CSE 484: Computer Security and Privacy

Software Security: Buffer Overflow Defenses

Spring 2021

Tadayoshi Kohno

yoshi@cs

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

Admin

- Homework 1: Due Tomorrow 11:45pm
- Lab 1:
  - Online: https://courses.cs.washington.edu/courses/cse484/21sp/assignments/lab1.pdf
  - Try to form groups and do “Environment and Sign-Up” before quiz section

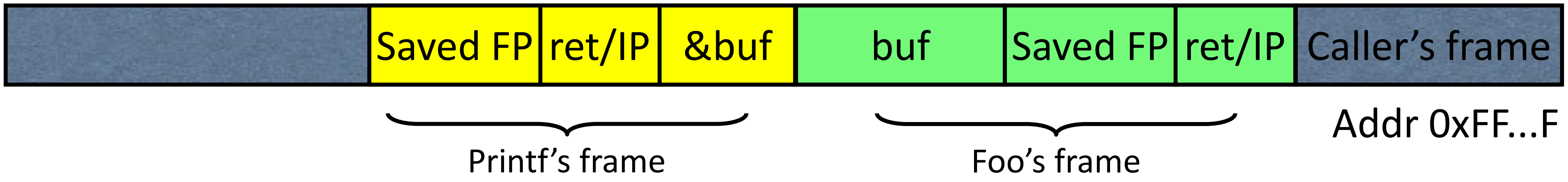
Summary of Printf Risks

- Printf takes a variable number of arguments
  - E.g., printf("Here's an int: %d", 10);
- Assumptions about input can lead to trouble
  - E.g., printf(buf) when buf="Hello world" versus when buf="Hello world %d"
  - Can be used to advance printf's internal stack pointer
  - Can read memory
    - E.g., printf("%x") will print in hex format whatever printf's internal stack pointer is pointing to at the time
  - Can write memory
    - E.g., printf("Hello%n"); will write "5" to the memory location specified by whatever printf's internal SP is pointing to at the time

How Can We Attack This?

```
foo() {
        char buf[…];
        strncpy(buf, readUntrustedInput(), sizeof(buf));
        printf(buf); //vulnerable
}
```

If format string contains % then printf will expect to find arguments here…

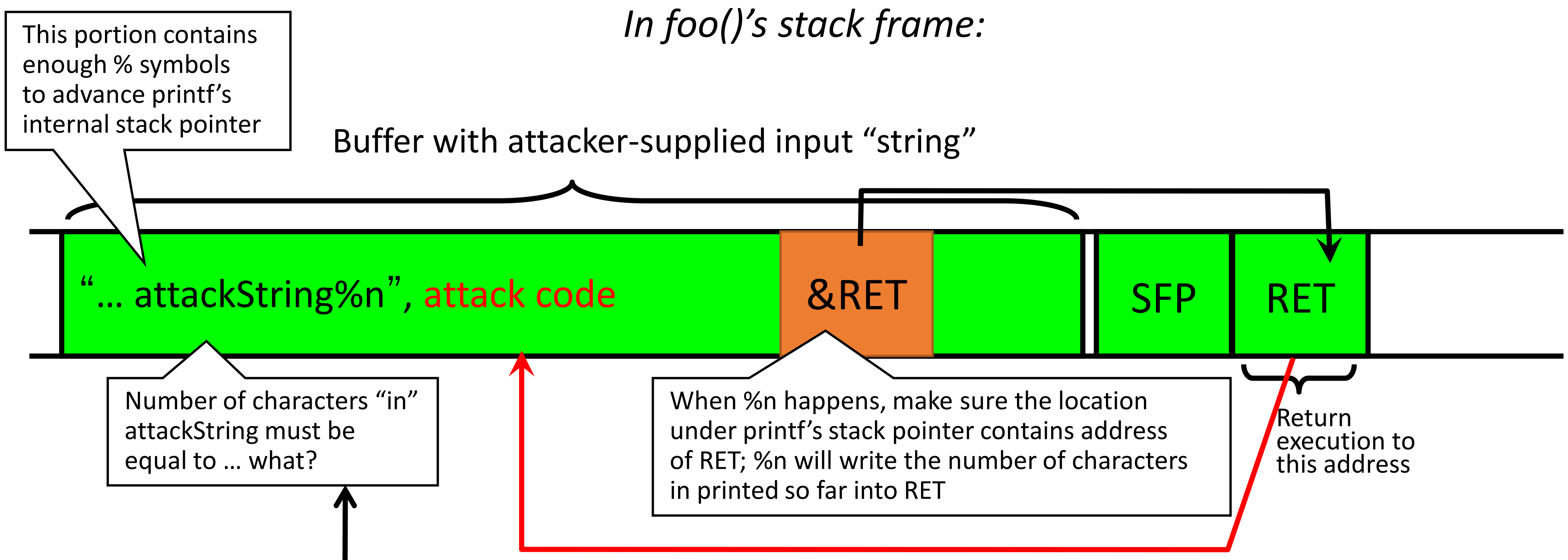


What should the string returned by readUntrustedInput() contain??

Go to Canvas Quiz for today!

Different compilers / compiler options / architectures might vary

Using %n to Overwrite Return Address



Why is "in" in quotes? C allows you to concisely specify the "width" to print, causing printf to pad by printing additional blank characters without reading anything else off the stack.

Example: printf("%5d%n", 10) will print three spaces followed by the integer: "   10"
That is, the %n will write 5, not 2.

Key idea: do this 4 times with the right numbers to overwrite the return address byte-by-byte. (4x %n to write into &RET, &RET+1, &RET+2, &RET+3)

The exploitation twilight zone

- During an exploitation attempt sometimes you have to ‘let it run’
  - Overflow a buffer
  - Change things
  - Let program run for ‘a bit’
  - Everything triggers!

- Printf exploit a perfect example

Recommended Reading

- It will be hard to do Lab 1 without:
  - Reading (see course schedule):
    - Smashing the Stack for Fun and Profit
    - Exploiting Format String Vulnerabilities
  - Attending section

Buffer Overflow: Causes and Cures

- Classical memory exploit involves code injection
  - Put malicious code at a predictable location in memory, usually masquerading as data
  - Trick vulnerable program into passing control to it

- Possible defenses:
  1. Prevent execution of untrusted code
  2. Stack “canaries”
  3. Encrypt pointers
  4. Address space layout randomization
  5. Code analysis
  6. …

Defense: Executable Space Protection

- Mark all writeable memory locations as non-executable
  - Example: Microsoft's Data Execution Prevention (DEP)
  - **This blocks many code injection exploits**
- Hardware support
  - AMD "NX" bit (no-execute), Intel "XD" bit (executed disable) (in post-2004 CPUs)
  - Makes memory page non-executable
- Widely deployed
  - Windows XP SP2+ (2004), Linux since 2004 (check distribution), OS X 10.5+ (10.4 for stack but not heap), Android 2.3+

What Does "Executable Space Protection" Not Prevent?

- Can still corrupt stack …
  - … or function pointers
  - … or critical data on the heap

- **As long as RET points into existing code, executable space protection will not block control transfer!**

  → return-to-libc exploits

return-to-libc

- Overwrite saved ret (IP) with address of **any library routine**
  - Arrange stack to look like arguments

- Does not look like a huge threat
  - …

- Canvas in-class activity, April 6!

return-to-libc

- Overwrite saved ret (IP) with address of **any library routine**
  - Arrange stack to look like arguments

- Does not look like a huge threat
  - …
  - We can call *any* function we want!
  - Say, exec ☺

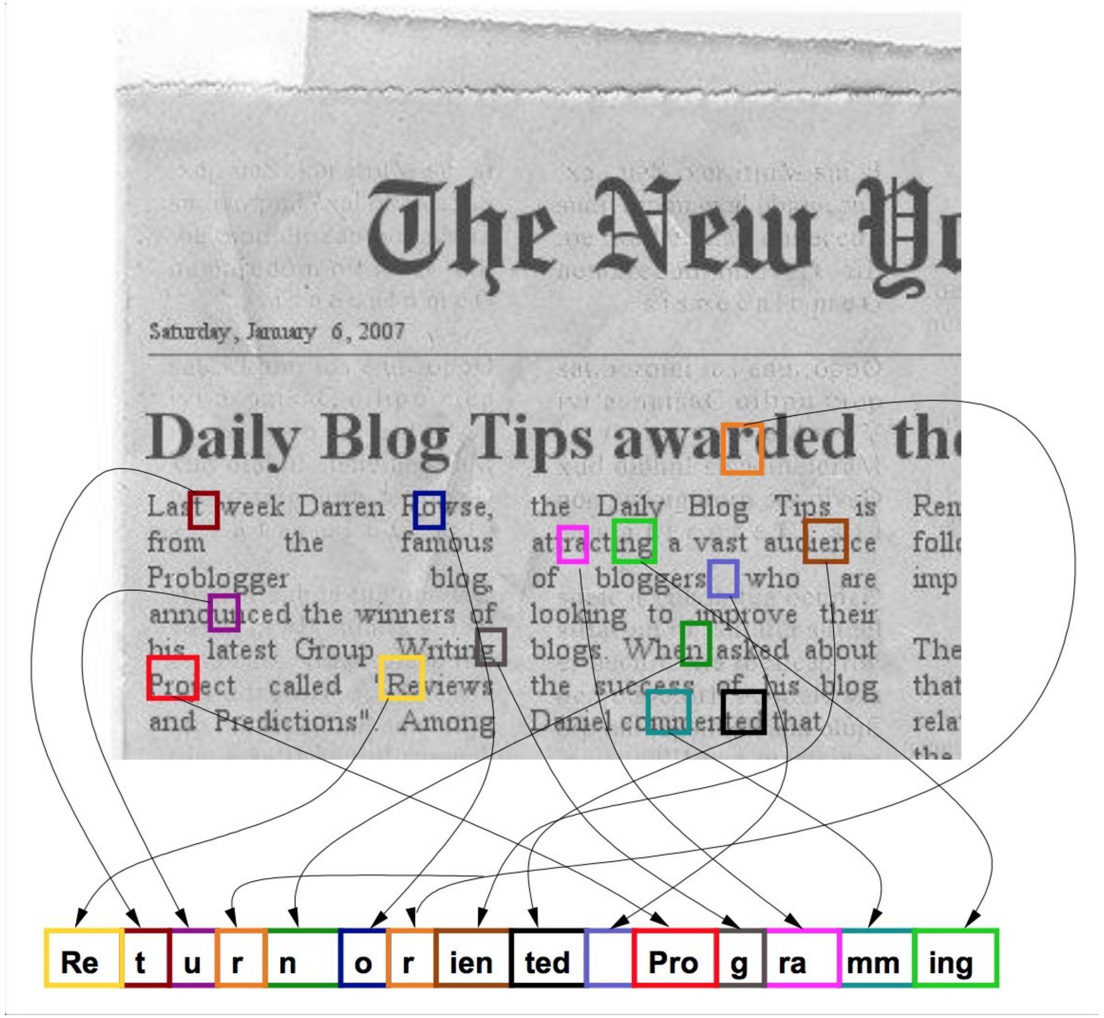
return-to-libc++

- Insight: Overwritten saved EIP need not point to the *beginning* of a library routine
- **Any** existing instruction in the code image is fine
  - Will execute the sequence starting from this instruction
- What if instruction sequence contains RET?
  - Execution will be transferred… to where?
  - Read the word pointed to by stack pointer (SP)
    - Guess what? Its value is under attacker's control!
  - Use it as the new value for IP
    - Now control is transferred to an address of attacker's choice!
  - Increment SP to point to the next word on the stack

Chaining RETs

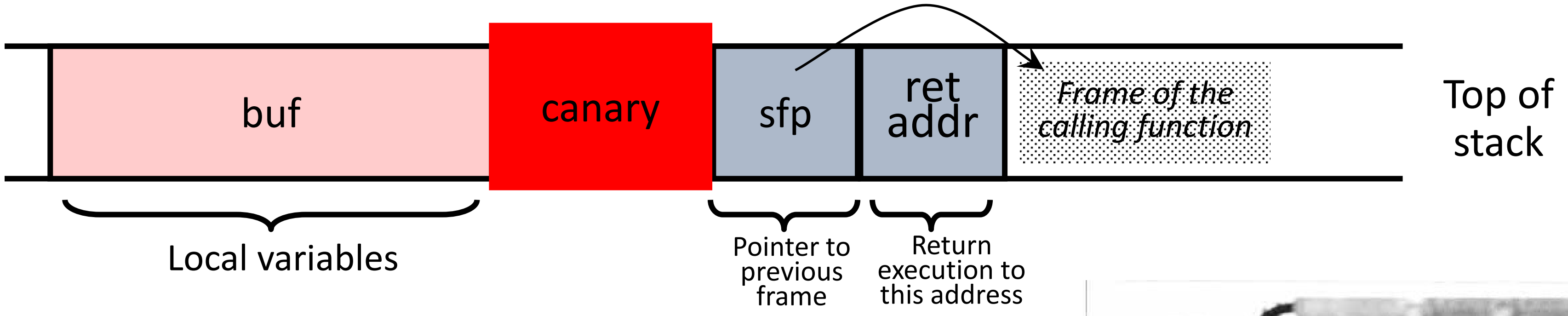
- Can chain together sequences ending in RET
  - Krahmer, “x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique” (2005)
- What is this good for?
- Answer [Shacham et al.]: everything
  - Turing-complete language
  - Build “gadgets” for load-store, arithmetic, logic, control flow, system calls
  - Attack can perform arbitrary computation using no injected code at all – return-oriented programming

Return-Oriented Programming



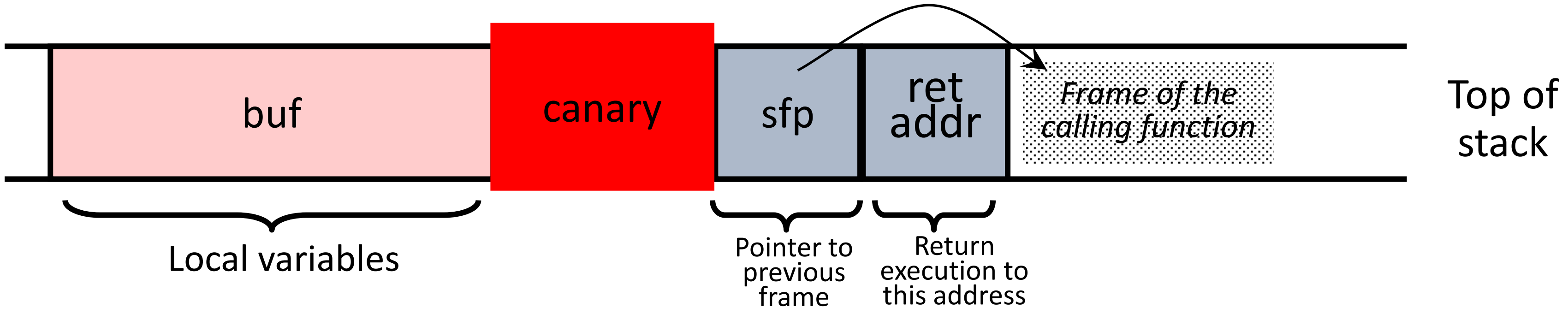
Defense: Run-Time Checking: StackGuard

- Embed "canaries" (stack cookies) in stack frames and verify their integrity prior to function return
  - Any overflow of local variables will damage the canary



Defense: Run-Time Checking: StackGuard

- Embed “canaries” (stack cookies) in stack frames and verify their integrity prior to function return
  - Any overflow of local variables will damage the canary



- Choose random canary string on program start
  - Attacker can’t guess what the value of canary will be
- Canary contains: “\0”, newline, linefeed, EOF
  - String functions like strcpy won’t copy beyond “\0”

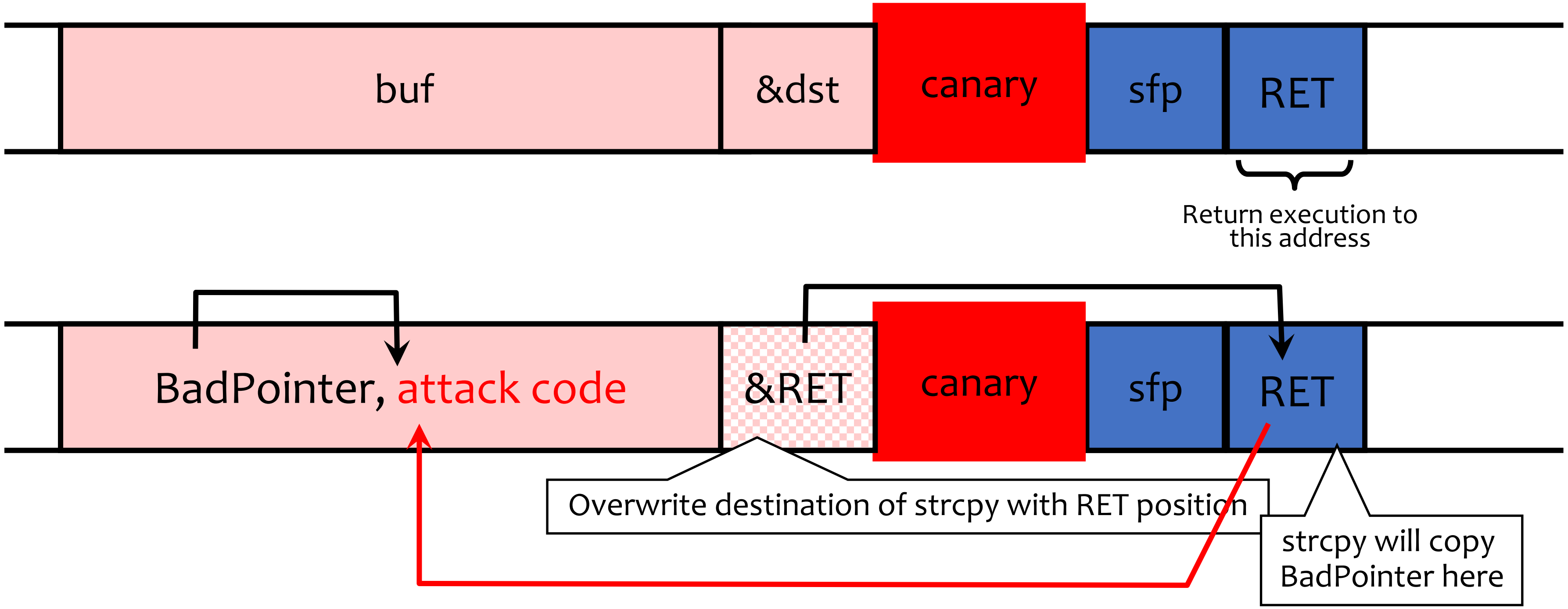
StackGuard Implementation

- StackGuard requires code recompilation
- Checking canary integrity prior to every function return causes a performance penalty
  - For example, 8% for Apache Web server at one point in time

- Canvas Quiz

Defeating StackGuard

- StackGuard can be defeated
  - A single memory write where the attacker controls both the value and the destination is sufficient
- Suppose program contains copy(buf,attacker-input) and copy(dst,buf)
  - Example: dst is a local pointer variable
  - Attacker controls both buf and dst



Admin (Reminders)

- Homework 1: Due Tomorrow 11:45pm
- Lab 1:
  - Online: https://courses.cs.washington.edu/courses/cse484/21sp/assignments/lab1.pdf
  - Try to form groups and do "Environment and Sign-Up" before quiz section