CSC501
Operating Systems Principles

Protection and Security I:
Code Injection Attacks

A Quick Recap

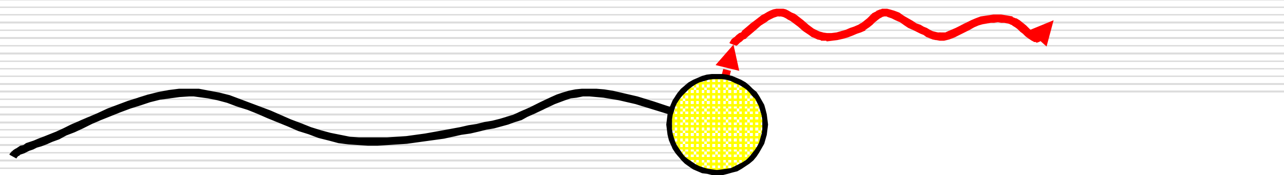
- Previous Lectures
  - Processes, Memory, I/O Devices and File Management

- Now:
  - Protection and Security:
    - Threat I: Code Injection
    - Threat II: Malware
    - Defense: Protection

Code-Injection Attacks

- What is it?
  - Wikipedia (http://en.wikipedia.org/wiki/Code_injection):

    "**Code injection** is the exploitation of a computer bug that is caused by processing invalid data. It is often used by an attacker to introduce (or "inject") code into a computer program to change the course of execution."



Code-Injection Attacks

- Why should we care?
  - Wikipedia (http://en.wikipedia.org/wiki/Code_injection):
    - "The results of a Code Injection attack can be disastrous."
  - About 60% of CERT advisories deal with unauthorized control information tampering
    - Buffer overflow is one of the most common ways for code injection attacks
    - Other bugs can also divert control

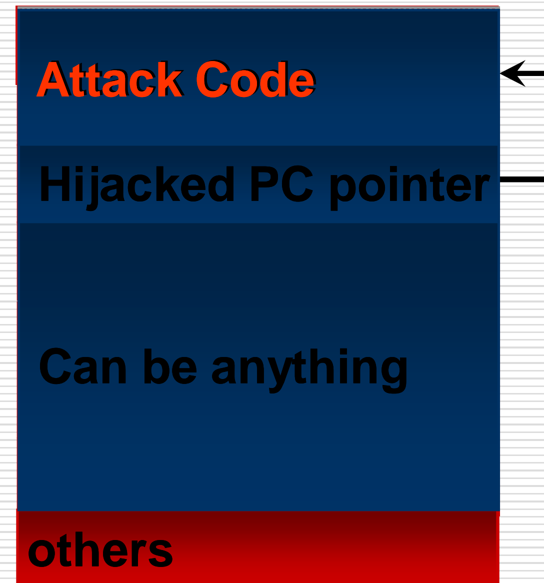
Understanding Code-Injection Attacks

- Suggested reading:
  - B. Kuperman, C. Brodley, H. Ozdoganoglu, T. Vijaykumar, A. Jalote, "*Prevention and Detection of Stack Buffer Overflow Attacks*," Comm. of the ACM, November 2005.
- A buffer overflow occurs when you try to put too many bits into an allocated buffer.
- When this happens, the next contiguous chunk of memory is overwritten, such as
  - Return address, function pointer, frame pointer etc.
- Also an attack code is injected and executed, which can lead to a serious security problem.

Understanding Code-Injection Attacks

- Essential stages
  1. Inject attack code
  2. Hijack control flow
  3. Execute attack code

```
int foo (int arg1) {
    char buf[128];
    gets (buf);
}
```



- We are not talking about social engineering attacks

Stage 1: Inject Attack Code

- How?
  - Typically done by exploiting software vulnerabilities
    - Buffer overflows
    - Format-string vulnerability
    - Integer overflows
    - Double-free vulnerability
    - SQL injection vulnerability
    - XSS scripting vulnerability
    - …
  - Different vulnerabilities usually require different ways to exploit them

Stage 2: Hijack Control Flows

- How?
  - Mainly by overwriting control data
  - What are those control data?
    - Return address
    - Function pointers

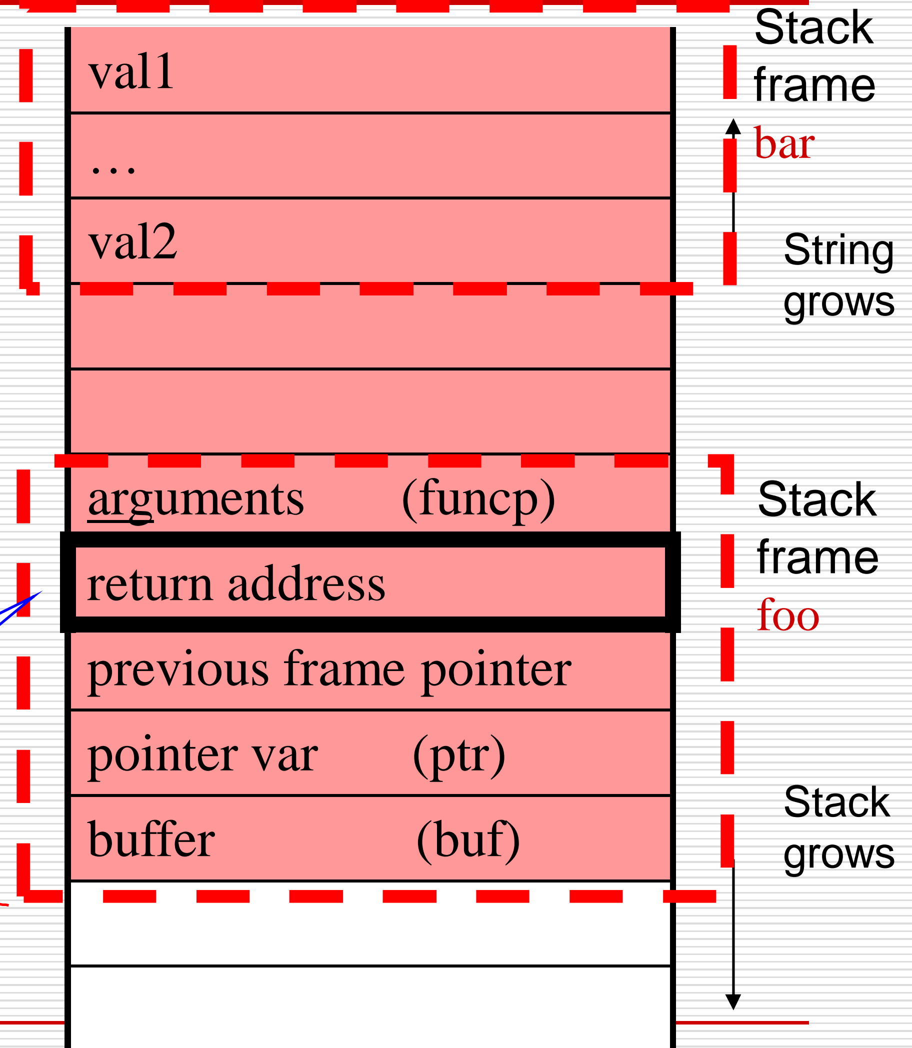
When function *foo* is called by *bar*…

```
int bar (int val1) {
    int  val2;
    foo (a_function_pointer);
}
```

Contaminated memory

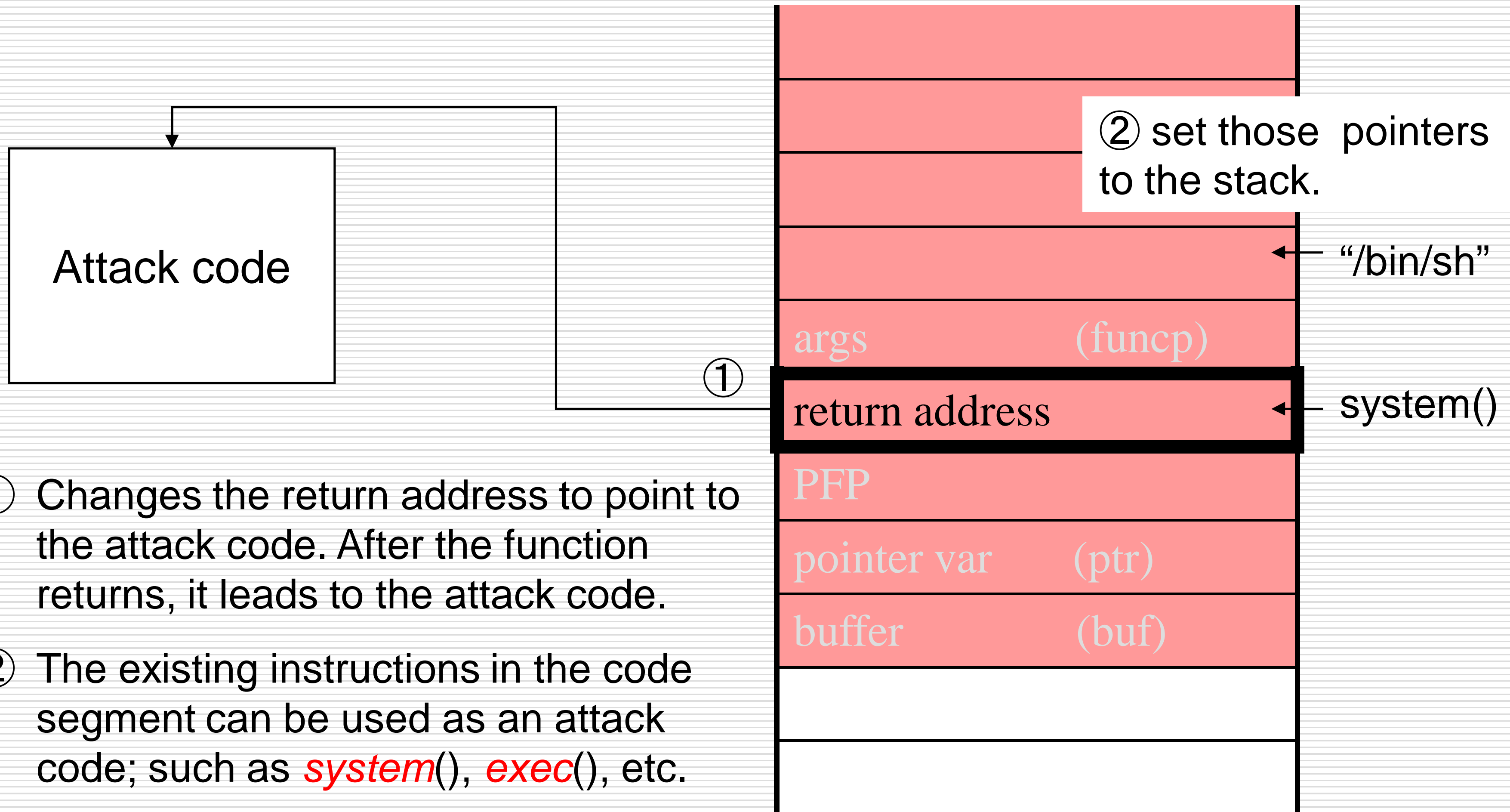
```
int foo (void (*funcp)()) {
    char* ptr = point_to_an_array;
    char buf[128];
    gets (buf);
    strncpy(ptr, buf, 8);
    (*funcp)();
}
```

Most popular target



Attack Scenario #1
--- by changing the **return address**

```
int foo (void (*funcp)()) {
    char* ptr = point_to_an_array;
    char buf[128];
    gets (buf);
    strncpy(ptr, buf, 8);
    (*funcp)();
}
```

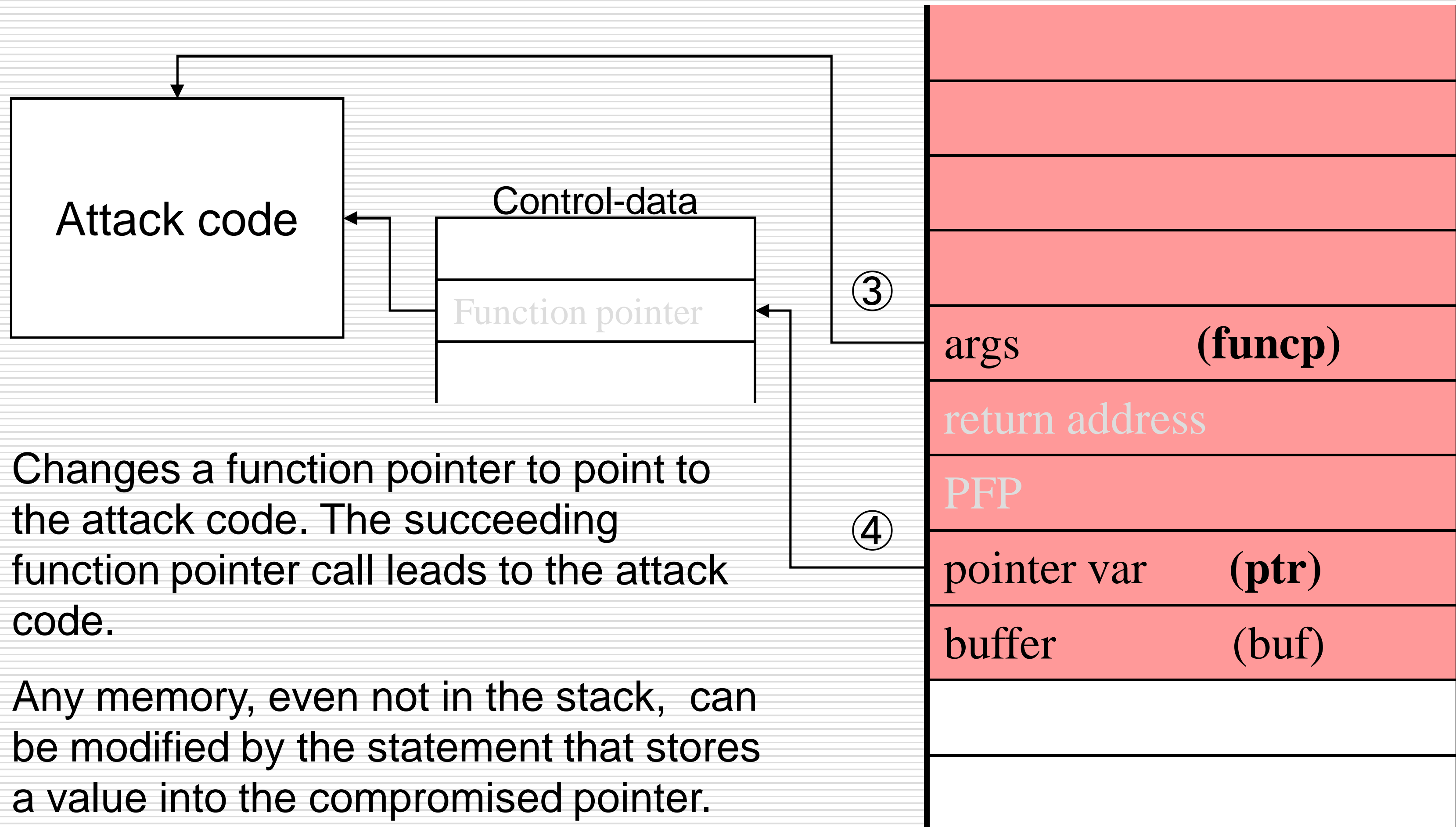


① Changes the return address to point to the attack code. After the function returns, it leads to the attack code.

② The existing instructions in the code segment can be used as an attack code; such as *system*(), *exec*(), etc.

Attack Scenario #2
--- by changing **pointer variables**

```
int foo (void (*funcp)()) {
    char* ptr = point_to_an_array;
    char buf[128];
    gets (buf);
    strncpy(ptr, buf, 8);
    (*funcp)();
}
```



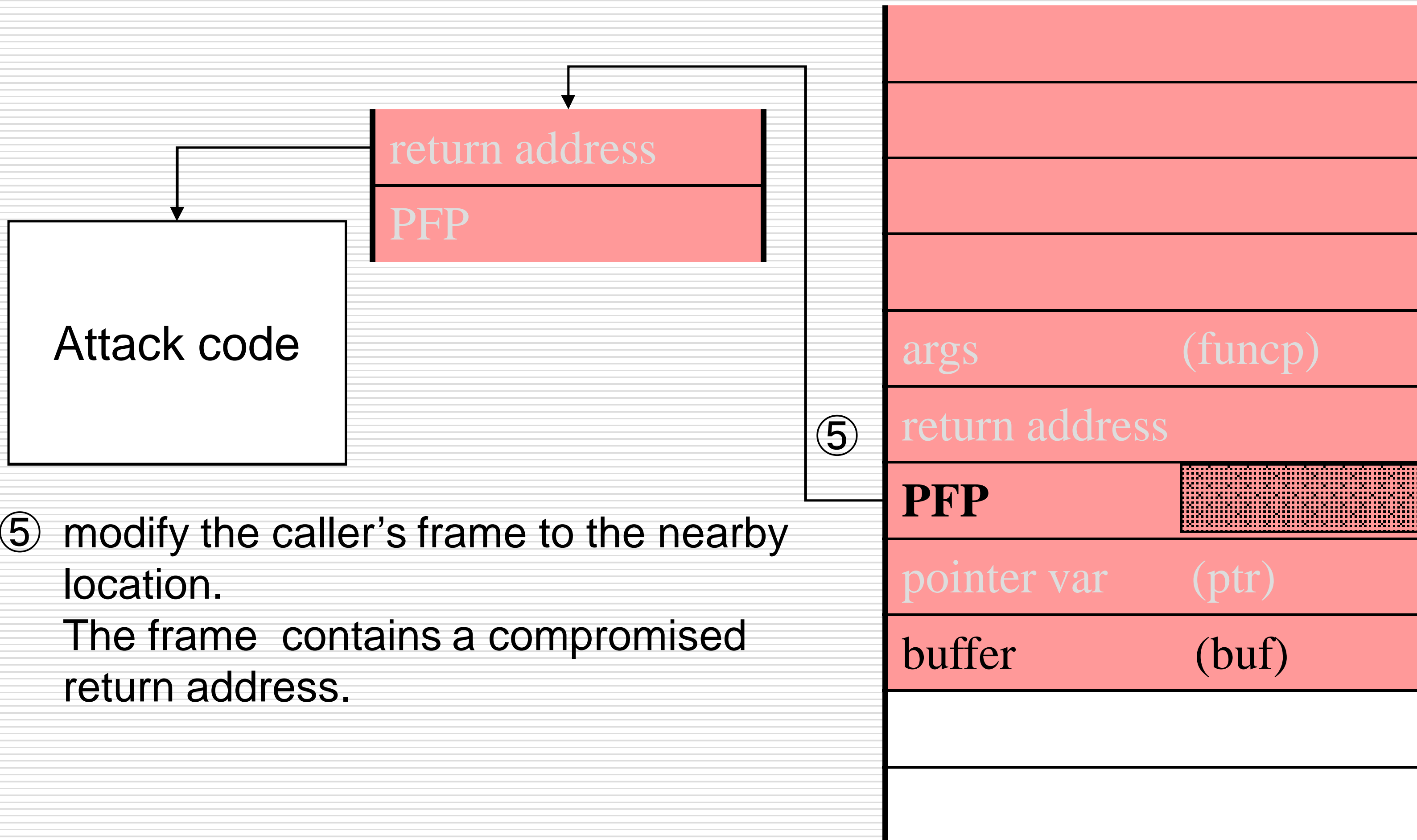
③ Changes a function pointer to point to the attack code. The succeeding function pointer call leads to the attack code.

④ Any memory, even not in the stack, can be modified by the statement that stores a value into the compromised pointer.

E.g. strncpy(ptr, buf, 8);

Attack Scenario #3 --- by changing the **previous frame pointer**

```
int foo (void (*funcp)()) {
    char* ptr = point_to_an_array;
    char buf[128];
    gets (buf);
    strncpy(ptr, buf, 8);
    (*funcp)();
}
```



⑤ modify the caller's frame to the nearby location.
The frame contains a compromised return address.

Question:

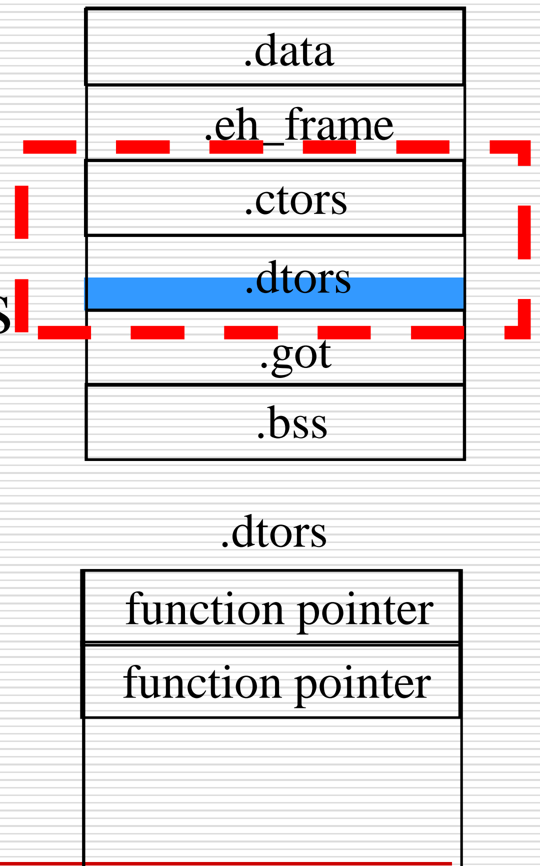
In addition to stack overflows, do we also have heap overflows, data/bss overflows?

Heap Overflow

- Heap overflows exist for many memory allocators
  - Dlmalloc (overflow, double free)
  - CSRI (overflow)
  - Quickfit (overflow)
  - Phkmalloc (overflow)
  - Boehm’s Garbage Collector (overflow, double free)

Data/BSS Overflow

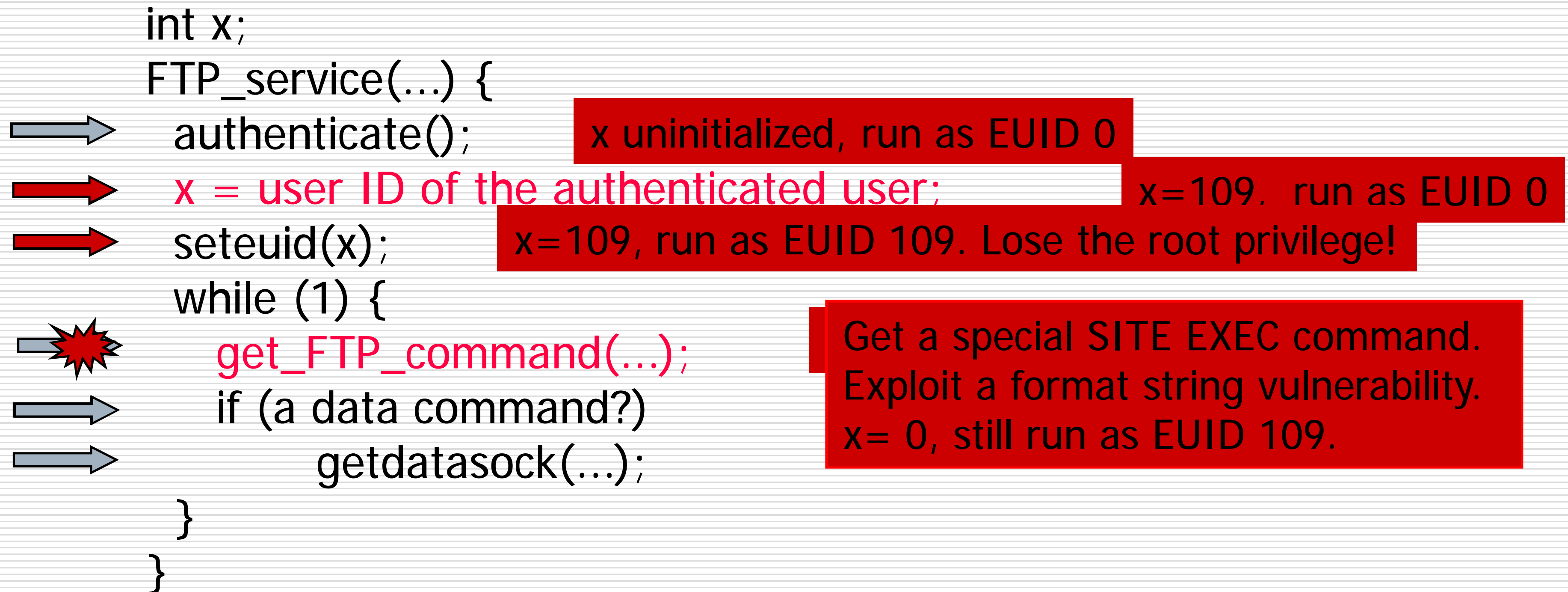
- Data contains global or static initialized data
- BSS contains global or static uninitialized data
- Stored together with other sections that control program execution
- An attacker can overflow into other sections



Stage 2: Hijack Control Flows (2)

- Control-data atttacks
  - Currently the most dominant form of memory corruption attacks [*CERT* and *Microsoft Security Bulletin*]
- Non-control-data attacks: attacks not corrupting any control data
  - i.e., attacks preserving the integrity of control flow of the victim process

Example 1: Non-Control-Data Attack against *WU-FTPD* Server (via a **format string** bug)



When return to service loop, still runs as EUID 0 (root). Allow us to upload */etc/passwd*

x=0, run as EUID 0

We can grant ourselves the root privilege!

seteuid(x);

x=0, run as EUID 0

Only corrupt an integer, not a control data attack.

Example 2: Non-Control-Data Attack against *NULL-HTTP* Server (via a **heap overflow** bug)

- Attack the configuration string of CGI-BIN path.
- Mechanism of CGI
  - Suppose server name = www.foo.com
    CGI-BIN = /usr/local/httpd/exe
  - Requested URL = http://www.foo.com/cgi-bin/bar
  - The server executes

The server gives me a root shell!
Only overwrite four characters in the CGI-BIN string.

- The attack
  - Exploit the vulnerability to overwrite CGI-BIN to /bin
  - Request URL http://www.foo.com/cgi-bin/sh
  - The server executes

Example 3: Non-Control-Data Attack against *SSH Communications* SSH Server (via an **integer overflow** bug)

```
void do_authentication(char *user, ...) {
    int auth = 0;
    ...
    while (!auth) {
   /* Get a packet from the client */
      type = packet_read();
      switch (type) {
      ...
      case SSH_CMSG_AUTH_PASSWORD:
       if (auth_password(user, password))
            auth =1;
      case ...
       }
       if (auth) break;
   }
  /* Perform session preparation. */
  do_authenticated(…);
}
```

auth = 0

auth = 0

auth = 1

Password incorrect, but *auth* = 1

auth = 1

Logged in without correct password

Stage 2: Hijack Control Flows (3)

- Control-data attack vs. non-control-data attack

Question:

Which one is easier to launch?

Question:

Which one is more powerful?

Stage 3: Execute Attack Code

- *Reliable* Execution
  - Injected code vs. existing code
  - Absolute vs. relative address dependencies
- Complexity vs. Features

Question:

What are those common features in attack code?

A Recap on Code Injection Attacks

- Stages
  - **1:** Inject attack code into buffer
  - **2:** Redirect control flow to attack code
  - **3:** Execute attack code

http://www.metasploit.com

Question:

Which stage is easier to accomplish?

File Edit View Bookmarks Widgets Feeds Chat Tools Help

Shut Down By FBI AN...

http://metasploit.com/

Google

THIS SITE

metasploit

HAS BEEN PERMANENTLY SHUT DOWN BY THE FEDERAL BUREAU OF INVESTIGATION AND U.S. DEPARTMENT OF HOMELAND SECURITY

Individuals involved in the operation and use of the Metasploit Framework are under investigation for violations related to the Computer Fraud and Abuse Act

It is unlawful to knowingly accessing a computer without authorization in order to obtain national security data, information contained in a financial record of a financial institution, or contained in a file of a consumer reporting agency on a consumer, information from any department or agency of the United States, or information from any protected computer if the conduct involves an interstate or foreign communication. Violators risk criminal prosecution under 18 U.S.C. 1030. First-time offenders will face up to ten years in federal prison, restitution, forfeiture and a fine.

Next Lecture

- Protection and Security II
  - Malware