Buffer Overflow & Format Strings

Seminar on Computer Security Threats and Counter Measures

Dominik Zindel

19.01.2007

Content

Goals

Common goal of attacks using a buffer overflow or a format string attack:

- Access hidden data.
- Change program flow to execute own program.
- Get more rights than supposed to.

SUID

- Unix-flag: `-rwsr-sr-x`.
- Execute file with owners (e.g. root) right.
- Program is executed in the context of the owner, not the current user.
- Useful for tools requiring root-rights, without giving user root-rights.

Memory

- Program counter: Register indicating address of next instruction.
- Frame pointer: Pointer to current stack frame.
- FP used by functions to locate the return address.
- FP pushed into stack at the beginning of a function.

Figure: Memory, Stack

Shellcode

- Array of character-coded assembly instructions.
- Performs execve(”/bin/sh”).
- Opens a shell.
- ```
  char sc[]=
  "\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50
  \x53\x89\xe1\x99\xb0\x0b\xcd\x80";
  ```
  ```

Shellcode: Character-coded and Assembler

```
"\x31\xc0"      /* xorl    %eax,%eax       */
"\x50"          /* pushl   %eax             */
"\x68""//sh"    /* pushl   $0x68732f2f      */
"\x68""/bin"    /* pushl   $0x6e69622f      */
"\x89\xe3"      /* movl    %esp,%ebx        */
"\x50"          /* pushl   %eax             */
"\x53"          /* pushl   %ebx             */
"\x89\xe1"      /* movl    %esp,%ecx        */
"\x99"          /* cdql                     */
"\xb0\x0b"      /* movb    $0x0b,%al        */
"\xcd\x80"      /* int     $0x80            */
```

Content

Buffer Overflow

- Programming error.
- Memory access exception → program termination.
- Store data beyond the boundaries of a fixed length buffer.
- Overwrite other data.

History of Buffer Overflows

- 1988: First exploitation, Unix service fingerd.
- 1995: Rediscovered by Thomas Lopatic.
- 1996: Step-by-step introduction by ‘Aleph One’ (Elias Levy).
- 2001: Code Red Worm (MS IIS).
- 2003: SQLSlammer worm (MS SQL Server 2000).

Premises

- Programming error.
- Array in C: not first-class object but represented as pointer $\longrightarrow$ difficult verification.
- High level of control in C.

Buffer Overflow: example

```
void function(char *str) {
  char buffer[16];
  strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  function(large_string);
}
```

Stack smashing

- Most popular technique. By Aleph One.
- Impossible to write to program counter.
- Overwrite the return address, let it point to the shellcode.
- Address of buffer unknown, but guessable (debugger).
- Detailed information about target-program & run-time behaviour required.

Stack smashing: nop-sled

- Make guessing easier: nop-sled.
- Not necessary to know exact return address.
- Large array of nop instructions placed before shellcode.
- EIP returns to return address (overwritten with nop) → slide down.

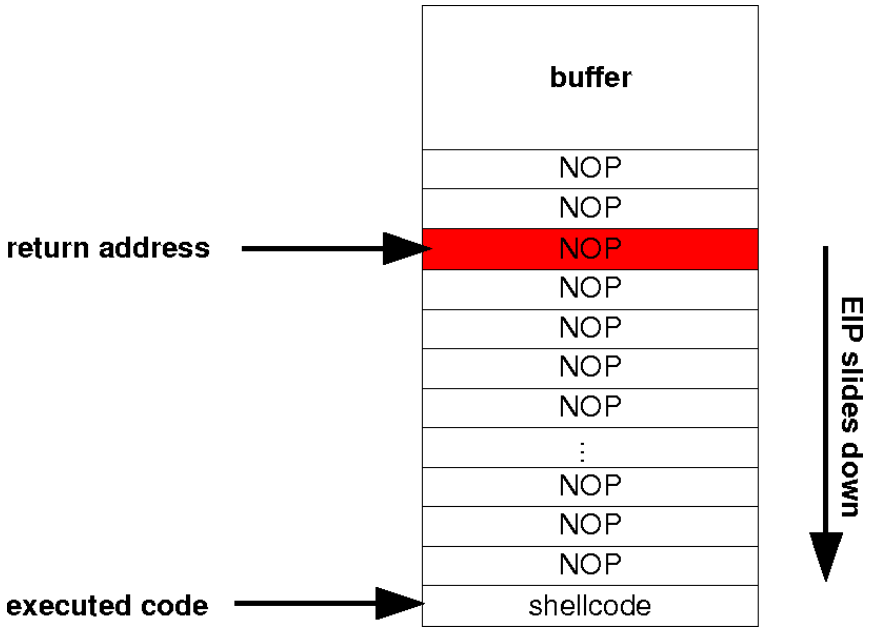


Figure: nop sled.

Frame Pointer Overwrite

- Sufficient to overflow buffer by one byte.
- Impossible to directly change frame pointer (register) $\rightarrow$ alter last byte of saved FP in the stack.
- Overflowed function returns to calling function.
- Calling function has wrong FP $\rightarrow$ pops chosen word into program counter on return.
- Chosen word: e.g. address of shellcode.
- Exact address of the buffer has to be known.

Indirect Alteration via Pointer

- Indirectly overwrite code pointers.
- Alter return address indirectly.
- Hard to discover.
- Hard to find.

```
void f (char **argv){
 char *p;
 char buffer[32];
 ...
 strcpy(buffer, argv[1]);
 strcpy(p, argv[2]);}
```

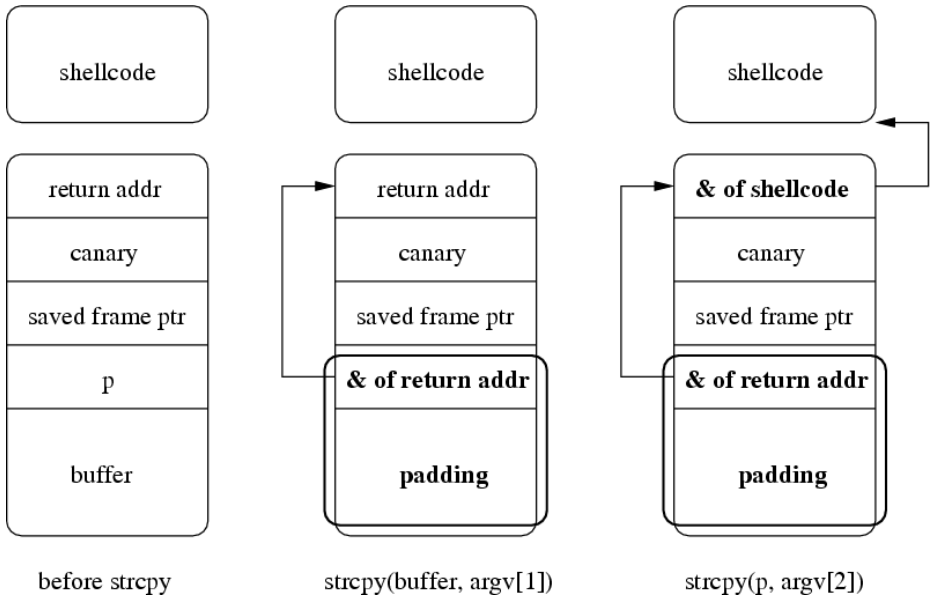


Figure: Indirectly overwrite return address

Content

Format Strings

- C functions that perform formatting, eg. `printf()`.
- `printf("Dear %s. I love you!",name)`
- Software vulnerability.
- Crash programs.
- Execute harmful code.

Basic principle

- # of directives $<$ # of parameters $\rightarrow$ underflow.
- # of directives $>$ # of parameters $\rightarrow$ overflow.
- Wrong directive $\rightarrow$ misinterpreted parameter.
- Control of format string $\rightarrow$ exploit behaviour.

Example Vulnerable & Safe Code

Vulnerable code:

```
void vulnfnc (char *user){
  ...
  printf(user);
}
```

Safe code:

```
void vulnfnc (char *user){
  ...
  printf("%s",user);
}
```

History of Format Strings

- Bug first noted in 1990: ‘interaction effect’.
- Attacks discovered around 1999/2000.
- About 380 vulnerable programs (MITRE’s CVE).
- ftpd, klogd, AOL IM 4.1.2010.
- MySQL Server 4.1 before 4.1.21 and 5.0 before 1 April 2006.

Premises

- Software vulnerability.
- Previously thought harmless.
- Unfiltered user input.
- # of arguments unknown at compile time $\rightarrow$ rely on format string.

View Memory

```
printf("%08x %08x %08x %08x %08x");
```

- Prints the values of 5 consecutive words on the stack.
- 8-digit padded hexadecimal number.
- View content of the stack.

Overwriting a Word

- %n directive: # of characters written by the format function so far.
- Number is written to location pointed by parameter.

```
printf("\x10\x01\x48\x08%08x%08x%08x%08x%08x%n");
```

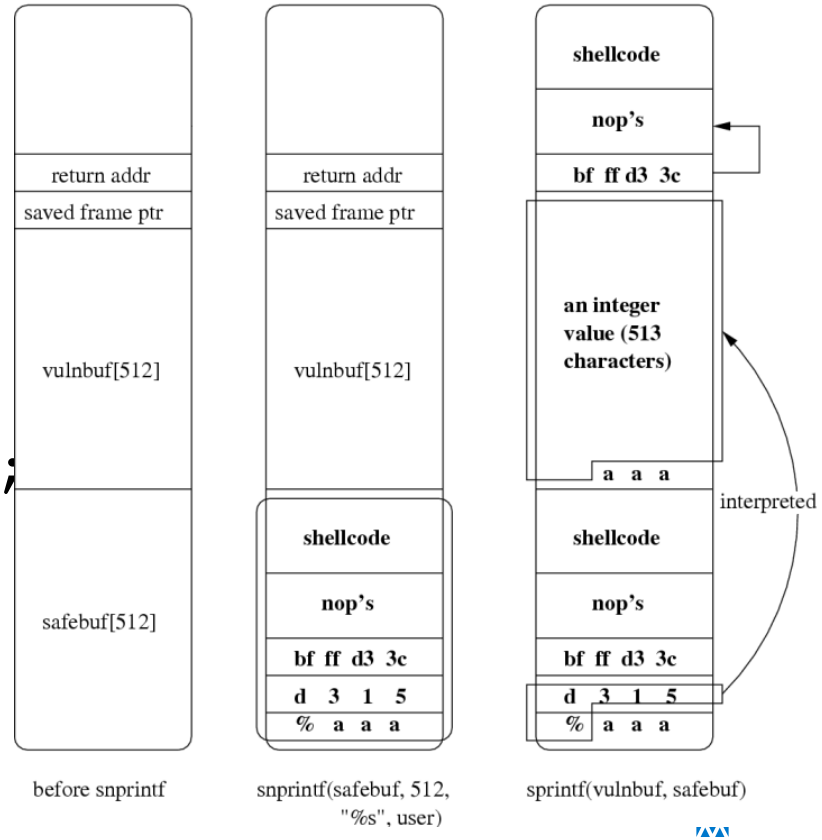
- Write small int at `0x08480110`.
- 44: 4 (`\x10\x01\x48\x08`) + 8 characters per `%08x`.
- Control number of characters written = value written at `0x08480110`

Minimum Field Width Specifier

- Overflow a buffer & overwrite the return address with address of a code pointer.
- Similar to stack-smashing attack.
- Overflow buffer using minimum field width specifier of a format directive.

Example

```
void func(char *user){
 char safebuf[512];
 char vulnbuf[512];
 snprintf(safebuf,512,"%s",user);
 sprintf(vulnbuf,safebuf);}
```



Example continued

- `safebuf`: cannot be overflowed (`snprintf`) → stack-smashing attack useless.
- safebuf = format string of `sprintf` → interpreted before copied to `vulnbuf`.
- If `safebuf` contains `"%513d"` → 513 characters written to `vulnbuf` → overflow → overwrite return address to address of shellcode.

Content

Conclusion

- Different but still similar attacks.
- Main point: overwrite return address.
- Knowledge about systems required.
- Big effects on system.

Questions, Remarks

References I

- Kyung-Suk Lhee, Steve J. Chapin: Buffer Overflow and Format String Overflow Vulnerabilities. 2002. http://citeseer.ist.psu.edu/lhee02buffer.html
- murat@enderunix.org: Buffer overflows demystified. http://www.enderunix.org/documents/eng/bof-eng.txt
- Lefty: Buffer Overruns, whats the real story?. http://doc.bughunter.net/buffer-overflow/buffer-overrun.html
- SecuriTeam: Writing Buffer Overflow Exploits - a Tutorial for Beginners. 2002. http://www.securiteam.com/securityreviews/5OP0B006UQ.html

References II

- Wikipedia: Buffer Overflow. http://en.wikipedia.org/wiki/Buffer_overflow
- Wikipedia: Format string attack. http://en.wikipedia.org/wiki/Format_string_attacks
- scut/team teso: Exploiting Format String Vulnerabilities v1.1. 2001. http://julianor.tripod.com/teso-fs1-1.pdf
- SANS institute: Inside the Buffer Overflow Attack:Mechanism, Method, & Prevention. 2002. http://www.sans.org/reading_room/whitepapers/securecode/386.php

References III

- Przemyslaw Frasunek: WUFTPD 2.6.0 remote root exploit. 2000. http://marc.theaimsgroup.com/?l=bugtraq&m=96179429114160&w=2
- Aleph One: Smashing The Stack For Fun And Profit. http://doc.bughunter.net/buffer-overflow/smash-stack.html
- CVE: Common Vulnerabilities and Exposures. http://cve.mitre.org/
- Klog: The Frame Pointer Overwrite. http://doc.bughunter.net/buffer-overflow/frame-pointer.html
- David Litchfield: Windows 2000 Format String Vulnerabilities. http://www.nextgenss.com/papers/win32format.doc