There may be bad data points in CommitStateDB when exec evm transaction with multi msgs #667

Open summerpro opened this issue on Dec 21, 2020 · 4 comments

Labels Status: On Ice Type: User Reported

summerpro commented on Dec 21, 2020 • edited Contributor

System info: [Include Ethermint commit, operating system name, and other relevant details]

- branch development

Additional info: [Include gist of relevant config, logs, etc.]

- The "storage" data from "CommitStateDB" are written to the "Store"(keeper) at the time of the "deliverTx"(handler) and are cleared at the "endBlock" stage.
- The "code" data from "CommitStateDB" are written to the "Store"(keeper) and cleared at the "endBlock" stage.
- If there are 2 msgs in a tx, in the deliverTx (handler) stage, where msg1 executes successfully and msg2 fails, then all store data （Keeper） will be rolled back. However, the storage data in commitStateDB and the code are still Reserved, if another tx2 is executed later in the deliverTx phase, the bad data points from tx1 will be used
- The problem encountered is Similar to issue668， issue669

solution:

1. Multiple msgs are not allowed in one tx
2. Update and clear all cache data of commitStateDB in the handler phase

👍 5

summerpro changed the title ~~There may be dirty data when exec evm transaction with multi msgs~~ There may be bad data points when exec evm transaction with multi msgs on Dec 21, 2020

summerpro changed the title ~~There may be bad data points when exec evm transaction with multi msgs~~ There may be bad data points in CommitStateDB when exec evm transaction with multi msgs on Dec 22, 2020

This was referenced on Dec 22, 2020

Bad data may be generated due to insufficient gas during the execution of evm transactions #668
Closed

Bad data may be generated due to error returned during the execution of evm transaction #669
Closed

Roll back CommitStateDB after failing to execute handler in evm module #677
Merged

summerpro mentioned this issue on Jan 8, 2021

fix bad-data-bug with multi msgs in module evm #694
Closed
11 tasks

zhongqiuwood mentioned this issue on Jan 12, 2021

Contributions from OKEX organization #707
Closed

summerpro commented on Jan 13, 2021 Contributor Author

Actually there're 2 issues. Since they were fixed, we now release detailed info as follows:

Storage Caching Lifecycle Inconsistency detailed vuln report:

Affected Version:
currently all versions (e.g. <= v0.4.0)

Vulnerability Overview:
Due to the inconsistency between the Storage caching cycle and the Tx processing cycle, Storage changes caused by a failed transaction are improperly reserved in memory. Although the bad storage cache data will be discarded at EndBlock, it is still valid in the current block, which enables many possible attacks like "arbitrary mint token" etc.

Details and Root Cause:

First, we found some ground truth:

1. The lifecycle of storage cache is consistent with block rather than tx
   a. when a contract is called the first time in the current block, its storage is read from KVStore to memory(CommitStateDB instance)
   b. storage cache is cleaned up at EndBlock stage

a. each tx's dirty data is cached in msCache. Only 'success' tx could write msCache to deliverState (which is further written to CommitMultiStore during Commit stage), 'failed' tx's msCache will be discarded.
b. 'failed' tx could modify storage cache if only it didn't get reverted in EVM. This can be done in multiple ways.

Clearly, 3a and 3b are not consistent with each other, thus, this is the root cause of the vulnerability.

Steps to Exploit:

The idea is triggering the vuln according to the idea of 3b, then expand the modification(e.g gain benefit) by accessing the compromised storage cache and stabilizing it in a 'success' tx, finally the previous modification was 'discard' when the block ends and the storage falls into an unreasonable state.

e.g.:

1. Suppose that there is an ERC20 contract WETH, the attacker controls 3 accounts A, B, C with 10WETH, 0WETH, 0WETH at the beginning.
2. Construct Tx1 containing 2 msg, msg1: A.transfer(B,10WETH), msg2: a 'failed' msg(e.g. like A transfers a lot of Photon to B, but the balance is insufficient)
3. Construct Tx2 contains 1 msg: B.transfer(C, 10WETH)
4. Broadcast Tx1, Tx2, make sure they are included in the same block in order.

Let's see what's happening here:

0. Init: A:10, B:0, C:0
1. Tx1.msg1 success, Tx1.msg2 failed, thus the changes made on storage in Tx1.msg1 is not written back to deliverState, so no state change occurs in the deliverState. But the Storage cache in memory is modified and retained.
   a. Storage cache: A:0, B: 10, C:0
   b. deliverState: A:10, B: 0, C:0
2. Tx2 success.
   a. Storage cache: A:0, B: 0, C:10
   b. deliverState: A:10, B: 0, C:10
3. Storage cache is discarded at EndBlock, deliverState is written to persistent storage, which achieves the malicious minting of ERC20.

Credit to: OKLink & Chaitin Tech

summerpro commented on Jan 13, 2021 Contributor Author

Contract Bytecode Wrongfully Stored in Failed Tx detailed vuln report:

Affected Version:
currently all versions (e.g. <= v0.4.0)

Vulnerability Overview:
When deploying contracts, the bytecode set in a FAILED transaction wrongfully remains in memory(stateObject.code)and is further written to persistent store at the Endblock stage, this issue may be utilized to build honeypot contracts.

Details and Root Cause:
First, we found some ground truths:

1. The bytecode set by a failed transaction is wrongfully written to persistent store
   a. The contract bytecode is written to stateObject.code(memory) if only the execution of evm.Create() succeed during deliverTx, regardless of success or failed of the tx.
   b. All the bytecode in memory will be written to persistent store at Endblock stage rather than in deliverTx.
   c. Even if the transaction fails, the bytecode from evm.Create() won't be discarded and will remain in memory.
2. The storage changes by failed transaction is not stored persistently due to the multistore rollback mechanism of Cosmos SDK.

Thus, deploying a contract in a failed tx will end up with bytecode being written to persistent store but the Storage changes will not. In other words, adversary can manage to omit the execution of constructor.

Steps to Exploit:
1.Construct a Tx containing 2 msg:

- msg1: we create a new contract
- msg2: a 'failed' msg(e.g. like A transfers a lot of Photon to B, but the balance is insufficient)

2.Broadcast it.

Although the execution of this whole Tx fails, the bytecode of the contract is still written to the KVStore at the Endblock stage, and the Storage was not stored persistently.

Credit to: OKLink & Chaitin Tech

fedekunze mentioned this issue on Jan 15, 2021

LTS and Contributor Guidelines #719
Closed

github-actions bot commented on Feb 27, 2021

This issue is stale because it has been open 45 days with no activity. Remove `stale` label or comment or this will be closed in 7 days.

github-actions bot added the stale label on Feb 27, 2021

fedekunze added Status: On Ice and removed stale labels on Feb 28, 2021

LernaJ added the Type: User Reported label on Apr 21, 2021

OS-WS commented on Apr 26, 2021

https://cve.mitre.org/cgi-bin/cvename.cgi?name=2021-25837

Is there any plan to fix these issues?

Assignees

No one assigned

Labels

Status: On Ice   Type: User Reported

Projects

None yet

Milestone

No milestone

Development

Successfully merging a pull request may close this issue.

fix bad-data-bug with multi msgs in module evm
summerpro/ethermint

4 participants