OS-agnostic identification of Processes and Threads in virtual machine

I.A. Vasiliev
P.M. Dovgalyuk
M.A. Klimushenkova

Where the task came from:

Cons of whole-system instrumentation:

- causes a slowdown of system under study
- produces redundant data during analysis process

Pros of selective instrumentation:

- analysis of specific processes and threads
- overhead minimization

Existing approaches

Virtuoso: works inside of the system under study; does not support ASLR

DECAF: makes use of the kernel structures; strictly dependent on the OS

PEMU/Exterior: uses non-reliable method for threads identification (masking lower 12 bits of the KSP)

Objective

Find a way to uniquely identify user **threads** and **processes** in the system under study within a virtual machine, without relying on the kernel data structures

Identification of processes

Each process has corresponding PGD value

It is sufficient to monitor the corresponding register (cr3)

Implementation of process identification

By monitoring the values of cr3 register all the time, we have the information about the current set of processes operating in the system and can freely begin and complete instrumentation and analysis of any number of processes at any time.

Identification of threads

Observation:

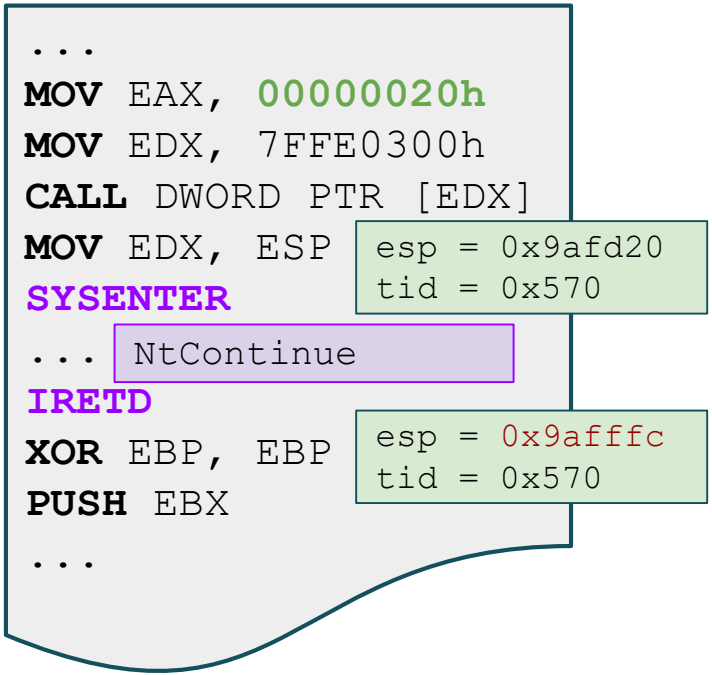
- Since the same PGD can correspond to different threads, this value by itself is not enough for identification of threads
- Each thread got to have its own stack for storing local variables and return addresses. However, the register, containing the SP value changes constantly during the the usual work process

Hypothesis: to identify the thread, you can use a pair of “register, containing the value of PGD” and “a range of SP values”

Changes of esp value in kernel mode

Hypothesis: thread switches occur in kernel space, which means that by monitoring the value of esp register before and after switching to kernel space, we can determine whether or not the thread was switched

Reality: practice test has shown, that this assumption is not always true, and often the esp value change does not correspond to thread switching



Defining ranges

Creation of the new range:

- **pop esp** or **mov esp** with a esp value not falling into any of the existing ranges
- switching from the kernel space to user space

Expansion of the existing range:

- other instructions, explicitly or implicitly changing the esp value, including **call** and **ret** group instructions

Additional conditions:

Problem	Solution
multiple ranges ⇳ single thread	Combining of intersecting ranges

Identification of fibers

Using this approach allowed identifying fibers during system operation, which would also be an impossible task using only information about switching from the kernel space or using information from kernel structures

Use cases:

- Debugging of specific processes, threads and even fibers
- Making a call stack to analyze program workflow
- Narrow approach to use of instrumentation, to achieve more precise results

Using system calls information

For processes: to detect the termination of the process, you can monitor the corresponding system call (NtTerminateProcess)

For threads: a problem, with multiple threads being corresponding to a single range, can be solved by monitoring a specific system call (NtTerminateThread)

The use of system calls, contrary to the use of kernel structures, allows to simplify the porting of the algorithm for previously implemented OS-families. Plus, the information about system calls is usually well documented, as opposed to information about kernel structures.

Thread Identification results

for **Windows XP**: ~95% accuracy of range to thread relation. An exception: rare situations when one thread have multiple corresponding ranges

for **Debian**: the first tests showed that the algorithm is functional, but it’s too early to talk about its accuracy, because no reliable source for thread comparison was selected yet

Hypothesis conclusions

The result obtained in the test environment can serve as confirmation of the hypothesis about the possibility of identifying threads, having minimal knowledge of the system and relying on the ranges of SP values.

Achieving OS-independence comes down to **minimizing** the use of system information.

At the current state, the implementation of the algorithm is not completely reliable and needs further improvement before it can be used in solving real problems.

Future work

- Finding ways to improve the accuracy of the implementation

If we succeed:

- Adaptation of this approach for Linux
- Adaptation of this approach for other process architectures