CSE 484 / CSE M 584: Computer Security and Privacy

Software Security: Buffer Overflow Attacks

Fall 2016

Adam (Ada) Lerner
lerner@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements

- Sign the ethics form by today at 5!
- Homework 1 is due on Monday.
- Please start forming groups for lab 1
  - You can use the forum to find group members

Announcements

- TA office hours have been moved to Mondays at 4:30 (after class), in the second floor breakout.
  - Sorry for the confusion!

Security: Not Just for PCs

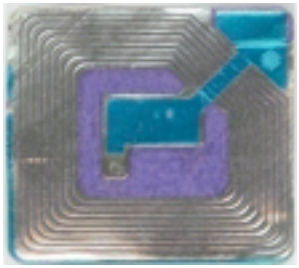
smartphones

voting machines

EEG headsets

medical devices

wearables



RFID

mobile sensing platforms

cars

game platforms

airplanes

Software Problems are Ubiquitous

Software Bug Halts F-22 Flight

Posted by kdawson on Sunday February 25, @06:35PM
from the **dare-you-to-cross-this-line** dept.

mgh02114 writes

> "The new US stealth fighter, the F-22 Raptor, was deployed for the first time to Asia earlier this month. On Feb. 11, twelve Raptors flying from Hawaii to Japan were forced to turn back when a software glitch crashed all of the F-22s' on-board computers as they crossed the international date line. The delay in arrival in Japan was previously reported, with rumors of problems with the software. CNN television, however, this morning reported that every fighter completely lost all navigation and communications when they crossed the international date line. They reportedly had to turn around and follow their tankers by visual contact back to Hawaii. According to the CNN story, if they had not been with their tankers, or the weather had been bad, this would have been serious. CNN has not put up anything on their website yet."

Software Problems are Ubiquitous

1985-1987 -- Therac-25 medical accelerator. A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an "improved" therapy system that could deliver two different kinds of radiation: either a low-power electron beam (beta particles) or X-rays. The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. A second "improvement" was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.

What engineers didn't know was that both the 20 and the 25 were built upon an operating system that had been kludged together by a programmer with no formal training. Because of a subtle bug called a "race condition," a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others are seriously injured.

Software Problems are Ubiquitous

January 15, 1990 -- AT&T Network Outage. A bug in a new release of the software that controls AT&T's #4ESS long distance switches causes these mammoth computers to crash when they receive a specific message from one of their neighboring machines -- a message that the neighbors send out when they recover from a crash.

One day a switch in New York crashes and reboots, causing its neighboring switches to crash, then their neighbors' neighbors, and so on. Soon, 114 switches are crashing and rebooting every six seconds, leaving an estimated 60 thousand people without long distance service for nine hours. The fix: engineers load the previous software release.

Software Problems are Ubiquitous

- Other serious bugs (many others exist)
  - US Vincennes tracking software
  - MV-22 Osprey

  - Medtronic Model 8870 Software Application Card

Adversarial Failures

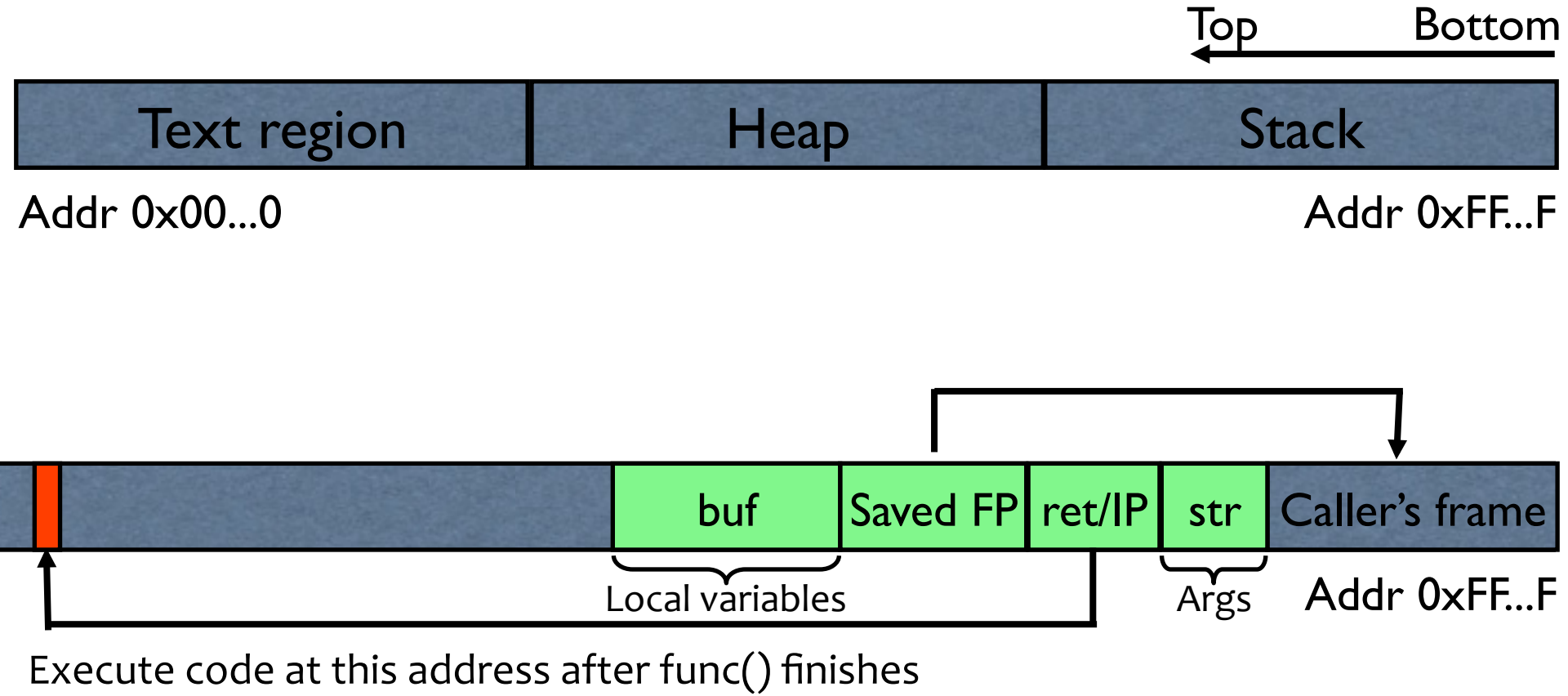
- Software bugs are bad
  - Consequences can be serious
- Even worse when an intelligent adversary wishes to exploit them!
  - Intelligent adversaries: **Force bugs into "worst possible" conditions/states**
  - Intelligent adversaries: **Pick their targets**

BUFFER OVERFLOWS

Adversarial Failures

- Buffer overflows bugs: Big class of bugs
  - Normal conditions: Can sometimes cause systems to fail
  - Adversarial conditions: Attacker able to violate security of your system (control, obtain private information, ...)

Reference for Q1



A Bit of History: Morris Worm

- Worm was released in 1988 by Robert Morris
  - Graduate student at Cornell, son of NSA chief scientist
  - Convicted under Computer Fraud and Abuse Act, sentenced to 3 years of probation and 400 hours of community service
  - Now an EECS professor at MIT
- Worm was intended to propagate slowly and harmlessly measure the size of the Internet
- Due to a coding error, it created new copies as fast as it could and overloaded infected machines
- $10-100M worth of damage

Morris Worm and Buffer Overflow

- One of the worm's propagation techniques was a buffer overflow attack against a vulnerable version of `fingerd` on VAX systems
  - By sending special string to finger daemon, worm caused it to execute code creating a new worm copy
  - Unable to determine remote OS version, worm also attacked `fingerd` on Suns running BSD, causing them to crash (instead of spawning a new copy)

Famous Internet Worms

- Buffer overflows: very common cause of Internet attacks
  - In 1998, over 50% of advisories published by CERT (computer security incident report team) were caused by buffer overflows
- Morris worm (1988): overflow in `fingerd`
  - 6,000 machines infected
- CodeRed (2001): overflow in MS-IIS server
  - 300,000 machines infected in 14 hours
- SQL Slammer (2003): overflow in MS-SQL server
  - 75,000 machines infected in **10 minutes** (!!)
- Sasser (2005): overflow in Windows LSASS
  - Around 500,000 machines infected

… And More

- Conficker (2008-08): overflow in Windows RPC
  - Around 10 million machines infected (estimates vary)
- Stuxnet (2009-10): several zero-day overflows + same Windows RPC overflow as Conficker
  - Windows print spooler service
  - Windows LNK shortcut display
  - Windows task scheduler
- Flame (2010-12): same print spooler and LNK overflows as Stuxnet
  - Targeted cyperespionage virus
- Still ubiquitous, especially in embedded systems

Attacks on Memory Buffers

- Buffer is a pre-defined data storage area inside computer memory (stack or heap)
- Typical situation:
  - A function takes some input that it writes into a pre-allocated buffer.
  - The developer forgets to check that the size of the input isn't larger than the size of the buffer.
  - Uh oh.
    - "Normal" bad input: crash
    - "Adversarial" bad input : take control of execution

Stack Buffers

- Suppose Web server contains this function

```
void func(char *str) {
        char buf[126];
        ...
        strcpy(buf,str);
        ...
    }
```

- No bounds checking on strcpy()
- If str is longer than 126 bytes
  - Program may crash
  - Attacker may change program behavior

Answer Q2

- Suppose Web server contains this function

```
void func(char *str) {
        char buf[126];
        ...
        strcpy(buf,str);
        ...
    }
```

- No bounds checking on strcpy()
- If str is longer than 126 bytes
  - Program may crash
  - Attacker may change program behavior

Example: Changing Flags

- `authenticated` variable

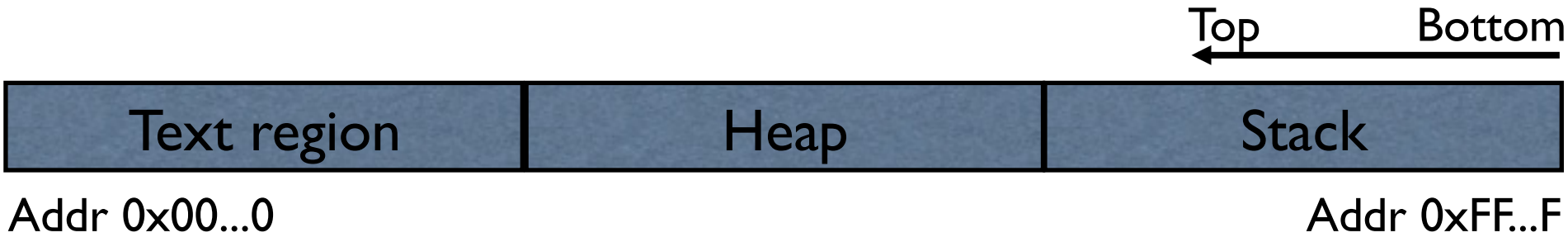
Example: Changing Flags

- `authenticated` variable

- Morris worm also overflowed a buffer to overwrite an authenticated flag in fingerd

Memory Layout

- Text region: Executable code of the program
- Heap: Dynamically allocated data
- Stack: Local variables, function return addresses; grows and shrinks as functions are called and return



Redirecting Program Flow

- Instead of "normal" string, attacker sends 2 things as input:
  - Assembly code she wants to execute
  - The address where she expects that code to appear

Redirecting Program Flow

- Instead of "normal" string, attacker sends 2 things as input:

"Shellcode"

  - Assembly code she wants to execute
  - The address where she expects that code to appear

Stack Buffers

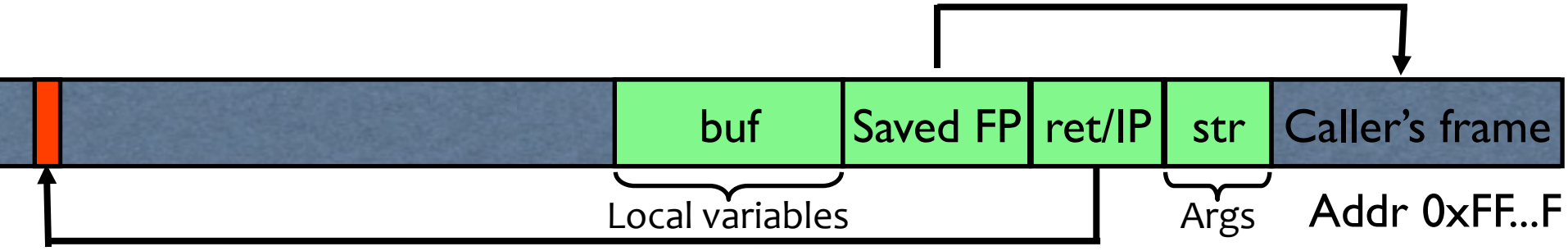
- Suppose Web server contains this function:

```
void func(char *str) {
        char buf[126];
        strcpy(buf,str);
    }
```

Allocate local buffer
(126 bytes reserved on stack)

Copy argument into local buffer

- When this function is invoked, a new frame (activation record) is pushed onto the stack.



Execute code at this address after func() finishes

What if Buffer is Overstuffed?

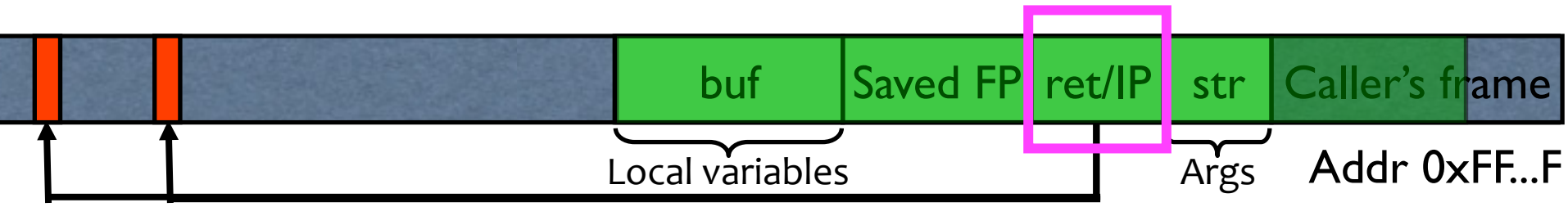
- Memory pointed to by str is copied onto stack…

```
void func(char *str) {
        char buf[126];
        strcpy(buf,str);
    }
```

strcpy does NOT check whether the string at *str contains fewer than 126 characters

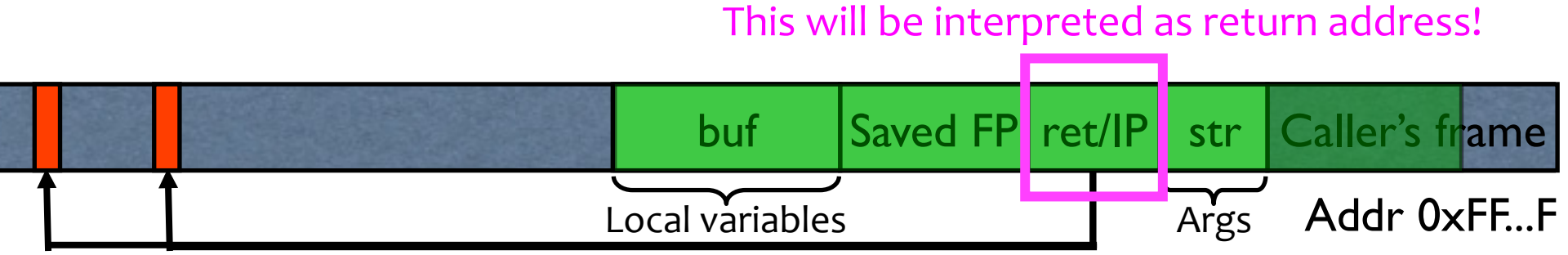
- If a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations.

This will be interpreted as return address!



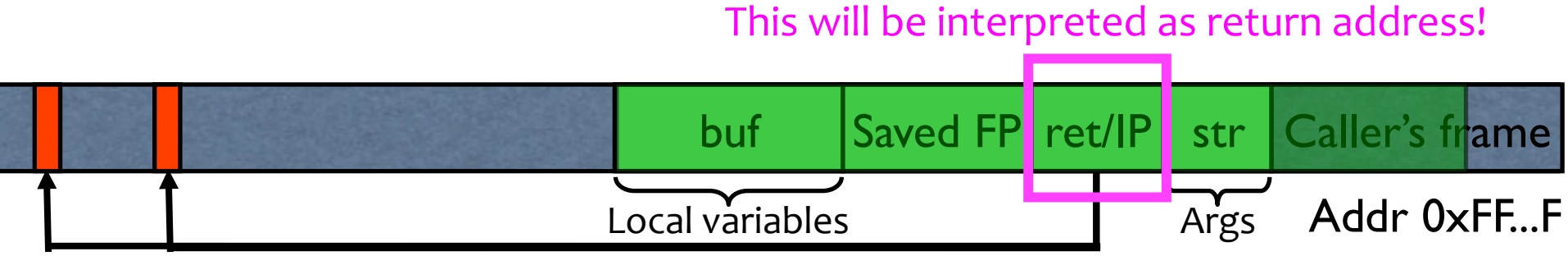
What if Buffer is Overstuffed?

- What if the string is read in from an attacker on the network?

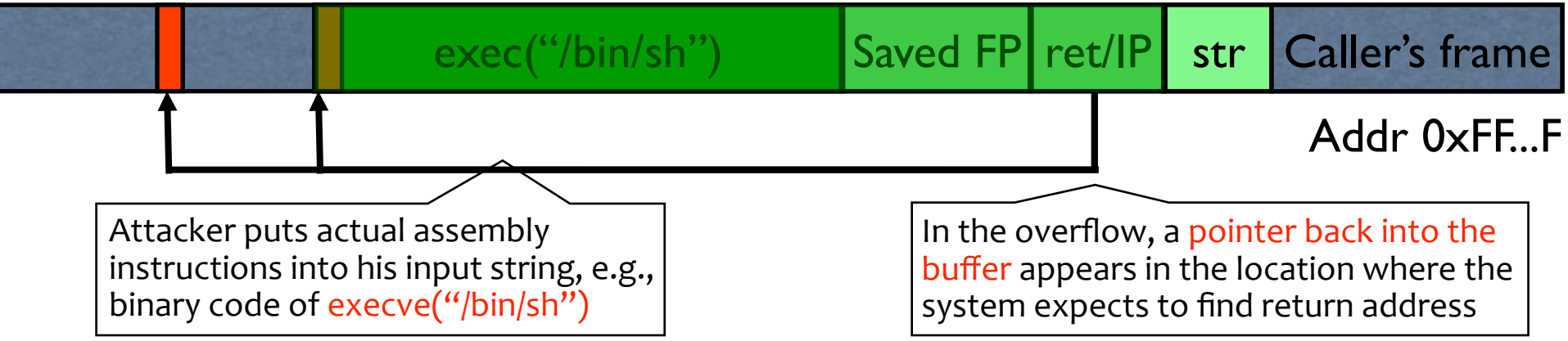


What if Buffer is Overstuffed?

```
exec("/bin/sh") asdf…asdf 0xFFFFFFA2
```

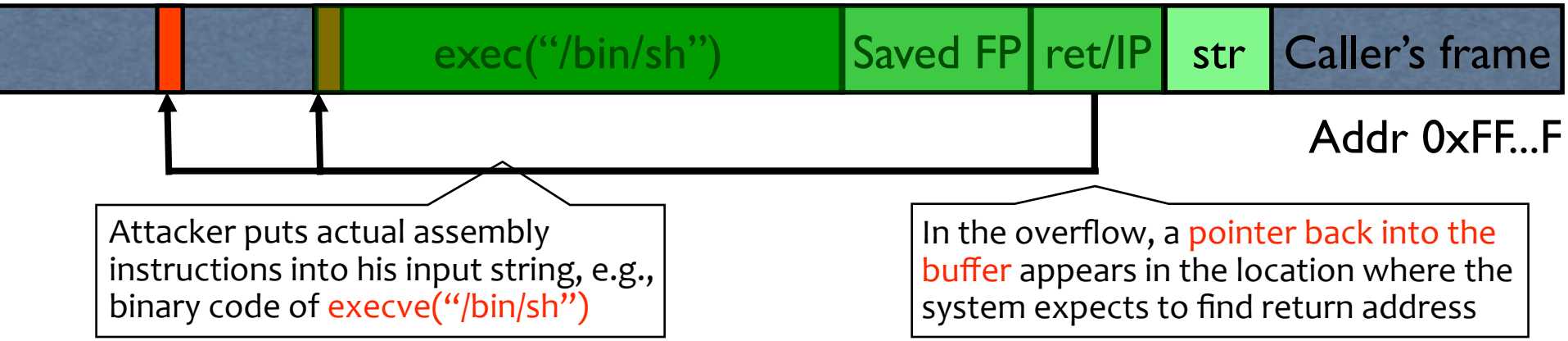


Executing Attack Code



- When function exits, code in the buffer will be executed, giving attacker a shell
  - Root shell if the victim program is setuid root

Stretch Break



- When function exits, code in the buffer will be executed, giving attacker a shell
  - Root shell if the victim program is setuid root

Buffer Overflows can be Hard

- Overflow portion of the buffer must contain correct address of attack code in the RET position
  - The value in the RET position must point to the beginning of attack assembly code in the buffer
    - Otherwise application will (probably) crash with segmentation violation
  - Attacker must correctly guess in which stack position his/her buffer will be when the function is called

Problem: No Bounds Checking

- strcpy does not check input size
  - strcpy(buf, str) simply copies memory contents into buf starting from *str until "\0" is encountered, ignoring the size of area allocated to buf
- Many C library functions are unsafe
  - strcpy(char *dest, const char *src)
  - strcat(char *dest, const char *src)
  - gets(char *s)
  - scanf(const char *format, …)
  - printf(const char *format, …)

Does Bounds Checking Help?

- strncpy(char *dest, const char *src, size_t n)
  - If strncpy is used instead of strcpy, no more than n characters will be copied from *src to *dest
    - Programmer has to supply the right value of n
- Potential overflow in htpasswd.c (Apache 1.3):

```
strcpy(record,user);
strcat(record,":");
strcat(record,cpw);
```

Copies username ("user") into buffer ("record"), then appends ":" and hashed password ("cpw")

- Published fix:

```
strncpy(record,user,MAX_STRING_LEN-1);
strcat(record,":")
strncat(record,cpw,MAX_STRING_LEN-1);
```

Answer Q3

- strncpy(char *dest, const char *src, size_t n)
  - If strncpy is used instead of strcpy, no more than n characters will be copied from *src to *dest
    - Programmer has to supply the right value of n
- Potential overflow in htpasswd.c (Apache 1.3):

```
strcpy(record,user);
strcat(record,":");
strcat(record,cpw);
```

Copies username ("user") into buffer ("record"), then appends ":" and hashed password ("cpw")

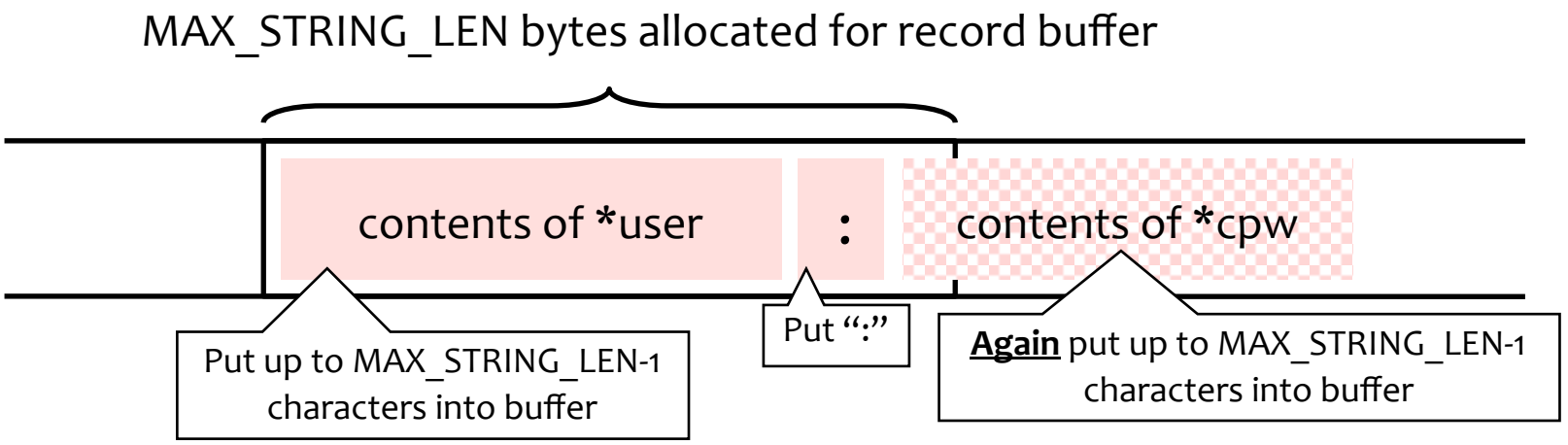
- Published fix:

```
strncpy(record,user,MAX_STRING_LEN-1);
strcat(record,":")
strncat(record,cpw,MAX_STRING_LEN-1);
```

Misuse of strncpy in htpasswd "Fix"

- Published "fix" for Apache htpasswd overflow:

```
strncpy(record,user,MAX_STRING_LEN-1);
strcat(record,":")
strncat(record,cpw,MAX_STRING_LEN-1);
```



What About This?

- Home-brewed range-checking string copy

```
void mycopy(char *input) {
     char buffer[512]; int i;

     for (i=0; i<=512; i++)
         buffer[i] = input[i];
}
void main(int argc, char *argv[]) {
     if (argc==2)
        mycopy(argv[1]);
}
```

Off-By-One Overflow

- Home-brewed range-checking string copy

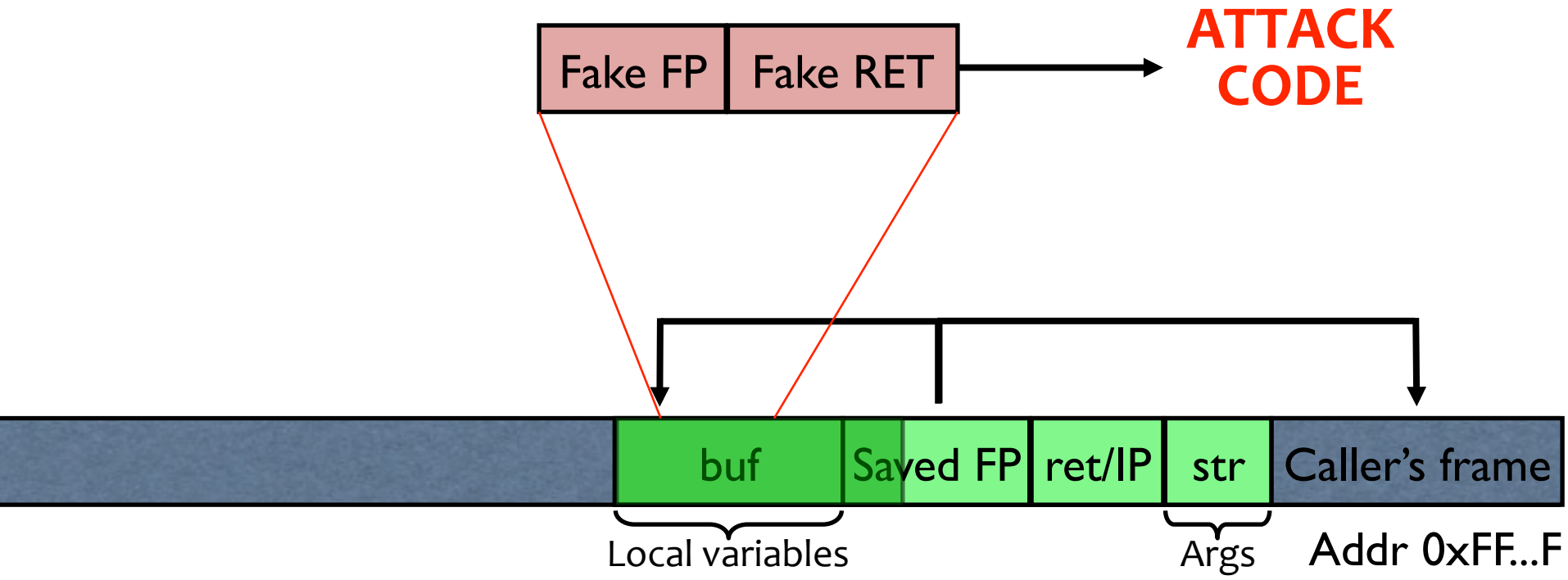
```
void mycopy(char *input) {
     char buffer[512]; int i;

     for (i=0; i<=512; i++)
         buffer[i] = input[i];
}
void main(int argc, char *argv[]) {
     if (argc==2)
        mycopy(argv[1]);
}
```

This will copy **513** characters into buffer. Oops!

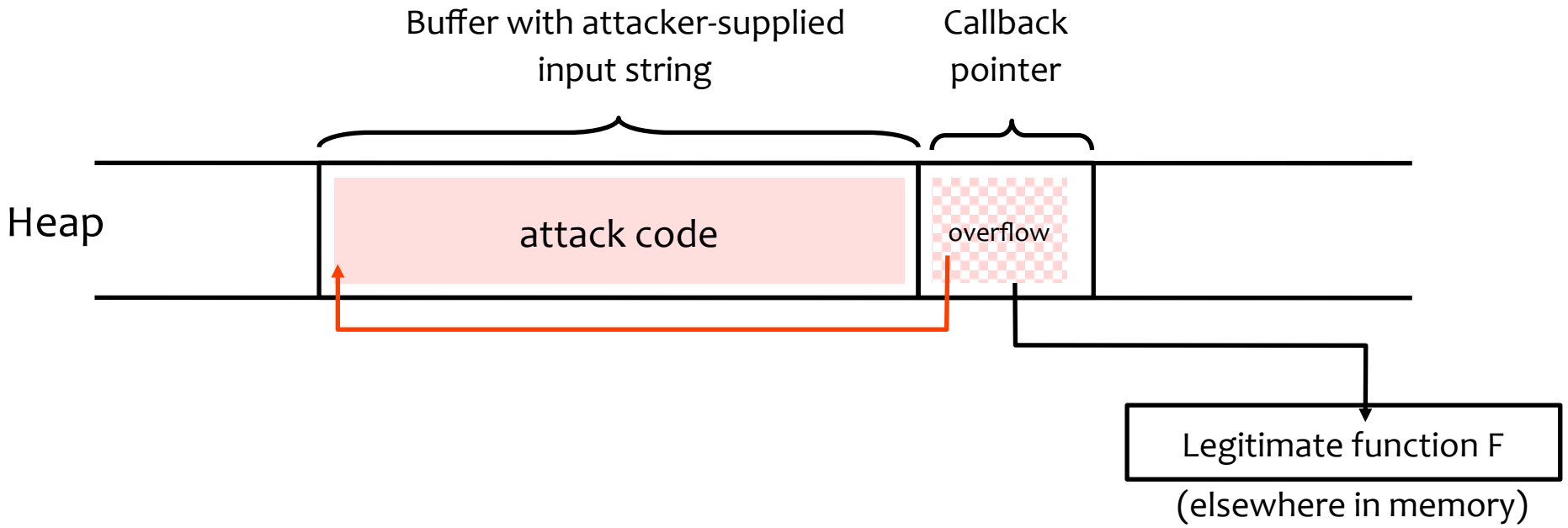
- 1-byte overflow: can't change RET, but can change pointer to previous stack frame
  - On little-endian architecture, make it point into buffer
  - RET for previous function will be read from buffer!

Frame Pointer Overflow



Another Variant: Function Pointer Overflow

- C uses function pointers for callbacks: if pointer to F is stored in memory location P, then another function G can call F as (*P)(…)



Other Overflow Targets

- Format strings in C
  - More details next time
- Heap management structures used by malloc()
  - More details in section

- These are all attacks you can look forward to in Lab #1 ☺

Looking Forward

- Ethics form due at 5!
- Homework #1 due Monday, Oct 10
- Next few classes:
  - Friday: guest lecture by David Aucsmith
  - Monday: more buffer overflows
  - Wednesday: guest lecture by Emily McReynolds
- Section tomorrow about Lab 1