Return oriented programming (ROP)

Cat and mouse

- **Defense: Make stack/heap nonexecutable** to prevent injection of code
  - **Attack response: Jump/return to libc**
- **Defense: Hide the address of desired libc code** or **return address** using ASLR
  - **Attack response: Brute force search** (for 32-bit systems) or **information leak** (format string vulnerability)
- **Defense: Avoid using libc code entirely** and use code in the program text instead
  - **Attack response:** Construct needed functionality using **return oriented programming (ROP)**

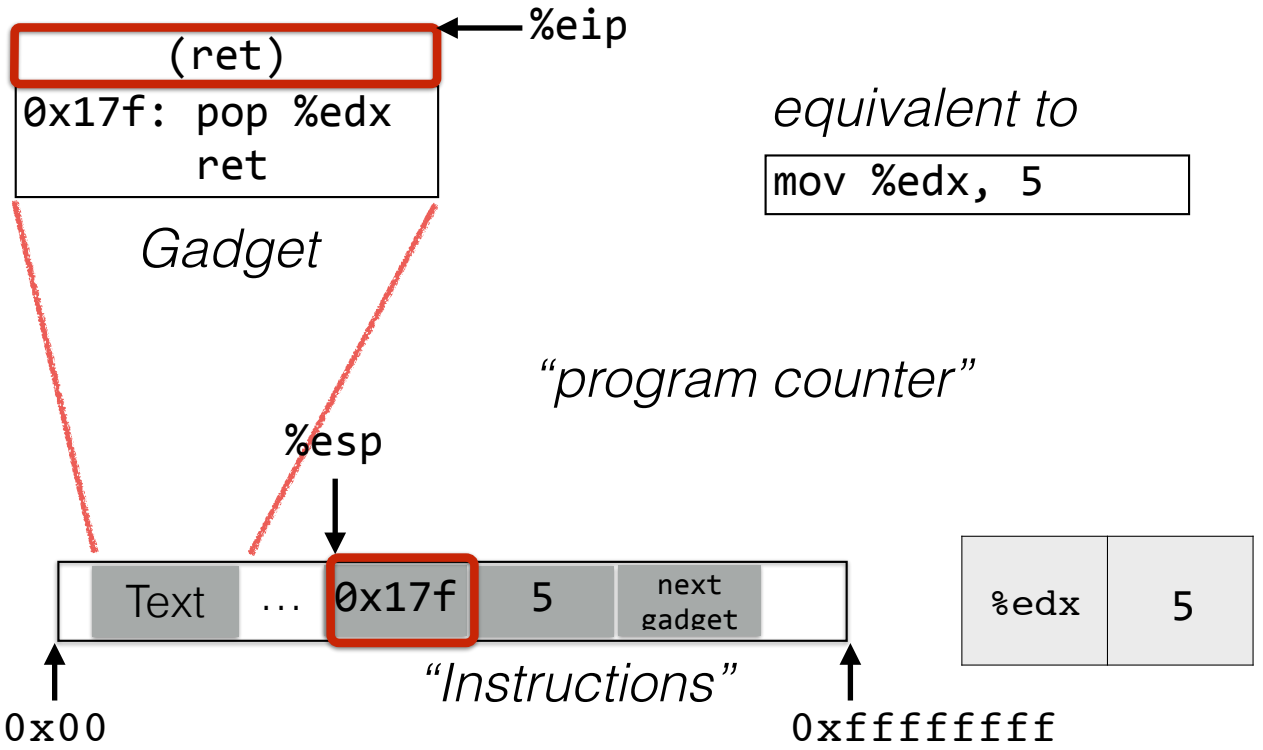
Return-oriented Programming

- Introduced by Hovav Shacham in 2007
  - *The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)*, CCS'07
- Idea: rather than use a single (libc) function to run your shellcode, **string together pieces of existing code, called *gadgets***, to do it instead
- Challenges
  - **Find the gadgets** you need
  - **String them together**

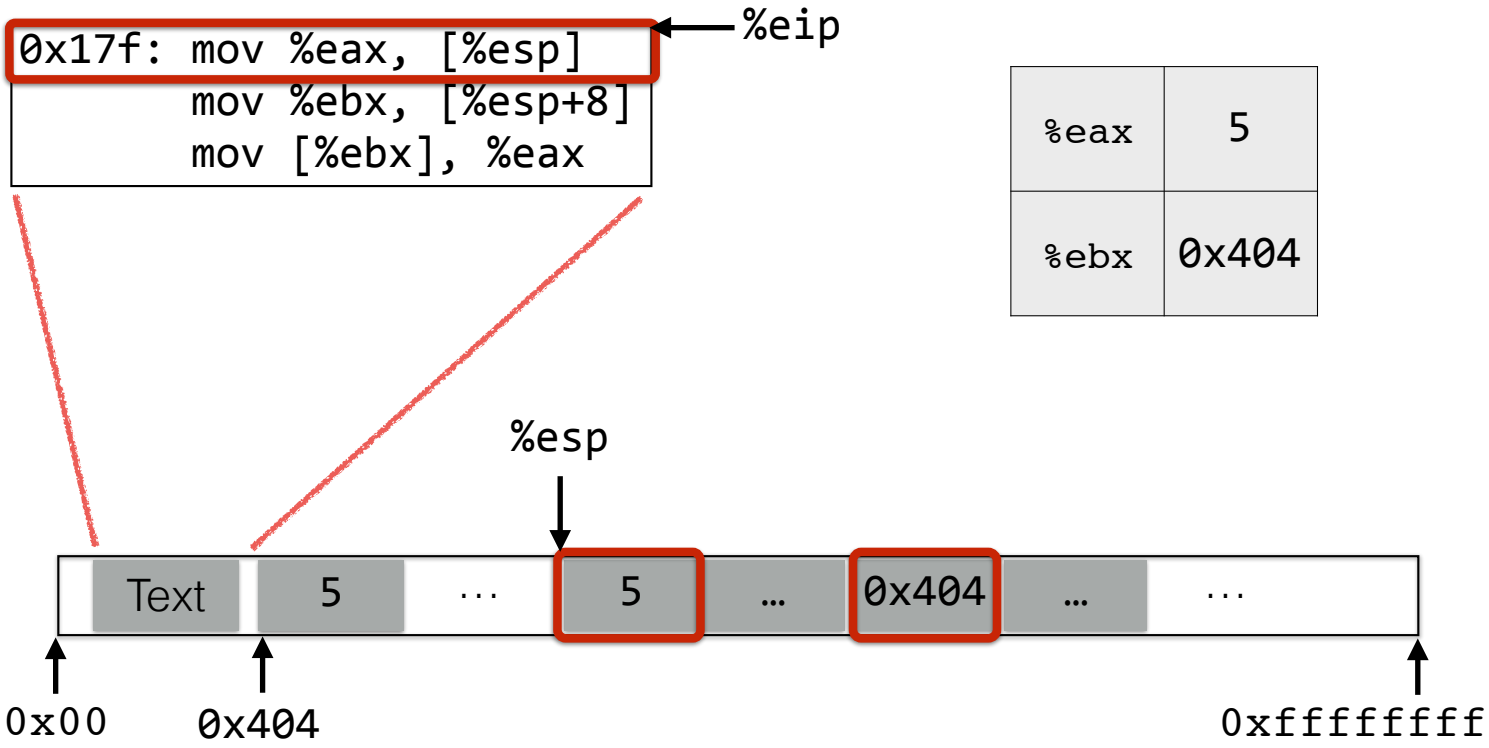
Approach

- Gadgets are instruction groups that end with **`ret`**
- Stack serves as the code
  - **`%esp`** = program counter
  - Gadgets invoked via **`ret`** instruction
  - Gadgets get their arguments via **`pop`**, etc.
    - Also on the stack

Simple example



Code sequence



Equivalent ROP sequence

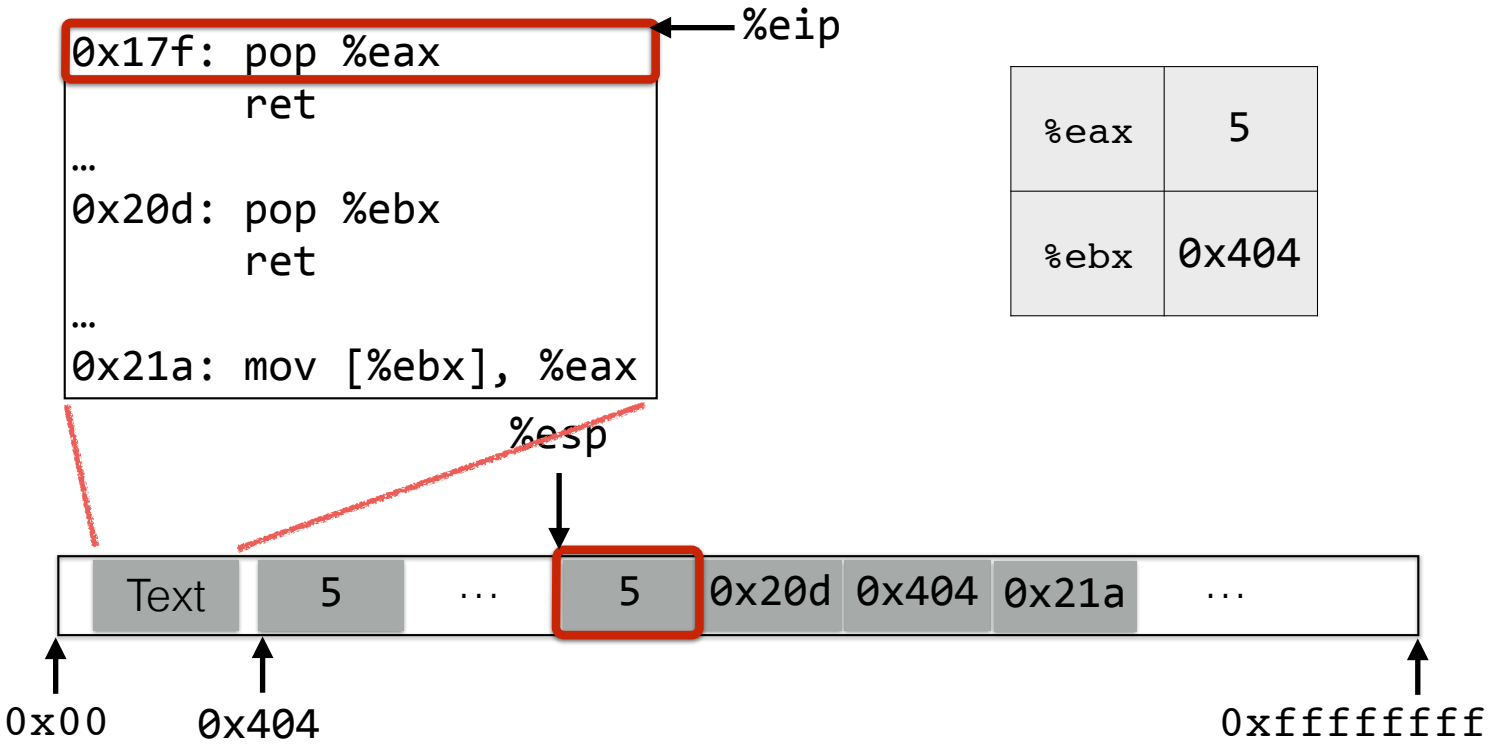


Image by Dino Dai Zovi

Whence the gadgets?

- How can we find gadgets to construct an exploit?
  - **Automate a search of the target binary for gadgets** (look for `ret` instructions, work backwards)
    - Cf. https://github.com/0vercl0k/rp
- Are there sufficient gadgets to do anything interesting?
  - Yes: Shacham found that for significant codebases (e.g., libc), **gadgets are Turing complete**
    - Especially true on x86's dense instruction set
  - Schwartz et al (USENIX Security '11) have automated gadget shellcode creation, though not needing/requiring Turing completeness

Blind ROP

- **Defense: Randomizing the location of the code** (by compiling for position independence) on a 64-bit machine makes attacks very difficult
  - Recent, published attacks are often for 32-bit versions of executables

- **Attack response**: **Blind ROP**

  If server restarts on a crash, but does not re-randomize:

  1. Read the stack to **leak canaries and a return address**
  2. Find gadgets (at run-time) to **effect call to `write`**
  3. **Dump binary to find gadgets for shellcode**

http://www.scs.stanford.edu/brop/

Defeat!

- The blind ROP team was able to **completely automatically**, only **through remote interactions**, develop a **remote code exploit for nginx**, a popular web server
  - The exploit was carried out on a 64-bit executable with full stack canaries and randomization
- Conclusion: **give an inch, and they take a mile?**
- Put another way: **Memory safety is really useful!**