ECE 471 – Embedded Systems
Lecture 27

Vince Weaver

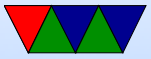
https://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

10 November 2025

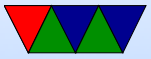
Announcements

- HW#9 assigned, due on 21st
- Project topics responded to
- Midterm #2 on Wednesday November 19th



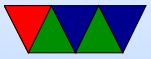
Computer Security
and why it matters for embedded systems

- Most effective security is being unconnected from the world and locked away in a box. Until recently most embedded systems matched that.
- Modern embedded systems are increasingly connected to networks, etc. Embedded code is not necessarily prepared for this.
- Internet of Things: IoT (the S is for Security)



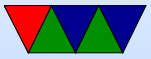
Computer Security – The Problem

- Untrusted inputs from user can be hostile.
- Users with physical access can bypass most software security.



What can an attacker gain?

- Fun / Mischief / Curiosity
  "Some people want to watch the world burn"
- Profit
- A network of servers that can be used for illicit purposes (SPAM, Warez, DDOS, bitcoin mining)
- Spying on others (companies, governments, etc)

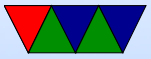


Sources of Attack

- Untrusted user input
  Web page forms
  Keyboard Input

- USB Keys (CD-ROMs)
  Autorun/Autostart on Windows
  Scatter usb keys around parking lot, helpful people plug into machine.

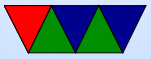
- Network



cellphone modems
ethernet/internet
wireless/bluetooth

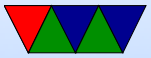
- Backdoors
  Debugging or Malicious, left in place

- Brute Force – trying all possible usernames/passwords



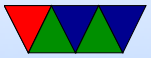
Types of Security Compromise

- Crash
  "ping of death"
- DoS (Denial of Service)
- User account compromise
- Root account compromise
- Privilege Escalation
- Rootkit (modify OS kernel to hide attacks)
- Re-write firmware? VM? Above OS?

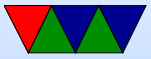


Unsanitized Inputs

- Using values from users directly can be a problem if passed directly to another process
- If data (say from a web-form) directly passed to a UNIX shell script, then by including characters like ; can issue arbitrary commands: `system("rm %s\n",userdata);`
- SQL injection attacks; escape characters can turn a command into two, letting user execute arbitrary SQL commands; xkcd `Robert '); DROP TABLE Students;--`

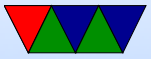


https://xkcd.com/327/



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.

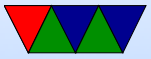


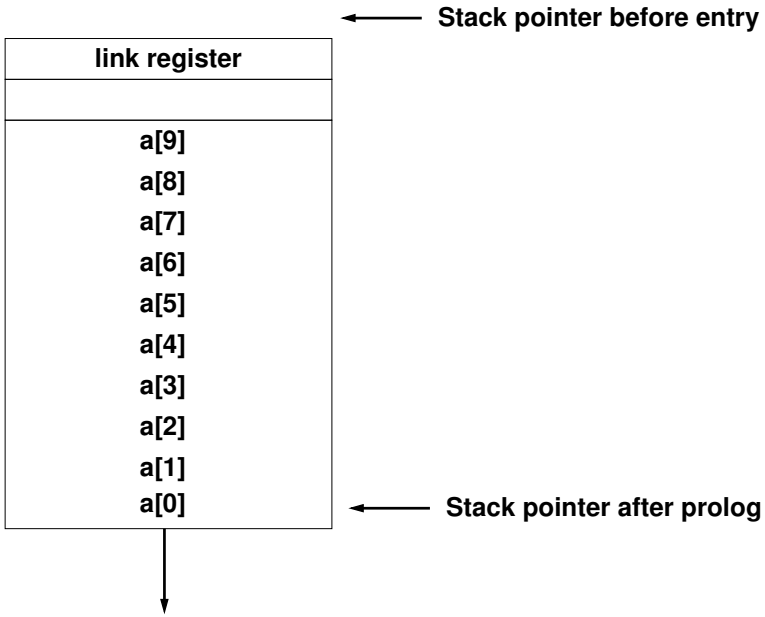
Buffer Overflow Example

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

    memcpy(a,values,size);

    return;
}
```

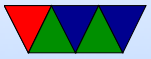
Maps to

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

    memcpy

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



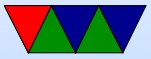


A value written to a[11] overwrites the saved link register. If you can put a pointer to a function of your choice there you can hijack the code execution, as it will be jumped to at function exit.

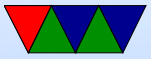


Mitigating Buffer Overflows

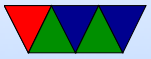
- Extra Bounds Checking / High-level Language (not C)
- Address Space Layout Randomization
- Putting lots of 0s in code (if strcpy is causing the problem)
- Scanning for unusual characters (can you write all-ASCII shellcode?)
- Running in a "sandbox"
- Shadow stack – recent processors support this. Either separate stack for return values and variables, or else



have two stacks and compare before returning

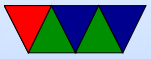


Coding Mistakes with Security Implications



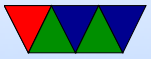
Dangling Pointer / Null Pointer Dereference

- 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.

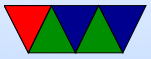


Privilege Escalation

- 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.
- 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.

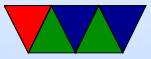


- ping (requires root to open raw socket)
- X11 (needs root to access graphics cards)
- web-server (needs root to open port 80).
- `LD_PRELOAD` can interact poorly with these



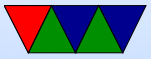
Information Leakage / Side Channel Attacks

- 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)

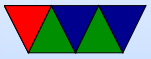


Information Leakage: Meltdown and Spectre

- 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



Continued Next Lecture