ECE 598 – Advanced Operating Systems
Lecture 23

Vince Weaver

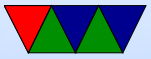
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

24 April 2018

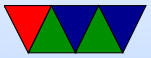
Announcements

- HW#10 was posted.
  Shared memory issue was other cores booting in VM mode.
  Got IPI working.
  Stuck trying to get per-core current_process
- Projects on Thursday. Last minute volunteers?
  Would be nice to have at least one more group.
- Midterm on May 1st (Tuesday). Not cumulative.



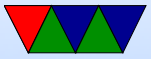
Project Presentations

- Aim for 8 minutes with maybe 2 min for questions
- Can present with slides if you want
- Mostly just showing off your project idea and what you accomplished.
- Intro (background), Experimental Setup, Results, Comments on what worked and what didn't, Future Work, Conclusion, Questions
- Feel free to give demo if possible.

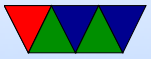


Midterm Review

- Virtual Memory: what it's good for, how two addresses can have the same address
- Filesystems: Fat, ext4, others. Why use fat? Virtual filesystems like /proc
- Multi-core/Locking/Deadlock
- IPC
- Security
- Maybe a brief graphics question

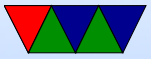


OS Security Continued

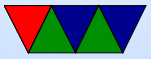


Users/Passwords

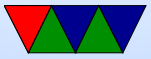
- Multi-user systems — are they needed? How do they work?
- Linux UID, GID
  How big is UID? Was 16 bits (64k) eventually updated to 32-bits (which broke some things) as some large universities ran into the limit
- How are users authenticated
  - /etc/passwd – hashed password, one-way hash
    can you crack a password? dictionary attack?



  - /etc/shadow
  - NSS, LDAP, PAM, NSS
- How enforced? Syscalls such as kill and such, check uid. Virtual memory usually cannot see any other processes. Our OS trickier, on ARM with large pages can have up to 16 different owner/protections but with us it has 1MB granularity.

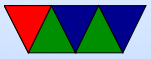


Escalating Privileges



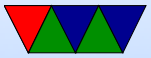
Security Through Obscurity

- KASLR randomization – knowing where parts of a program are can make exploits easier, so try to randomize where kernel and executables run
- Doesn't work well on 32-bit, only so many random places
- Also binaries released with distros are a bit predictable
- Makes debugging harder



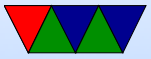
Hardware Against You

- Side Channel Attacks
  - Accurate timing (perf, high-res timers, etc)
  - Meltdown
  - Spectre
- DRAM Rowhammer – bang on memory cells, can have nearby bits flip to 0. How can that be an issue



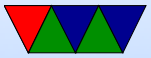
Hardware Help You

- VM/Pagetables – provide permissions
- Read/Write permissions to pages
- No-execute, seem obvious, came later (why?) No room in pages. Added with move to PAE or 64-bit PTEs often
- Newer support (ARM, etc) for kernel-noexec pages, i.e. to keep kernel from accidentally/on purpose jumping to and executing user code
- Can kernel just write to user provided addresses? Bad idea. Linux has copy_to_user and copy_from_user



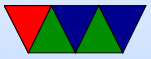
that tries to sanitize addresses and inputs before using.

- Newer support for fast hardware bounds checking



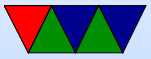
ROP programming

- Just mark all code as execute only (no read/write)
- What if can chain together series of small code snippets ending in return, and push onto the stack, then return from it?
- How to fix that? Somehow enforce code entry points



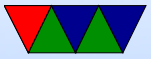
Pi Secure Mode

- Hardware can provide security features
- Pi has separate secure mode. OS can run in unsecure mode with limitations, separate vectors/page tables
- Only secure code can update important registers
- A pain if you're trying to do fancy stuff like turn off caches from userspace

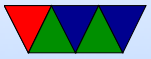


DRM

- Can you make truly "secure" machine only running code you want?
- First you have to trust the hardware and firmware
- Also need to trust the compiler (see: Reflections on Trusting Trust, Thompson 1984)
- Firmware only runs code signed with key
- Bootloader signed with key, as is operating system and any drivers
- In theory could also enforce signing of apps

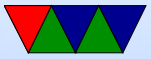


- Only code officially blessed by central authority can run on your machine.
- Might be more secure, but it would spell end to general purpose computing
- Media companies like it (no stealing movies)
  Video game companies (no pirating)
- Where does it go wrong? What if key leaks?
  What if you have a buffer-overrun and hackers can get it to run unsigned code? [this is how jailbreaks for phones and consoles often work]



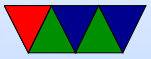
Sandboxing

- Try to run “safe” subset of instructions
- Google NaCL
- Containers
- Virtualization



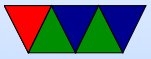
Finding Bugs

- Source code inspection
- Watching mailing lists
- Static checkers (coverity, sparse)
- Dynamic checkers (Valgrind). Can be slow.
- Fuzzing



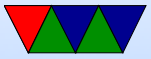
Reporting Linux Bugs

- All bugs are potentially security bugs
- Figuring out if any bug (and hundreds are fixed per week) is a security bug takes time
- False positives/negatives
- Having to manually mark them as such can help black hats find them easier

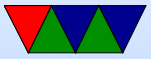


Fuzzing

- 1988. Barton Miller. Noticed unix programs crash due to line noise. General case OK, but die if you feed them out-of-range data.
- Idea is to check for potential crashes in programs by feeding them random inputs and see how they handle it.
- Trinity for Linux (by Dave Jones) is one current project
- perf_fuzzer (by me) is another



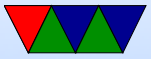
- Often the best way to get crashes isn't truly random inputs, but almost-correct inputs



perf_fuzzer

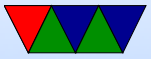
- Wrote my fuzzer in response to a perf_event bug found using trinity (I had contributed perf_event support to trinity).

- That bug was a 64-bit config value was only checked for validity with a 32-bit cast (so only bottom 32-bits). A user could then use the full 64-bit value to point to memory and increment values. The clever hacker managed to increment the IDT instruction vector table, point the undefined instruction interrupt to root shell



code, and then executed an undefined instruction.

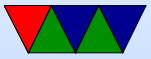
- I wanted to see if perf_event (a research interest of mine) had any other vulnerabilities.



Fuzzing – Bugs I found

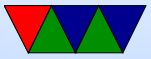
- ARM config too big – the config field was used as an offset into a (small) array without checking the size. Kernel Panic
- ARM dangling pointer – a small struct with function pointers was cast to a larger struct with function pointers, and one of the function pointers off the end was called. Kernel Panic

  Also, by luck on very specific 3.11-rc kernels the dangling pointer pointed to 0x80000000, which is user mappable.



Put code there to set uid to 0 and exec a shell and you have root. (Got a CVE, but getting this fixed took forever. Luckily it's very unlikely anyone was ever affected by this).

- Have found other, more boring, bugs. Usually just computer lockups.



Academic Fuzzing

- Just “regular” fuzzing is considered old-news, so despite being able to find new bugs all the time it’s hard to get academic funding or research for it
- Linux and hacker conferences are interested, although those are somehow not counted as exciting by other academics.