Security and Design audit
Decentraland PassThrough contract

Monday, 07-Jan-2019

Agustín Aguilar

Introduction

The Decentraland Team requested the review of the project on the repository https://github.com/decentraland/pass-through, the commit referenced for this audit is 0x38e8809.

The audited contracts are:
- PassThroughStorage.sol: Defines the storage layout for PassThrough.sol.
- PassThrough.sol: Forwards all calls of the contract to a designated estateRegistry, forbidding some methods to be called by a limited user, the “operator”.

The contracts are well written, no high severity issues were found.

Issues

Medium severity

1 - Outdated full source directory

The *full/* directory should contain the complete project joined on a single file, this file is present but it’s outdated and contains a different version of PassThrough.sol, this could lead to the wrong deployment of the contract.

Note:
This report assumes the *full/* directory to be outdated, as the modifications on PassThrough.sol are posterior.

2 - Inability of forwarding calls to multiple contracts

The PassThrough contract may be thought as a multi-signature wallet shared between the owner and operator. This “wallet” fails as is not possible to call any other contract than the designated *estateRegistry*.

This limitation could lead to a loss of funds or limited functionality, any accidental sending of ETH, Tokens, or even parcels could end up with those assets locked on the contract.

Proposal:
Create a method to forward calls to any contract, this method should include the ability to send ETH and should check if the forwarded call is blacklisted or not.

Note:
If the proposal is implemented, consider replacing the current scheme of the blacklisted methods to a scheme allowing to blacklist multiple methods of distinct contracts

Low severity

Two low severity issues were found on the smart contract code and design, and they don’t pose a threat to the security and integrity of the contracts.

3 - Manual calculation of method signature

The PassThrough uses *disableMethod* to blacklist signatures of being forwarded to *_estateRegistry,* it takes a String as a parameter and using *abi.encodeWithSignature* calculates the signature of the method.

Disabling methods using the method name and parameters may be an error-prone process, and in the case of an input error, the *operator* may end up accessing methods that are supposed to be blacklisted.

Proposal:
Make *disableMethod* take a *bytes4* parameter. Using interfaces, function types and selector to blacklist methods inside the constructor.

See:
https://solidity.readthedocs.io/en/v0.4.21/types.html#function-types

4 - Restricted time limited disableMethod functionality

When a method is disabled, it can’t be called for the next 2 years. The 2 years number is hardcoded on the contract but serves no purpose, as the owner could call multiple times disableMethod to prolong the lock time forever.

Any future plan to extend or reduce the duration of the blocking could be challenging to implement on the existing contracts, and would probably require a new deployment and migration.

Proposal:
Make disableMethod take a second parameter as the duration desired for the locking.

Notes

- Check owner and operator in reverse order

If the operator is meant to forward more messages than the owner, it will be recommended on line 36 to reverse the order of the check “owner or operator”, and avoid to read the owner from storage on every forwarded call, saving 400 gas per call.

- Unnecessary switch on assembly code

The line 56, checks the result of the forwarded call using a switch, but there are only 2 possible states, replace the assembly *switch* with an *if* for better readability.

- Keeping 10 000 gas off the forwarded call should not be required

When the call is forwarded to the estateRegistry, the contract forwards all the remaining gas minus 10 000 gas that is kept on the PassThrough contract. This behavior is irrelevant, on an eventual Out of gas exception on the estateRegistry, the PassThrough contract will use that extra gas to revert the transaction.

- Using the old Solidity 0.4.24 compiler

It’s recommended to use the last version of the Solidity compiler, currently being Solidity 0.5.1

- Possible collisions on forwarded methods

If any method of the *estateRegistry* has the same signature of any of the public methods of the PassThrough, that method is not going to be callable, because the PassThrough contract catches the call before it is forwarded. *Note: Related to Issue 1*

- Update free memory pointer after allocating memory

The assembly coded uses *mload(0x40)* to read the free memory pointer, then allocates on that memory the result of the call to the estateRegistry, but it never updates the free memory pointer with the new state of the memory, using something like:

mstore(0x40, add(prt, size))

Note:
In this case, the memory is never used again after the execution of the method; nonetheless, it is a good practice to maintain proper discipline and special care when working with assembly code.

Final thoughts

The PassThrough contract is well written and servers correctly its purpose, no critical vulnerabilities or bugs were found on the contracts.

Is important to meet the following conditions before production release:

1 - Update the joined contract on the *full/* directory, to avoid any confusion on the last stable version of the contract.

2 - Validate that all blacklisted methods match the signature of the compiled and already deployed *estateRegistry*.

3 - Check that no methods on the *estateRegistry* contract have a signature collision with a method on Owanble.sol, PassThroughStorage.sol or PassThrough.sol

January 2019, Madrid, Spain - by Agustin Esteban Aguilar