Overshadow: A virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems

What is this paper doing?

- This paper describes a mechanism for protecting an application from the kernel by running in a virtual machine. It handles this primarily by encrypting pages so that the OS cannot read them and rewriting system calls. This is potentially useful because should an attacker use an exploit and gain a root shell he won't be able to read application code or data since it is encrypted.

How does an application boot-strap?

- The exact procedure is hinted upon but never full fully described. Initialization of the program is handled by shim. It is not clear if encryption is a requirement to run at all on this system. An application's executable is decrypted when loaded into memory as pages are needed.

Running in Overshadow vs Integrity Mode

- Integrity mode allows the system to keep code and data in clear text but check for changes to pages by rechecking the hash. This removes the overhead of encryption and decryption of all the pages that are used.

Is both an encrypted and decrypted copy of everything kept?

- See Figure 1, basic cloaking protocol. No, pages of memory are decrypted as needed by the application, should the kernel attempt to access those pages they are encrypted unless the application has authorized it.

How does shim get into address space?

- Where shim is loaded in the memory space is not discussed in the paper.

When the (malicious) kernel is doing stuff, what can it break, could it modify shim?

- Fig. 3 – Control flow for faults and interrupts
    - 1. Application fires interrupt
    - 2. Calls VM
    - 3. Jumps to kernel
    - 4. Kernel jumps to shim
    - 5. Shim jumps back to VMM
    - 6. Back to application

- Can we do bad things at step 3 to modify or hack into shim? Shim is a wrapper for hypercalls, jumps back to VMM before going back to application.

Fig. 4 – control Flow for handling syscalls

- Step 3 is modifying system calls from step 1 for the kernel. An example of this is converting file I/O commands into mmap() operations. Interactions with the kernel are expensive and this can cause poor performance with syscall heavy applications as seen in Figure 6.