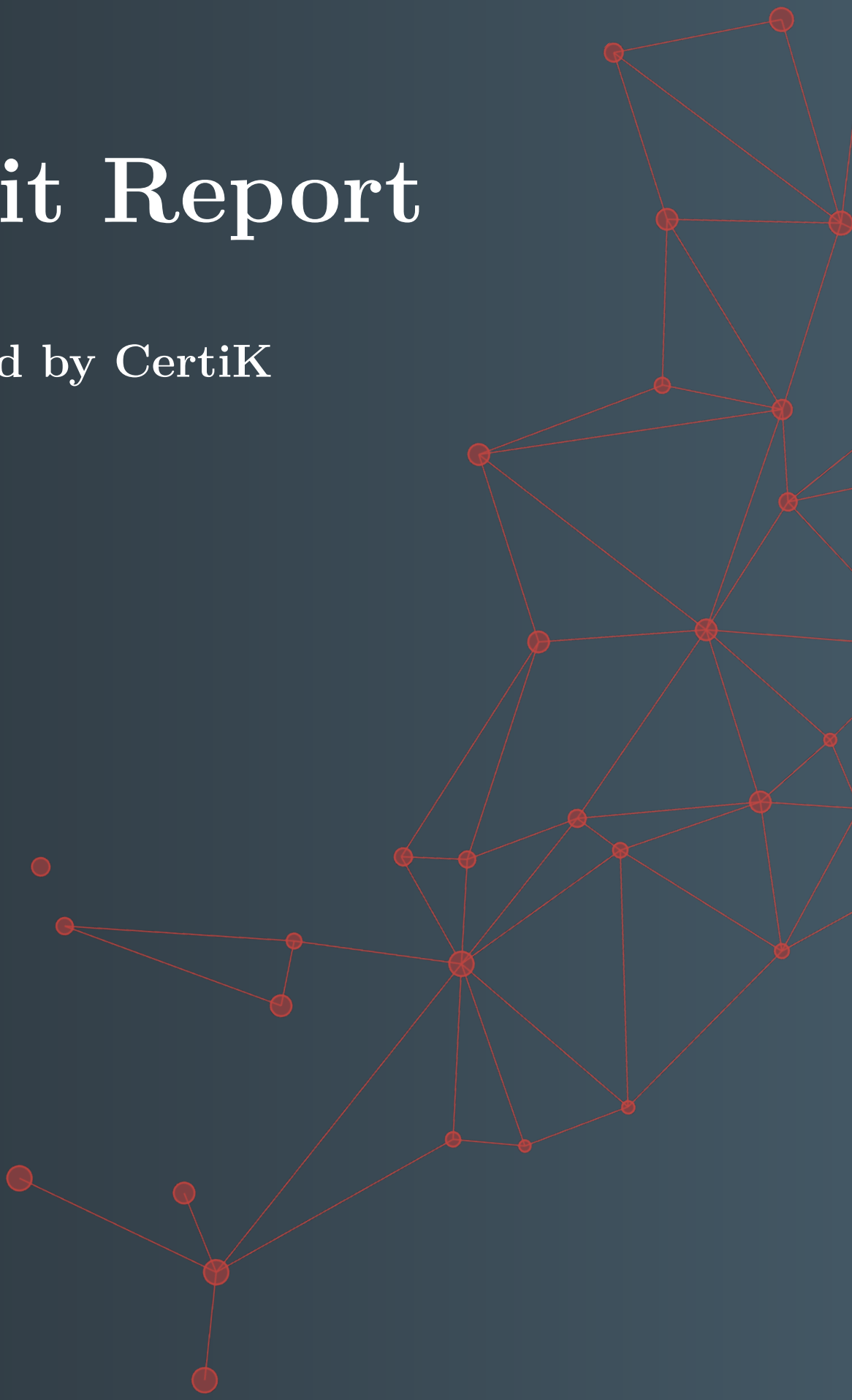
CERTIK

Audit Report

Produced by CertiK

for Matic



Contents

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Matic (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement").

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that the project is checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and verifications on the project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor.

Executive Summary

Matic Network is a Layer 2 scaling solution that achieves scale by utilizing sidechains for off-chain computation while ensuring asset security using the Plasma framework and a decentralized network of Proof-of-Stake (PoS) validators. Matic strives to solve the scalability and usability issues while not compromising on decentralization and leveraging the existing developer community and ecosystem. A series of thorough security assessments have been carried out, the goal of which is to help Matic protect their users by finding and fixing known vulnerabilities that could cause unauthorized access, loss of funds, cascading failure, and/or other vulnerabilities. Alongside each security finding, recommendations on fixes and best practices have also been given.

Testing Summary

SECURITY LEVEL

Smart contracts Audit

This report has been prepared as a product of the smart contract audit request by Matic.

This audit was conducted to discover issues and vulnerabilities in the source code of smart contract implementation.

TYPE	Smart contracts
SOURCE CODE	https://github.com/maticnetwork/pos-portal/
LANGUAGE	Solidity
REQUEST DATE	July 24, 2020
REVISION DATE	Aug 19, 2020
METHODS	A comprehensive examination has been performed using Whitebox Analysis. In detail, Dynamic Analysis, Static Analysis, and Manual Review were utilized.

Review Notes

Overview

A primary focus for the audit is to have a thorough look at the smart contracts that power the PoS (proof-of-stake) based bridge mechanism for Matic Network. Specifically we want to make sure that the exit mechanism is correctly implemented and cannot be exploited to withdraw the tokens deposited on the sidechain more than once per transfer.

Scope of Work

- The audit work was scoped to a specific commit `c810a2400e54f61014943544719246f0c8b66401` of the source code per the agreement
- The codebase are divided into modules of smart contracts based on their functionalities:

Child

Smart contracts	Description	PASS
ChildChainManager	• Operates on child chain • Keeps a mapping for corresponding token addresses between the root chain and child chain • Syncs deposit transactions	

ChildToken	• Smart contract to add functionalities to deposit from and withdraw to the root chain on top of ERC20, ERC721, ERC1155, Mintable721 and MaticWETH tokens.	

Root

Smart contracts	Description	PASS
RootChainManager	• Deposits tokens from the root chain to the child chain • Implements the general exit mechanism	
RootToken	• Implementation of ERC20, ERC721, ERC1155, Mintable721 tokens on root chain	
StateSender	• Sends data to the child chain	
TokenPredicates	• Locks tokens after deposit from the root chain to the child chain • Implements specific exit mechanism for each token	

Lib

Contains implementation of the RLP encoding, Merkle tree proof verification, Merkle Patricia Tree proof verification.

Common

Contains standard solidity libraries for smart contract upgradability and access control.

Audit Summary

The codebase of the project was identified to be carefully designed and detailed, as well as properly documented. In total we found **one critical issue** in the exit mechanism (Exhibit 1) that enables malicious attackers repeated withdrawal from the childchain. All other issues were of negligible importance and mostly referred to coding standards and inefficiencies.

In the second round we have found one major issue that enables replay attacks, one between the child chain and main chain, second between different contracts on the child chain.

Audit Revisions

On 11th August 2020 the pull request `preliminary-audit-fixes` with commit `f33407cbecfd0dbbd3da2da849efbc60e6018c7d` was submitted. This pull request fixed almost all listed issues except 5, 11, 15. The changes were approved by the Certik audit team on the same day.

On 19th August 2020 the pull request `feature/meta-transaction` with commit `8dcc8b4b6476bfdfaa10eff76e1eeed178f252ee` was submitted. The pull request has fixed all the remaining issues. The changes were approved by the Certik audit team on the same day.

Audit Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Repeated exit	Security	Critical	MerklePatriciaProof.sol Lines 150-153

Description:

RootChainManager.sol lines 250-255 the `exitHash` is determined by three factors, one of which is `inputDataRLPList[8]`, the branch mask, i.e. the path in the receipt merkle patricia trie from the root to the corresponding burning transaction receipt.

The path is in bytes and translated to hex by `_getNibbleArray()` in `MerklePatriciaProof.sol`. There are 2 cases depending whether the path in hex has odd or even length. In case on line 150 we accept every beginning nibble except 1,3 and erase the first two nibble in the hex array, which means two arrays 2055 and 4055 would produce the same `_getNibbleArray()` to bypass `verify`, but they produce different `exitHash` so one can exit more than once.

Recommendations:

One needs to check in the `else` case that the first two nibbles are 20 (we don't allow paths ending in extension nodes here).

Alleviation:

In commit `f33407c` the computation of exitHash the component inputDataRLPList[8] is changed to `MerklePatriciaProof._getNibbleArray(inputDataRLPList[8].toBytes())`, which makes the branchMask unique.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Redundant return statement	Code optimization	Informational	MerklePatriciaProof.sol Lines 102, 107

Description:

The default return value is `false` so the return statements on lines 107 and 102 are not needed and the `else` case on line 101 can be omitted.

Recommendations:

Omit the unnecessary code.

Alleviation:

The recommendation has been assimilated in commit `f33407c`.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Integer underflow	Arithmetic	Informational	RootChainManager.sol Line 342

Description:

The subtraction `blockNumber - startBlock` is unsafe and can cause integer underflow, but we believe this cannot be exploited in any way as it would certainly cause the Merkle proof verification to fail.

Recommendations:

Use SafeMath for arithmetic operations.

Alleviation:

SafeMath usage has been added in commit `f33407c`.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Unlocked compiler version	Compiler version	Informational	All smart contracts headers

Description:

An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers.

This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendations:

We advise that the compiler version is instead locked at a specific version possible that the full project can be compiled at.

Alleviation:

Compiler version has been locked in commit `f33407c`.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Interface structure	Coding style	Informational	IChildToken.sol

Description:

The IChildToken interface contains only the deposit function and is inherited in every Child Token contract. Every Child Token contains an additional withdraw function which is essential because it enables token withdrawal from the child chain to the main chain so we believe this function belongs to the general pattern as well, i.e. IChildToken interface.

Recommendations:

Add withdraw to the IChildToken interface.

Alleviation:

All child tokens are not expected to have the same interface for withdraw. For eg. ERC20 tokens will have single param for amount and ERC1155 will have 2 params for amount and tokenId. Due to this reason, not adding withdraw function to IChildToken interface.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Multiple lookups from storage	Gas optimization	Informational	RootChainManager.sol Lines 275, 281

Description:

The value in storage `childToRootToken[childToken]` is looked up twice in exit function. Each lookup costs 200 gas each. It would be better to assign this value to a memory variable, because both memory assignment and look up cost 3 gas each.

Recommendations:

Use memory assignment to avoid multiple storage lookups.

Alleviation:

The recommendation has been assimilated in commit `f33407c`.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Depositor role	Function logics	Informational	All Child Token contracts

Description:

In the constructors of `ChildToken` contracts the depositor role is assigned to the contract's creator. The depositor is an important role as it is the only address that is authorized to deposit the tokens from root contract to child contract. From the natspec it is evident that only the `ChildChainManager` contract should possess this role and this contract does not deploy the `ChildToken` contracts so one would needs some transactions to pass over the depositor rights.

Recommendations:

Assign the depositor role directly to the ChildChainManager in the constructor.

Alleviation:

In commit `f33407c` the depositor role has been initiated to `childChainManager`.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
`require` error message	Coding style	Informational	RLPReader.sol Line 104, 168, 208,

Description:

`require` can be used to check for conditions and throw an exception if the condition is not met, in which case the error message provided by the developer will appear. This is why a very descriptive error message is needed.

Recommendations:

Adding an error message describing the failed condition.

Alleviation:

In commit `f33407c` error messages have been added.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
`isList()` verification	Logics	Informational	RLPReader.sol Line 165-183

Description:

The function `numItems()` calculates the number of elements in an `RLPItem` representing a list, so it should check whether the input indeed represents a list. For example it would return the length of a string. On the other hand adding a `require` check is not necessary and would only cost more gas, since the only place this function is used is in `toList()` and the aforementioned condition is already checked. The exhibit is here just for completeness in case the `RLPReader.sol` library is expanded and the function `numItems()` is used in additional functions.

Recommendations:

Add `require` check if necessary.

Alleviation:

In commit `f33407c` the check and issue description have been added to the comments of `numItems()` function

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary `if` clause	Logics	Informational	RLPReader.sol Line 229

Description:

Checking in the function `numItems()` whether `item.len == 0` to return zero is not needed since if `item.len == 0` then it does not encode a list, because the RLP encoding of an empty list is `0xC0`. Moreover the function `numItems()` is only used in `toList()` and right before calling `numItems()` the condition `isList()` is checked which excludes the possibility of an empty item.

Recommendations:

Omit the `if` clause.

Alleviation:

In commit `f33407c` the recommendation has been assimilated.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Inefficient comparison	Logics	Informational	RLPReader.sol Line 234

Description:

In function `numItems()` by the definition of `RLPItem` it is clear the the value of `currPtr` can never exceed `endPtr` and the while loop on line 234 would stop when `currPtr` and `endPtr` are equal, hence instead of `currPtr < endPtr` we can use `CurrPtr != endPtr` as each not equal comparison costs 3 less gas (negligible for short `RLPItem`, this exhibit is just here for completeness) than less than comparison.

Recommendations:

Use not equal instead of less than.

Alleviation:

The improvement is negligible so the Exhibit was not applied.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Exclusion of empty bytes input	Implementation	Informational	RLPReader.sol Lines 53-64

Description:

In several functions of `RLPReader` library we check that `item.len` is not zero. Indeed any RLP encoding (even of empty string or empty array) is non-empty, this means we should check this condition in the function `toRlpItem()` to exclude this case before hand.

Recommendations:

Check that the input bytes are not empty in the function `toRlpItem()`.

Alleviation:

In commit `f33407c` the length check has been added.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Non unique uint encoding	Implementation	Minor	RLPReader.sol Lines 173-205

Description:

The functions `toUint()` and `toUintStrict()` only take `RLPItem` representing `uint` as input. To get the data bytes only the length of the payload offset is calculated, whereas the content of this prefix is not considered, which means invalid `RLPItem` with prefixes of incorrect length or short list would still get through.

Recommendations:

Check the payload offset content in `toUint()` and `toUintStrict()`.

Alleviation:

In commit `f33407c` the prefix check has been added.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Incorrect encoding of value “false”	Implementation	Minor	RLPReader.sol Lines 155-164

Description:

According to RLP encoding specification the special value “false” is encoded as “0x80”, whereas in current implementation of the function “toBoolean” it is “0x00”.

Recommendations:

Change the implementation to follow the specification.

Alleviation:

toBoolean() is not being used anywhere in the system, the function is removed completely.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Replay attack between child chain and root chain	Implementation	Major	NetworkAgnostic.sol 37-87

Description:

Matic wants to enable users to send transactions to matic network without changing the network in their wallet so the chainId is the same for both child chain and main chain. This opens an attack vector when a user wants to send a transaction only to the child chain, then the attacker can submit the signed transaction to the other chain against that user's will. The requires the same nonce, which can happen, and the same contract address on both chains to have an exploitable function, for example when the two contracts are deployed from the same address.

Recommendations:

Change the chainID and check the chainID before transaction execution.

Alleviation:

Using same chainId for child token contracts is exploitable. Changed NetworkAgnostic feature to native meta transaction in commit `8dcc8b4`. The opcode chainid is used so it is always the native chain id. This chain id is used as salt in EIP712 domain separator so that metamask allows signing tx with different chain id than currently selected network.