

# ECE 471 – Embedded Systems

## Lecture 25

Vince Weaver

<https://web.eece.maine.edu/~vweaver>

[vincent.weaver@maine.edu](mailto:vincent.weaver@maine.edu)

6 November 2024

# Announcements

- Don't forget HW#8
- Project ideas due Friday: even if you've told me about it, send an e-mail if you haven't yet (note this is worth points on your project grade)
- No class Monday (Veterans' Day)



# 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



# Signed Firmware

- Best you can do is trust it to be the same firmware released by your vendor (you still have to trust them)
- Use cryptographic signing. Hardware will only run code “signed” by a trusted entity.
- A signed firmware can run a signed bootloader which can run a signed operating system which can run signed apps



# Signed Firmware Tradeoffs

- Downside: no longer general purpose, average person cannot run code they wrote unless they can get it signed
- Code still has to be well written. “jailbreaks” on phones and video game consoles are due to trusted code having bugs and then jumping into unsigned code.
- Will you still be able to run Linux?  
Trust Microsoft to keep signing bootloader for us?
- Walled gardens, restricted App stores (see Apple / EPIC lawsuit)



- Maybe makes sense if the firmware in your microwave locked down, but also might mean no one can ever fix it



# Signed Firmware on “General Purpose” Computers

- Hardware manufacturer has keys that only allow booting trusted signed boot firmware (UEFI)
  - Signed UEFI only allows booting trusted signed bootloader
  - Signed bootloader only allows booting trusted signed operating system
  - Signed operating system only allows signed device drivers



- Signed operating system only allows running signed apps
- Already here in a lot of ways
  - Get warnings if MacOS apps compiled/released by someone without a signature
  - Windows 12 requiring hardware with TPM hardware
  - Some smartphones and game systems have been here a while
  - Linux on trusted hardware, Microsoft owns the keys and for now will sign a bootloader that will run Linux. Do we trust them?





# Signed Firmware Tradeoffs

- Right to Repair Laws
- People concerned that car repair and cell phone repair becoming impossible as the companies lock things down with microcontrollers so you can only use officially approved parts
- Actually tractors and farm equipment big deal too
- Maine Ballot initiative on this November 2023



# Firmware on Desktop/Laptop Systems

- What is the Pi GPU doing?
- What about the T2 processor on macs?
- New for ARMv8: ARM Trusted Firmware (ATF). Two standards, vendors have possibly made a mess of it already.
- Other platforms have it too. DRM to keep you from copying movies or video games.
- Windows 11 requiring TPM2 module



# Firmware Hacking

- How can you figure out what the firmware is doing?
- Reverse engineering
- Looking for Security Issues
- Bypassing security measures



# Firmware Security

- Encryption, but key has to live somewhere
  - If it's in ROM or RAM, can be dumped unless careful
  - Also might travel bus in open (original Xbox)
  - Decapping, people dissolve tops off chips and look at with microscope to see contents
- Extra logic gates (work by profs here at UMaine)
- Glitching
  - If give improper power (too little/too much), temperature (too cold/too hot), noise/sparks, etc, can



cause code to jump to places it shouldn't

- Poor user code
  - Save game bugs popular cause of jailbreaks on consoles
  - Console code can be super tight but if you sign a game from a customer and it has a bug, all might be lost



# Locking down Hardware

- Many embedded boards, like STM32, you can “blow the fuses” after programming so that you can’t read out or re-flash ROM
- Recent news: locked down STM32 boards, but when debugger hooked up still shows address of interrupts. What if move interrupt table into protected code and trigger interrupt? Code appears as interrupt address?



# Why Lock Down Hardware

- Companies say it's for safety, security. Are there other reasons?
  - Profit. Can they lock you into one phone vendor? Lock you into one app store?
  - Secrecy. Don't want secrets of how their code works getting out.
  - "Piracy". Companies super worried you'll save video / music / games and then copy to your friends without paying for them. Digital Rights Management (DRM)



# Question

- Instead of epoxying shut USB, just have OS ignore any USB ports?
  - In theory you could do that. Which is cheaper, unskilled labor epoxying USB or else writing your own kernel (or convincing Microsoft) to disable USB?
  - According to spec if OS is ignoring USB, are you safe?
  - (Ignoring the possibility of a USB device with super-capacitors on board to explode your system)
  - Past DMA attacks with Firewire which could bypass





## OS and read/write memory

- What if hardware vendor for whatever reason talks to the USB device even though OS hasn't asked it to? It probably shouldn't, but how can you audit that? What about the firmware on the USB controller itself?
- The USB controller hooks up to a main CPU bus somehow, maybe over PCIe, maybe directly to CPU. It can potentially do lots of sneaky things.
- Why would there even be a controller for USB? Well when you plug in has to negotiate power, speed, and other stuff too. Probably a microcontroller



- It all depends how paranoid you want to be
- Even things like sandboxes and if you have all the code, can you \*prove\* the code is correct? That the compiler is trustworthy? This is actually a field of study, trying to prove code is correct. It's not trivial.
- Doesn't matter how good your software is if SW/HW compromised
- Can you analyze all 10 billion transistors in modern CPU?
- USB keyboard needs to work, even w/o OS



# HW#8 – C string review

- String manipulation is famously horrible in C.
- There are many ways to get the "YES" and "t=24125" values out of the text file for HW#8.
- Any way you choose is fine.



# C String Review

- This is tricky to get right
- It's relevant to Computer Security, the next topic we will cover



# What is a C string? – essentially a hack

- A NUL (zero) (note: not NULL) terminated array

- |   |   |   |   |   |    |
|---|---|---|---|---|----|
| H | e | l | l | o | \0 |
|---|---|---|---|---|----|

- Note this is really:

0x48	0x65	0x6c	0x6c	0x6f	0x00
------	------	------	------	------	------

- Note in C, arrays are essentially just pointers
- Can statically declare: (compiler puts the 0 on end for you)

```
char string1 [6]=" Hello " ;
```

```
char string1 []=" Hello " ; // autosize
```



```
char *string2=" Hello ";
```



# C String Review

- Many issues with array of bytes vs string, especially in other languages. Complicated if Unicode or UTF8. Windows / java and wchar (16-bit chars)
- You can use either pointer or array access to get a value ( `string[0]` is the same as `*string` )
- Note that double quotes indicate a string, while single quotes indicate a single character



# Upsides of C strings

- Fast and simple to deal with in assembly language
- Can quickly make short and cryptic functions to manipulate them
- ???





# Downsides of C strings

- No way to tell the maximum size from the pointer
- Can only find out current size of string by iterating to find end
- The C library has a lot of helper functions, many of which are flawed in deep ways



# Other String Implementations

- Pascal-style strings, first byte is the length
  - Always know length, no need to `strlen()`
  - Maximum size (if 8-bit than max 256 chars)
- Higher level / object oriented languages (python, C++?) still have some sort of array of chars inside, but wrap it with extra info to provide safer access to things



# C string pitfalls – Writing off the End

- What happens when web form on your device's web interface asking "name" and you allocate 64 bytes but don't check, and someone types 4096 bytes
- What's the worst case?
- Crash your program?
- Corrupt data?
- Complete system compromise?



# Can the C-library string functions save you?

- The standard `strcpy(char *dst, char *src)`
  - will happily go off the end if destination smaller than source
- `strncpy(char *dst, char *src, int size)`
  - added destination-size parameter, also pads dest with 0
  - NOTE: will leave off (!) the NUL terminator if not fit
- `strncpy(char *dst, char *src, int size)`
  - always terminates destination



- if destination full, you lose a byte as it is silently truncated and last byte made NUL
- No error is indicated if this happens
- why a problem? example: say want to remove file.txt~ but got truncated to file.txt instead?
- <https://lwn.net/Articles/507319/>
- Kernel has `strncpy(char *dst, char *src, int size)`
  - always returns valid string
  - returns a negative ERRNO on failure



# HW#8 Challenge – Reading from File



# Method One – File I/O Using fscanf()

- The “stream” file interface in C lets you use buffered I/O and is slightly higher level than `open()/close()`
- Open a file with: `FILE *fff;`  
`fff=fopen("filename","r");`  
Check for errors! `fff==NULL` if it fails to open
- close a file with `fclose(fff);`
- you can read a string using `fscanf(fff,"%s",string);`



# notes on scanf() functions

- printf() like interface

```
char string [256];  
int x;  
scanf(" %d %s" ,&x , string );
```

- Types to read like in printf, d for integer, s for string
- Useful trick, %\*s the asterisk means read but don't output, useful for skipping things
- Result goes to a pointer. Note a string is already a pointer so no need for an ampersand





- `scanf()` reads from standard input (keyboard)
- `fscanf()` reads from file
- `sscanf()` reads from a string



# Method Two – Read Entire File into RAM

- There are multiple ways to read files into a string in C  
Assume `char string[1024];`
  - `fd=open("filename",RD_ONLY);`  
`result=read(fd,string,1023); close(fd);`
  - `FILE *fff; fff=fopen("filename","r");`  
`fread(buffer,size,count,fff); fclose(fff)`
- If you are treating things as a string, be sure to NUL-terminate `string[result]=0;`



# Hardcoded sizes

- In the last example I was being lazy and hardcoded a 1k size instead  
Can you make that dynamic?
- Use `stat()` to get filesize, then use `malloc()` to allocate space? Be sure to `free()` when done



# Other ways to access file contents

- Advanced: use `mmap()`
- You can also use `fgets(buffer, size, fff);` to bring in one line at a time
- What about `gets()`? Dropped from C libraries as being too unsafe! No size so just writes forever



# Finding a location / substring in a larger string

- If you trust the Linux kernel developers to keep a “stable ABI” you can assume the temperature will always be a fixed offset and hard code it. This can be a bit dangerous.
- You can use the `scanf()` series of functions to parse the string (either `fscanf()` directly, or `sscanf()` on the string) One helpful hint, putting a ‘\*’ in a conversion (like `%*s` tells `scanf` to read in the value but ignore it.



- You can use the `strstr()` search for substring C-library function to search for substrings, i.e. `strstr(string, "NO");` (haystack, needle)
- Maybe in conjunction with `strtok()`?
- You can manually parse the array.

Using array syntax, something like:

```
i=0; while(string[i]!=0) {  
if (string[i]=='t') break; i++ }
```

Using pointer syntax, something like:

```
char *a; a=string; while(*a!=0) {  
if (*a=='t') break; a++; }
```



# Pointing into a string

- If you searched for "t=" you might now have a pointer a to something like "t=12345". To point to 12345 you can just add 2 to the string pointer.
- `printf("%s\n", string+2);`
- `printf("%s\n", &string[2]);`



# Converting string to decimal or floating point

- `atoi(char *string)` converts string to integer. What happens on error?
- `strtol()` will give you an error but is more complex to use
- `atof()` and `strtod()` will do floating point





# Comparing strings

- Can you just use ==? NO!
- Be careful using `strcmp()` (or even better, `strncmp()`) they have unusual return value less than, 0 or greater than depending. 0 means match  
So you want something like  

```
if (!strcmp(a,b)) do_something();
```

