ECE 471 – Embedded Systems
Lecture 25

Vince Weaver

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

`vincent.weaver@maine.edu`

1 November 2023

Announcements

- Don't forget HW#8 (1-wire)
- Don't forget Project topics by Friday
  will respond to them via e-mail
- Please send an e-mail with your topic even if you've talked to me about it in person

Quick Rundown of Project Topic Possibilities

- There's a list of possible projects and a link to past projects toward the end of the assignment pdf
- There's also a list of parts. Lots of sensors, displays, and other things available
- If you do want to borrow parts, give me a bit of warning as sometimes can take a bit to find it and make sure it's in working condition

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)

Big Event Where This Matters

- Election next Tuesday
- Are voting machines embedded systems?
- Places with Electronic Voting Booths (Maine generally has paper ballots which are a bit better)
- Can you “hack” an election?
- Have been found trivial to hack. Running windows, with exposed USB connector.
- How did researchers get access to them. (eBay)
- Attacks often have to be local unless you happen to hack

main database

- Paper ballots tend to be more secure
- Social Engineering issues.
- What about vote-by-mail? Ruins anonymous voting, as people can bribe/watch you vote
- Internet voting (?!)

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.

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
- Profit
- A network of servers that can be used for illicit purposes (SPAM, Warez, DDOS)
- 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
- 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;--`

Buffer Overflows

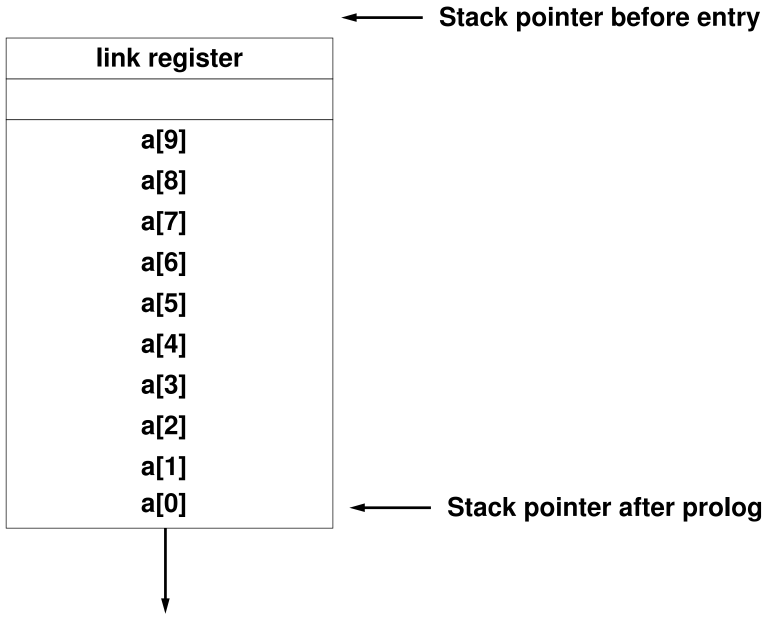
- User (accidentally or on purpose) copies too much data into a fixed sized buffer.
- Data outside expected area gets over-written. This can cause a crash (best case) or if user carefully constructs code, can lead to user taking over program.

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”

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).