Function Pointers, Vulnerabilities

CS 2130: Computer Systems and Organization 1
December 5, 2025

Announcements

- Homework 10 due Monday
- Quiz 10 open today, due Sunday on Gradescope
- Final exam: 7-9pm Dec 12, Wilson 301 (different room!)
  - Cumulative, see practice tests
  - Exam conflict form in email
- Remember to fill out course evaluations
  - 5 pts extra credit on final exam if completed by **Wednesday, Dec 10 at 5pm**!

Using `write`

pig latin example continued

Example Code

Consider the following code:

```
void apply(double (*f)(double), double *l, unsigned n) {
  for(int i=0; i<n; i+=1)
    l[i] = f(l[i]);
}
```

What are its parameters? How do we call it?

Example Code

```
int main() {
  double vals[5] = { M_PI, M_E, 2130, 1, 0 };
  for(int i=0; i<5; i+=1) printf("%f\t", vals[i]);
  puts("");
  apply(sqrt, vals, 5);
  for(int i=0; i<5; i+=1) printf("%f\t", vals[i]);
  puts("");
  apply(sin, vals, 5);
  for(int i=0; i<5; i+=1) printf("%f\t", vals[i]);
  puts("");
  apply(cos, vals, 5);
  for(int i=0; i<5; i+=1) printf("%f\t", vals[i]);
  puts("");
}
```

Function Pointers

```
void apply(double (*f)(double), double *l, unsigned n) {
  for(int i=0; i<n; i+=1)
    l[i] = f(l[i]);
}
```

`double (*f)(double)` means:

- `*f` – `f` is a pointer
- `(double)` – with a single `double` argument
- `double` – that returns a `double`
- `( )` – to make it parse as `*f` instead of `double *`

Function Pointers

```
const char *(*fv)(const char *) = findVowel;
```

A **function pointer** is a pointer that references code

- In assembly, the address of the function is just a label
  - Follow calling conventions
  - Push return address
  - Jump to that label
- C tries to hide that with this function pointer syntax
- Be aware of operator precedence!

Vulnerabilities…

…and when to report them

Memory

Common Memory Problems (from reading)

- Memory leak
- Uninitialized memory
- Accidental cast-to-pointer
- Wrong use of ‘sizeof‘
- Unary operator precedence mistakes
- Use after free
- Stack buffer overflow
- Heap buffer overflow
- Global buffer overflow
- Use after return
- Uninitialized pointer
- Use after scope

Vulnerabilities

Vulnerabilities

Vulnerability: a program for which something like this could happen (security holes)

- Ex: stack buffer overflow possibility
- Not necessarily malicious (like when we talked about backdoors)

Exploit: a way to use a vulnerability or backdoor that has been created

- Ex: the magic long word to type into our program

Vulnerabilities

Warning

Anytime you can modify memory the programmer did not expect you to be able to modify, there's something you can do to give yourself power or rights the programmer didn't mean to give you

Memory

Common Memory Problems (from reading)

- Memory leak
- Uninitialized memory
- Accidental cast-to-pointer
- Wrong use of ‘sizeof‘
- Unary operator precedence mistakes
- Use after free
- Stack buffer overflow
- Heap buffer overflow
- Global buffer overflow
- Use after return
- Uninitialized pointer
- Use after scope

Vulnerabilities

What should you do when you find a vulnerability?

Good Practices

Good practices when finding a vulnerability:

1. Tell the owner
2. Wait (a reasonable amount of time for a fix)
3. Publish