Kernel Security

CS390R - UMass Amherst

Course Information

- Project 5 due April 24
- You should have started work on presentations

Presentation Dates

- Day 1
  - Valerie Soltan & Aden Klotz
  - Asher Imran & Casey Ryan
  - Gary Wei & Ethan Hurlburt
  - Ronan Salz & Josh Barrett
  - Andrew Li & Ibrahima Keita
  - Thant Kyaw Hset, Hanlong Zheng & William Mironchuk
- Day 2
  - Jayesh Ramana & Hamza Shahzad
  - Robert Tacescu & Zachary Tower & Julia Kazmer
  - Jason Canuel & Joshua Hernandez
  - Jake Parkinson & Connor Andrews
  - Emily Nishikimoto & Sergio Ly
  - Richard Zhong & Aadam Lokhandwala
  - Manuel Bauche & Brendon Ky

Today’s Content

- Debugging
- Important files/instructions
- Applications of kernel Security (rootkits, custom drivers)
- Kernel bugs
- Mitigations
- Buffer-overflow -> Ret2usr
- UAF -> Heap-spray

Debugging

- Turn off pwndbg extension
- Add -s argument to qemu launch script
- gdb vmlinux
- target :remote 1234

Important files and instructions

- Files
  - /proc/kallsyms - lists all symbols loaded into the kernel and their addresses
    - Start of text section can be viewed at the top (head)
    - Last loaded kernel module symbols can be viewed at the bottom (tail)
- Instructions
  - swapgs - swap gs register between kernel and user mode
  - iretq - performs swap to usermode, requires RIP | CS | RFLAGS | SP | SS to be set.

Kernel bugs

- Basically same as userland
  - Buffer overflow
  - Use-after-free
  - Double free

Mitigations

- Stack Canary - Same as usermode canaries
- KASLR - Same as usermode aslr
- SMEP - Same as usermode NX
- SMAP - Usermode pages are non read/writable while in kernel mode
- KPTI - Separates kernel and user mode page tables while in kernel mode
- FG-KASLR - Fine grained kernel aslr

Exploitation technique - ret2usr

- Basically equivalent to user mode ret2shellcode
- Mitigated by most of the previously mentioned mitigations
- Goal is to increase privilege level of process while in kernel mode before executing system(“/bin/sh”) to spawn a shell.
  - Accomplished using ‘commit_creds()’ and ‘prepare_kernel_cred()’
- Store shellcode in userland and jump to it
  - Since we are jumping from kernel, we maintain privs

Let’s see how such an exploit might look

Exploitation technique - ret2usr /2

1. Save state (cs, ss, rsp, rflags)
2. Open device used by vulnerable driver
3. Overflow on read() syscall to leak aslr/canary
4. Overflow on write syscall to overwrite ‘ret’ with shellcode
5. shellcode calls ‘commit_creds(prepare_kernel_cred(0))’

Kernel Heap - UAF

1. Very different from userland heap exploits
2. Relias on allocating existing kernel structures into memory that we control using heap bug
3. We can then oftentimes read/corrupt pointers to defeat mitigations and achieve arbitrary read/write