LINKING, LOADING, & ATTACKS

- COMPILER TAKES HIGH-LEVEL LANGUAGE → OBJECT CODE

- LINKER BRINGS OBJECT FILES TOGETHER TO RESOLVE REFS. (TO SYMBOLS)

TOOLCHAIN

PREPROCESSOR → COMPILER → [ASSEMBLER] → LINKER → OS
(cpp) (gcc) (gas) (ld) (Linux

- gcc IS ACTUALLY A COMPILER "SUITE." THESE THINGS (MOSTLY) INVOKED FOR YOU

LIBRARIES

- PIECES OF CODE COMPILED "SOMEWHERE ELSE"
- UTILITY FUNCTIONS THAT ARE USEFUL IN OTHER CONTEXTS THAN YOUR PROGRAM!
- E.g. MATH, COMPRESSION, PHYSICS, SIG. PROCESSING, DATA ANALYSIS, STATS, PARSING, GAME DEV. ETC.

How does it work now?

- GIVE ME YOUR CODE (OR OBJECT FILE) AND I LINK WITH MY PROG. (SYMBOL RESOLUTION)

- ALSO OTHER LIBRARIES!

- THIS IS STATIC LINKING. EVERYTHING IS COMBINED INTO ONE BIG EXECUTABLE IMAGE.
- USES A LOT OF SPACE! (DISK & MEMORY!)
- WHAT IF TWO PROGRAMS USE THE SAME LIBRARY?
- WHAT HAPPENS IF THE LIBRARY IS UPDATED?
  - RE-LINK! TEDIOUS. HARD TO MAINTAIN SOFTWARE!

DYNAMIC LIBRARIES / SHARED LIBS.
& LINKED!

- LIBRARIES LOADED AT RUNTIME.
- ALL LIBS REQUIRED ARE KEPT TRACK OF IN THE EXECUTABLE. DYNAMIC LINKER + LOADER GOES OFF AND FINDS THEM IN THE FILESYSTEM AT LOAD TIME.
- SYMBOLS RESOLVED LAZILY! (NOT UNTIL WE USE THEM)
- WHAT MAKES THEM SHARED? THE OS. IF 2 PROGS USES THE SAME LIB, OS ONLY LOADS INTO MEM ONCE!

THIS IS DYNAMIC LINKING + LOADING

- WHEN WE STATICALLY LINK, LIBS CAN REQUEST TO BE LOADED AT A PARTICULAR LOCATION (RECALL .ORIG)

- WHAT IF 2 LIBS REQUEST SAME LOCATION?

- SAME PROBLEM EXISTS FOR DYNAMIC LIBS! ONLY THE LOADER NOW HAS THE PROBS.

SOLUTION: POSITION INDEPENDENT CODE!

- ~~ADDRESSES~~ DON'T USE ABSOLUTE ADDRESSING, REQUIRES FIXUPS FOR EVERY MEM REF!

- USE PC-RELATIVE ADDRESSING EVERYWHERE

- WE CAN NOW LOAD LIBS ANYWHERE, WILL STILL WORK. EASILY RESOLVES CONFLICTS.

- MOST SHARED LIBS ARE COMPILED THIS WAY (gcc -fPIC)

THE TRAGIC AND (SEMI-) UBIQUITOUS BUFFER OVERFLOW

```
int get_user_data (int user_id, char * user_name)
{
   int credit_card_num = get_cc_num (user_id);
   char name[100];
   strcpy (name, user_name);
   do_more_backend_processing (credit_card_num, name);
   return 0;
}
```

ROP ATTACKS: (TYPE OF CONTROL FLOW ATTACK)

- RETURN-ORIENTED PROGRAMMING
- BOILS DOWN TO: MANUFACTURING RETURN ADDRS.!