revision of **arrays**

8 weeks of revision to learn what an **array** is

This lecture

Going to be revision of every week

If you have **any** questions about **any**thing i say

Stop me and ask…

If you dont stop me and ask when you are confused don’t blame me **when** you fail the exam

Get ready to watch a lecture irl at 1.5x speed

But first….

- myexperience

Week 0 - **Arrays**

- I’ll give you a short and a long answer to this question.
-
- **Short Answer:**
  - An array can be defined as an **ordered** collection of **items** indexed by **contiguous integers**.
- **Long answer:**
- **Arrays** a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Week 1 - **Arrays** in C

Didnt do anything

<replace this with a talk on which Tooling in exam you can use>

Week 2 - buffer overflows into **arrays**

- Dangerous functions
  - gets fgets strcpy
- Using cyclic or gdb to find distance to ret addr
- Overwrite important variables/structures or return address

Week 2 - stack canaries at end of **arrays**

- Some magic number pulled in from libc
- Randomised on program startup
  - Stays the same if you fork (can attack web servers)
- 32 bit have two options
  - If local, brute force is an option
  - Need a leak
- 64 bit.. Can’t bruteforce
  - Need a leak
- Always ends in null byte.. (why?)

Week 2 - Reverse engineering **arrays** in x86

- CDECL calling convention
- Function prologue/epilogues
- Some instructions
  - Mov, lea, jumps, cmp
- REP instruction
- How does a loop look like in C vs asm?
- EBP offset vs ESP offset
- Disassemblers vs debuggers
- Recognising patterns
  - moving char vs moving int
  - Signed vs unsigned

Week 3 - Reverse engineering

- Top down vs bottom up approach
- Do we want to look at a high level and look for design flaws
- Do we want to look at individual functions and find exploits
    - Look at where input is/ where it goes
- Strace + ltrace
- How do you recognise datastuctures
    - Arrays
    - Structs
    - Pointers
    - Ints
    - Chars

Week 3 - Shellcoding **arrays**

- What does shellcode do?
    - Mov eax, x; int 0x80
- When can we actually use shellcode?
- Why would we need a NOPsled?
- Egghunter? Syscall proxy?
- Bypassing filters
    - Learn to do this ***hint***
    - What happens if we can’t have this in our payload
        - Newlines
        - Certain bytes
        - Ascii only payload??

Week 4 - Format Strings to create dynamic **arrays**

- Rarely found in wild because GREP exists
- Can be used to do two main things
    - Leak sensitive data
    - Write to places
    - Read/Write primitive
- What is the $
- Step 1) Find location of buffer on stack…
    - Helps us craft arbritrary pointers
- Step 2) Leak / Write data to there
- What does %x do?
- What does %s do?
- What does %n do?
    - What does hhn do? And why do we care?
- What can we overwrite? (got/function pointers/return address??)

Week 5 - Source code auditing to find **arrays**

- Bad API usage
  - Think memset, strncpy(dst, src, strlen(src))
  - Format strings
  - Gets
  - Leaking stuff with strncpy (it doesnt set a NULL byte)
- Heap
  - UAF/Double Free / Custom malloc implementations?
- Logic bugs
- Integer overflows / Underflows
  - Can lead to buffer overflows in the future,
  - Or incorrect program state
- Type conversions
  - Converting between char and an int
  - Signed to unsigned
  - Pointers to float… etc

Week 5 - Source code auditing

- Using sizeof incorrectly
- Pointer arithmetic, char* vs int* and ++
- Race conditions
  - Not using locks
- **A big one is forgetting early exits/returns on errors**
- Similar to RE, top down vs bottom up approach

Week 5 - Fuzzers hidden in **arrays**

- Probably won’t be in the exam
- While we are here..
  - How are assignments going?
    - Don’t forget about them
- Something awesomes for free bonus marks?
- Do something unique

Any fuzzzer questions?

Before we go into the later topics

Any questions?

ROP/HEAP is just a mixture of the previous weeks content

Week 7 - ROP chaining **arrays** together

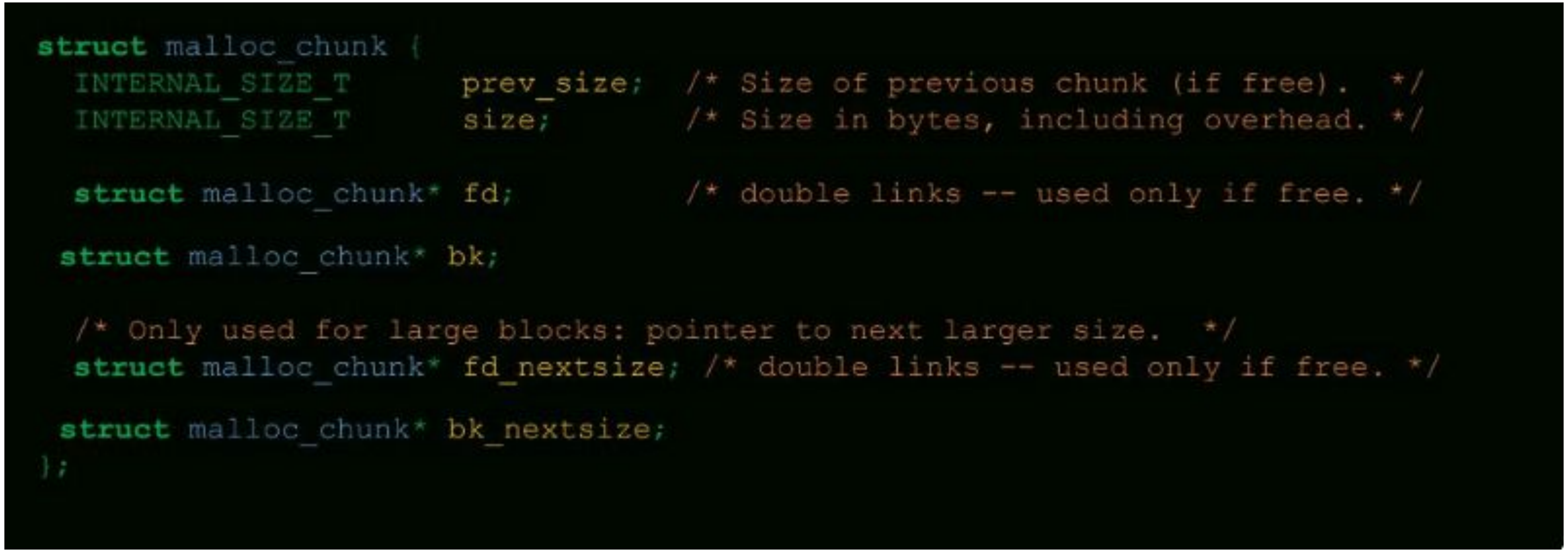
- Why do we use ROP?
- Ret2code vs ret2libc vs pure ROP
- What can we do by Chaining functions
  - Using mprotect to get working shellcode
- Pop pop ret
- What happens if you can’t find any good gadgets??
  - Leak libc
- If NX and no win functions
  - Will either be ret2libc or ROP
- Will 100% be in final exam
- Stack pivots
  - Very useful
  - ropper -f file --stack-pivot

Week 7 - ROP

- How to get a leak if you have ROP
  - puts(puts)
  - puts@plt (puts@GOT)
  - puts(*puts)
- Retsled vs Nopsled
  - Why would we need this?
- More on pivoting..
  - Xchg instruction
  - Add esp, …
  - Ret <xxx> instruction
-

Week 8 - HEAPs of **arrays**

- What is the role of malloc/free
- The malloc chunk is important to understand
- Lower 3 bits of size representing things
  - Why are chunks multiples of 8



```
struct malloc_chunk {
  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
 struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
 struct malloc_chunk* bk_nextsize;
};
```

Week 8 - HEAP exploitation in the **array**

- Difference between tcache fastbins smallbins unsorted bins???
  - Size
  - Coallecse?
  - Speed?
  - Security?
- Usually we group together multiple bugs to exploit a program
- Use after free
  - Either read or write after free.. Or both
  - If read what can we do?
  - If write what can we do?
- Double free
  - How is this useful
  - Overlapping chunks
- Forging chunks. Why? What can we do?

Week 8 - HEAP - **H**elp **A**ll **Arrays P**ermafrost

- Heap spraying?? Sometimes useful.. Probably won't be done in this course
- One_gadget
- What if one_gadget doesn't work
  - HEAP into ROP
- Use a stack pivot to get a ROP chain going
- If you have a small buffer on stack, but can't overflow
  - Add esp, 0x30
  - Now esp points into your buffer
  - Roproprop
- Smallbin/Largebin use in exploitation (Pre-tcache or V-Large size chunks)

Week 9 - Revision of **arrays**

- Tooling is important
- Speed is the most important thing for this exam
- Challenges from week 2-6 should be solvable in < 30 min by now
    - If they aren’t go do them over and over
        - Until they are
    - Maybe not the source code or RE ones
        - But the exploitation challenges should mostly be super easy right now
- This weeks wargames are harder ROP / HEAP
- If you can master ROP and HEAP chals, the exam will be **trivial**
- Lab tomorrow will be on harder rop

What now ? - after this course

- **What binary stuff didn’t we cover?**
- Actual logic errors
  - Logic errors in compilers/browsers
  - Type confusions
- Race conditions
- Anything to do with hacking a kernel
- Automated reversing
  - Angr?
  - z3?
- Automating our exploitation
  - Angrop
- **Fuzzing**
  - **Modern day exploit research relies on fuzzing...**