EECS 388 Discussion

Homework 4 & Buffer Overflows Part 2

Client Puzzles

- **Number of Computations**
  - 2^n for the client but only 1 for the server
- **Negating the Attacker's Advantage**
  - When n=6: $2^6 = 64$ computations for client
- **Adjusting Client Puzzles**
  - Scale n with the system load

Low Orbit Ion Cannon

- **How does the attack work?**
  - Repeatedly send HTTP GET requests
- **How does “Hive Mind mode” work?**
  - Executes commands typed in an IRC channel
- **Defenses without client puzzles?**
  - Look for irregular traffic patterns

Operation Payback

- **Operation Payback**
  - A DDoS attack on pro-copyright industry websites
- **Christopher Wayne Cooper**
  - Part of Operation Payback
  - Charged with conspiracy and intentional damage to a protected computer
- **LOIC Hive Mind vs Political Protest**
  - Thoughts?

Buffer Overflow Part 2

- **Important registers: eip, ebp and esp**
  - What was the purpose of each?
- **Remember useful GDB commands**
  - disassemble
  - x
  - info reg
  - ni
  - si

x86 Calling Conventions

- **Function Call:**
  - push the eip
  - push the ebp
  - mov %esp, %ebp
  - sub <CONSTANT>, %esp
- **Function Return:**
  - leave
    - mov %ebp, %esp
    - pop %ebp
  - ret
    - Pop address off the stack and jump to it

Overwriting a Return Addr

- Given what we know about how function calls work in x86, how can we change the return address of a function in order to call a new function?
  - To call a function, you need to load its start address into the eip.
  - How do we find the start address of the function we want to call?
  - How do we find the original return address?
  - How do we replace it with the new address?
    - Related: How can we see where the buffer is?

Shellcode

- Using what we know about buffer overflows, how can we make a vulnerable program execute forever?
- We toyed with changing the return address to another functions, but are there other possibilities?
- How can we hijack control of a program to perform a specific action?