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

-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**

- Libraries loaded at runtime
- All libs required are kept track of in the executable. Dynamic linkers + loaders goes off and finds them in the filesystem at load time.

- Symbols resolved lazily! (not until we use them)

- **Why do we make them shared?** The OS.
  - If 2 programs 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 (e.g., .text).

- What if 2 libs request same location?

- Same problem exists for dynamic libs! Only the loader now has the pic.

Solution: Position Independent Code!

- Address/Offset: 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)
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;

R.O.I. attakcs: (type of control
flow attack)
-Return-oriented programming
- boils down to: manufacturing return address /