SAND2013-5265C

Exceptional service in the national interest

Sandia National Laboratories

KPeeler
Hypervisor-Based Malware Unpacker

Evan G. Tobac

Ken Chiang

Kris Watts

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

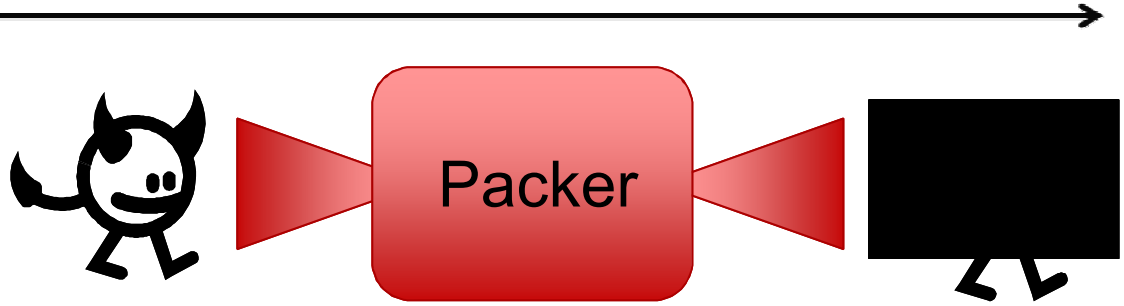
Outline

- Problem
- Kpeeler Architecture
- KVM-Stalker Internals
- Unpacking Heuristics
- Results
- Closing thoughts

Problem

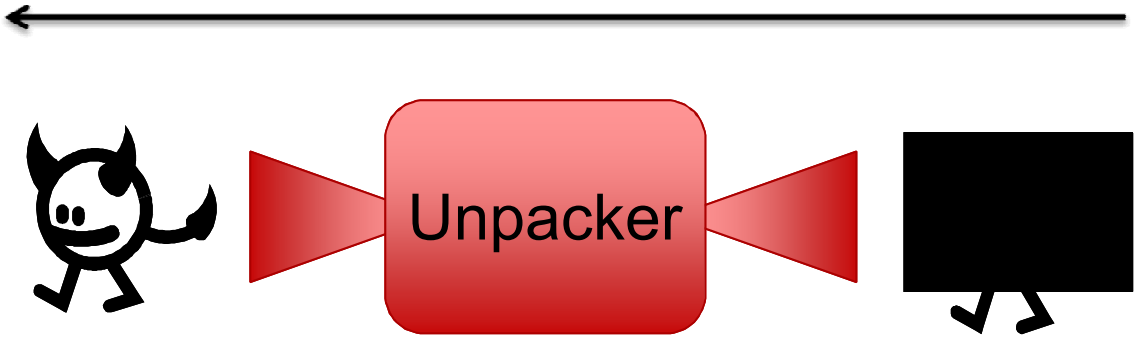
Sandia National Laboratories

- Malware is often packed
  - Malware contents are compressed, encrypted, etc.
  - Result is an executable which first unpacks, then runs malware
- Malware analysis is now difficult
  - Opcodes, data, signatures, behavior are obfuscated
  - Hashing, clustering, filtering, comparing fails
  - Horns and tail are hidden



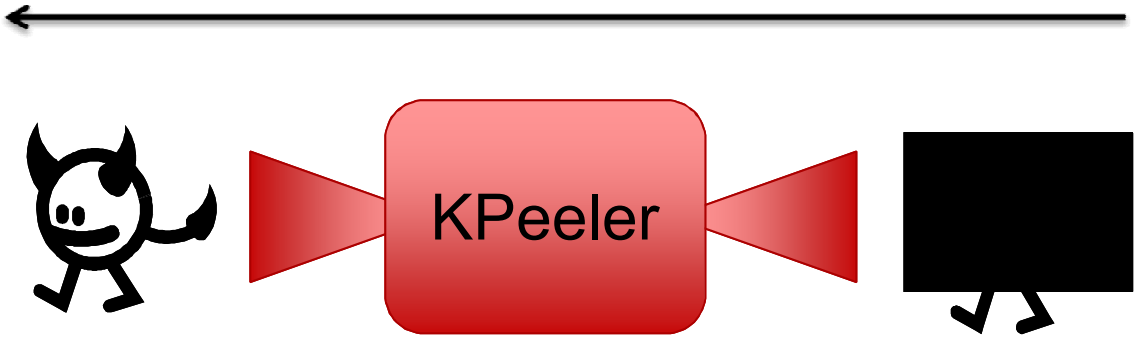
Problem

- What we need is an unpacker
  - Change packed executables back to their original form



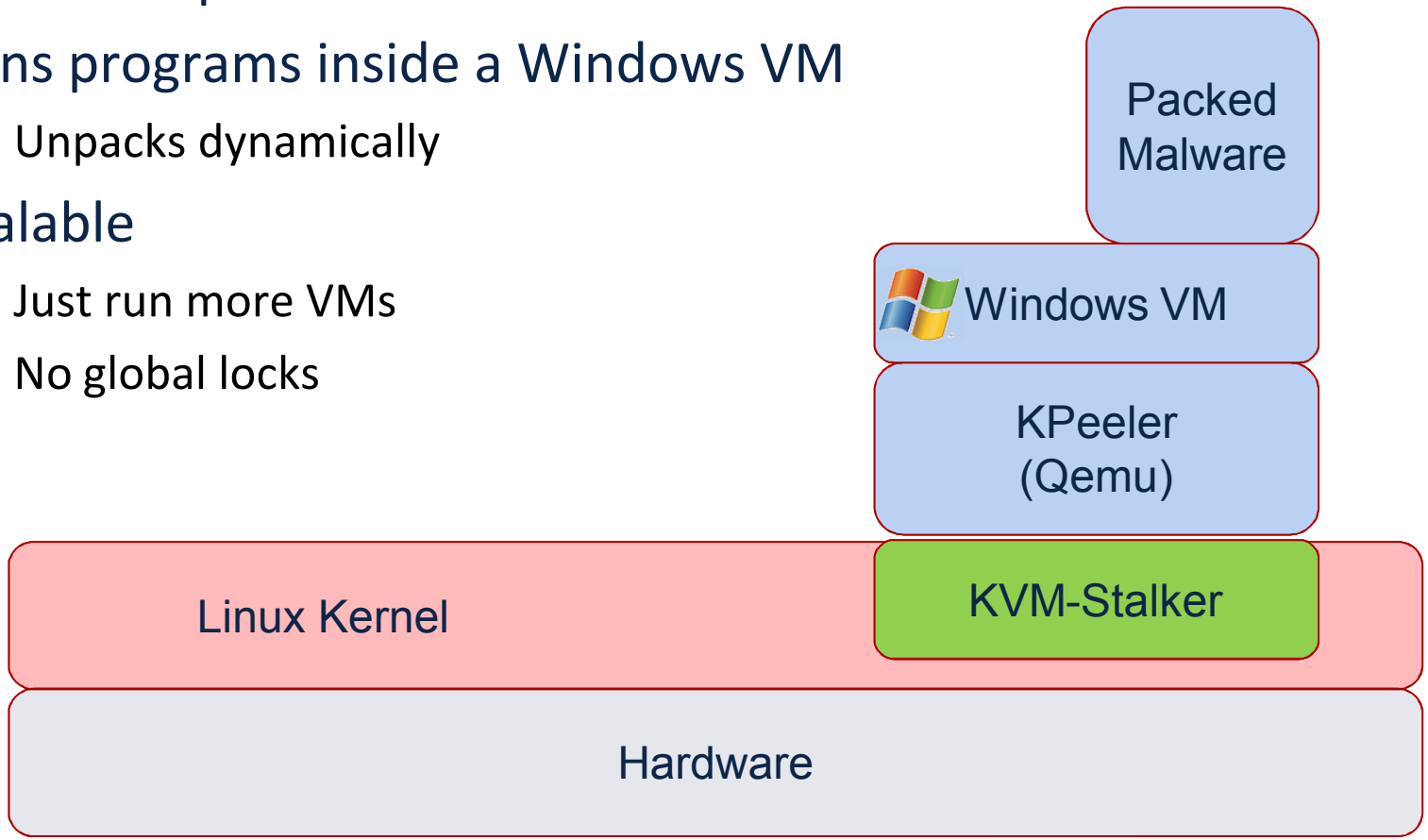
Problem

- What we need is an unpacker
  - Change packed executables back to their original form



Architecture

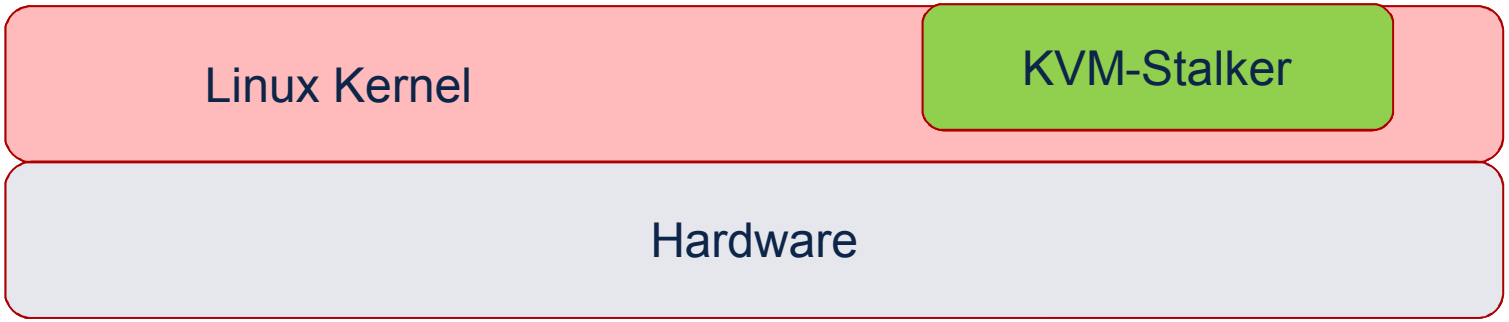
- KPeeler is a modified Qemu
- Runs on top of a modified KVM
- Runs programs inside a Windows VM
  - Unpacks dynamically
- Scalable
  - Just run more VMs
  - No global locks



Architecture – KVM-Stalker

Sandia National Laboratories

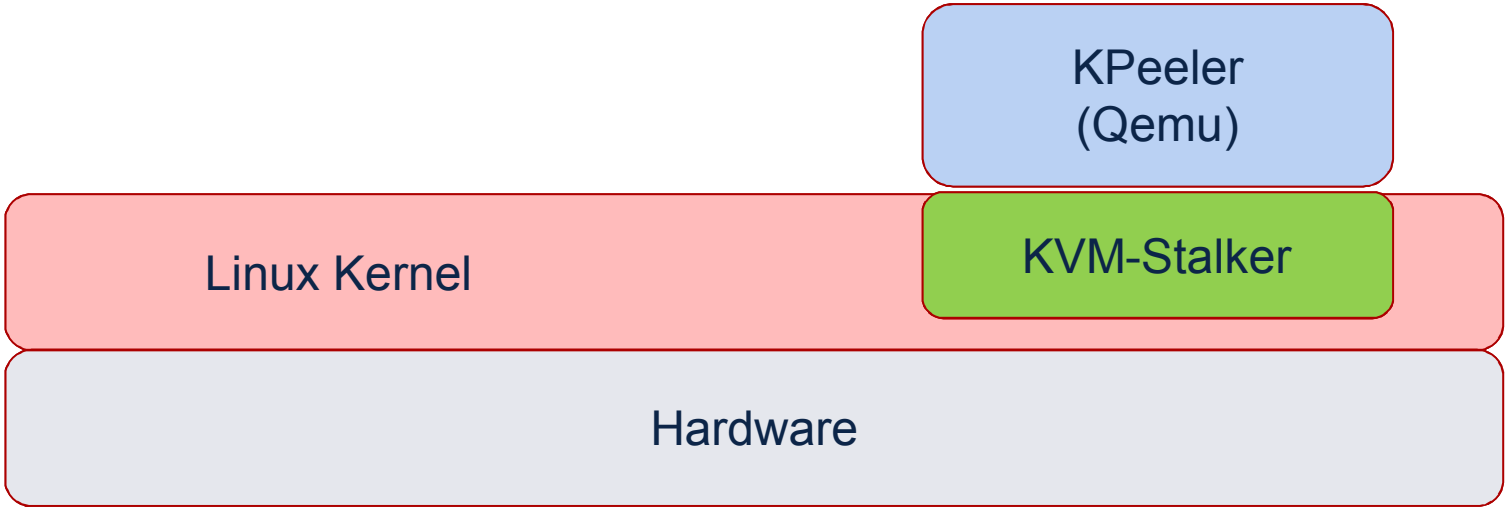
- Foundation is a modified KVM called KVM-Stalker
  - Hypervisor built into Linux Kernel
  - Gives control to guest VM, regains it on exceptions, I/O, etc.
- Tracks executing instructions, system calls, and memory writes of running programs
  - Provides info to KPeeler for analysis
- Obtaining info at hypervisor more easily avoids detection
  - Some malware won't unpack if it knows it's being debugged



Architecture – KPeeler

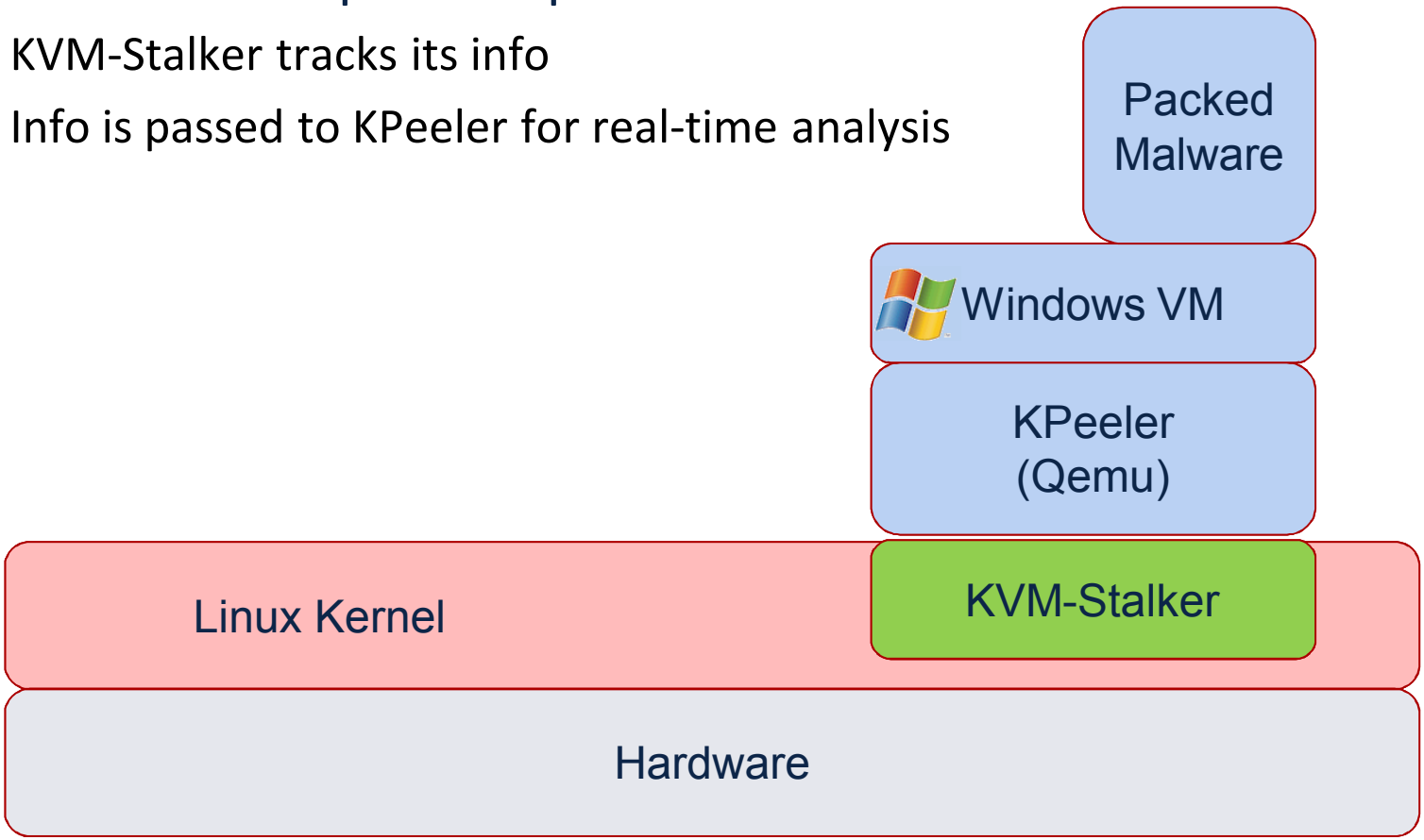
Sandia National Laboratories

- Receives info from KVM-Stalker
  - Communicates via IOCTLS
- Analyzes and attempts to extract malware content
  - Tracks guest's picture of memory
- Extensible heuristics for determining if malware is unpacked
- Logs info about running malware



Architecture

- KPeeler starts a Windows VM
- Windows runs a piece of packed malware
  - KVM-Stalker tracks its info
  - Info is passed to KPeeler for real-time analysis



KVM-Stalker Internals

Sandia National Laboratories

- KVM-Stalker has access to all Windows VM data
  - Virtual CPU data
  - All Registers
  - Memory
  - Exceptions, interrupts, traps
- Communicates with KPeeler/Qemu with IOCTLS
  - Called by KPeeler code
  - Can pass data back and forth between kernel and user space
  - e.g. `kvm_vcpu_ioctl(envPtr, KVM_GET_SREGS, &sregs);`

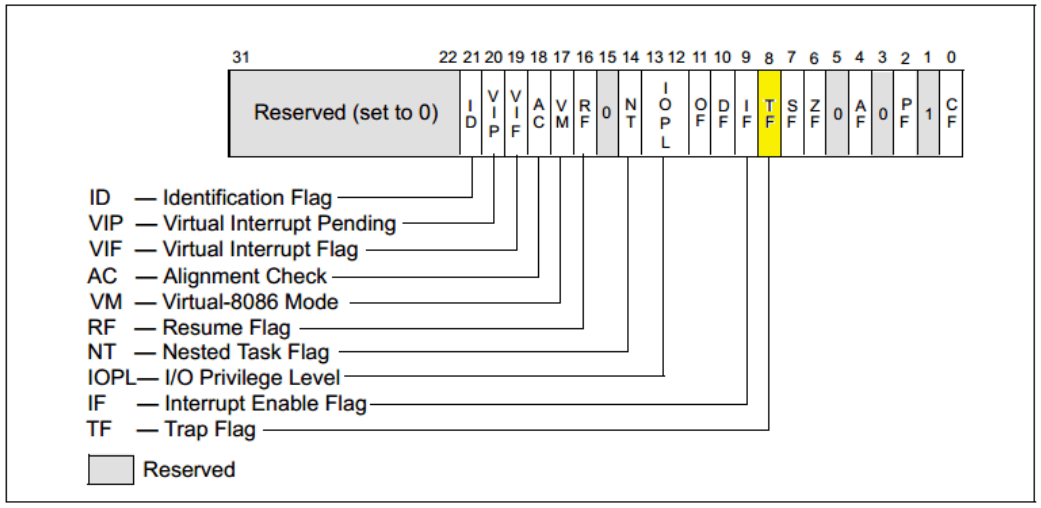
KVM-Stalker Internals

Sandia National Laboratories

- Want to make program info available to KPeeler for analysis
  - Instructions, system calls, writes to memory...
- General strategy
  - Make these events cause an exception to be handled by KVM-Stalker
  - Store event info in a kernel data structure
  - Give control to KPeeler
  - Use IOCTL to transfer event data to KPeeler for analysis/logging
  - Continue program execution

KVM-Stalker Internals – Instr. Trace

- Executing instructions give important unpacking insight
- KVM-Stalker sets trap flag in VM's EFLAGS
  - Generates a debug exception after every instruction
  - KVM-Stalker catches exception, vmexits to KPeeler
- KPeeler calls an IOCTL to get info about the instruction
  - `kvm_vcpu_ioctl(envPtr, KVM_STALKER_GET_INSTRUCTION_STATE, p)`
  - Processes and logs instruction



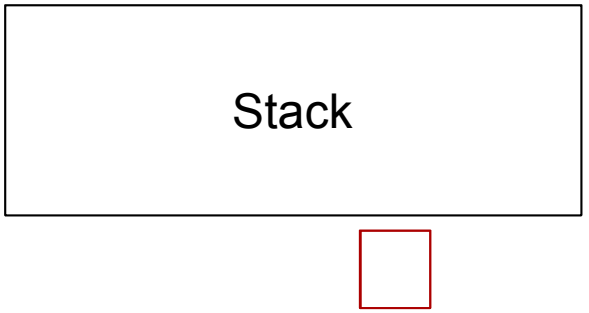
KVM-Stalker Internals – Instr. Trace

Sandia National Laboratories

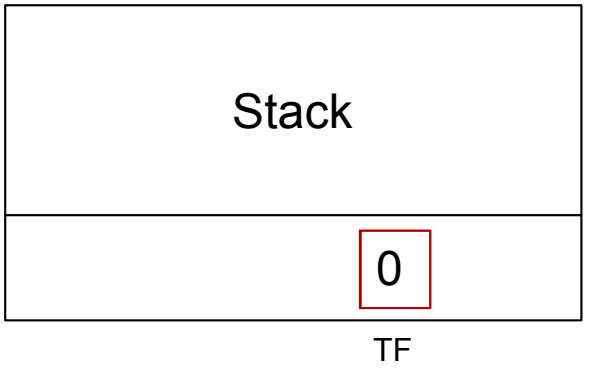
- Ignore VM kernel instructions we single step
  - ```
    static int handle_exception(struct kvm_vcpu *vcpu)
    {
        ...
        if(!kvm_stalker_is_userspace(vcpu, rip)) {
            return 1;     //Ignore and continue on
        }
        ...
    }
    ```
    ```

KVM-Stalker Internals – Instr. Trace

Sandia National Laboratories

- VM can clear trap flag, so make sure it's on
  - ```
    static int handle_exception(struct kvm_vcpu *vcpu)
    {
        ...
        if(!(vmx_get_rflags(vcpu) & X86_EFLAGS_TF)) {
            vmx_set_rflags(vcpu, (vmx_get_rflags(vcpu) | X86_EFLAGS_TF));
        }
        ...
    }
    ```
    ```

KVM-Stalker Internals – Instr. Trace

- VM can check trap flag
  - VM pushes EFLAGS onto stack with pushf
  - VM then checks trap flag on stack



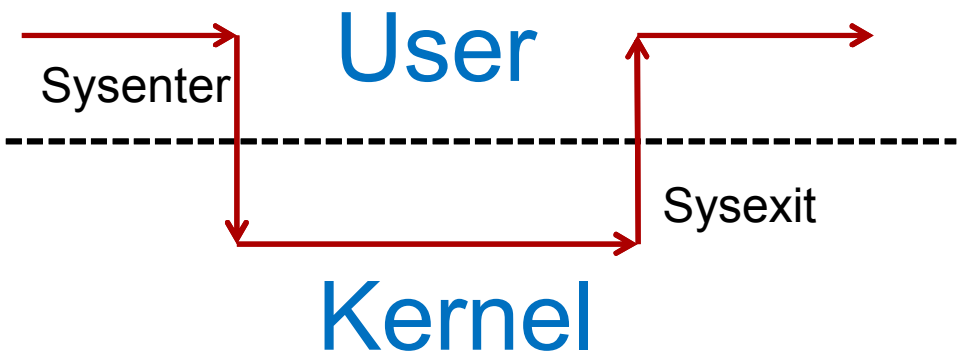
- We edit the stack to clear the trap bit
  - ```
    if (vcpu->stalker_pushf_correction_needed)
    {
        ...
        read_guest_mem(vcpu, rsp_va, flags, 4); //Read EFLAGS from stack
        *flags &= ~X86_EFLAGS_TF;     //Correct the TF value
        write_guest_mem(vcpu, rsp_va, flags, 4);   //Write EFLAGS to stack
    }
    ```


    ```

KVM-Stalker Internals – Instr. Trace

- VM can check trap flag
  - VM pushes EFLAGS onto stack with pushf
  - VM then checks trap flag on stack



- We edit the stack to clear the trap bit
  - ```
    if (vcpu->stalker_pushf_correction_needed)
    {
        ...
        read_guest_mem(vcpu, rsp_va, flags, 4); //Read EFLAGS from stack
        *flags &= ~X86_EFLAGS_TF;     //Correct the TF value
        write_guest_mem(vcpu, rsp_va, flags, 4);   //Write EFLAGS to stack
    }
    ```


    ```

KVM-Stalker Internals – Syscalls

Sandia National Laboratories

- Programs make system calls to ask the kernel to perform privileged tasks



- System calls can be used to allocate and free memory
  - This is important for unpacking
  - Want to create an exception for sysenter and sysexit

KVM-Stalker Internals – Syscalls

Sandia National Laboratories

- IA32_SYSENTER_EIP is a machine specific register
  - Tells what code to jump to during a sysenter
  - We set this to a bogus value
    - But save the real value!
- Sysenters now cause an exception handled by KVM-Stalker
- In sysenter handler
  - Set eip to real IA32_SYSENTER_EIP so sysenter can complete
  - Collect arguments, syscall number, etc. into syscall data structure
  - Set return value to bogus value so sysexit also causes an exception
    - But save the real value in the syscall data structure!

KVM-Stalker Internals – Syscalls

Sandia National Laboratories

- The system call completes
- Causes an exception when we sysexit to our bogus address
  - Again, exception is handled by KVM-Stalker

- In sysexit handler
  - Find syscall data structure that holds the real return address
    - Fix eip to correct return address
    - Add the return value of system call tot syscall data structure
  - Make sycall data structure available to KPeeler
  - Give control back to KPeeler for analysis

KVM-Stalker Internals – Mem Trace

Sandia National Laboratories

- Tracking memory changes is a big part of unpacking process
- Set pages of memory to read-only in page table
- Now VM raises exception on memory writes
  - Save info about memory write in kernel data structure
  - Set page back to read-write to allow write to go through

```
static int page_fault(struct kvm_vcpu *vcpu, gva_t addr, u32 error_code,
                      bool prefault)
{
   ...
   if (write_fault && user_fault){
      vcpu->memwrite_data.write_at_RIP = kvm_rip_read(vcpu); //Save rip
      vcpu->memwrite_data.address_of_write = addr; //Save addr of write
      ...                                          //Save other misc.
      mmu_spte_update(sptep, *sptep | PT_WRITABLE_MASK); //Make writable
   }
   ...
}
```

KVM-Stalker Internals – Mem Trace

Sandia National Laboratories

- On next single step exit
  - Use an IOCTL to get memory write info from previous instruction
    - Also inform KVM-Stalker to re-mark page as read-only
  - Update KPeeler's view of memory
    - Mark memory as modified

```
//Put memory write data into mw, re-mark page read-only
kvm_vcpu_ioctl(envPtr, KVM_STALKER_GET_MEMWRITE_INFO, mw);
for (i = 0; i < mw->write_size; i++) {
   mem_map[mw->address_of_write + i] = mw->write_data[i]; //Record write
   mod_map[mw->address_of_write + i] = true; //Record memory as modified
}
```

Unpacking Heuristic

Sandia National Laboratories

- KPeeler gets program info from KVM-Stalker
- Every time KPeeler single steps through another instruction
  - Log and process system calls
    - Update view of memory if necessary
  - Log and process all memory writes
    - Update view of memory
    - Mark written areas as modified
  - Log and process instructions
    - Watch for jumps

Unpacking Heuristic

Sandia National Laboratories

- Watch for jumps from non-modified to modified memory
  - Program might be jumping from unpacking to running malware
- Look for signs that unpacking is finished
  - Windows executable magic numbers
    - MZ, PE
  - Intact executable header/structure
  - Strings, checksum for modified block different from original program
  - Other extensible tactics
- Once satisfied, extract memory into unpacked malware file

Results

Sandia National Laboratories

- Pulled 50 samples from Sandia's FARM database
  - Selected samples based on low bytecode variance (< 0.4)

- KPeeler – 9/50
- TitanCore – 23/50
- RL!dePacker – 39/50

Ending Thoughts

Sandia National Laboratories

- Still a young project, work in progress
  - Lots of future work
  - Created to be easily extensible
- Unpacking rate can scale up with more VMs
  - No global locks
- Currently being incorporated into FARM as a tool

Thank You

Sandia National Laboratories

- Questions?