Unpleasant Things in C

A few examples

Proper Bounds Checking: All's OK, right?

```
char *read_data(int sockfd) {
        char *buf;
        // Read a 32-bit data length from the networks stream
        int length = network_get_int(sockfd);

        if ((buf = (char *)malloc(MAXCHARS)) != NULL)
                die("malloc: %m");
        if (length < 0 || length + 1 >= MAXCHARS)
                { free(buf); die("bad length: %d", value); }

        if (read(sockfd, buf, length) <= 0)
                { free(buf); die("read: %m"); }

        buf[length] = '\0';
        return buf;
}
```

Not So Fast…

- What is 2147483647 in hex?
- 0x7FFFFFFF
- What happens if *length* is set to this value?
- What happens when you add 1 to this value in *length*?
- 0x80000000
- What number does this represent in twos-complement?
- -2147483648
- What happened to our bounds check?

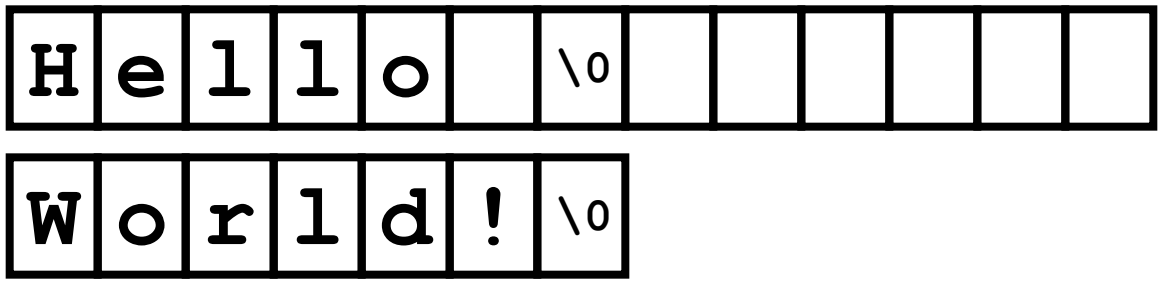
Not So Much?

```
char *read_data(int sockfd) {
        char *buf;
        // Read a 32-bit data length from the networks stream
        int length = network_get_int(sockfd);

        if ((buf = (char *)malloc(MAXCHARS)) != NULL)
                die("malloc: %m");
        if (length < 0 || length + 1 >= MAXCHARS)
                { free(buf); die("bad length: %d", value); }

        if (read(sockfd, buf, length) <= 0)
                { free(buf); die("read: %m"); }

        buf[length] = '\0';
        return buf;
}
```

strcat

```
char *strcat(char *a, char *b);
```

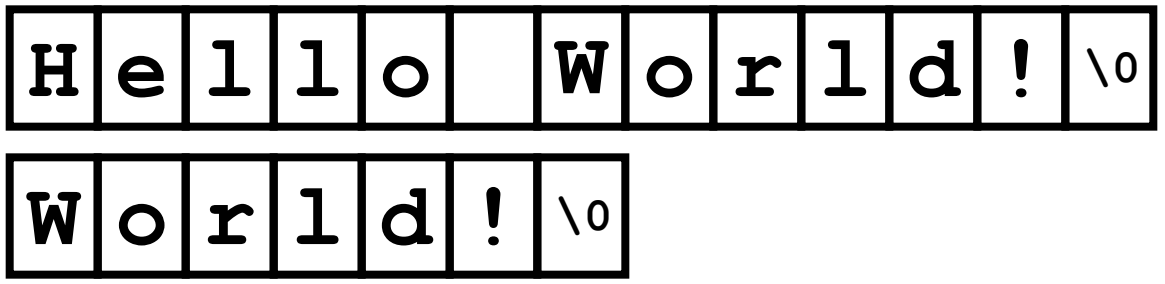
```
char x[13] = "Hello ";
char *y = "World!";
strcat(x, y);
```



strcat

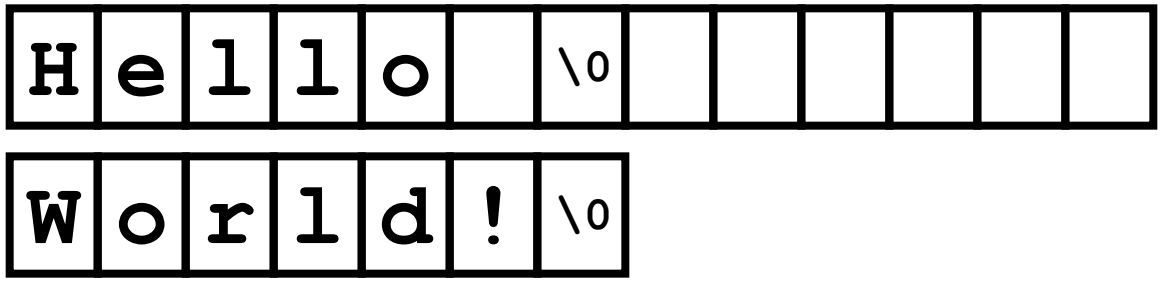
```
char *strcat(char *a, char *b);
```

```
char x[13] = "Hello ";
char *y = "World!";
strcat(x, y);
```



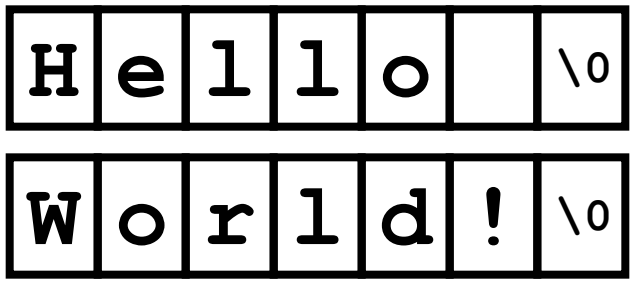
strcat

```
char *strcat (char *destination,
   const char *source){

   char *d = destination;
   while (*d) ++d;
   while ((*d++ = *source++) != '\0');
   return destination;

}
```



Overflowing a Buffer

```
char *strcat(char *a, char *b);


char x[] = "Hello ";

char *y = "World!";

strcat(x, y);
```



Dinosaurs That Should Be Extinct

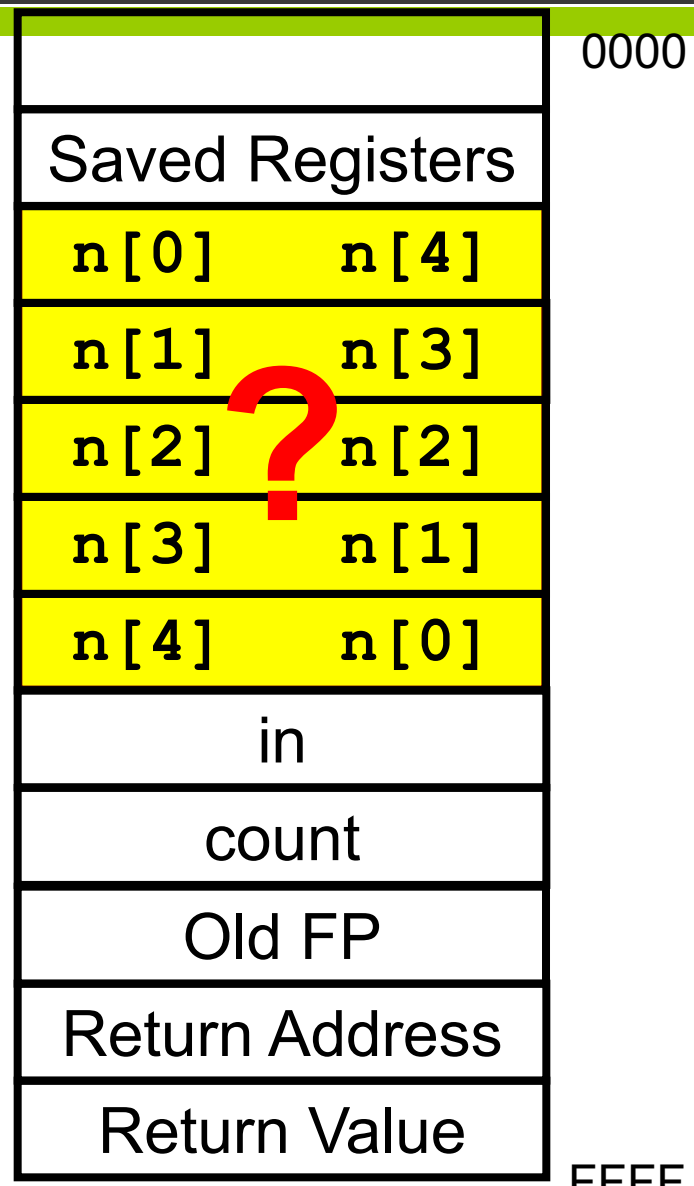
- Avoid strcpy() – use strncpy(dst, src, size)
- Avoid strcat() – use strncat(dst, src, size)
- Avoid sprintf() – use snprintf(dst, size, format,…)
- Avoid any string function that copies to a destination that doesn't have a *size* bound!
- (How many words does an extra argument cost on LC-3? Two, but the two-argument call only costs five words, so that extra argument is a 40% increase in memory footprint. Very similar to PDP-11. How does that look if you only have 32768 words of memory? Like you got your memory cut to 23408 words! Hence the original decision.)

Stack Games -- or Gaming the Stack

- Let us consider a function that reads some input values into an array
- …whose author might have forgotten to check an array bound
- Remember the stack frame from the LC-3?

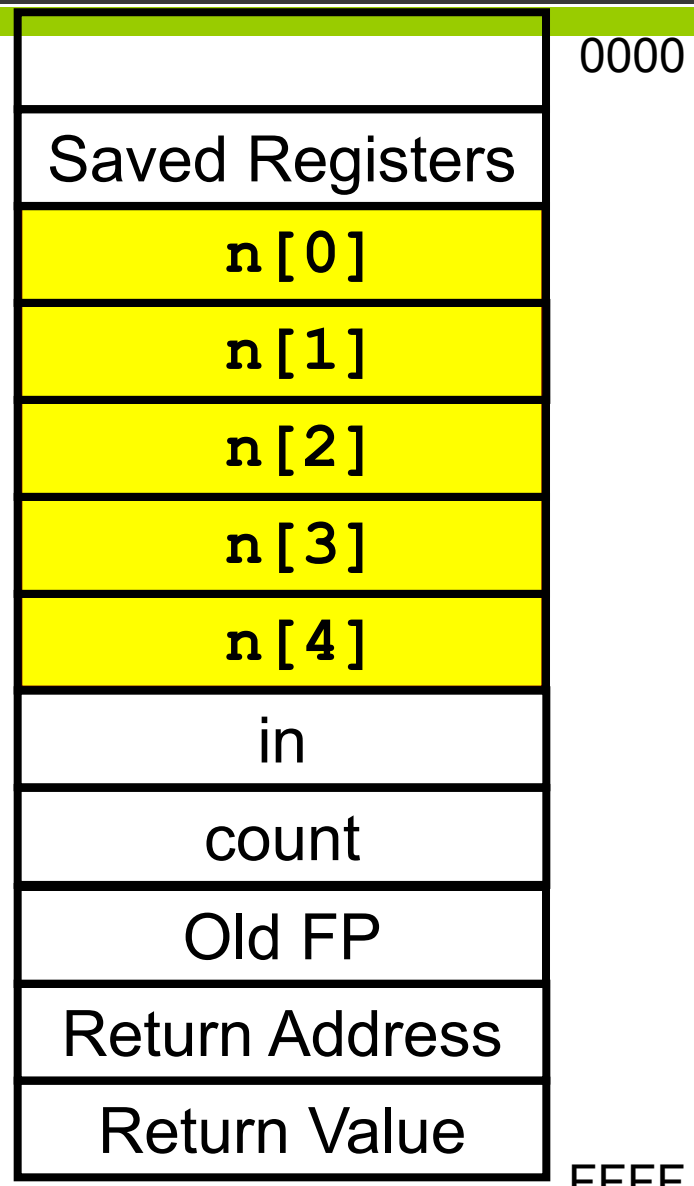
Which Which Way Does a Stack Array Grow?

```
W = smash();

int smash()
{
  int count = 0;
  int in;
  int n[5];
  do {
   printf("Enter max of 5 vals");
   printf("Enter -99 to quit");
   scanf("%d", &in);
   if (in != -99) {
      n[count] = in;
       count++; }
  while (in != -99);
  // do stuff
}
```



It Does Appear Backwards If the Stack Grows Down

```
W = smash();

int smash()
{
  int count = 0;
  int in;
  int n[5];
  do {
   printf("Enter max of 5 vals");
   printf("Enter -99 to quit");
   scanf("%d", &in);
   if (in != -99) {
      n[count] = in;
       count++; }
  while(in != -99);
  // do stuff
}
```

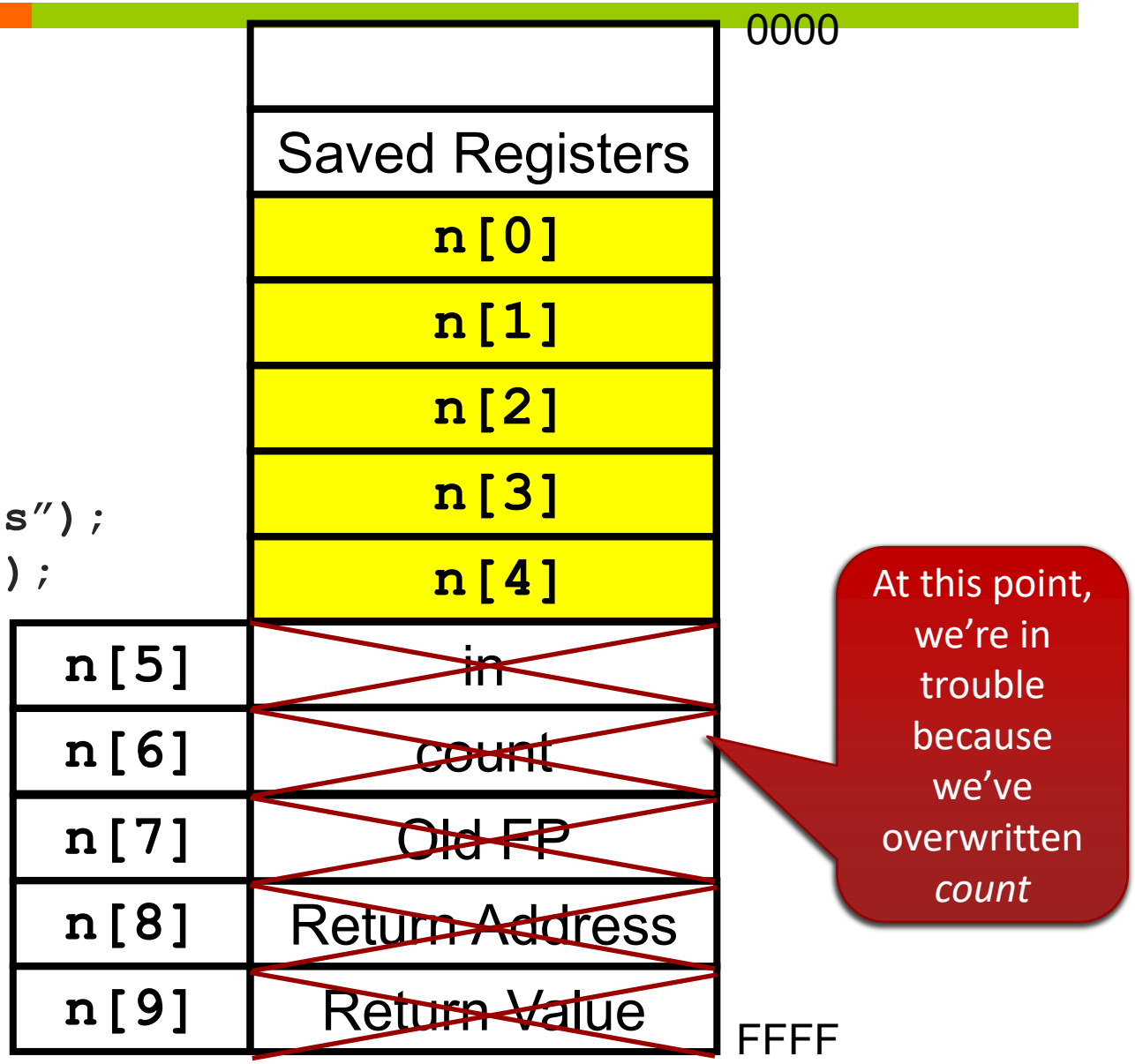


What if…

- We enter more than 5 values anyway?

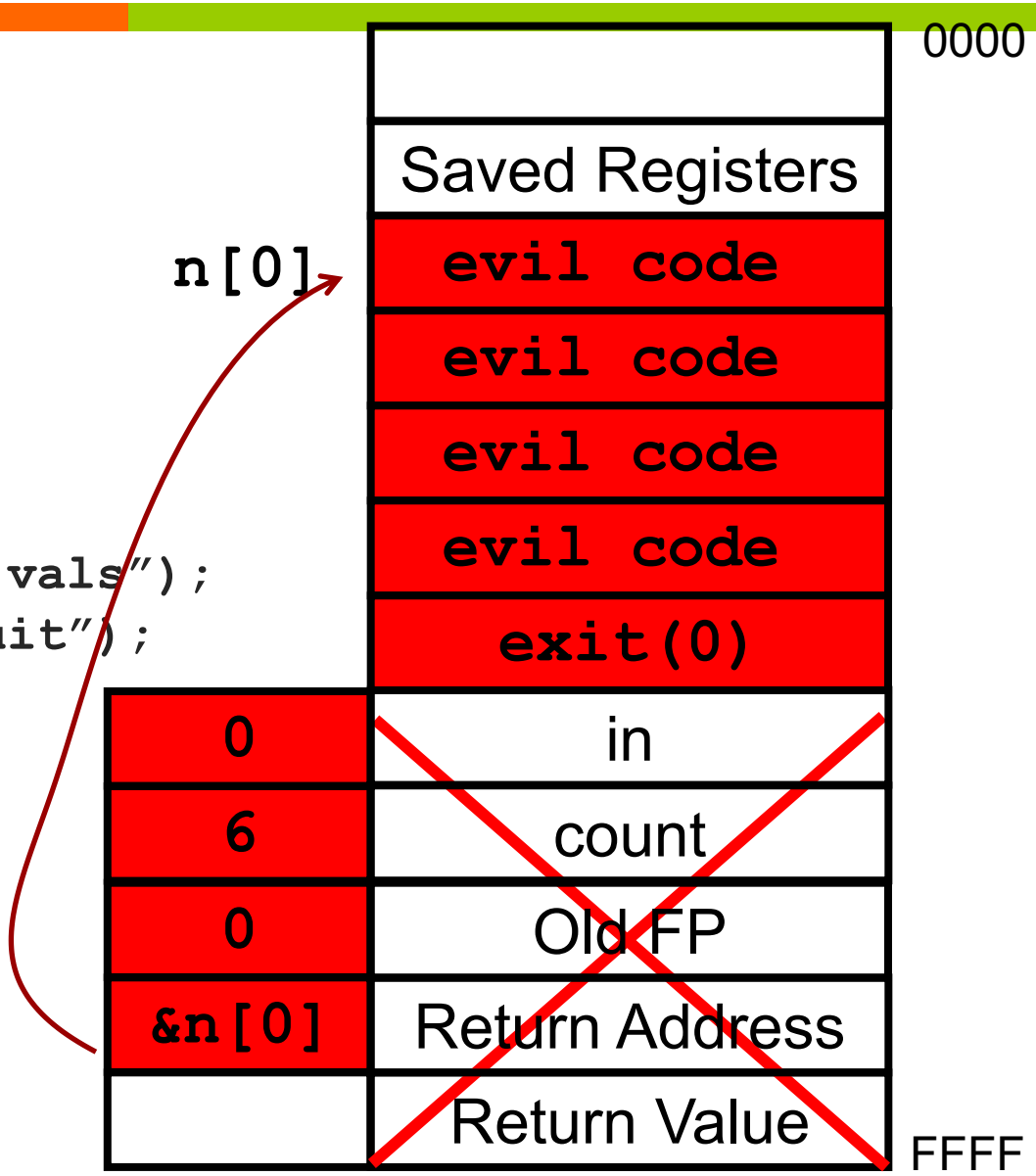
Entering More Than Five Values….

```
W = smash();

int smash()
{
   int count = 0;
   int in;
   int n[5];
   do {
    printf("Enter max of 5 vals");
    printf("Enter -99 to quit");
    scanf("%d", &in);
    if (in != -99) {
       n[count] = in;
        count++; }
    while (in != -99);
   // do stuff
}
```



What If We're Intentionally Exploiting This Vulnerability

```
W = smash();

int smash()
{
  int count = 0;
  int in;
  int n[5];
  do {
   printf("Enter max of 5 vals");
   printf("Enter -99 to quit");
   scanf("%d", &in);
   if (in != -99) {
      n[count] = in;
      count++; }
   while (in != -99);
  // do stuff
}
```



What Happens…

- …when our function returns?
- What might that evil code do?
- The industry term for a little interlude that can bring in other malicious code is *shellcode*
- And what if our program is running with administrator privileges?

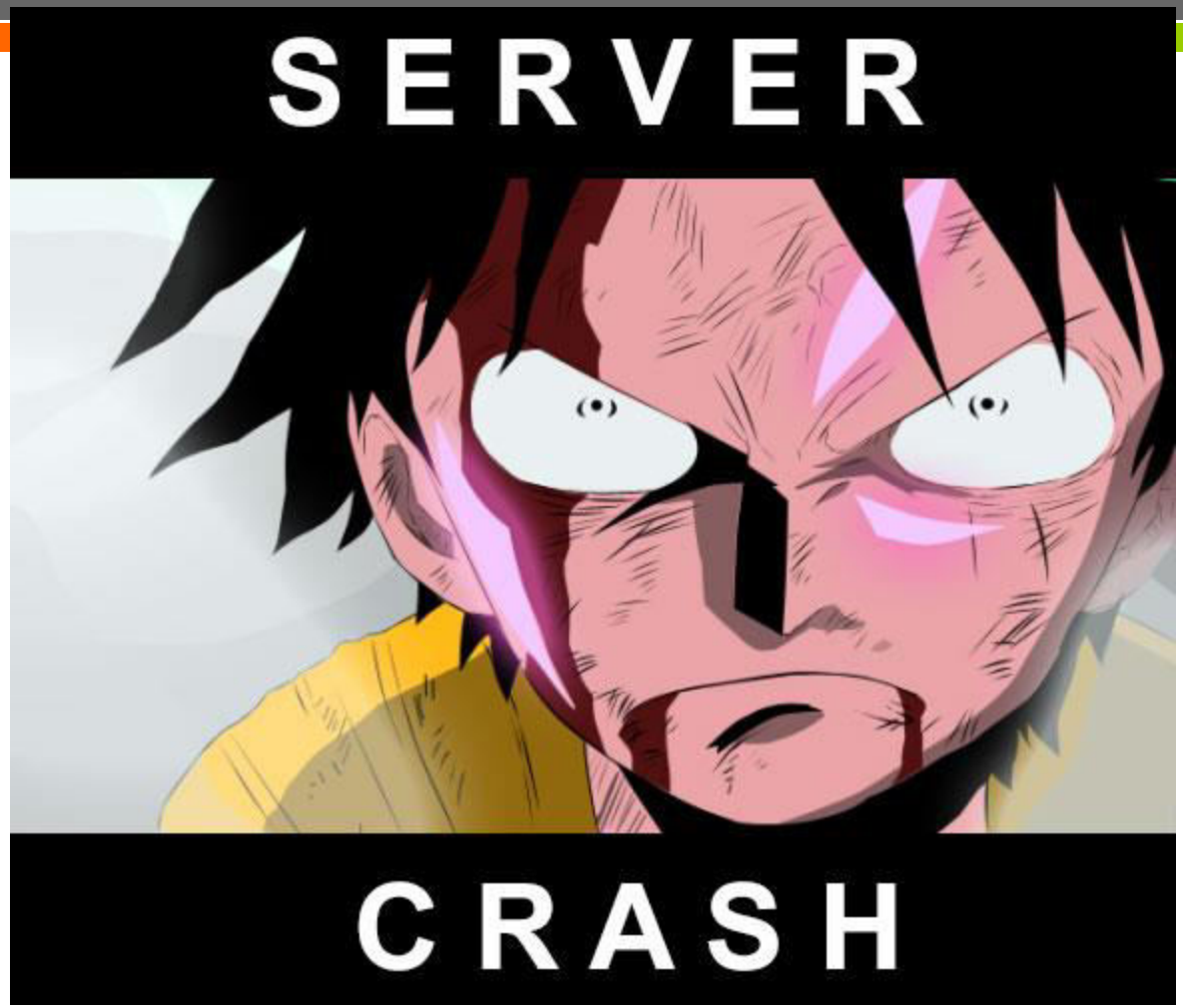
What Can Be Done?

- Bounds check!
- Non-executable stack
- Canaries – help from the compiler
  - Terminator – use '\0', -1 to make it harder to use strcpy
  - Random – memory allocator chooses a random number at startup
  - Random XOR – XOR the metadata from a block into the random number

A Real Example

- Here's a vulnerability announced 7-21-2021 for Linux. It's pretty deep, but it occurs because the code computes a 64-bit size and then passes it to functions that use a 32-bit size.
- (For completeness, Microsoft also announced a similar “privilege escalation” vulnerability in Windows 10 on the same day.)
- So in this case an integer size mismatch in the operating system code allows a "bad guy" to take over the entire system.
- https://blog.qualys.com/vulnerabilities-threat-research/2021/07/20/sequoia-a-local-privilege-escalation-vulnerability-in-linuxs-filesystem-layer-cve-2021-33909

What If Somebody Doesn't Like You?


SERVER
CRASH

Questions?