Exit-Less Isolated Execution

Yushi Omote
Japan Society for the Promotion of Science
omote@osss.cs.tsukuba.ac.jp

Takahiro Shinagawa
The University of Tokyo
shina@ecc.u-tokyo.ac.jp

Kazuhiko Kato
University of Tsukuba
kato@cs.tsukuba.ac.jp

Commodity operating systems today rely on a large code base that is hard to be verified, have broad attack surfaces and easily get compromised due to mis-configuration or security bugs. However, the OSs have highest privilege and unrestrained access to the entire system, and applications have no way of preventing their security-sensitive information (e.g. passwords, encryption keys...) from being revealed or tampered by compromised OSs. Hence mechanisms for securely isolating applications from commodity OSs are highly required [1–3].

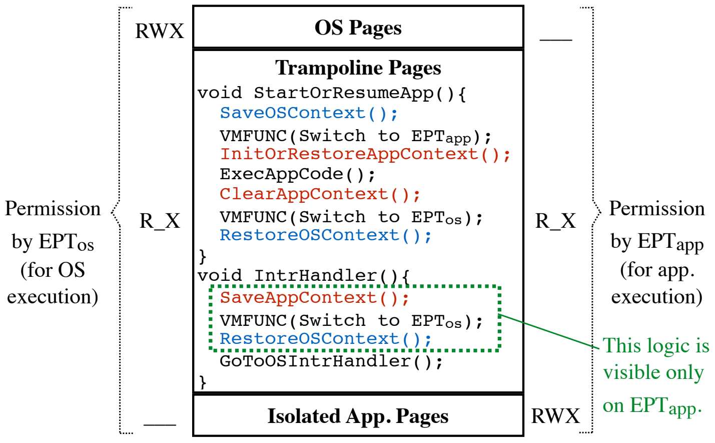
Previous research [2, 3] proposes easy-to-verify thin hypervisors that isolate applications from OSs. The hypervisors force OS and application domains to be mutually inaccessible by use of nested paging. When execution control is passed between them (due to jumps, interrupts or system calls), the hypervisors trap the events and securely perform context switching between them, properly switching permission of nested pages.

However, the context switching by the hypervisors causes costly CPU-mode changes to the hypervisor mode (*VM exits*), which consume many CPU cycles and heavily pollute cache. Unfortunately, VM exits due to the context switching can be continual because it happens whenever applications are scheduled or invoke system calls, which are more frequent on I/O-intensive workloads. Frequent VM exits result in high overhead on the system performance.

To avoid the overhead, we propose ELIE (**E**xit-**L**ess **I**solated **E**xecution): a hypervisor-based scheme for isolating applications from OSs without VM exits. ELIE completely eliminates VM exits from the normal execution path of isolated applications while supporting context switches for scheduling, interrupts and system calls in a compatible manner for commodity OSs by leveraging a new Intel CPU feature: the *VMFUNC* instruction.

The VMFUNC instruction enables any guest processes to switch nested page tables (*EPTs*) without a VM exit. Once EPTs are pre-configured by a hypervisor, guest processes can freely choose one of the EPTs by simply executing the VMFUNC instruction with the ID of an EPT to use as an argument. ELIE hypervisor pre-configures the two different EPTs (EPT_{app} for application execution and EPT_{os} for OS execution) that share trampoline pages whose permission is read-only executable on both EPTs (see Figure 1). The hypervisor puts the code in the trampoline pages to force secure context switching between OSs and applications using VMFUNC instructions. The basic approach of ELIE is that, whenever the context switching is required, the hypervisor cleverly leads OSs and applications to enter this trampoline logic without VM exits.

The first challenge is how to handle interrupts to applications without VM exits. When the application on EPT_{app} is interrupted, the CPU forces a jump to an OS page (an interrupt handler) but causes a VM exit because the OS page permission is unreadable on EPT_{app}. The hypervisor avoids the VM exit using *IDT shadowing*; the hypervisor forces all interrupts to be handled by its shadow IDT instead of the guest IDT, and the shadow IDT points to the trampoline interrupt handler (`IntrHandler()` in the figure), which executes the VMFUNC instruction to switch to EPT_{os} before jumping to the guest interrupt handler. To properly resume the application after finishing the guest interrupt handler, the address of the trampoline code (`StartOrResumeApp()` in the figure) is pushed to the OS stack as the return address. On the other hand, when the OS on EPT_{os} is interrupted, the trampoline interrupt handler just jumps to the guest interrupt handler.


RWX
OS Pages
Trampoline Pages
Permission by EPT_{os} (for OS execution)
R_X
R_X
Permission by EPT_{app} (for app. execution)
This logic is visible only on EPT_{app}.
Isolated App. Pages
RWX

```
void StartOrResumeApp(){
  SaveOSContext();
  VMFUNC(Switch to EPTapp);
  InitOrRestoreAppContext();
  ExecAppCode();
  ClearAppContext();
  VMFUNC(Switch to EPTos);
  RestoreOSContext();
}
void IntrHandler(){
  SaveAppContext();
  VMFUNC(Switch to EPTos);
  RestoreOSContext();
  GoToOSIntrHandler();
}
```

Figure 1. The summary view of the trampoline logic of ELIE hypervisor.

The second challenge is how to keep the integrity of the guest page table (*GPT*). ELIE hypervisor does not control the GPT to avoid VM exits (e.g. due to CR3 hooks). Hence although the application asks the OS not to remap application pages on the GPT (e.g. `mlock()`), the malicious OS can intentionally reorder the pages by modifying the GPT and break the integrity of application code and data, which causes malfunction of the application. The hypervisor prevents such attack by configuring EPT_{app} so that the copy of the correct GPT always appears at the guest physical address pointed to by the CR3 value of the application process.

Evaluation results of ELIE prototype show a round-trip context switching takes only 325 cycles while 2,321 cycles with conventional exit-full switching on average.

References

[1] X. Chen et al. Overshadow: A Virtualization-based Approach to Retrofitting Protection in Commodity Operating Systems. In *Proc. of ASPLOS'08*, 2008.

[2] Y. Li et al. MiniBox: A Two-way Sandbox for x86 Native Code. In *Proc. of ATC'14*, 2014.

[3] J. M. McCune et al. TrustVisor: Efficient TCB Reduction and Attestation. In *Proc. of SP'10*, 2010.