Buffer Overflow Attacks

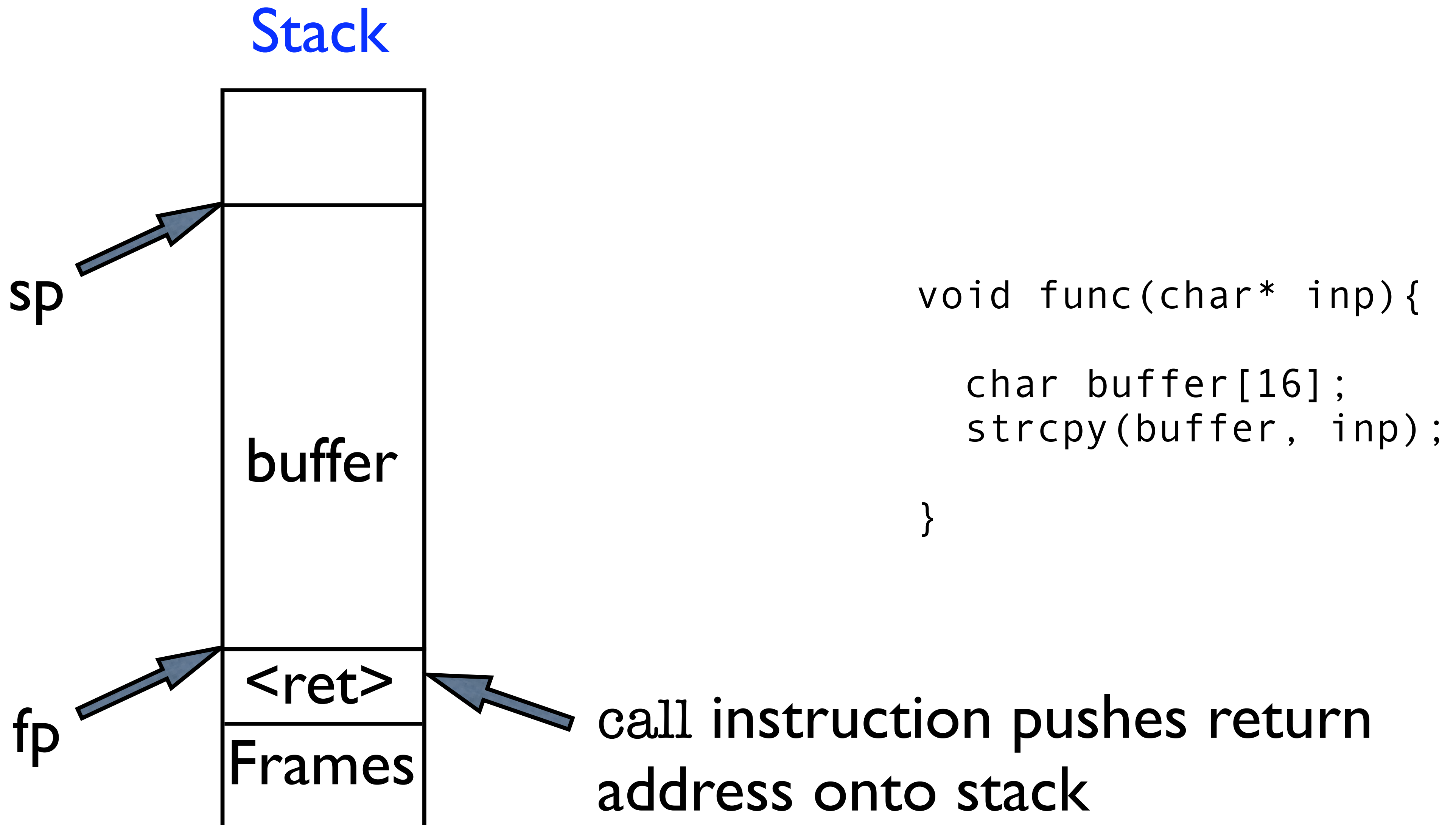
1. Smashing the Stack
2. Other Buffer Overflow Attacks
3. Work on Preventing Buffer Overflow Attacks

Smashing the Stack

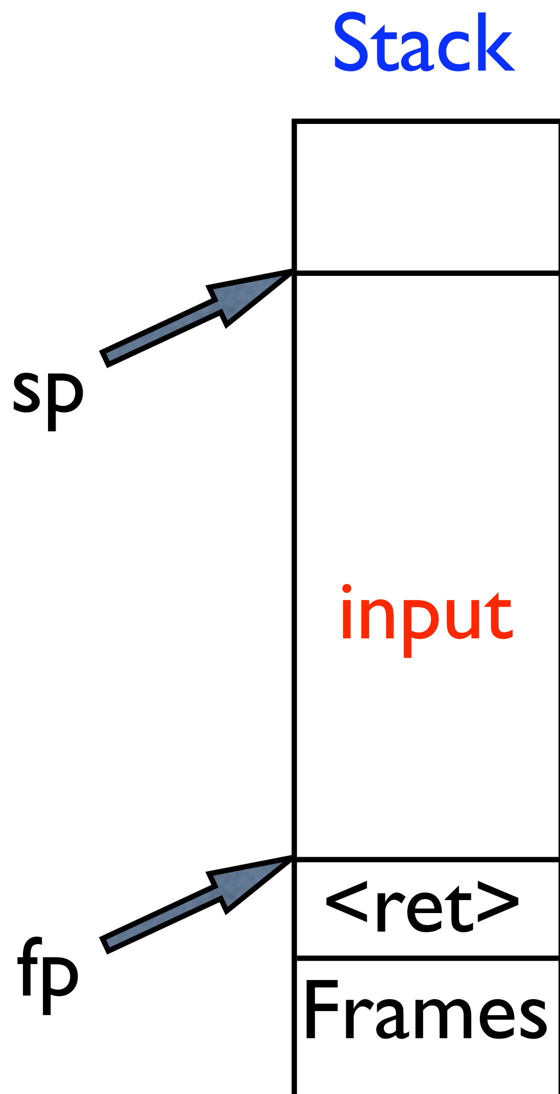
An Evil Function

```
void func(char* inp){

    char buffer[16];
    strcpy(buffer, inp);

}
```

Calling `func`

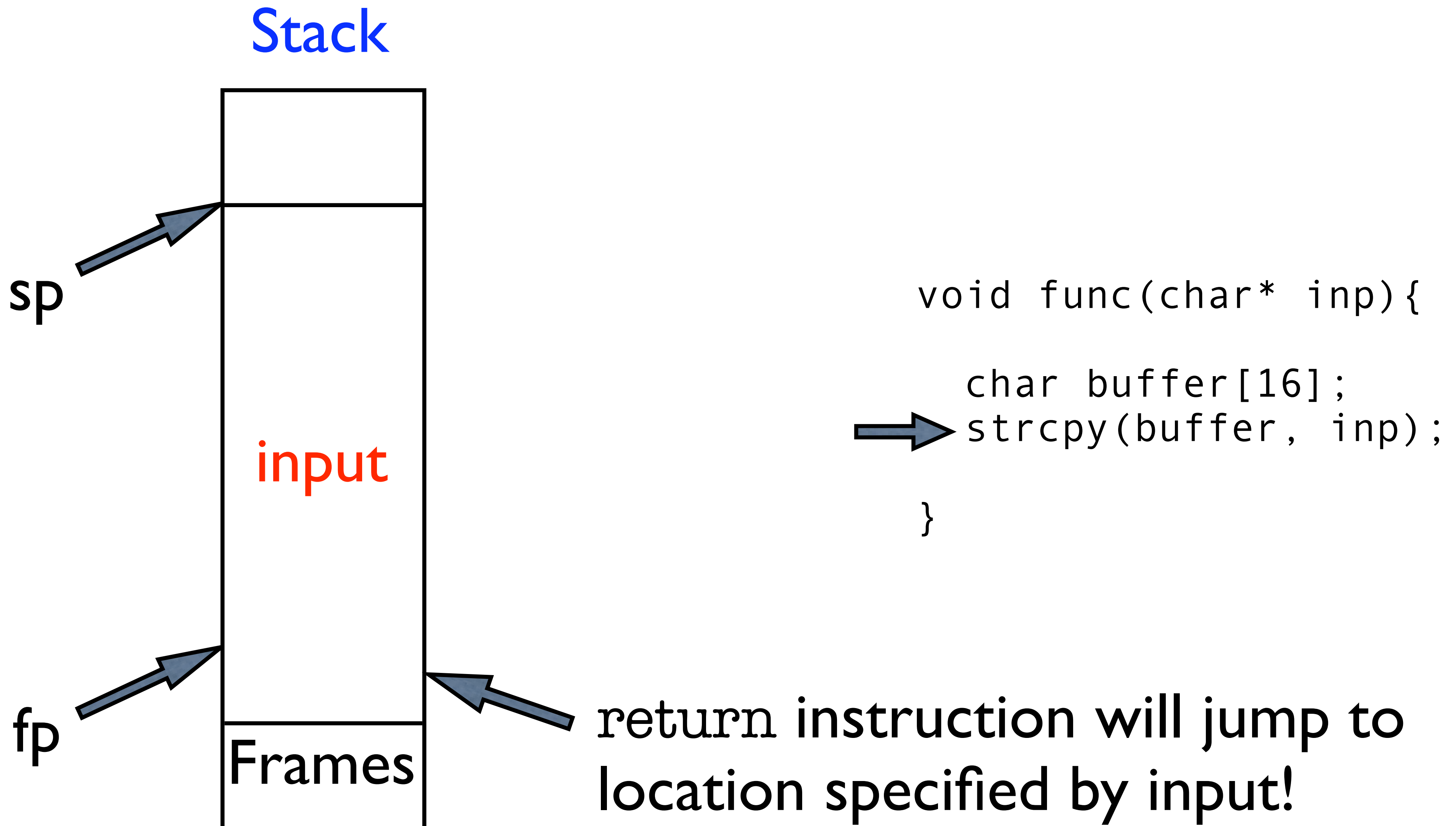


Operation of `func`

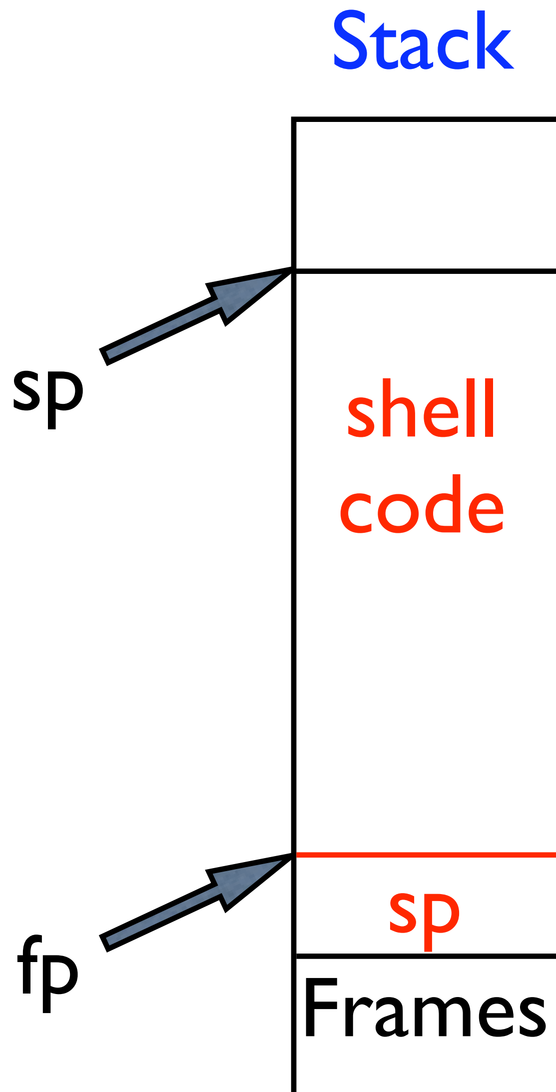



```
void func(char* inp){

  char buffer[16];
  strcpy(buffer, inp);

}
```

C doesn't check bounds...



The Exploit




```
void func(char* inp){

  char buffer[16];
  strcpy(buffer, inp);

}
```

Two Questions

1. How does attacker know where buffer starts on the stack?

Two Questions

1. How does attacker know where buffer starts on the stack (referred to as ra)?
2. How does attacker know where to put ra in the input to overwrite <ret>?

Two Questions

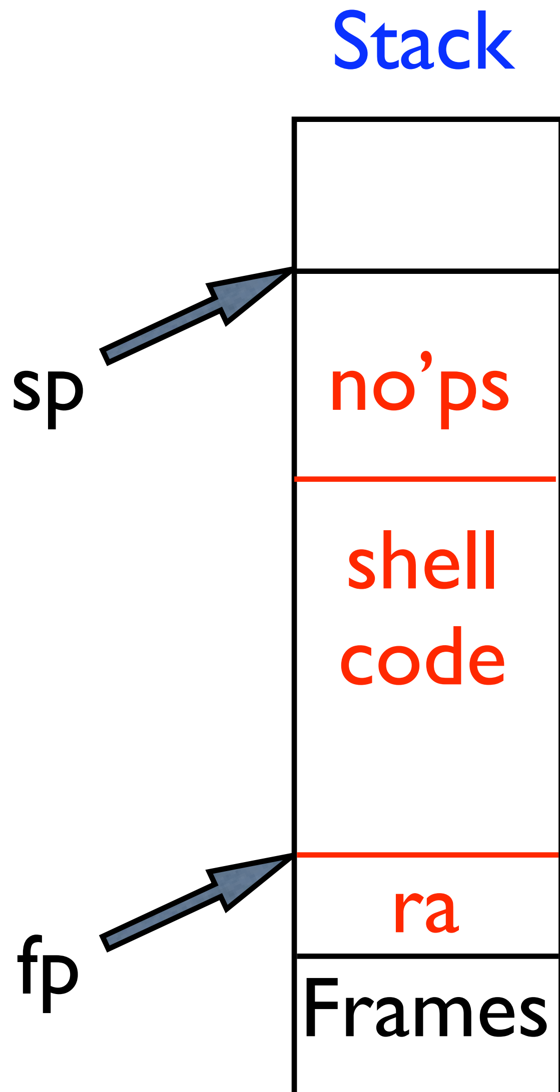
1. How does attacker know where buffer starts on the stack?
2. How does attacker know where to put ra in the input to overwrite <ret>?

Answer: she doesn't, but she can guess...

Guessing ra

- Stack has same starting location in all programs
- Most programs don't ever put more than a few thousand bytes on the stack
- So, just guess ra!

Adding a "Launching Pad"

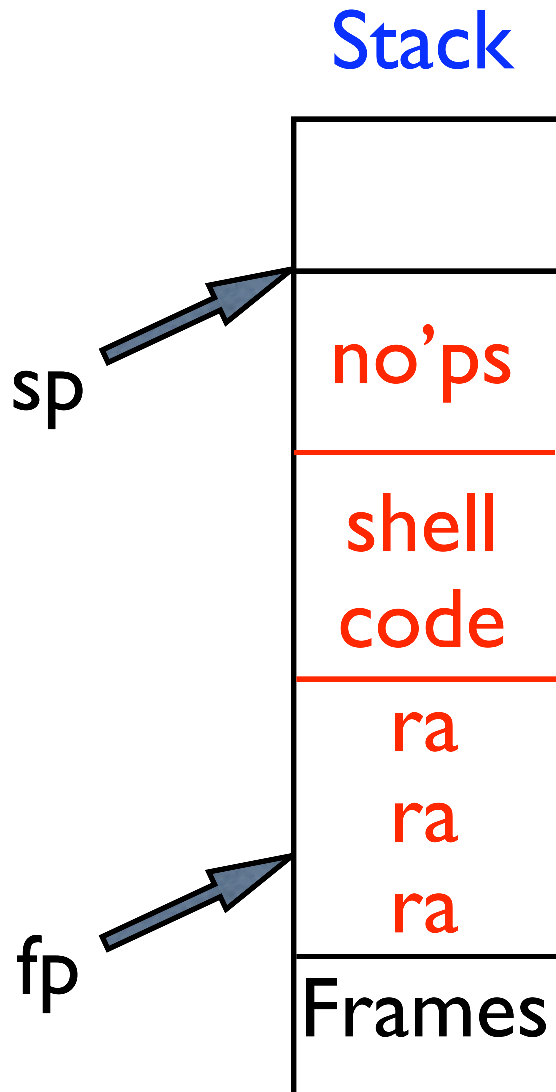



```
void func(char* inp){

  char buffer[16];
  strcpy(buffer, inp);

}
```

Guessing Where to Put ra in the Input

- This is really just trial-and-error...
- But we can improve our chances on this as well.

Improvement




```
void func(char* inp){

  char buffer[16];
  strcpy(buffer, inp);

}
```

Final Version of Input

no'ps	shellcode	ra ra ra

A good value of ra is worked out through trial and error, as is a good length for the input

Other Buffer Overflow Attacks

Arc Injection

- Input overwrites return address to point to a piece of code *not* supplied by the attacker
- For example, `system` call in libc:

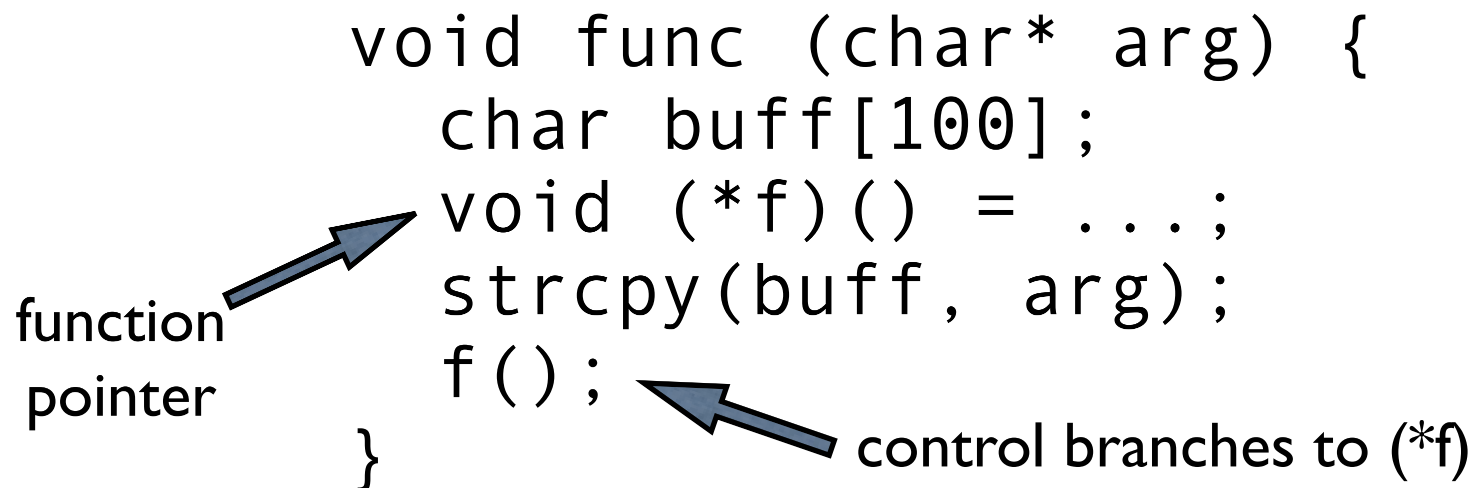
```
void system(char * arg){
  check_validity(arg);
  R = arg;
  execl(R, ...)
}
```

register

target for attacker

Pointer Subterfuge

- Attacker overwrites a pointer instead of the return address
- Example: overwriting a function pointer



Heap Smashing

- Exploits buffer overflows in heap rather than on stack
- Often combined with pointer subterfuge (eg, overwriting a function pointer that is not a local variable)
- Gaining in popularity

Preventing Buffer Overflows

Stackguard (Cowan et al.)

- Instruments code during compilation:
  1. Augments `call` instructions to write a ‘canary word’ next to return address, and push it onto a ‘canary stack’
  2. Augments `return` instructions to first check that canary word is intact

ProPolice (Etoh & Yoda)

- Extension to Stackguard's method: they place canary word between local char[]'s and other local variables
- This protects against pointer subterfuge on the stack

StackShield

- Similar to StackGuard, but saves a copy of return address to check against
- If real return address has been overwritten, can either terminate program or use copy and let program keep running

CRED: Bounds Checking for C (Ruwase & Lam)

- Does bounds checking for all string/char[] accesses
- Only tool to detect all attacks in a testbed that includes all previously mentioned attacks
- Adds 26%-130% overhead to the program

Baratloo et al.

- Two dynamically loaded libraries: Libsafe replaces unsafe library functions with safe versions, and Libverify does return address verification
- Works on executables
- Slowdown is < 15% for all tested programs

Baratloo et al.

- Weaknesses of Libverify:
  1. Only prevents attacks that overwrite the return address
  2. For technical reasons, has to copy entire function to heap on each call

References

- Aleph I. "Smashing the stack for fun and profit." http://www.phrack.org/phrack/49/P49-16
- Baratloo, Singh, and Tsai. "Transparent runtime defense against smash stacking attacks." http://www.usenix.org/publications/library/proceedings/usenix2000/general/full_papers/baratloo/baratloo.pdf
- Cowan et al. "StackGuard" http://www.cse.ogi.edu/DISC/projects/immunix/StackGuard/usenixsc98.ps.gz

References

- Etoh & Yoda. "Protecting from stack-smashing attacks." http://www.trl.ibm.com/projects/security/ssp/main.html
- Ruwase & Lam. "A practical dynamic buffer overflow detector." http://suif.stanford.edu/papers/tunji04.pdf
- Stackshield. http://www.angelfire.com/sk/stackshield