Leaks and Array out of bounds

Stuart Nevans Locke

Overview

- Leaks
  - Common leak targets
- Arrays
  - Char[] and leaks
  - Array Out of Bounds
    - Relative OOB
    - Absolute accesses

Leaks

- Often the first thing you need to do when crafting an exploit
  - Pointer leaks used to defeat ASLR
  - Libc is (practically) always ASLRed
    - We want to call system(“/bin/sh”)
- If PIE is enabled, you need a leak before you fully overwrite anything
  - Partial Overwrites still exist
- We might also want to leak stack canary
  - Assuming we have stack overflow

Leaks

- So what are some leak targets?
  - Places that have values we might want to leak
    - Stack
      - What can we get from a stack leak?
    - GOT
      - What can we get from a GOT leak?
    - Heap
      - I promise, we can leak stuff from there too
      - Talk about next semester

Leaks

- Leak Primitive
  - Often used in more complicated exploits
  - Two types:
    - Relative leak
      - Takes an **offset** to leak, leaks at least one byte
    - Absolute leak
      - Takes a **pointer** to leak, leaks at least one byte
- I’ll show an example of both later

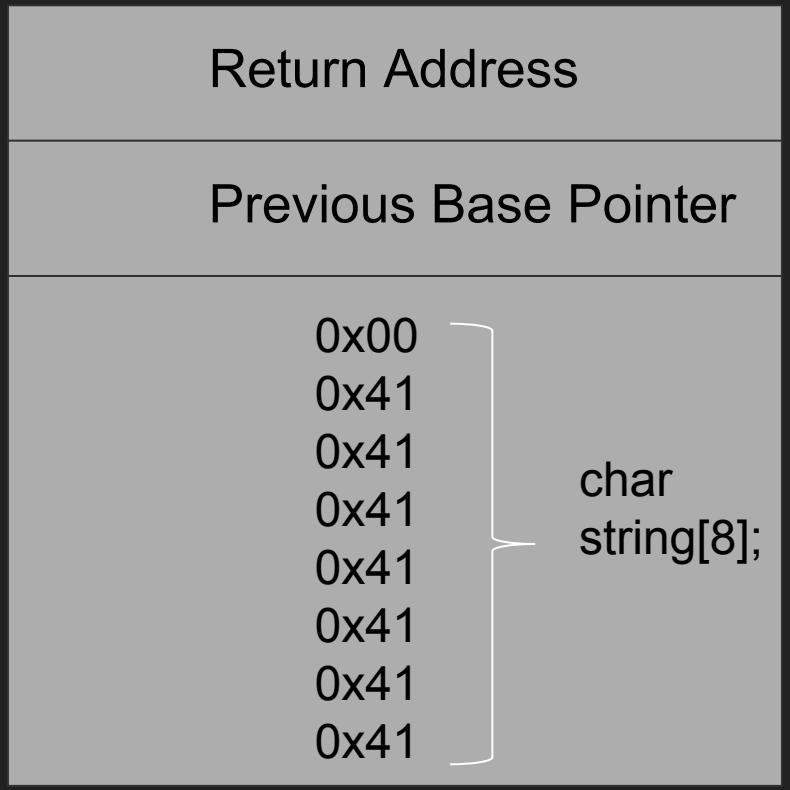
Arrays

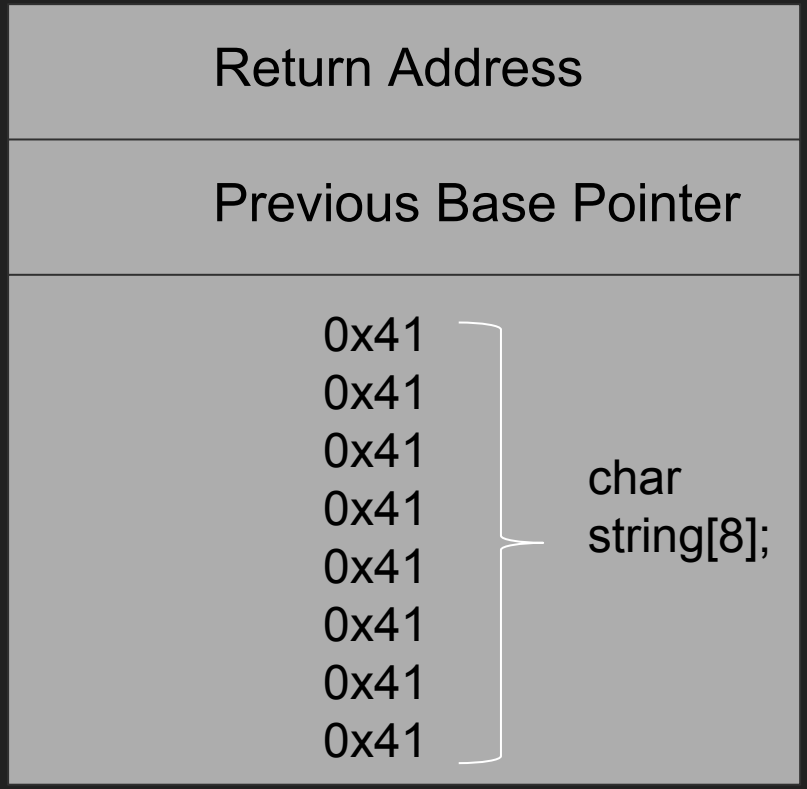
- In C, as we know, arrays have no set size
- Accessing arrays is actually syntactic sugar
  - arr[offset]
  - *(arr+offset)
  - offset[arr]

Char[] and leaks

- char string[16] = “Hello\n”; printf(string);
  - How does c know where strings end?
    - Null terminators
      - Every string ends with a null byte
      - In a 16 byte string, there are really only 15 bytes of usable space
- How to abuse this
  - Write 16 bytes to a 16 byte string buffer
    - Then print it
      - C will continue printing characters until a null byte

Char[] Leak


Return Address
Previous Base Pointer
0x00
0x41
0x41
0x41
0x41
0x41
0x41
0x41
char
string[8];


Return Address
Previous Base Pointer
0x41
0x41
0x41
0x41
0x41
0x41
0x41
0x41
char
string[8];

Array out of Bounds

- Array out of Bounds (OOB) are conceptually very simple
  - You don’t bounds check your index in an array
- arr[idx] = data;
  - Write using Array OOB
- puts(arr[idx]);
  - Leak using Array OOB

Relative OOB

- int array[32]; printf(“%i\n”, array[idx]);
  - If we make idx 32, we have a relative OOB
    - Meaning we read 4 bytes past the bounds of the array
- Allow us to do Array OOB relative to the location of the variable
  - You can consider a buffer overflow a relative out of bounds write
- Can only look at data “near us” in memory

Absolute Read/Write

- Absolute Access
  - Can read/write to absolute addresses
    - Meaning you could give a GOT address and leak/write to it
- Turning Relative OOB into Absolute
  - Depends on the section of memory it is in
    - .data or .bss
      - Stored at a static address. Just subtract the known address.
    - Stack
      - Leak a stack pointer from the stack. (Base Pointer is a great way to do this.)
      - Subtract the leaked stack pointer.

Questions?

We have demos

If interested, I could also go through my process of what I do when I first get a binary