ECE 435 – Network Engineering
Lecture 5

Vince Weaver

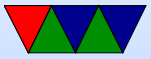
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

18 September 2018

Announcements

- HW#2 was posted.



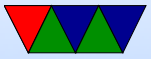
HW#1 Review

- Sockets Code
  - Need to clear out the old message and not print remnants.
  - How to copy a string in c?

    `strcpy(dest,src)` – potential buffer overflow if not terminated

    `strncpy(dest,src,size)` – better, but not always terminated

    `strlcpy()` – even better, not always available



Custom coded: always dangerous:

```
i=0; // important
while(src[i]) { dst[i]=src[i]; i++} dst[i]=0;
```

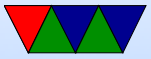
- Looping in the server:

  Want to accept *one* file descriptor, then loop reading it.

  Don't want to loop accept() and get new connection over and over.

  What does `accept()` do?

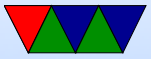
  If you keep `accept()`ing, you get new connection each time This also leaks file descriptors (Though unless >



4096 or more won't notice)

What you want to do is loop reading from the file descriptor until it closes.

- Looping in the client:
  Be sure the socket call is *outside* the forever loop. Otherwise you are starting a new connection each time, not reusing an existing one.
- Don't ignore compiler warnings.
  What if `toupper()` not found?
  manpage. Need to include `ctype.h`



- Specifications could Be Better
  - When you type "bye" it would exit both sides.
    (bye by itself? cr/lf? byet?
- Something Cool

  Command Line args argc=number. Always at least 1.

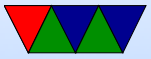
  argv[]. argv[0]=executable name

  argv[1]=first argument

  `atoi()/strtod()/strtol()` Why different?

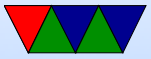
  `atoi()` has no way of reporting error, just returns 0.

  `strtod()` can report errors but is more complex.
- General



- Error handling: don't just segfault. Even if can't do anything, print message and try to exit gracefully.

- Comment your code!
- OSI reference model
  - Bits and voltages – physical layer (1?)
    Not hardware layer
  - Routing packets – network layer (3?)



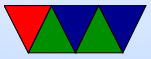
HW#2 Issues

- Get the header printing first, then worry about correctness of headers (dates, length))
- Know how to search for a string and point to location after it?
  - Find a string and point to beginning of it.

```
char *pointer;
pointer=strstr(haystack,needle);
```

  - Look for `"GET "`

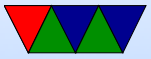
    Actually points to beginning of GET. How to skip ahead?



- pointer+=4 is one way. (pointer math, ugh)
- How to get to first space?
- `strtok(pointer," ");`
  Will split the string into chunks, put 0 at end.
- Also can do this manually;

```
pointer2=pointer;
while(*pointer) {
   if (pointer==' ') {
      *pointer=0;
      break;
   }
   pointer++;
}
printf("%s\n",pointer2);
```

- Know how to construct a string on the fly? `strcat()`,



`sprintf()`

`strcpy()` first bit in.

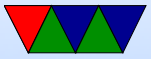
`strcat()` additional strings.

If you want formatting you can do things like

```
sprintf(temp_string,"File size=%d\r\n",filesize);
strcat(out_string,temp_string);
```

Create big enough buffer.

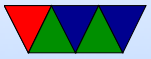
- How to find size of a file?
  Can read it in, and count. Or can use the stat (`man stat.2`) need .2 (or `man -a`) as there's a command line tool called stat that comes ip first.
- How to read/write file. There are a large number of



ways to do this. `open()/read()/write()/close` `fopen()/fread/fwrite/fclose` (careful! Buffered! And maybe need fdopen() to print to file descriptor).

```
fd=open(filename,O_RDONLY);
if (fd<0) fprintf(stderr,"Error opening %s\n",filename);
while(1) {
   result=read(fd,buffer,256);
   if (result<=0) break;
   write(network_fd,buffer,result);
}
```

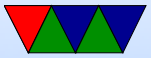
Be sure to close afterward.



Certificate Authorities

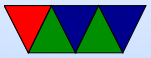
Problem: how do you verify the public key belongs to the person who they say it is? (on website? what if someone intercepts and replaces, mitm style)

- Certificate authorities
- Signed data block from official organization
- Hashed?
- Can be revoked
- Digital Signature Algorithm

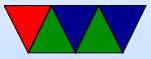


Other Encryption Concerns

- Redundancy, some way to validate plaintext is valid. Example: if encrypting a binary blob where each byte indicates something (12 34 means order 34 cows or something), random garbage might decode to valid message
- Freshness – replay attacks. What if you record old message (Bank deposits $100 to account) and replay. Will have valid encryption.
- Block chain ciphers

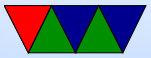


- Stream Ciphers



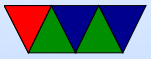
Encryption Problems

- Keys leaked (DVD/game console issues)
- poor random numbers used (Debian problem)
- differential cryptanalysis (start with similar plaintexts and see what patterns occur in output) [DES IBM/NSA story]
- Power/Timing analysis – note power usage or timing/cache/cycles when encryption going on, can leak info on key or algorithm
  Bane of perf



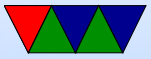
SSL/TLS

- Secure Socket Layer / Transport Layer Security
- Handshake protocol followed by key exchange
- Browser says hello, which hashes/algorithms it supports
- Server picks one and sends back
- Server then sends a certificate (signed by authority) saying who it is, and what its public key is
- Client verifies certificate (via the CA public key it has stored)
- client generates a random number, encrypts with servers



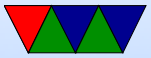
public key, sends to server, used as symmetric key

- What could go wrong, what if someone gets a hold of server private key? could decrypt logged data. Diffie-Hellman key exchange – random number plus unique session key prevents problems if server private key leaked



Other tools that use encryption

- How do you encrypt an e-mail, or a hard-drive, etc
- PGP – pretty good privacy
  OpenPGP RFC 4880
  Encrypt message with symmetric key, send along the key encrypted via asymmetric
  was illegal for a while (more than 40 bit encryption an exportable munition)
  people got RSA algorithm in perl tattoos
- GPG – free software replacement for PGP



- Can also PGP sign a message. Not encrypted, but signed with your key to verify it was in fact sent by you. Takes hash of the input, then encrypts the hash with key. Also, downloads from servers (like debian)