Everscale Bug Report - Stack Serialization

Author: Evgeniy Shishkin <evgeniy.shishkin@gmail.com>

Description

In Everscale it is possible to create big tuples with relatively little gas consumption. In our previous bug report, we showed that printing such tuples may result in a great validator performance degradation. This time, we found that serialization (i.e. transforming an in-memory data structure to a series of bytes) of such objects may cause a stack overflow of the validator's node and, hence, complete network stop.

This problem is not related to big tuples per se, but incorrect gas consumption calculation logic in the serialization code. This erroneous logic prevents TVM from stopping execution when gas goes out of limit. What is worse, there is a recursive call happening during the serialization. This fact leaves open the possibility for a stack overflow.

To reproduce this bug, you have to:

1. Create a deep tuple (say, 15000 enclosed elements)
2. Create a random continuation
3. Put the tuple into the stack of the newly created continuation (SETCONTARGS)
4. Serialize the continuation (STCONT)

Now we go a bit deeper into the most interesting parts.

Tuple serialization

It is surprisingly difficult to find a place where a Tuple object located on the stack would be serialized. However, we managed to find at least one such place. It is done in a context of more broad Everscale-specific instruction STCONT, i.e. serialization of a continuation. To trigger the tuple serialization, you need to put it into the continuation stack. This can be done using the SETCONTARGS instruction.

Continuation serialization

Everscale VM has some undocumented instructions. It doesn't present both in the Durov's whitepaper and in the C++ node code. One such instruction is STCONT, i.e. serialization of a continuation located on a stack. To the best of our knowledge, without this instruction this problem would not manifest itself.

Incomplete gas consumption computation logic

Consider the code fragment located here. In this block, the continuation's stack get serialized. Authors obviously acknowledged the fact that serialization has to be properly charged for the gas. However, the implementation lacks a gas overflow check.
The tuple serialization logic contains the same error. What is worse, it contains recursive call here. For a big tuple, it leads to a stack overflow.

Affected components

The main origin of the problem is in the incomplete gas consumption computation logic, in all serialization handers.

https://github.com/tonlabs/ton-labs-vm/blob/50170e72c307bbf1e9e2071fa83bcb0be4a03bb6/src/stack/mod.rs#L358

and

https://github.com/tonlabs/ton-labs-vm/blob/50170e72c307bbf1e9e2071fa83bcb0be4a03bb6/src/stack/continuation.rs#L142

Result of Attack

The result of the attack is a full network stop in the worst case. It is only a matter of deploying the contract to the master chain.

Mitigation Strategies

The most easy fix would be to introduce gas overflow checks in the serialization handlers.

DREAD Score

Criteria	Rational
Damage potential: 10	Full network stop will limit the ability of users to manage their funds. Validators cut off the election cycle lose their earning potential and rewards as well.
Reproducibility: 10	Easy to reproduce on any - main or development - network. Does not depend on capabilities.
Exploitability: 10	The bug can be exploited by a simple transaction with proper parameters

Affected users: 10	All users of the network are affected.
Discoverability: 5	The in depth analysis of the VM code has to be carried out in order to find this issue. Not that easy.

Total: (10 + 10 + 10 + 10 + 5)/5 = 9 **Critical**

Steps to reproduce

The Fift script with the code triggering the problematic code is put here:
https://gist.github.com/unboxedtype/b90ac2ce4ce79e9f4245d0f610a1e3a0

You may also need this library:
https://gist.github.com/unboxedtype/3546a5a5e3562fb2c344cc082a2c201b