ECE 471 – Embedded Systems
Lecture 26

Vince Weaver

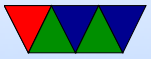
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

6 November 2020

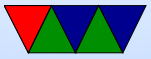
Announcements

- Don't forget project topics
- HW#9 will be posted, due in two weeks



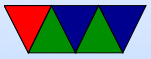
HW#9 Info

- Read temperature probe, print temp on i2c display
- Re-use code from past assignments
- Follow spec on functions to put code in, how to print results
- Testing: 4 cases described, and set up to allow unit tests

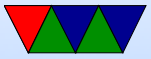


HW#7 Review – Code

- Follow directions: temp probe channel 2
- Why did we memset the structure before filling in the values?
- Converting 2 bytes into one. Be sure to mask
- What is the max frequency? someone setting to 500kHz by accident, a few degrees different. Data sheet unclear
- Don't just cut/paste all code in. Don't `ioctl(spi_fd, SPI_IOC_RD_MAX_SPEED_HZ, &f);` what does that do?
- Errors: exiting. Not print plausibly real invalid values.

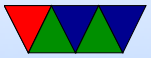


In our case, printing 0V when actually 3.3V not an issue, but imagine if it were 10,000V and you print 0V



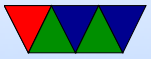
HW#7 Review – Questions

- Anti-lock brakes hard/soft/firm realtime?
  Hard. If things go wrong would be disaster
- Stereo change channel hard/soft/firm realtime?
  Soft. Prefer it not to be late, but still want to happen
- Video coming in at 60fps decoding?
  Firm, if frame decoded late it is useless
- Disadvantage of SPI?
  More wires, no standard, no errors
- Advantage of SPI?



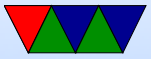
Lower Power, Full Duplex, No max speed

- TMP36 on end of cable.
  Voltage Drop, Noise?
  Datasheet has two options, convert to current, or an extra resistor.
- Minimum frequency of 10kHz or results invalid. Maybe cannot go this fast if bitbanging via GPIO. Also context switch in middle, Linux not realtime?



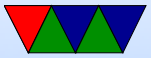
HW#7 Review – Linux "fun"

- /dev/null
- /dev/full
- /dev/zero, holes in files
- /dev/random – give explanation on sources of randomness (entropy), pseudo-randomness, etc.
- Mention related DOS/Windows compatibility issue



Buffer Overflows

- User (accidentally or on purpose) copies too much data into a fixed sized buffer.
- Data outside expected area gets over-written. This can cause a crash (best case) or if user carefully constructs code, can lead to user taking over program.

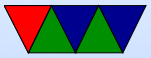


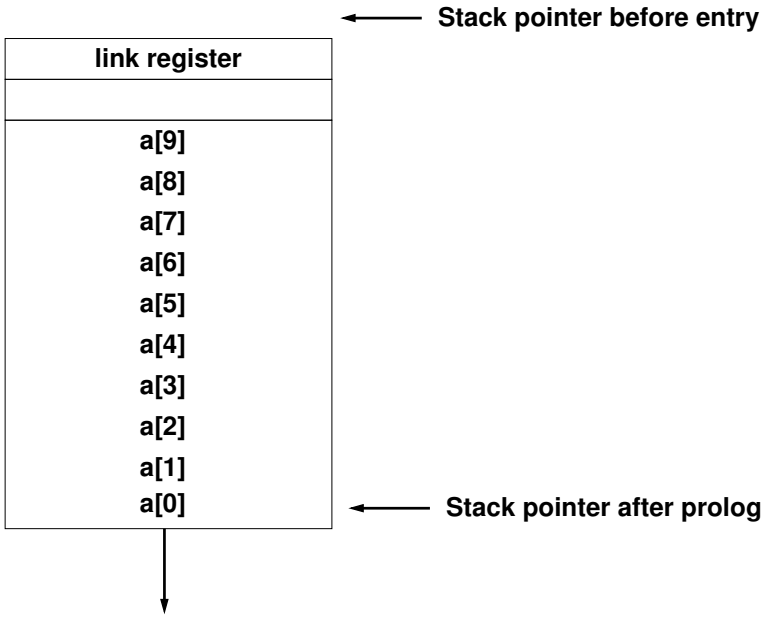
Buffer Overflow Example

```
void function(int *values, int size) {
    int a[10];

    memcpy(a,values,size);

    return;
}
```

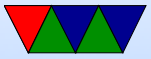
Maps to

```
    push    {lr}
    sub sp,#44

    memcpy

    add sp,#44
    pop {pc}
```



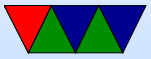


A value written to a[11] overwrites the saved link register. If you can put a pointer to a function of your choice there you can hijack the code execution, as it will be jumped to at function exit.



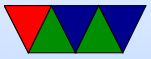
Mitigating Buffer Overflows

- Extra Bounds Checking / High-level Language (not C)
- Address Space Layout Randomization
- Putting lots of 0s in code (if strcpy is causing the problem)
- Running in a "sandbox"



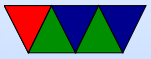
Dangling Pointer / Null Pointer Dereference

- Typically a NULL pointer access generates a segfault
- If an un-initialized function pointer points there, and gets called, it will crash. But until recently Linux allowed users to mmap() code there, allowing exploits.
- Other dangling pointers (pointers to invalid addresses) can also cause problems. Both writes and executions can cause problems if the address pointed to can be mapped.

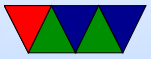


Privilege Escalation

- If you can get kernel or super-user (root) code to jump to your code, then you can raise privileges and have a "root exploit"
- If a kernel has a buffer-overrun or other type of error and branches to code you control, all bets are off. You can have what is called "shell code" generate a root shell.
- Some binaries are setuid. They run with root privilege but drop them. If you can make them run your code

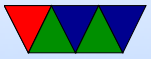


before dropping privilege you can also have a root exploit. Tools such as ping (requires root to open raw socket), X11 (needs root to access graphics cards), web-server (needs root to open port 80).



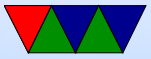
Information Leakage

- Can leak info through side-channels
- Detect encryption key by how long other processes take? Power supply fluctuations? RF noise?
- Timing attacks
- Meltdown and Spectre

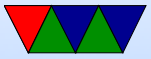


Finding Bugs

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

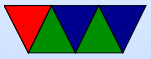


Computer Security



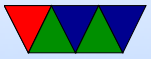
Social Engineering

- Often easier than actual hacking
- Talking your way into a system
- Looking like you know what you are doing
- “The Art of Deception”



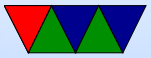
Worrisome embedded systems

- Backdoors in routers.
- Voting Machines, ATMs
- pacemakers
- Rooting phones
- Rooting video games
- Others?



Voting Machines

- Maine has paper ballot — not too bad
- Often are old and not tested well (Windows XP, only used once a year)
- How do researchers get them to test? e-bay?
- USB ports and such exposed, private physical access
- Can you trust the software? What if notices it is Election Day and only then flips 1/10th the vote from Party A to Party B. Would anyone notice? What if you have source code?



- What if the OS does it. What if Windows had code that on Election Day looked for a radio button for Party A and silently changed it to Party B when pressed?
- OK you have and audit the source code. What about the compiler? (Reflections on Trusting Trust). What about the compiler that compiled the compiler?
- And of course the hardware, but that's slightly harder to implement but a lot harder to audit.