CHAPTER 10 – BUFFER AND STACK OVERFLOWS

World famous worms causing buffer overflows and memory crash/access. Difference between worm and virus is the worm finds its victim and virus is passive. Code Red/Morris worm took down millions of computers worldwide by exploiting a Microsoft memory buffer overflow error. All the websites that had the vulnerability showed a skull.

Prevention techniques are known

Buffer overflow basics:

- int buffer[10];
  buffer[20] = 37;
- If an attacker knows where the OS is going to put the buffer, he can choose to overwrite what's at buffer[20] or anywhere else
- The problem with array allocation is that the program grows up and the array grows down
- Buffer overflow can cause overwriting of important memory like system code

- Caused by programming error
- Allows more data to be stored in a fixed sized buffer than should be allowed
  - Memory allocation is increasing and buffer size overall is decreasing
  - Most languages check for overloading the buffer but C does not
  - Corrupting/Overwriting/Accessing adjacent memory
  - Program may work fine the majority of the time and vulnerability may not be known
    § Boolean flag for authentication to access data could be overflowed and accessed and modified!

Programming language history:

- Machine level data is array of bytes
- Modern high level languages are not vulnerable to buffer overflows
  - Does incur overhead but needs this for buffer overflow checking
- Stack holds local data/local variables/function parameters/return addresses
- Slide on function calls and stack data
  - Rerun through stack frames/frame pointers/variables
  - What happens if Buffer overflows? Program returns to incorrect return address and a crash is likely
  - Allocating 2 bytes of an array and overflowing it with a string

Buffer Overflow:
Simple Buffer Overflow
- Knowing where an authentication flag is after a buffer allocation, an attacker can overwrite it allowing anyone to authenticate
- Read the paper “smashing the stack” to learn more
- Overload the buffer on purpose
- Return to the wrong address on the stack
- Could lead to crash or ending up at an address with sensitive data
- You want return address to hit your evil code
- Example in class shows the example program may have to be run many times before buffer overflow occurs using gdb tool
  - Most of the time you do not know where the compiler will place your malicious block of code in the heap.
  - Precede malicious code with NOP “landing pad” and insert ret multiple times

QUESTIONS:
- Would it ever be possible for a program to overflow memory out of the heap and into kernel code in a modern OS – Yes, definitely. You probably have never seen a heap overflow because it is dynamic, but it is possible to overflow. Heap allocation has a limit that you could exceed if you were trying. There are a lot of defenses against this