Privilege Escalation

Coleman Kane
Coleman.Kane@ge.com

February 9, 2015
Privilege Escalation or Elevation is the act of gaining access to resources which were intended to be protected by authorization mechanisms built into the targeted system. Generally, these are divided into two families:

- Horizontal
- Vertical
Generally speaking, this type of escalation occurs when the human-computer or computer-computer interface is intended to serve as "gate keeper" for access to internal-only resources on the system. In other words, the attacker exploits flaws in the UI to gain unrestricted access to everything that the underlying application would be able to access.

Some examples:

- SQL Injection attacks
- XSS attacks
- Escape to shell
Use of existing access (whether granted or illicit) to gain resource access beyond what the current user or role is supposed to have access to. In many cases, the act being described is elevating from "normal user" to "administrator" or "root" privileges on a system.

Examples:

- Win7Elevate / UAC Bypass
- "setuid" root binary UNIX exploit
- Device Driver exploits
- syscall exploits
Implemented by modern CPUs to enable arbitrary execution of native code while protecting systems

- HW offers multiple run modes, or "rings", where direct access to some features are permitted
- Code "asks permission" to switch to more privileged "ring"
- HW verifies authorization criteria are met
- In most systems, 2 modes: Supervisor & User
- Don’t confuse with "root"/"administrator" and "non-root" user processes
General OS division of code execution:

- **Supervisor:** Bootloader, kernel code, device driver code
- **User:** Applications, user commands, event "root" user programs

Some exceptions include Windows NT & Mac OS X (XNU kernel), which both can run "device drivers" as limited userland service applications that make kernel requests for their device interactions. However, they continue to offer the option to run as supervisor as well.
Most OS’s conceptualize permission groups based upon "privileged" and "unprivileged" user accounts

- Applications and services are executed by user, retaining privileges of the starting user, with the option to "downgrade" permission for security

- OS provides "API" for secure device/resource interactions by user ring code

- "Root" user typically granted access to whatever is asked for, while "non-root" is forced to operate with a limited set of privileges

- "setuid" and similar typically provide controlled temporary task-specific privilege elevation
Common non-root local-system privileges consist of:

- Starting new processes
- Reading/writing data controlled by the user or access-granted to the user
- Process mgmt for any processes running by that user
- Initiating network communications, or listening on TCP/UDP ports >1024
- Access to local OS APIs & libraries
In addition to all non-root privileges, root generally has expanded privileges:

- Adding/removing users, setting passwords, privileges
- Read/write ANY file (regardless of whether access explicitly granted)
- Installing/removing device drivers
- Replacing programs, libraries, etc...
- OS grants special access when root uses APIs / libraries
- Access to all devices, memory
- Process management across entire system
This structure presents us with two common vectors to gain vertical privilege escalation:

- Using OS APIs/syscall vulnerabilities to inject arbitrary code into Supervisor mode
- Using user-mode vulnerabilities in applications running as "root" to inject arbitrary code/commands as "root"
Virtualization is an increasingly popular feature of modern consumer CPUs, however it has been around for a long time. This feature adds a new super-supervisor level to the management of the operating system(s). The isolation enables a system owner to host multiple full OS’s with fine-grained resource access control, where a complete compromise of one OS cannot escape into other hosted operating systems.
Generally speaking, virtualization is facilitated by a hypervisor, which can be configured to selectively grant access to the following resources (among others):

- Number of CPU’s and % of CPU time
- Upper bound on virtual-physical memory (HV will allocate virt mem that VM thinks is physical mem)
- Virtualized I/O devices
- Direct hardware access (controlled)
- APIs provided by hypervisor available for each VM
Here are some examples of hypervisors:

- **VirtualBox** [http://virtualbox.org](http://virtualbox.org) (will frequently be used for examples)
- **VMWare** [http://www.vmware.com](http://www.vmware.com)
- **Xen** [http://www.xenproject.org](http://www.xenproject.org)
- **KVM** [http://www.linux-kvm.org](http://www.linux-kvm.org)
- **bhyve** [http://bhyve.org](http://bhyve.org)

The isolation and abstraction layers in hypervisors are the primary elements enabling "cloud computing". This provisioning approach also relies upon the inherent "Ring -1" security features to allow multiple customers the flexibility of executing virtual systems on shared hardware owned by a third party. However, some new exposure risks may present themselves if you are making assumptions common to traditional "bare metal" systems.