Selective Instrumentation mechanism and its application in a Virtual Machine

I.A. Vasilev
P.M. Dovgalyuk
V.A. Makarov
M.A. Klimushenkova

The relevance of selective instrumentation

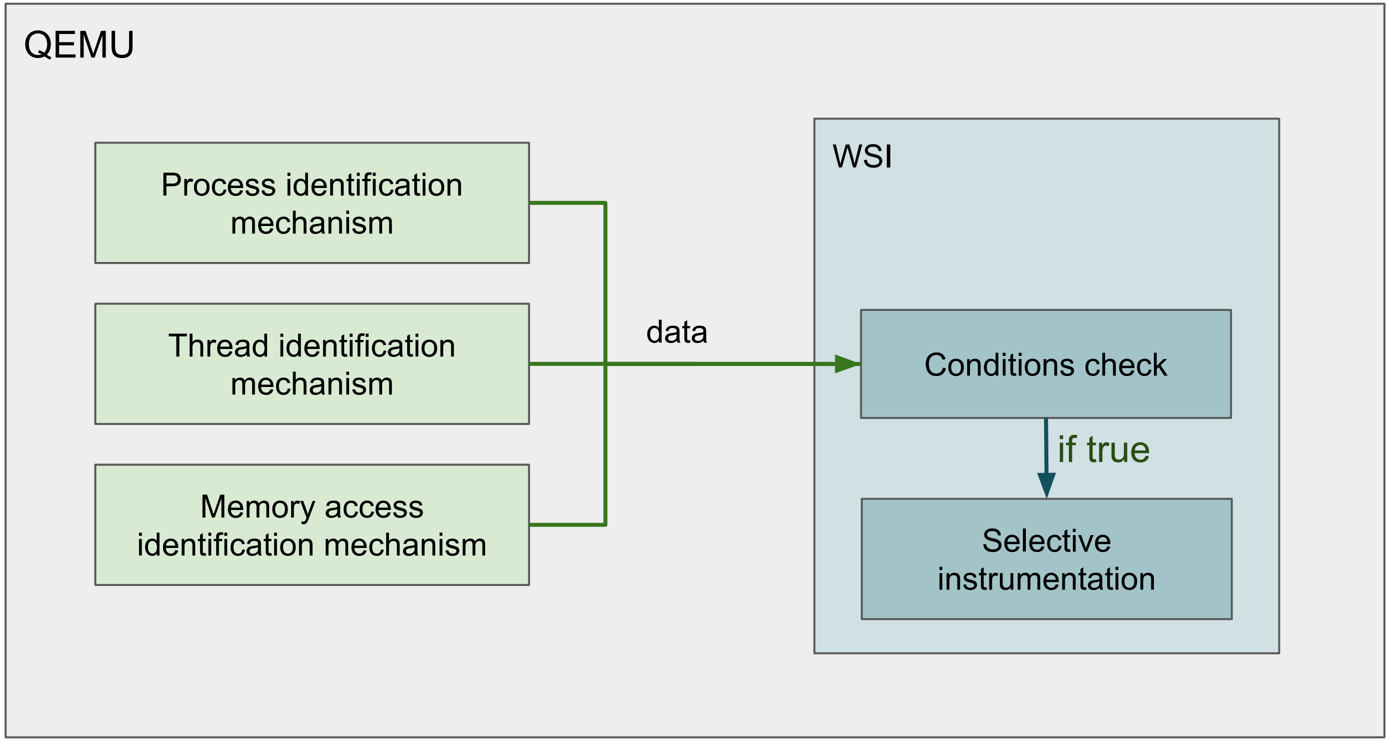
Whole-system Instrumentation (WSI):

- + allows OS analysis and interprocess interaction analysis
- + analysis is performed isolated from program under the study
- - often provides redundant information, which complicates and slows down the analysis process

Selective Instrumentation:

- + approach, that narrows WSI mechanism and allows to instrument exactly the necessary data and at the necessary moment

How selective instrumentation works


QEMU
Process identification mechanism
Thread identification mechanism
Memory access identification mechanism
data
WSI
Conditions check
if true
Selective instrumentation

Existing approaches

Application level instrumentation has an easy access to system related information through the use of system API.

For whole system instrumentation:

- depends on agent-app inside of the system (Virtuoso);
- uses kernel structure data (DECAF)
- uses heavily os-dependant methods (PEMU)

Objective

To develop method(s) for process, thread and memory access identification, with an ability to use gotten information for selective instrumentation and analytical routines in general.

Developed method should **not** rely on kernel structures and should be relatively easy for adaptation for new systems.

Process identification

OS allocates a PGD for every new process, because each process needs its own address space.

Thus, each process must have a unique PGD value, which can be used to identify it.

Watching for changes in this register we can detect switches between processes.

Implementation

Watching for changes in the corresponding register we are able to keep the table of currently active processes. The value of this register should be used as an id to determine an object for instrumentation process. Before actually performing instrumentation current process is compared to ones, that are set as processes of interest by an analyst, and by this comparison mechanism decides whether or not to perform instrumentation.

The use of termination system calls allows to maintain an up-to-date process table.

Thread identification

As one PGD value can (and most likely will) correspond to a couple of threads, this value alone can not be used for identification.

But we also know, that every threads needs its own stack for variables and return addresses storage.

To identify a thread one can use a pair of ***PGD value*** and ***SP values range.***

Implementation

At any operation of instruction translation we are looking for ones, that are explicitly or implicitly changing SP value.

Those changes are divided into two groups: one that contains operations, leading to creation of a new range, and other that corresponds to the extension of already existing ranges.

Create: pop esp; mov esp; return from kernel mode SP.

Extend: every other situation, when SP is somehow changed.

Improvement of implementation

If two ranges eventually begin to intersect, they need to be joined as one.

To determine thread termination, it is needed to detect a corresponding system call.

The resulting approach also allows one to track fibers.

Memory instrumentation

We added callbacks for memory related operations, both for store in and load from memory.

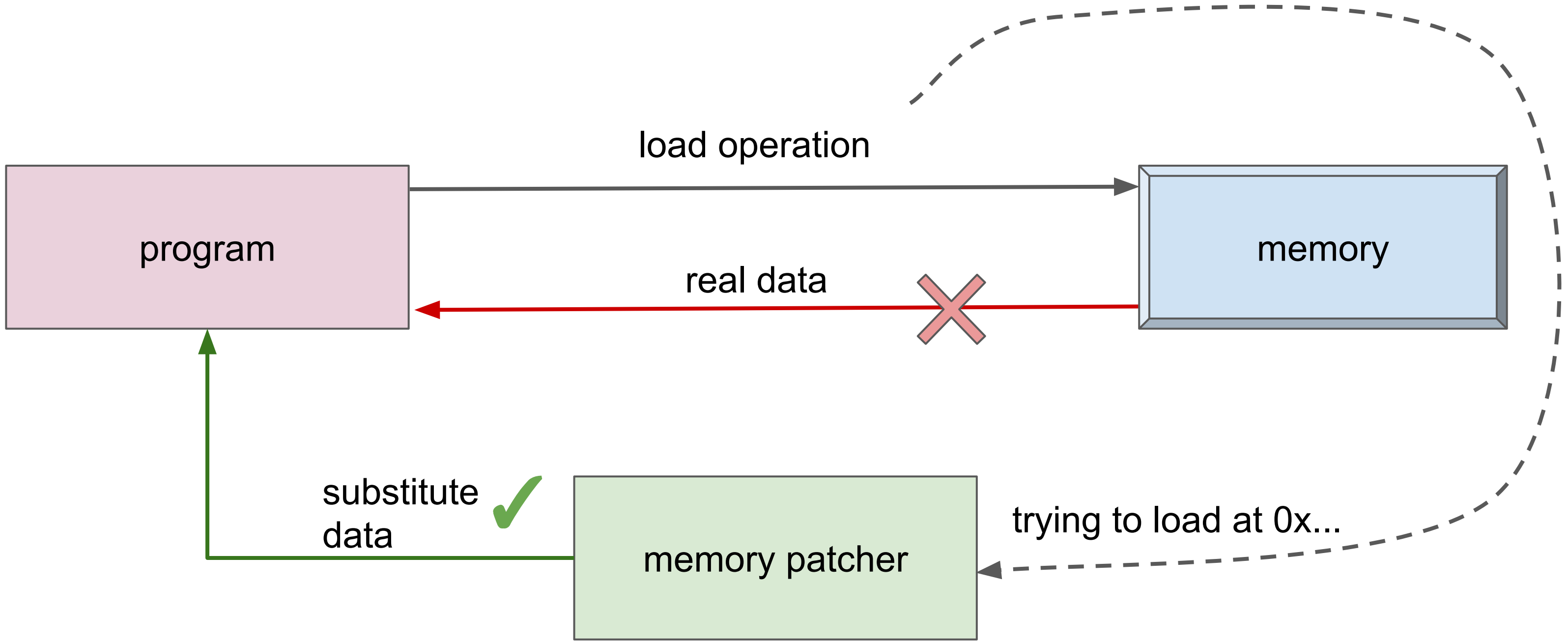
Corresponding callbacks were added both for translation and execution stages.

Thus, at any occurance of memory operations one can get corresponding address, operation type, gotten or stored data, and then use gotten information to perform analytical tasks.

Use cases:

- Debugging of specific processes, threads and even fibers
- Memory patching to change loading values
- Making a call stack to analyze program workflow
- Narrow approach to use of the instrumentation, which allows the achievement of much more precise results

Memory patcher plugin



Call stack plugin

Uses process identification and thread identification information.

1. Plugin checks executed instructions looking for a call to a new function, and for each detected call it saves current context, thread and address.
2. Plugin looks for return functions and associate them with closest call instructions.

In result, analyst can get a view of a current call tree for a specific context.

It also allows detection of workflow interceptions. To detect these plugin checks for each return instruction if the new address is the same, as where the call was. If they differ, it may be an indication of a malicious behavior.

Evaluation

We implemented approach for process identification for ARM and x86

Approach for thread identification is implemented for x86 architecture and was successfully tested on Windows-family operating systems and Linux operating systems. ARM implementation is yet at a stage of development.

Memory instrumentation relies completely on platform and architecture-agnostic capabilities of qemu, so it was successfully tested for x86, ARM and MIPS architectures.

Future work

- Thread identification for ARM
- Development of other instrumentation plugins