Marmo Contracts Audit

OPENZEPPELIN SECURITY | APRIL 23, 2019

Security Audits

The RCN team asked us to review and audit their Marmo contracts. We looked at the code and here are the results.

The audited code is located in the marmo-contracts repository. The version used for this report is `d3fb5922a4f01e47d585343d08cccfad659b3584`.

Following are our assessment and recommendations, in order of importance.

Critical Severity

None.

High Severity

Assembly and bytecode without extensive documentation

The Marmo contracts include multiple assembly blocks and a big amount of bytecode. While it is not a security vulnerability right now, this is at the same time the most complicated and the most critical part of the system, it needs to be documented with extra care.

These assembly blocks and bytecode are not extensively documented. Developers may misunderstand the purpose of the code and cause unexpected errors when attempting to modify it.

Consider clearly documenting the intent of each block of assembly code, as well as exhaustively documenting every opcode and every parameter. This will guide future contributors and reviewers when trying to understand, extend or fix the code.

Update: *Partially fixed. On pull request #28 the bytecode of the proxy was moved to a library , making the MarmoStork constructor clearer.*

Implementations have full control over the Marmo wallet

The `relay` function of the `Marmo` contract uses `delegatecall` to forward the intent to an `_implementation` contract chosen by the caller. This implementation contract can do anything, executing within the context of the wallet. So the caller must trust that the implementation will execute the intent. A malicious implementation could do a lot of harm, like overwrite the signer, transfer the ether in the wallet to an address under their control, or call `selfdestruct`.

This gives implementations a lot of power and forces users to verify its deployed code every time they want to relay an intent. Users could call the wrong implementations by mistake or by deception and lose control over their own wallets.

One possible solution would be to whitelist the `EXTCODEHASH` of verified implementations, and to block calls to unverified implementations. That brings limitations and an extra layer of

Update: *This is by design. The RCN team does not plan to control a Marmo ecosystem, but to encourage users to deploy their private Marmo instances to relay their own intents. Also, the system is designed to be used through an SDK which will prevent many problems that could happen if the intents are generated manually.*

Medium Severity

Missing test coverage report

There is no automated test coverage report. Without this report it is impossible to know whether there are parts of the code never executed by the automated tests; so for every change, a full manual test suite has to be executed to make sure that nothing is broken or misbehaving.

Consider adding the test coverage report, and making it reach at least 95% of the source code.

Canceled event not emitted

The `Marmo` contract has a `Canceled` event that should be emitted when an intent is canceled. However, this event is not emitted by the `cancel` function. This will make more difficult for clients to follow the status of intents, forcing them to either listen for all the transactions of the contract or to poll calling `isCanceled`.

Consider emitting `Canceled` at the end of the `cancel` function.

Update: *Fixed in pull request #29.*

MarmoStork does not check the size of the implementation contract address explicitly

The constructor of the MarmoStork contract takes an array of bytes in the `_source` argument, and uses it as the destination address for a `delegatecall`. An ethereum address has 20 bytes, but the size of this array is not checked at the beginning of the function.

The length of this `_source` argument is then used to generate the bytecode of the contract that will delegate the calls. If an array that is not a valid address is passed to this constructor, the resulting bytecode will have an unintended behavior. The part generated from the length of the array is surrounded by other hardcoded bytecode, so it would be very difficult to craft an attack just

addresses, it can even result in the [illegible] opcode instead of the expected [illegible].

Later in the constructor, this argument is converted to an address calling the `toAddress` function which does require the length to be 20 or less. So a bigger array will end up reverting the constructor, but this does not happen explicitly and because of that it has the risk of being removed by mistake.

Consider changing the type of the `_source` parameter to `address`. This will make the expectations clear and prevent any issues derived from invalid addresses. Alternatively, consider requiring the array to be 20 bytes at the start of the constructor, following best practices to fail early and to structure your functions starting with the conditions. Both options have an added benefit: a fixed size would simplify the bytecode generation because more terms can be replaced by constants.

Update: *Fixed in pull request #30. The constructor of MarmoStork now takes and address as parameter.*

Low Severity

README is empty

The README.md files on the root of the git repositories are the first documents that most developers will read, so they should be complete, clear, concise and accurate.

The README.md of the Marmo contracts has no information about what is the purpose of the project nor how to use it.

Consider following Standard Readme to define the structure and contents for the README.md file. Consider including an explanation of the core concepts of the repository, the usage workflows, the public APIs, instructions to test and deploy it, and how it relates to the parts of the project.

Make sure to include instructions for the responsible disclosure of any security vulnerabilities found in the project.

Update: *The README file is no longer empty, but it is still missing the important information that Standard Readme recommends.*

values make the code harder to understand and to maintain.

Consider defining a constant variable for every hard-coded value, giving it a clear and explanatory name. For complex values, consider adding a comment explaining how were they calculated or why were they chosen.

Update: *Partially fixed. Pull requests #33 and #37 fixed the two examples given in this issue. But there are still hard-coded values without explanation like* `32` *in* `Marmo.sol` *line 125,* `0x20` *in* `MarmoStork.sol` *line 82, and more.*

Restricted Address Range is in Draft

The `Marmo` contract can be made unusable by setting the signer to the invalid address `65536`. This address was chosen because it is the first one after the restricted range, as defined by EIP1352: Specify restricted address range for precompiles/system contracts.

This EIP is still a draft, so there is a little risk of it changing in the future and making `65536` one of the restricted addresses. If the code has to be released to production before the EIP is finalized, consider using a random address to invalidate the contract, instead of one so close to the restricted range.

Update: *Fixed in pull request #33. Marmo now uses a higher address as an invalid signer.*

Duplicated Code to Get the Signer

In the `Marmo` contract there is a `signer` function that returns the address of the signer. This code is duplicated in the `init` function.

Consider calling `signer` from the `init` function, instead of duplicating the code.

Update: *The RCN team decided not to fix this in order to save one* `jump` *operation.*

Misleading comment about destroying the wallet

The `Marmo` contract has a special `INVALID_ADDRESS`. According to the comments, the purpose of this address is to destroy the wallet. However, setting the signer to this invalid address

Consider updating the comments to more accurately describe the purpose and effect of the `INVALID_ADDRESS`.

Update: *Fixed in pull request #38.*

Invalid address check uses assert

The `relay` function of the `Marmo` contract fails when the signer is the invalid address. This failure is executed with the `assert` function, so if somebody calls an invalidated contract by mistake, they will lose their gas.

Consider using `require` instead of `assert` to be more forgiving and return the remaining gas when an invalidated contract is called. Also, the semantics of the `require` statement are closer to the intent of this feature.

Update: *Fixed in pull request #34.*

Relayed event is emitted before the action is executed

The `relay` function of the `Marmo` contract emits a Relayed event. This event is emitted before the `delegatecall` is executed. While this is not a security vulnerability because the function is doing proper reentrancy protection, it is safer and clearer to emit events immediately after the action they are signaling is executed.

Consider moving the emit of the `Relayed` event immediately after the `delegatecall` statement.

Update: *This is by design. The RCN team wants to have the execution of an intent between two events.*

Block on intent receipt can overflow

An intent receipt encodes the block in which it was relayed using 95 bits. Theoretically, the block numbers can be bigger than the maximum number that can fit in 95 bits, which means that the block can overflow as mentioned in a comment.

information.

Consider being extra safe by reverting when the block is bigger than the value that can be stored, or by storing the block in a `uint256` variable. Or alternatively, consider documenting more thoroughly the effects of storing the block in 95 bits.

Update: *The RCN team decided not to implement a revert in this case due to the costs and complications of addressing something with this low probability.*

Re-implementing ECDSA signature recovery

The Marmo project includes an implementation of the ECDSA signature recovery function. This function is already part of the OpenZeppelin package, which has been audited and is constantly reviewed by the community.

Consider using the recover function from OpenZeppelin to reduce the code surface area.

An important point related to this implementation is that it is subject to signature malleability, which means that multiple signatures will be considered valid. This also affects the OpenZeppelin implementation, and it is currently being discussed with the community to find the best solution. This issue does not affect the Marmo contracts because the signatures are not used as if they were unique. However, consider documenting this to make it clear that higher layer applications should not consider the signatures unique either.

Update: *The RCN team decided not to add external dependencies to their project*.

No way to check whether a wallet has been revealed

The `MarmoStork` contract has no way to check whether a wallet has been revealed.

Consider adding a mapping to save all the addresses of wallets that have been revealed. Consider emitting an event after a wallet is revealed.

Update: *The RCN team considers this functionality unnecessary because* `extcodesize` *allows to check if an address has code.*

implementation, and the signed transaction data as the id of the intent. This means that an intent can be relayed only once.

This is correct to prevent replay attacks. However, note that there will be many valid cases to call the same transaction multiple times. Consider adding a nonce to the intent data, to allow users to relay the same transaction securely.

Update: *The Marmo SDK adds a salt to the transaction data to allow sending the same intent multiple times.*

Notes & Additional Information

- In the package.json file, ethlint and solium are listed as dependencies. Solium has been deprecated starting with version 1.2.0 on 2018–12–25, and was renamed to Ethlint. Consider removing the Solium dependency.
- The Truffle config file is full of comments and commented options coming from the template. Consider removing all the things that are not relevant for the developers of the Marmo project.
- The docstrings of the contracts are not following the Ethereum Natural Specification Format (NatSpec). Consider following this specification on everything that is part of the contracts' public API.
- In the `Marmo` contract, there are two cases that call `revert("Unknown error");` (L95 and L138). While it is a good safeguard to catch unexpected conditions, these states should be impossible to reach. With a unit test suite that covers all the possible code paths, the impossibility of those states can be verified. Consider adding all the required unit tests to safely remove the `revert` statements. If you prefer to stay extra safe and keep these statements, consider modifying the code to use `assert` instead, to accurately reflect your intentions.
- An intent receipt encodes the block in which it was relayed. When the intent is canceled, the block value is set to 0. Consider using the receipt to also record the block in which the intent was canceled, which might be useful for user interfaces or other projects using the system.
- To favor explicitness and readability, several parts of the contracts may benefit from better naming. Our suggestions are:
  – `MarmoStork` to `MarmoWalletFactory`.

– `reveal` to `createMarmoWallet`.
– `p` to `marmoWallet`.
– `Marmo` to `MarmoWallet`.
– `signer` to `existentSigner`.
– `relayedBy` to `getIntentRelayer`.
– `relayedAt` to `getBlockOfIntentExecution`.
– `Relayed` to `IntentRelayed`.
– `Canceled` to `IntentCanceled`.
– `isCanceled` to `isIntentCanceled`. Maybe even consider here modifying the implementation to avoid the negative, and call it `isIntentValid`.
– `relay` to `relayIntent`.
– `_implementation` to `_executor`.
– `MarmoImp` to `MarmoExecutor`.
– `Receipt` to `IntentExecuted`.

Note that most of the projects similar to Marmo are using the term meta-transactions instead of intents. While intent is a good name for this concept, consider renaming it to be more aligned with the ecosystem and what people are starting to expect.

Conclusion

No critical and two high severity issues were found. Some changes were proposed to follow best practices and reduce the potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Marmo contracts. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits