Protecting Kernels from Untrusted Modules using Dynamic Binary Instrumentation

Akshay Kumar
ak.kumar@mail.utoronto.ca
Peter Goodman
pag@cs.toronto.edu
Angela Demke Brown
demke@cs.toronto.edu
Ashvin Goel
ashvin@eeceg.utoronto.edu
University of Toronto

Kernel modules cannot be trusted
Two thirds of all kernel vulnerabilities reside in kernel modules [CVE 2010].

Kernel modules can be:
- Malicious
- Buggy
- Exploited

Modules can compromise:
- Control-flow integrity
- Data integrity
- Both (e.g. stack integrity)

Existing kernel protection methods
Existing solutions cannot secure against all native kernel modules. They either:
- Secure only virtualized modules (HUKO, Gateway, etc.)
  - Many native modules cannot be run under virtualization
- Secure only modules whose source code is available (BGI, LXFI, etc.)
  - Many modules are provided as pre-compiled binaries by third-party vendors

Goals and Approach
Goals:
- Secure all kernel modules
- Secure pre-compiled binary modules
- No overhead when running in the kernel

Approach:
- Secure modules by modifying their binary code at runtime using DynamoRIO Kernel (DRK)
- Instrument only while the module code is running

Kernel modules will be secured in three steps:
1. Isolate modules in separate protection domains
2. Mediate all control transfers between the kernel and its modules
3. Verify all memory accesses by modules

Challenges
Securing kernel modules is challenging:
- Interrupt handling
- Complex kernel interface
- Sensitive kernel data exposed through shared data, macros, etc.
  - BGI and LXFI lead the way
- Difficult to maintain integrity of kernel stack
  - Call/return consistency is manageable
  - Data consistency is challenging

Architecture

Dynamic Binary Instrumentation

Original Code
```
mutable_func:
pushq  %rbp
...  
cmpl  $0, %eax
jle   LBB1_2
callq  %rax
jmp   LBB1_3
LBB1_2:
movl   $0, -16(%rbp)
LBB1_3:
...  
popq   %rbp
ret
```

Instrumented Code
```
mutable_func:
pushq  %rbp
...  
cmpl  $0, %eax
jle   LBB1_2
check_call_trax
runtime_call_trax
LBB1_2:
check_write %rbp, -16
movl   $0, -16(%rbp)
jmp   LBB1_3
LBB1_3:
...  
popq   %rbp
ret
```

Summary
- Protect the kernel from malicious or misbehaving modules
- Use DynamoRIO Kernel to secure pre-compiled binary modules
- Run non-module kernel code natively without overhead