1) Discussion on IPsec
2) overflowing a variable is buffer overflow
   1) Depends on where the variable lives
   2) depends on the behavior of the program
   3) Affects C and C++ like languages
3) Where can a variable live where it is susceptible to BO
   1) everywhere: stack and heap
   2) Seg fault
4) memory resident
   1) normally happens on the stack (stack smashing)
   2) stack stores return addresses and local variables
   3) also uses frame pointers
   4) overflow buffer with executable statement
      1) must know the architecture
      2) C just keeps writing
      3) To pass an array in C you pass a pointer which is unsafe
5) NoOp is used to exploit
   1) noOp sled – anywhere you jump will slide to the beginning of an executable
5) Heap
   1) harder to exploit because no return addresses
   2) store function pointers which leads to problems
   3) Memory management unit can make units as non-executable
4) Things to help
   1) protect address space
   2) run things randomly
   3) store things randomly
   4) Guard pages
   5) Canaries
6) useful for crashing programs
7) bugs that are really hard to guard against
8) Morris worm used a buffer overflow exploit
9) Lots of buggy legacy code has these mistakes
10) Lots of consequences of a BO
    1) corruption of programs
11) Attacker needs to identify where buffer is stored in memory
12) C has lots of vulnerabilities but for those same reasons that make it vulnerable make the perfect language for what it does.
13) Metasploit project provided useful information on how these exploits work.
14) Target trusted system utilities
    1) This will escalate privileges
    2) Network service daemon
    3) commonly used library code.
15) Defense
1) runtime vs compile time
2) high level language that does range checks
3) Safe coding techniques
4) add more code to detect if a problem has occurred
5) Stackshield
6) Address space randomization
16) Replacement Stack Frame attack
   1) off-by-one attack
17) Return to System Call
18) Heap overflow
   1) not storing code that can be executed so store function pointers
   2)