CERTIK

CertiK Preliminary Audit Report for Onchain

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 PolyNetwork (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

PolyNetwork is built to implement interoperability between multiple chains in order to build the next generation internet infrastructure. Authorized homogeneous and heterogeneous public blockchains can connect to PolyNetwork through an open, transparent admission mechanism and communicate with other blockchains. Popular blockchain networks such as Bitcoin, Ethereum, Neo, Ontology, and Cosmos are already a part of PolyNetwork.

A series of thorough security assessments have been carried out on the cross chain manager smart contract. The goal of the audit was to help the Onchain project 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

TBA after remediations

Smart contracts Audit

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

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/polynetwork/eth-contracts
LANGUAGE	Solidity
REQUEST DATE	July 24, 2020
REVISION DATE	Aug 14, 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 cross chain transaction mechanics of Polynetwork. Specifically we want to make sure that the utility libraries for data manipulation are correctly implemented, meta transactions from any chain are correctly processed and executed only once on Ethereum chain.

Scope of Work

- The audit work was scoped to a specific commit `631db01f333b581f9402f922ab0db48aa8f3f8a7` of the source code per the agreement
- The codebase are divided into modules of smart contracts based on their functionalities:

Core

Group	Description	PASS
Assets	• Represent token assets of other chains on Ethereum • Currently in scope BTC, ONGX, ONTX as ERC20	
CrossChainManager	• Store and sync data with PolyChain • Encodes transaction from Ethereum chain • Decodes transaction to Ethereum chain	

Libs

Smart contracts	Description	PASS
Common	• Data manipulation, serialization and deserialization in low-level inline assembly	
EthCrossChainUtils	• Verification of data from PolyChain	
Other	• Access control, pause functionalities, safe math	

Audit Summary

The codebase of the project was identified to be overall well designed and detailed. In total we found **one critical issue** in the signature verification (Exhibit 1) that enables malicious attackers bypass the signature threshold and take control of Polychain connection on Ethereum. There are two other major issues in function implementation which lead to undesired behaviour.

Audit Revisions

On 21st August the Polychain dev team submitted the commit `94f35cb9a0b8298c39f248e173f2a18fef780564`, which contains remediations for most of the listed issues, especially the critical and major ones. The Certik audit team has verified and approved the changes.

Audit Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Unique signatures not guaranteed	Security	Critical	`EthCrossChainUtils.sol` L109-125

Description:

In `EthCrossChainManager.sol` the public function `changeBookKeeper()` can be called to change the set of consensus nodes. In order for the transition to be successful the caller has to provide enough signatures (at least ⅔ of the current consensus nodes). The validity of these signatures is verified by `verifySig()`, but this function does not check whether a signature has been used or not, this means it suffices to acquire one valid signature and repeats enough times to bypass this check.

Recommendations:

Check whether a valid signature/address has been used.

Alleviation:

This issue has been addressed in commit `94f35cb` by additional helper function `containMAddresses()`.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Untight storage packing	Gas optimization	Information	`EthCrossChainUtils.sol` L6-18

Description:

The EVM storage saves data in chunks of 32 bytes, which means if we pack data fields of a struct tightly regarding this restriction a higher gas and storage efficiency will be reached, as `SSTORE` is an expensive opcode. The struct Header contains several fields of types `uint32`, `uint64`, which can be placed together to occupy one 32 bytes slot.

Recommendations:

Put the numerical fields next to each other.

Alleviation:

The recommendations were assimilated in commit `98f9c58`.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Incorrect proof size computation	Function logics	Major	`EthCrossChainUtils.sol` L50

Description:

In function `merkleProve()` the variable denotes the size of the Merkle proof and is computed as `(_auditPath.length - off)/32`, but each proof component consists of not just the node hash (32 bytes) but also the node's position either left or right. This position is represented by one byte (`0x00` for left, `0x01` for right), so the divisor should be 33 instead of 32.

Recommendations:

Change 32 to 33 in the size computation.

Alleviation:

The recommendations were assimilated in commit `a09718c.`

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Possible redundant data in Merkle proof	Function logics	Informational	`EthCrossChainUtils.sol` L50

Description:

The function `merkleProve()` does not check whether `_auditPath.length - off` is a multiple of the proof component length (32 in the code but as explained in exhibit 3 it should be 33), so there is a possibility of passing along a small piece of redundant information, which gets ignored by proof verification, but this shouldn't cause any issue.

Recommendations:

Check that `_auditPath.length - off` is a multiple of proof component length.

Alleviation:

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Integer overflow	Arithmetic	Informational	`EthCrossChainUtils.sol` L99

Description:

In function `verifyPubkey()` the arithmetic expression in line 99 can overflow and leads to unexpected behaviour. However we believe this cannot cause any issue, since the outputs of `verifyPubkey()` would most likely not pass additional verification in `initGenesisBlock()` and `changeBookKeeper()`, the only places this function is used.

Recommendations:

Use `SafeMath` for the arithmetic expression.

Alleviation:

A check was added in commit `98f9c58`.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Redundant computation	Coding style	Informational	`EthCrossChainUtils.sol` L76

Description:

On line 76 the variable `buff` is an empty byte array, as it was declared one line above, so including it as input in `abi.encodePacked()` is redundant.

Recommendations:

Omit `buff` in `abi.encodePacked()`.

Alleviation:

The recommendations were assimilated in commit `98f9c58`.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Multiple lookups from storage	Gas optimization	Informational	`EthCrossChainUtils.sol` 81, 82

Description:

In function `_getBookKeeper()` the value of `Utils.slice(_pubKeyList, i*POLYCHAIN_PUBKEY_LEN, POLYCHAIN_PUBKEY_LEN)` is used to derive the `keepers` addresses and `nextBookKeepers`. It is computed twice in the same function, so to save gas we can save the value in a memory variable instead.

Recommendations:

Use memory assignment to avoid multiple storage lookups.

Alleviation:

The recommendations were assimilated in commit `8a0d92c`.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Possible redundant data in signature list	Function logics	Informational	`EthCrossChainUtils.sol` L113

Description:

The function `verifySig()` does not check whether `_sigList.length` is a multiple of `POLYCHAIN_SIGNATURE_LEN`, i.e. 65, so there is a possibility of passing along a small piece of redundant information, which gets ignored by signature verification, but this shouldn't cause any issue.

Recommendations:

Check that `_sigList.length` is a multiple of `POLYCHAIN_SIGNATURE_LEN`.

Alleviation:

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Redundant variable declaration	Gas optimization	Informational	`EthCrossChainUtils.sol` L119

Description:

In function `verifySig()` there is a loop which checks the validity of signatures in `_sigList`.

It is more gas efficient to declare `signer` outside the loop and assign the output of `ecrecover()` to it instead of declaring it in each iteration.

Recommendations:

Declare the variable `signer` before the `for` loop.

Alleviation:

The recommendations were assimilated in commit `928eeb1`.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `for` loop	Gas optimization	Informational	`EthCrossChainUtils.sol` L114-123

Description:

Function `verifySig()` verifies there are enough valid signatures using a `for` loop. The function returns `True` if and only if the number of valid signatures is at least a certain threshold. To save gas we can put a check inside the loop to return `True` as soon as the number of valid signatures reaches the threshold

Recommendations:

Put a threshold check inside the loop.

Alleviation:

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Function naming convention	Coding style	Informational	`EthCrossChainUtils.sol` Line 161

Description:

Function `deserializMerkleValue()` should be name `deserializeMerkleValue()`.

Recommendations:

Rename the above mentioned function.

Alleviation:

Recommendations were assimilated in commit `3c95562`.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Unupdated library description	Documentation	Informational	`ZeroCopySink.sol` `ZeroCopySource.sol`

Description:

The description of these two libraries are not updated and still corresponds to the `SafeMath` library.

Recommendations:

Update the library description.

Alleviation:

The library descriptions were updated in `0230d5a`.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Untight memory packing	Memory optimization	Informational	`ZeroCopySink.sol` L59, L84, L108, L133, L159

Description:

The functions `WriteUint()` convert unsigned integers to bytes using low-level inline assembly, which necessitates manual memory allocation. After allocation we need to update the free memory pointer at slot `0x40`, for example line 59 `mstore(0x40, add(buff, 0x40))`. Here it is possible to change the second `0x40` to `0x21` because `uint8` only occupies one byte (the number is left-aligned thanks to `shl(246, v)`), hence the resulting bytes array would occupy only `0x21` bytes in memory (additional `0x20` bytes for memory pointer). Similar modifications can be made to other WriteUint() functions.

Recommendations:

Change the free memory pointer update as explained above.

Alleviation:

The recommendations were assimilated in `38c18fd`.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Integer Overflow	Arithmetic	Minor	`ZeroCopySource.sol` L23, L46, L60, L78, L99, L126, L154, L184, L243, L257

Description:

`offset` denotes the position in the byte array, from which to read the next piece of data. The validity of `offset` is checked by whether its sum with the data length is less or equal to the length of the byte array. By choosing the `offset` value sufficiently large the addition can overflow, thus bypasses the check and leads to unexpected behaviour.
For example in function `NextUint64()` if offset is `uint256(-8)` then `offset + 8` would be 0, so the check on line 126 would be bypassed. Note that the integer overflow phenomenon also occurs in inline assembly, thus the function would return the number represented by the last 8 bytes of the 32 bytes slot pointed by `buff`, which most likely would be 0.

Recommendations:

Use `SafeMath` to avoid integer overflow.

Alleviation:

The recommendations were assimilated in commit `1ee9102.`

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary comparison with zero	Arithmetic	Informational	`ZeroCopySource.sol` L172 `ZeroCopySink.sol` L143 `Utils.sol` L28, L36, L319, L321

Description:

The comparison `v >= 0` in of these places is unnecessary because v is of type `uint256`, thus automatically greater or equal to 0, whereas the comparison `v > 0` can be simplified to `v != 0` to save some gas.

Recommendations:

Omit `>= 0` comparison and change `> 0` to `!= 0` for `uint` variable.

Alleviation:

Recommendations were assimilated in commit `4b50b50`.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Possible incorrect number range prefix	Function logics	Minor	`ZeroCopySource.sol` L265-278

Description:

In `ZeroCopySink.sol` lines 173-183 the function `WriteVarUint()` outlines a prefix system which specifies the range of the following number: no prefix for `[0, 0xFD)`, prefix `0xFD` for `[0xFD, 0xFFFF]`, prefix `0xFE` for `(0xFFFF, 0xFFFFFFFF]` and prefix `0xFF` for `(0xFFFFFFFF, 2**64)`. The function `NextVarUint()` in `ZeroCopySource.sol` then uses this system to convert the bytes to numbers. The problem is the resulting number is not checked whether it indeed lay in the range specified by the prefix. This can cause a mismatch in bytes encoding for example `Uint64(NextVarUint(0xFD0100, 0))` would give 1, so encoding it back to bytes with `WriteVarUint(1)` would give `0x01`, which is different from `0xFD0100`.

Recommendations:

Make sure that the resulting number lies in the intended range.

Alleviation:

Recommendations were assimilated in commit `008d923`.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Incorrect implementation	Function logics	Major	`Utils.sol` 328

Description:

The natspec indicates that the height of the query must be greater than the height of the init genesis blockheight, but in case of `_len = 1` the function returns immediately `(0, true)`. We believe it is necessary to compare the target number with the single element of the array first.

Recommendations:

Compare the target number with the single element of the array in case of `_len = 1`.

Alleviation:

The function was not used anywhere, hence deleted in commit `008d923`.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Incorrect implementation	Function logics	Minor	`Utils.sol` 345

Description:

We believe that in the function `findBookKeeper()` natspec the description "output is the index whose value in `_arr` is the closest to the target `_v`" is not complete. If the function execution reaches line 345 then we have the following situation: `left = right + 1`, `right < _v < left`. Firstly this means the condition `_arr[left - 1] < _v` is redundant. Secondly the output would always be `left - 1 = right`, even though `_v` could be closer to `_arr[left]` than to `_arr[left - 1]`. For example the input `_arr = [1,4]`, `_v = 3` would return index 0 with value 1 instead of index 1 with value 4.

Recommendations:

Omit the redundant check `_arr[left - 1] < _v`. Furthermore either change the natspec, so it correctly describes the function `findBookKeeper()` or fix the function, so it follows the natspec. Namely check which endpoint has value closer to `_v`.

Alleviation:

The function was not used anywhere, hence deleted in commit `008d923`.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Misleading comment	Documentation	Informational	`Pausable.sol` L28, L59, L67

Description:

The natspec indicates that certain addresses can be assigned pauser role, but the contract does not have this feature.

Recommendations:

Change the natspec or add pauser role feature.

Alleviation:

The recommendations were assimilated in commit `311356c`.

Exhibit 20

TITLE	TYPE	SEVERITY	LOCATION
Incorrect length check	Function logics	Informational	`Utils` L271

Description:

In function `compressMCPubkey()` line 271 we require that the `key.length >= 34`, but on line 273 we try to access `key[66]`. Isn't it better to require directly `key.length >= 67`?

Recommendations:

Change the requirement from `key.length >= 34` to `key.length >= 67`.

Alleviation:

The recommendations were assimilated in commit `311356c`.

Exhibit 21

TITLE	TYPE	SEVERITY	LOCATION
Redundant require check	Function logics	Informational	`LockProxy.sol` L77

Description:

In function `lock()` we require that `amount >= 0`, even though `amount` is of type `uint256`, so this condition is automatically true or was it meant to be `amount > 0` instead?

Recommendations:

Omit or change the require check.

Alleviation:

The recommendations were assimilated in commit `311356c`.

Exhibit 22

TITLE	TYPE	SEVERITY	LOCATION
Misleading variable name	Coding style	Informational	`LockProxy.sol` L76

Description:

`fromAssetHash` is a token address, why is it called hash?

Recommendations:

Change the name or explain the variable's purpose in natspec.

Alleviation:

The recommendations were assimilated in commit `311356c`.

Exhibit 23

TITLE	TYPE	SEVERITY	LOCATION
Unintended transfer behaviour	Function logics	Minor	`LockProxy.sol` L143-153

Description:

In case of `fromAssetHash = address(0)` and `msg.value = 0` the if condition on line 144 will fail and the execution will enter the else branch unnecessarily and make an expensive external call to `address(0)`.

When `fromAssetHash` is a token address but `msg.value > 0` ether is still sent to the LockProxy contract but not accounted for.

Recommendations:

In if condition only check `fromAssetHash` and `check msg.value` later inside the branch.

Alleviation:

The recommendations were assimilated in commit `311356c`.

Exhibit 24

TITLE	TYPE	SEVERITY	LOCATION
Argument length checks missing	Coding style	Informational	`EthCrossChainManager.sol` L25

Description:

In the natspec the function `initGenesisBlock()` should only receive nonempty `rawHeader` and `pubKeyList` as inputs, but this is not checked anywhere.

Recommendations:

Check to make sure that the inputs are nonempty.

Alleviation:

Exhibit 25

TITLE	TYPE	SEVERITY	LOCATION
Redundant check	Security	Major	`ERC20Extended.sol` L61

Description:

The function `bindAssetHash()`'s input chainId is of type uint64, hence the check `chainId >= 0` is unnecessary.

Recommendations:

Omit the redundant check

Alleviation:

Exhibit 26

TITLE	TYPE	SEVERITY	LOCATION
Unintuitive return value	Coding style	Informational	`BTCX.sol` L24

Description:

The function `setMinimumLimit()`'s return value is of type bool, which was not set during the execution, so it will always be assigned the default bool value, which is `False`. This behaviour can be confusing, especially when one expects a successful function call to return `True`.

Recommendations:

Return `True` when the function executes successfully.

Alleviation: