ECE 471 – Embedded Systems
Lecture 26

Vince Weaver

https://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

9 November 2026

Announcements

- HW#9 assigned, due on 20th
- Project topics responded to (22 groups???)
- Reminder no class Wednesday for Veteran's Day
- Midterm #2 on Wednesday November 18th

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;
}
```

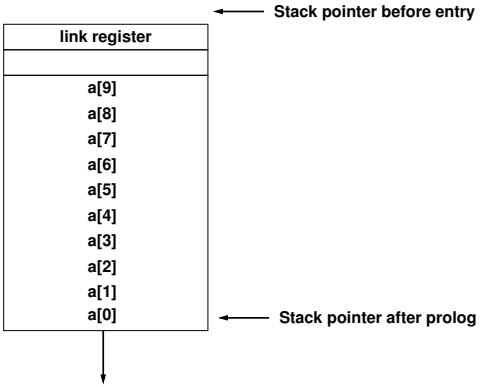
Maps to

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

        memcpy

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

Buffer Overflow Example



A value written to a[11] overwrites the saved link register. If you can put a pointer to custom/evil code there 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)
- Scanning for unusual characters (can you write all-ASCII shellcode?)
- Running in a "sandbox"
- Shadow stack – recent processors support this. Either separate stack for return values and variables, or else have two stacks and compare before returning
- Control Flow Enforcement hardware

Coding Mistakes with Security Implications

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.

Setuid

- 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.
  - ping (requires root to open raw socket)
  - X11 (needs root to access graphics cards)
  - web-server (needs root to open port 80).
  - `LD_PRELOAD` can interact poorly with these

Information Leakage / Side Channel Attacks

- Can leak info through side-channels
- Detect encryption key by how long other processes take?
  Power supply fluctuations? RF noise?
- Timing attacks
- If code takes different paths through code can notice this via linked info
  Solution: cycle-invariant code, takes same amount of time for all paths through code (really hard to write code like this)
- Recent CPU architecture extensions to help with this (ARM64 DIT data independent timing)

Information Leakage: Meltdown and Spectre

- Can use timing to find if address is in cache
- If speculative execution, can do things like

```
if (*secret&1) a[0]=1;
else a[4096]=1;
```

then use timing to see which one was brought in

Attacks on Public Sourcecode

- Can you sneak purposefully buggy/exploitable code into open source?
- Can you sneak bad code (or use typo-squatting) to trick people in large public repositories (like javascript/npm)
- To-do at U of Minnesota where researches tried (unsuccessfully it turns out) to sneak questionable code into the kernel

Intentionally Misleading Sourcecode

- "Trojan Source" in the news: can use unicode (including left-right reversal) to have code that looks correct but compiler will compile differently
  `x!=y vs y=!x`
- Should code allow non-ASCII?
- Underhanded C-code competition

Finding Software Bugs

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

perf_fuzzer

- Fuzzers intentionally try invalid/dangerous input by generating random inputs causing crash
- Can often get even better results by taking good inputs and slightly modifying them (off by one, etc.)
- I wrote the `perf_fuzzer` which found many bugs in Linux kernel with the `perf_event_open()` syscall

Reporting Bugs

- So you found a security bug...
- Who do you contact?
- What's responsible disclosure?
- Bug bounties
- Can be a hassle reporting properly, and companies are always suspicious and can even accuse you of evil hacking
- Current Events (2025): google reporting lots of bugs in free ffmpeg package but not willing to help fix them
- AI is ruining everything (2026)

Social Engineering

- Often easier than actual hacking
- Talking your way into a system
- Looking like you know what you are doing
  - Wear a safety vest?
  - Wear a lab coat and clipboard?
  - Wear a UPS uniform and carry a bulky package?
- Mitnick's "The Art of Deception"