ECE 471 – Embedded Systems
Lecture 23

Vince Weaver

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

vincent.weaver@maine.edu

2 November 2026

Announcements

- HW#8 was posted, pick up hardware if you haven't
- Don't forget project ideas for Friday (6th)
  E-mail, one per group, with group members and topic
  Please e-mail even if I said it was OK in person
- Midterms finally graded, average was 88

Current Events: Daylight Savings

- Clock math always hard for computers
- Can you hardcode start/stop days for Daylight Savings?
- Some states don't follow it (mostly AZ these days)
- Congress can (and has) changed it on fairly short notice
- Other countries this happens yearly
- This causes a lot of OS / firmware updates
- This year (2026) a lot of Western Canada decided to stay in permanent daylight savings at the last minute

Avoiding DST problems

- Linux/UNIX often had system clock UTC + offset rather than trying to have it set to local time
- Old Windows/DOS would use local time which could cause problems

Embedded System Security: Voting

- Election Tuesday
- Are voting machines embedded systems?
- Can elections be hacked?
- Most reports of hacking are social engineering type (i.e. Foreign Country buys deceptive Facebook ads)
- Recent reports seem to be poorly-calibrated touchscreens (which could be malicious but also could be bad UI design)

Voting Machine Types

- Vary a lot by state and even town (good is some ways, hard to target them all)
- Old fashioned: purely paper, machines with levers
- Like Maine: mark paper ballot, use embedded system to count, have paper trail if need recount
- Touch-screen voting (often encouraged for accessibility), OK if it prints paper ballot you can later hand count
- Fully electronic with no paper trail, difficult to audit
- Even worse: all internet voting

Mail in Voting

- Mail in votes have their own issues, make votes possibly non-anonymous so people can pressure/bribe you to vote a certain way

Voting Machine Hacking

- Some voting machines 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 (hopefully they aren't on Wifi)

Why hard to audit/secure?

- Often are old and not tested well (Windows XP, only used once a year)
- Run by small teams in towns w/o much IT experience
- Often bought from lowest bidder
- 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?

Why hard to audit/secure?

- 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. (10 billion transistors)

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.

Why review C Strings?

- Continuing problem with C programs
- It's relevant to Computer Security, a topic we will cover soon

What is a C string? – essentially a hack

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

- Note this is really:

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

- Note in C, arrays are essentially just pointers

Declaring a C string

- Can statically declare: (compiler puts the 0 on end for you)

```
char string1[6]="Hello";
char string2[]="Hello";  // autosize
char *string3="Hello";
```

- The above are all ways of declaring a NUL-terminated 5-character string
- Note that double quotes indicate a string, while single quotes indicate a single character

C String Issues

- Not own type, just array of bytes.
- 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)

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

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?

For example, lets look at the standard string copy function

- `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

FreeBSD / kernel string functions

- `strlcpy(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 got truncated to file.txt instead?
  - `https://lwn.net/Articles/507319/`
- Kernel has
  `strscpy(char *dst, char *src, int size)`
  - always returns valid string
  - returns a negative ERRNO on failure
  - They removed strncpy() from kernel entirely in 7.2!

HW#8 Challenge – Reading from File

Method One – File I/O Using fscanf()

- The "stream" file interface in C lets you used 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 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 File Line at a time

- use fopen() as before, but read in one line at a time
- the `fgets(char *string, int size, FILE **stream)` function is best at this
- You can then use `sscanf()` on the string to parse it
- Aside: the related `gets(char *string)` function is now forbidden! Can you guess why?

Method Three – Read Entire File and Scan

- This takes multiple steps:
  1. Read entire file into a string
  2. Use strstr() or similar to find the string you want
  3. Use this to grab either the YES string or the t=12345 value
  4. For the latter case, offset by two and then use atoi() or similar to convert to integer

Read Entire File into array

- There are multiple ways to read files into an array 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 the array as a string, be sure to manually 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

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.

Searching substring

- 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();`

Other ways to access file contents

- Can read in one character at a time with `fgetc()`
- Advanced: use `mmap()` to map a file directly into RAM without using open/read/write

Handed Back Midterm Exam

- Average was an 88