire.sol:60
Fixed ✔	Likelihood Low	

Description

As the “wireFrom” function can be called by anyone directly (and not through the “receiveApproval” and “wire” functions) a user may generate a new wire using any sender address.

For example, a user A may generate a Wire request using a victim (user B) as a sender and himself (user A) as the receiver. As long as the victim B has previously approved the transfer of funds from his account to the attacker’s A, the Wire will be created and the funds transferred to the attacker.

Recommendations

Mark the “wireFrom” function as internal so that it can only be called from the “receiveApproval” and “wire” functions, which specify the sender directly or indirectly as the message sender. This would require a little refactoring, since currently the “wireFrom” function is called from the Minds engine to handle recurring subscriptions.

MND-103 Outdated Solidity version

Total Risk	Impact	Location
Low	Low	All *.sol* contract files
Fixed	Likelihood	
✔	Low	

Description

Currently the code specifies with the pragma statement that is built with an older version of the Solidity compiler, which is not the latest production release. The latest versions have added additional warnings that can help to detect problems, solve bugs and enforce new rules to enhance security.

In the case of the Minds contracts the pragma indicates that compiler version 0.4.13 or above must be used, but the latest version is 0.4.24.

Recommendations

Add the latest version to the pragma statement:

```
pragma solidity ^0.4.24;
```

The notation used above allows for the use of versions between 0.4.24 and 0.5.0.

References

For more information on the use of the version pragma and how to handle the different versions accepted, see the following reference links:

- https://solidity.readthedocs.io/en/develop/layout-of-source-files.html#version-pragma
- https://docs.npmjs.com/misc/semver#versions

MND-104 Use “emit” keyword to raise Events

Total Risk	Impact	Location
Low	Low	All *.sol* contract files
Fixed	**Likelihood**	
✔	Low	

Description

From version 0.4.21 and later, Solidity includes the “emit” keyword which should be used when emitting Events in order to differentiate between the calling of functions and events. Often this can be confusing as for example ERC-20 tokens have the “transfer” function and the “Transfer” event, which are usually called consecutively:

```
transfer(address to, uint value);
Transfer(address from, address to, uint256 _value);
```

Recommendations

Use the “emit” keyword when triggering Events.

References

Proposal: add `emit` keyword; make it the only way to emit events

- https://github.com/ethereum/solidity/issues/2877

MND-105 Use the “constructor” keyword to define constructor functions

Total Risk	Impact	Location
Low	Low	All *.sol* contract files
Fixed	Likelihood	
✔	Low	

Description

As of the current Solidity version (0.4.24), the “`constructor`” keyword should be used to define constructor functions of a contract in place of the now deprecated function with the same name as the contract. This was done in order to prevent errors when renaming the contracts and forgetting to rename the constructor function too.

Recommendations

Use “`constructor`” to define the constructor function instead of using a function with the same name as the contract.

References

For more information on the use of the “constructor” keyword, see the following reference links:

- https://github.com/ethereum/solidity/pull/3635
- https://solidity.readthedocs.io/en/v0.4.23/contracts.html#constructors

MND-106 Missing visibility modifiers in several methods

Total Risk	Impact	Location
Low	Low	All *.sol* contract files
Fixed	**Likelihood**	
✔	Low	

Description

Many of the public methods in the Minds contracts lack visibility modifiers, such as "`public`". It's a good smart-contract programming practice to clearly distinguish between private and public methods, to prevent confusion.

Recommendations

Add the "`public`" modifier to the public methods.

MND-107 Use ‘view’ instead of deprecated ‘constant’

Total Risk	Impact	Location
Low	Low	MindsWire.sol, MindsBoost.sol
Fixed ✔	Likelihood Low	

Description

The modifier `constant` for functions has been deprecated, and `view` should be used instead.

Recommendations

Just replace `constant` with `view` in any function definition that uses the `constant` modifier.

References

For more information on the deprecation of the `constant` keyword for functions and the introduction of `view` and `pure`, see https://github.com/ethereum/solidity/issues/992

5. Observations

After the assessment we are left with some observations about the code that don't really qualify as security issues, but might be valuable as general remarks or head ups:

- The function "`hasPledged`" in `MindsTokenSaleEvent.sol` always returns true, so calling it from "`isValuePledged`" is superfluous;
- The function "`wireFrom`" in `MindsWire.sol` and "`boostFrom`" in `MindsBoost.sol` require `amount >= 0`, but it might be more sensible (and consistent with the comments) to require `amount > 0`, explicitly excluding boosts and wires of 0 value;
- The function "`countWires`" in `MindsWireStorage.sol` doesn't modify the contract state so, as any function that doesn't modify the storage state at all, it can be declared as *view*; this allows them to be executed locally, avoiding the cost of broadcasting a transaction.

6. Disclaimer

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. This report should not be considered a perfect representation of the risks threatening the analysed system, networks, and applications.