ECE 471 – Embedded Systems
Lecture 24

Vince Weaver

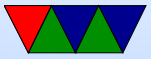
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

2 November 2020

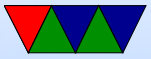
Announcements

- Don't forget HW#8
- Keep thinking about projects, topic due Friday.
- Class on Wednesday will start off with a 10-minute research talk doubled up with ECE100
- They just released the Raspberry Pi 400



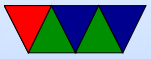
HW#8 – C string review

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 you choose is fine.



Method One – Read String Using fscanf()

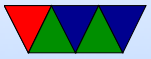
- The "stream" file interface in C lets you used buffered I/O and is slightly higher level than `open()`/`close()`
- Open a file with: `FILE *fff; fff=fopen("filename","r`
- close a file with `fclose(fff);`
- you can read a string using `fscanf(fff,"%s",string);` see details below
- There are multiple ways to read files into a string in C
  Assume `char string[1024];`
  - `fd=open("filename",RD_ONLY);`



`read(fd,string,1023); close(fd);`

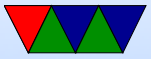
- `FILE *fff; fff=fopen("filename","r"); fread(bu`
  `fclose(fff)`
  You can also use `fgets(buffer,size,fff);`
- Advanced: use `mmap()`

- C strings
  - In C, characters are NUL (0) terminated character arrays (usually 8-bit bytes). Usually ASCII or UTF8
  - Other languages might be unicode, 16-bit, wchar
  - You can use either pointer or array access to get a value (string[0] is the same as *string)



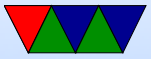
- Note that double quotes indicate a string, while single quotes indicate a single character
- It is very easy to accidentally go off the end of a string and corrupt memory
- Alternatives? Fancy libraries? Pascal strings (where first char is the length?)
- Always be sure your strings are terminated, otherwise bad things can happen (and not all C string manipulation functions do this properly, see strcpy(), strncpy(), strlcpy()

- 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, maybe in conjunction with `strtok()`



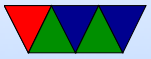
- You can manually parse the array.
  Using array syntax, something like:

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

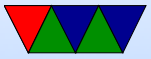
  Using pointer syntax, something like:

```
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()` 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()`



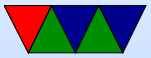
they have unusual return value

less than, 0 or greater than depending. 0 means match

So you want something like

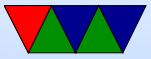
```
if (!strcmp(a,b)) do_something();
```

- Other file I/O: `fgetc()`



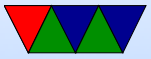
Computer Security
and why it matters for embedded systems

- 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

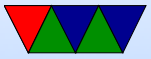


Big Event Where This Matters

- Election tomorrow
- Places with Electronic Voting Booths
- Have been found trivial to hack. Running windows, with exposed USB connector.
- How did researchers get access to them.
- Attacks often have to be local unless you happen to hack main database

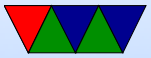


- Paper ballots tend to be more secure
- Social Engineering issues.
- What about vote-by-mail? Internet voting



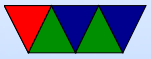
The Problem

- 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)

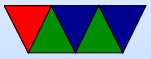


Sources of Attack

- 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.

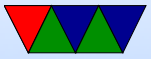
- Network



cellphone modems
ethernet/internet
wireless/bluetooth

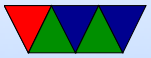
- Backdoors
  Debugging or Malicious, left in place

- Brute Force – trying all possible usernames/passwords

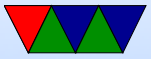


Types of Compromise

- Crash
  "ping of death"
- DoS (Denial of Service)
- User account compromise
- Root account compromise
- Privilege Escalation

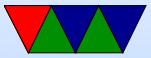


- Rootkit
- Re-write firmware? VM? Above OS?



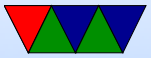
Unsanitized Inputs

- 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.

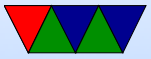


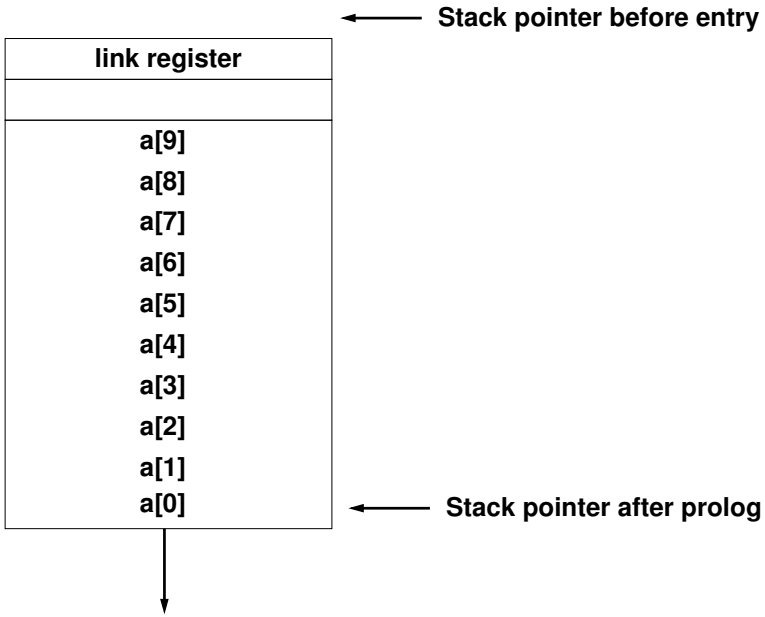
Buffer Overflow Example

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

    memcpy(a,values,size);

    return;
}
```

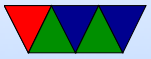
Maps to

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

    memcpy

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



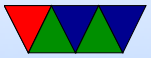


A value written to a[11] overwrites the saved link register. If you can put a pointer to a function of your choice there you can hijack the code execution, as it will be jumped to at function exit.



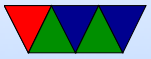
Mitigating Buffer Overflows

- Extra Bounds Checking / High-level Language (not C)
- Address Space Layout Randomization
- Putting lots of 0s in code (if strcpy is causing the problem)
- Running in a "sandbox"



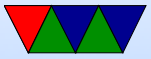
Dangling Pointer / Null Pointer Dereference

- 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.

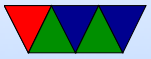


Privilege Escalation

- 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

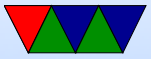


before dropping privilege you can also have a root exploit. Tools such as ping (requires root to open raw socket), X11 (needs root to access graphics cards), web-server (needs root to open port 80).



Information Leakage

- Can leak info through side-channels
- Detect encryption key by how long other processes take? Power supply fluctuations? RF noise?
- Timing attacks
- Meltdown and Spectre



Finding Bugs

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