PolyNetwork Hack Analysis

CertiK

·

Follow

5 min read
·
Aug 12, 2021

The hacker/s used fake credentials at the front desk, gaining the trust of the receptionist; the receptionist then handed over the keys to the vault.

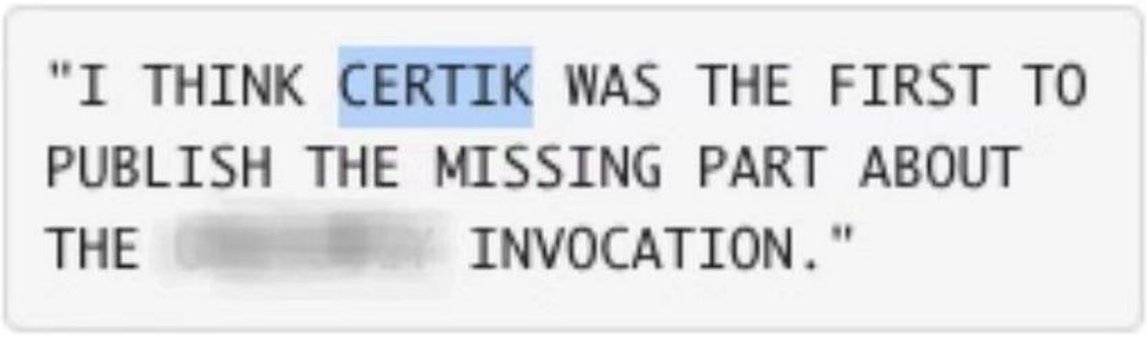
Incident Description

On Aug 10, 2021, PolyNetwork suffered a cross-chain attack that resulted in a total loss of $600M. The attacker created various malicious transactions on multiple chains and used the Relayer component to orchestrate the entire hack.

Using the vault example above, the hacker/s used fake credentials (invalid transaction on the Side Chain) to get a real key (signed Merkle certificate on the Alliance Chain) from the front desk receptionist (Relayer).

This report presents the full attack through an analysis and compartmentalization of the malicious transactions and the off-chain Relayer.

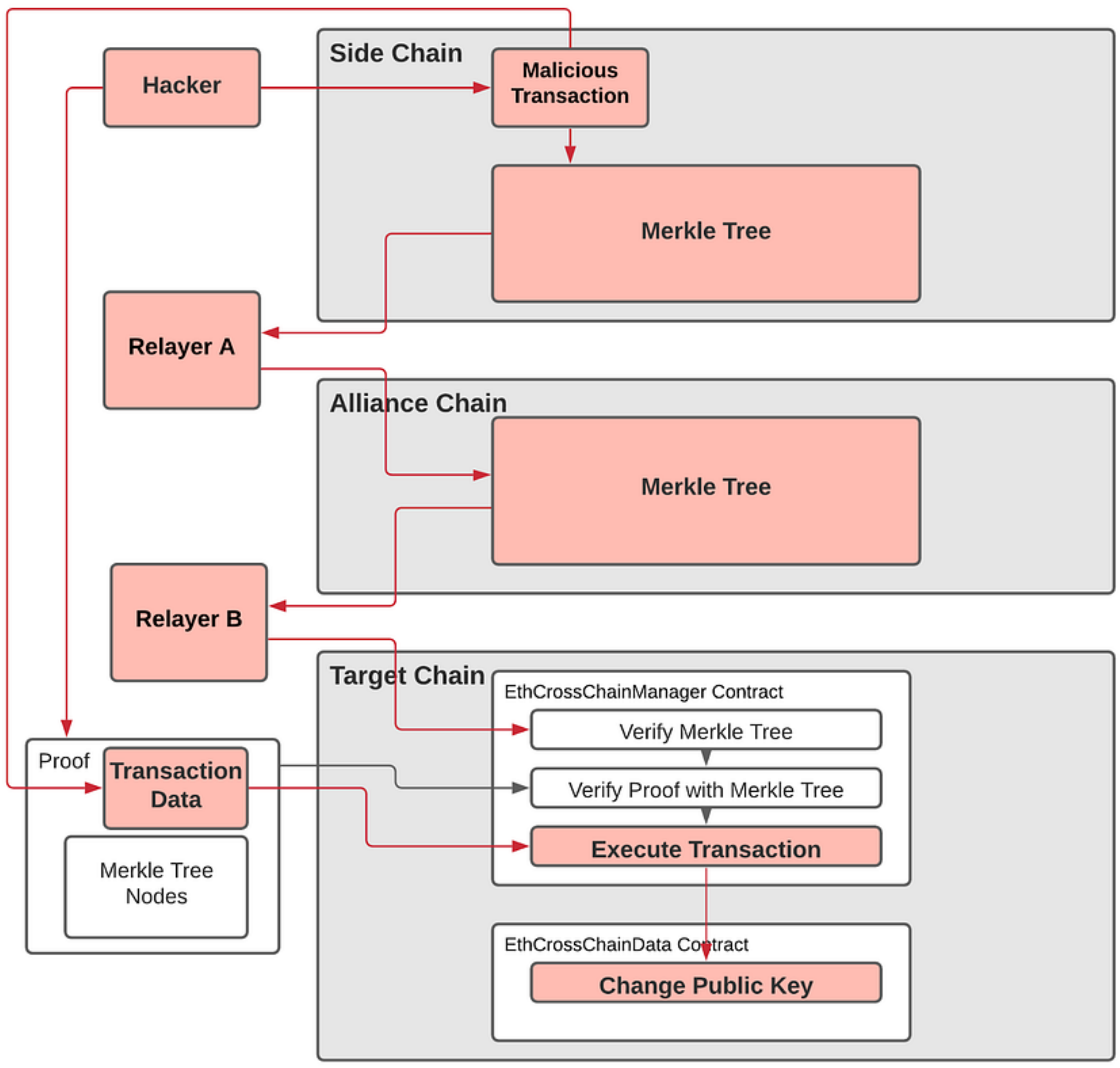
The note from the Mr. 600 Million White Hat indicates that CertiK's analysis finds the missing piece of this attack:
https://etherscan.io/tx/0x42446ccc66bb48eac7bd905ae7d79708f303849802b280eb4d65770c1bfc0997


"I THINK CERTIK WAS THE FIRST TO PUBLISH THE MISSING PART ABOUT THE [redacted] INVOCATION."

Vulnerability Analysis

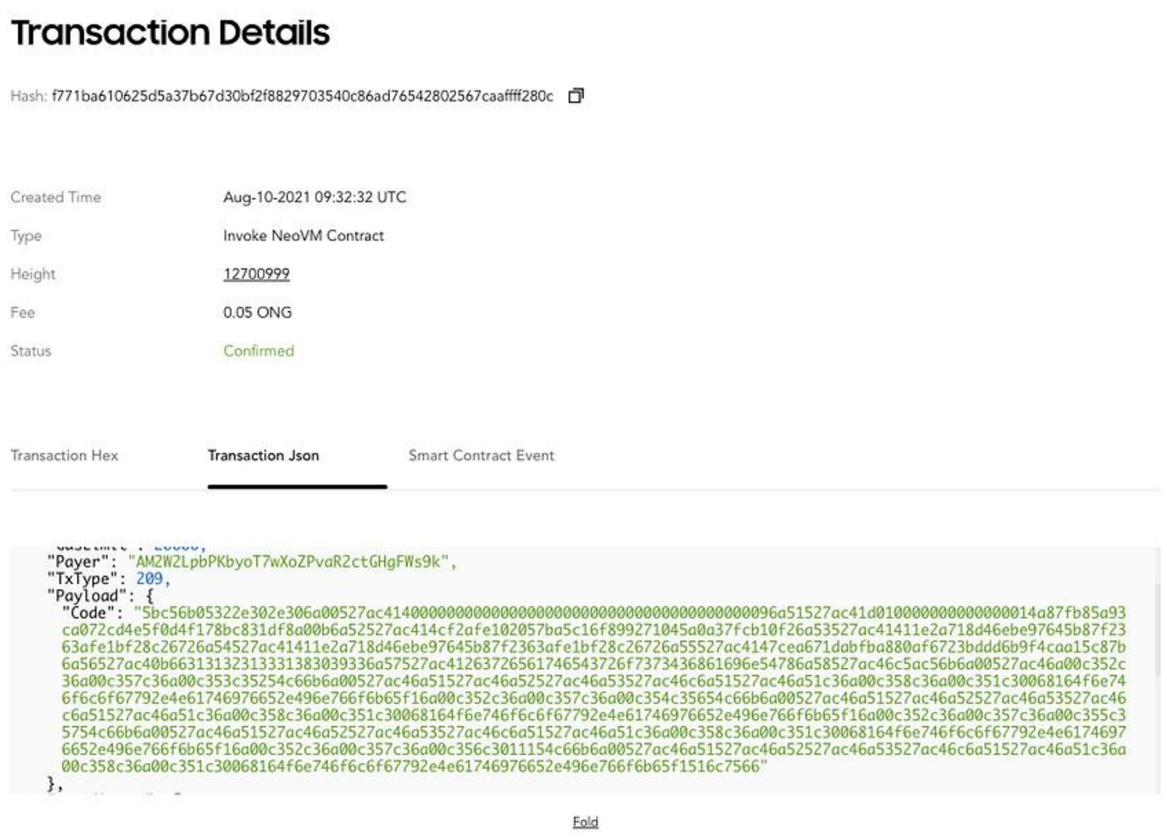
1. The hacker/s initialized a malicious transaction — "method of changing the keepers", on the Side Chain.
2. The malicious transaction was generated on the Side Chain **without sufficient verification.** The Relayer picked it up, included it in the Alliance Chain's Merkle tree, and **signed** it. This may have been the reason why the hacker/s had to **risk buying the Side Chain token** through a CEX to generate the malicious transaction on the Side Chain.
3. The hacker invoked the ECCM contract at the Target Chain with the **valid Merkle proof** from step 2, enabling the function to be executed and the keepers to be changed.
4. After the keeper privilege was gained, the hacker then performed a series of transactions to **unlock** assets from their designated addresses.

Note that some other Poly Network relayer chains **did not** sync the malicious transaction, so the assets are **untouched** even if a similar contract source code is used on the Target Chains.



Detailed Analysis

1. The hacker/s initiated a transaction on the Side Chain at Aug-10–2021 09:32:32 UTC: https://explorer.ont.io/tx/F771BA610625D5A37B67D30BF2F8829703540C86AD76542802567CAAFFFF280C#



Transaction Details

Hash: f771ba610625d5a37b67d30bf2f8829703540c86ad76542802567caaffff280c

Created Time	Aug-10-2021 09:32:32 UTC
Type	Invoke NeoVM Contract
Height	12700999
Fee	0.05 ONG
Status	Confirmed

```
  "Payer": "AM2W2LpbPKbyoT7wXoZPvaR2ctGHgFWs9k",
  "TxType": 209,
  "Payload": {
    "Code": "5bc56b05322e302e306a00527ac414000000000000000000000000000000000000000000096a51527ac41d010000000000000014a87fb85a93
    ca072cd4e5f0d4f178bc831df8a00b6a52527ac414cf2afe102057ba5c16f899271045a0a37fcb10f26a53527ac41411e2a718d46ebe97645b87f23
    63afe1bf28c26726a54527ac41411e2a718d46ebe97645b87f2363afe1bf28c26726a55527ac4147cea671dabfba880af6723bddd6b9f4caa15c87b
    6a56527ac40b66313132313331383039336a57527ac41263726561746543726f7373436861696e54786a58527ac46c5ac56b6a00527ac46a00c352c
    36a00c357c36a00c353c35254c66b6a00527ac46a51527ac46a52527ac46a53527ac46c6a51527ac46a51c36a00c358c36a00c351c30068164f6e74
    6f6c6f67792e4e61746976652e496e766f6b65f16a00c352c36a00c357c36a00c354c35654c66b6a00527ac46a51527ac46a52527ac46a53527ac46
    c6a51527ac46a51c36a00c358c36a00c351c30068164f6e746f6c6f67792e4e61746976652e496e766f6b65f16a00c352c36a00c357c36a00c355c3
    5754c66b6a00527ac46a51527ac46a52527ac46a53527ac46c6a51527ac46a51c36a00c358c36a00c351c30068164f6e746f6c6f67792e4e6174697
    6652e496e766f6b65f16a00c352c36a00c357c36a00c356c3011154c66b6a00527ac46a51527ac46a52527ac46a53527ac46c6a51527ac46a51c36a
    00c358c36a00c351c30068164f6e746f6c6f67792e4e61746976652e496e766f6b65f1516c7566"
  },
```

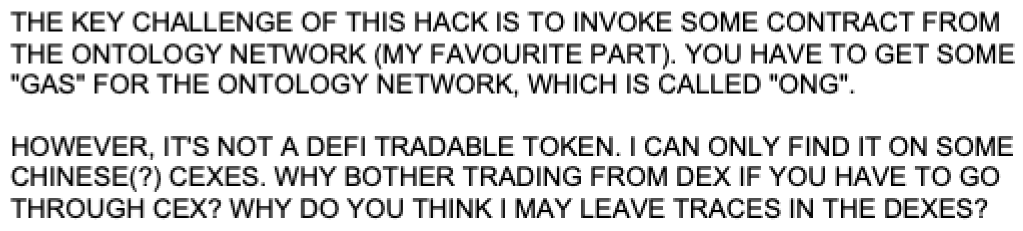
In the transaction json, we decoded the code section and obtained the parameter mapping below:

```
{
"args": "010000000000000014a87fb85a93ca072cd4e5f0d4f178bc831df8a00b"
"eth-eccd": "cf2afe102057ba5c16f899271045a0a37fcb10f2"
"bsc-eccd": "11e2a718d46ebe97645b87f2363afe1bf28c2672"
"heco-eccd": "11e2a718d46ebe97645b87f2363afe1bf28c2672"
"poly-eccd": "7cea671dabfba880af6723bddd6b9f4caa15c87b"
"method": "6631313231333138303933"
}
```

2. This transaction invoked a method "6631313231333138303933" whose signature is equal to 0x41973cd9 (the signature of putCurEpochConPubKeyBytes(bytes)). **This transaction should not be a valid transaction**, but was **generated on the Side Chain and picked up** by the Relayer and mined into the Alliance Chain block. This malicious transaction was included in **a Merkle tree** and **signed**. The Merkle tree was used to prove the existence of the transaction.

Cross-chain transaction:
https://explorer.poly.network/tx/1a72a0cf65e4c08bb8aab2c20da0085d7aee3dc69369651e2e08eb798497cc80

3. This attack used the loophole in the transaction validity on the Side Chain and the Relayer, which may have been the reason why the hacker/s had to **risk buying the Side Chain token** through a CEX to generate the malicious transaction on the Side Chain. See below the screenshot of the hacker's statement:


THE KEY CHALLENGE OF THIS HACK IS TO INVOKE SOME CONTRACT FROM THE ONTOLOGY NETWORK (MY FAVOURITE PART). YOU HAVE TO GET SOME "GAS" FOR THE ONTOLOGY NETWORK, WHICH IS CALLED "ONG".

HOWEVER, IT'S NOT A DEFI TRADABLE TOKEN. I CAN ONLY FIND IT ON SOME CHINESE(?) CEXES. WHY BOTHER TRADING FROM DEX IF YOU HAVE TO GO THROUGH CEX? WHY DO YOU THINK I MAY LEAVE TRACES IN THE DEXES?

4. On the Target Chain, EthCrossChainManager.verifyHeaderAndExecuteTx() was triggered. The first parameter proof includes:
a. fromChainId: 3 (Side Chain)
b. toChainId: 2 (Target Chain)
c. toContract: 0xcf2afe102057ba5c16f899271045a0a37fcb10f2
d. method: 0x6631313231333138303933
e. args: 010000000000000014a87fb85a93ca072cd4e5f0d4f178bc831df8a00b

5. The function checked the proof and recognized the data as **valid because it is indeed a part of the Merkle tree**. Later, it triggered EthCrossChainManager._executeCrossChainTx() to execute a transaction with toContract (0xcf2afe102057ba5c16f899271045a0a37fcb10f2), method (0x6631313231333138303933) and args (010000000000000014a87fb85a93ca072cd4e5f0d4f178bc831df8a00b). The method has the same function signature (0x41973cd9) as putCurEpochConPubKeyBytes(bytes), so EthCrossChainData.putCurEpochConPubKeyBytes() was triggered and executed, and lastly, the **consensus bookkeeper public key was successfully changed**.

6. After the hacker/s changed the public key, they were able to unlock the assets as they pleased.

Transaction on Ethereum:
https://etherscan.io/tx/0xb1f70464bd95b774c6ce60fc706eb5f9e35cb5f06e6cfe7c17dcda46ffd59581

Root Cause Summary

1. The malicious transaction was generated without strict verification on the Side Chain.

2. The Relayer picked up and processed the malicious transaction since it contains the "makeFromOntProof" event.

3. The Relayer published the transaction obtained from step 1 on the Alliance Chain.

4. A valid Merkle tree proof that includes the malicious transaction was generated from step 3 on the Alliance Chain. The Merkle tree definition is as follows (https://en.wikipedia.org/wiki/Merkle_tree) :

Merkle tree wiki: Hash trees can be used to verify any kind of data stored, handled and transferred in and between computers. They can help ensure that data blocks received from other peers in a peer-to-peer network are received undamaged and unaltered, and even to check that the other peers do not lie and send fake blocks.

5. The ECCM contract validated that the transaction does exist on the Side Chain using the Merkle proof. Note that the full validation of the transaction should have been done before constructing the Merkle proof that will be sent to the Target Chain. As shown in the design doc:

The management contract fetches the block headers from **chain A**, **verifies** whether or not the cross chain parameters and the proof are valid, and **then** transmits the necessary information to **chain B** in the form of an event;

Thus, the Target Chain (i.e., chain B) should validate that the received information is undamaged and unaltered using the Merkle proof, while the transaction information should be verified before it is sent to the Target Chain.