Buffer Overflow Attacks

Peter C. Chapin
Vermont Technical College

Bounds Checking

Programming languages intended for high performance often do not check the bounds when arrays are accessed.

```
char buffer[128]; buffer[128] = 'x';
```

Such checks usually involve runtime overhead.

C, C++, Fortran, Assembly Language (of course) all prefer high speed over safety of execution.

Out of bounds access → “undefined behavior”

C Makes Bounds Checking Hard

Adding bounds checking to C is difficult because of pointer arithmetic

```
int main(void)
{
  char buffer[128];

  f(buffer);
  f(buffer - 1);
  f(buffer + 64);
}
```

```
void f(char *buffer)
{
  buffer[128] = 'x';
}
```

Function `f` doesn't know `buffer` points into an array let alone how large the array is.

Adding Bounds Checking

Fortran: Many compilers will add bounds checking code as an option for debugging purposes.

C/C++: Adding such code is infeasible.

Except... Special techniques that attach extra information to pointers can be used.

An experimental version of gcc does this. [reference?]

However, performance hit is greater than that suffered by normal bounds checking languages.

Security Recommendation

*If security is a high priority, consider **not** using C or C++.*

If C or C++ must be used, apply tools that employ static and/or dynamic analysis methods to look for possible buffer overflows.

Do not rely on your ability to notice possible problems. Experience shows that technique does not work well.

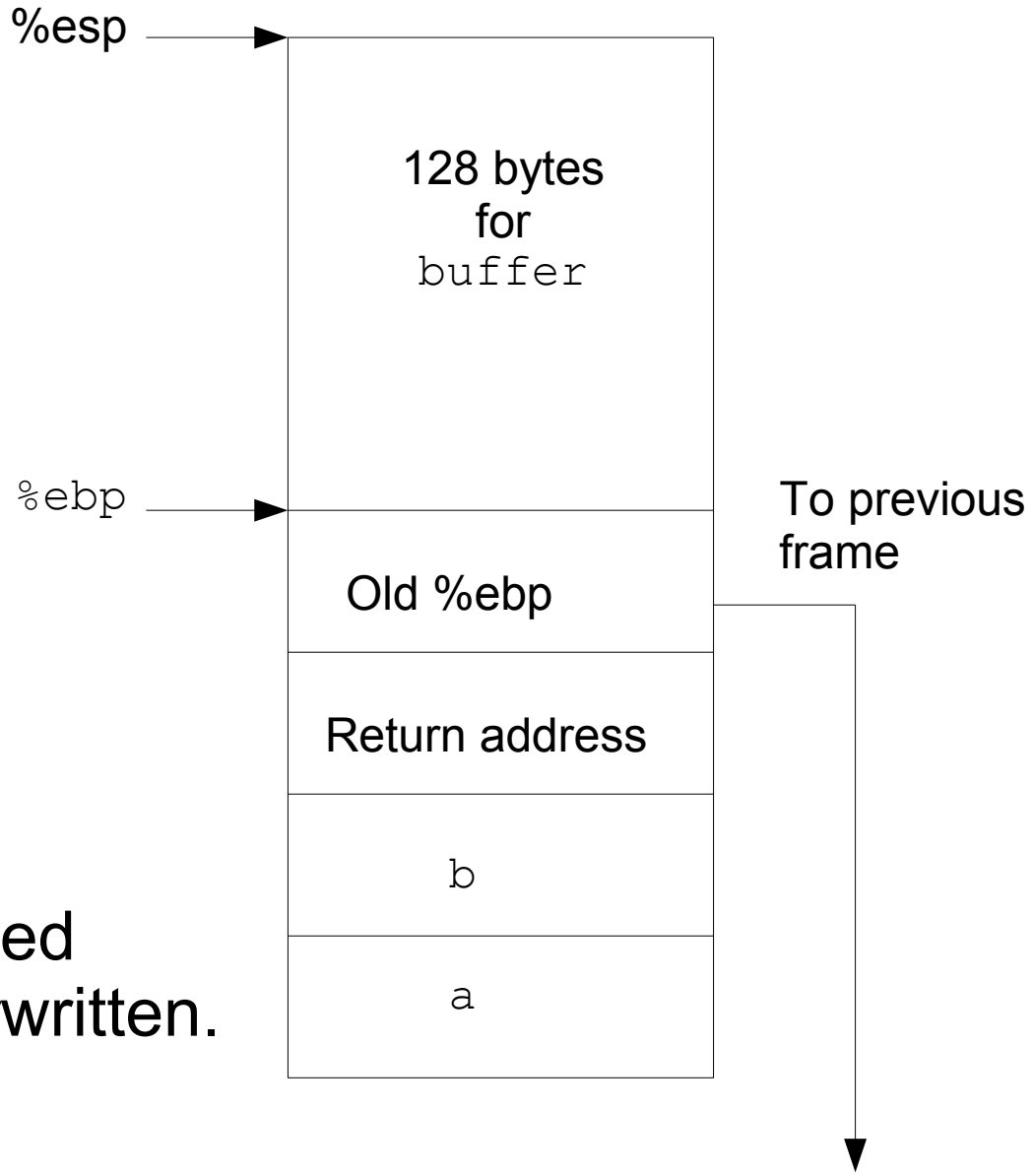
Stack Frames

```
int f(int a, int b)
{
  char buffer[128];

  push %ebp
  mov  %esp, %ebp
  sub  128, %esp

  ... [%ebp + 8]
  ... [%ebp - 128]

  mov  %ebp, %esp
  pop  %ebp
  ret
}
```



When a local buffer is overflowed
the return address can be overwritten.

Overflows Cause Crashes

When a “random” return address is used, the function returns to an unexpected location.

- It is *possible* that the program will continue executing.
- It is more likely that the program will crash.

An attacker who can control the return address can get the program to execute arbitrary code.

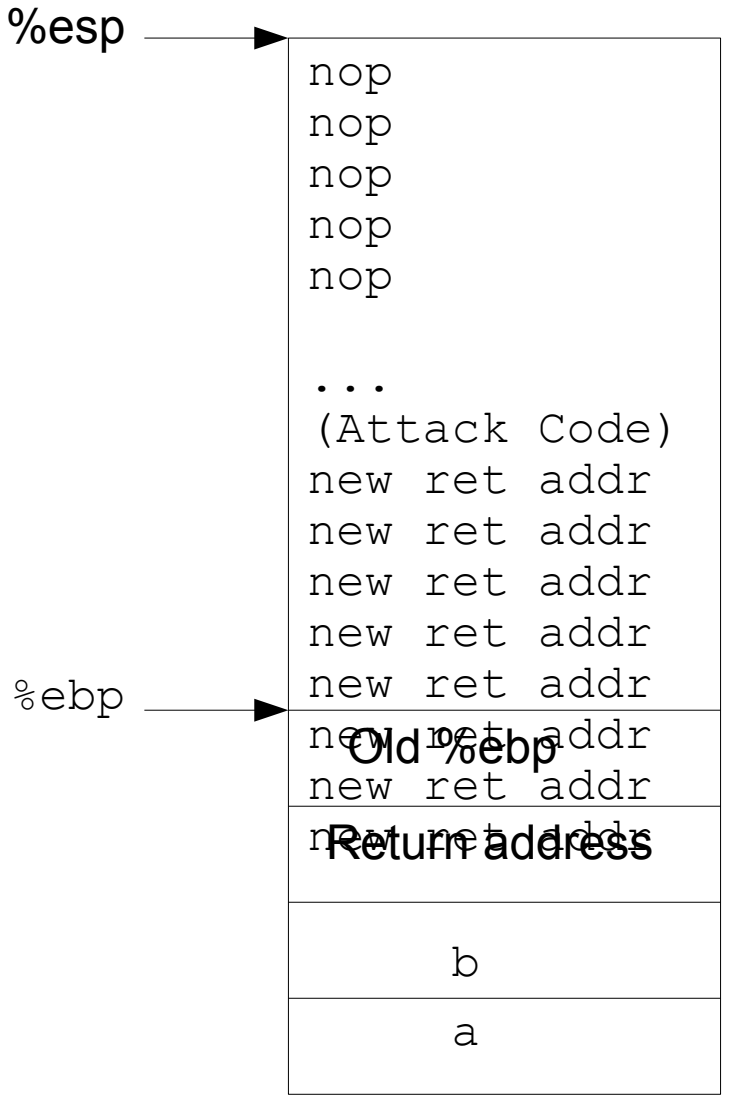
Attack Idea

1. Provide attack code as input to the program.

2. Flood buffer with desired return return address after the attack code.

3. Prefix attack code with `nop` instructions so return address need not be 100% accurate.

4. When function returns, program executes attack code.



Attack Code

Cleaning Up the Attack Code

What Return Address?

Countermeasures

Don't use C.

Use an operating system that won't let code execute on the stack.

Use a run time stack guard system of some sort (dynamic analysis).

Statically analyze code with a tool looking for problems.

At least: Write the code carefully (don't be lazy about bounds checking) and inspect the code manually looking for problems.