Buffer Overflow

String Library Code

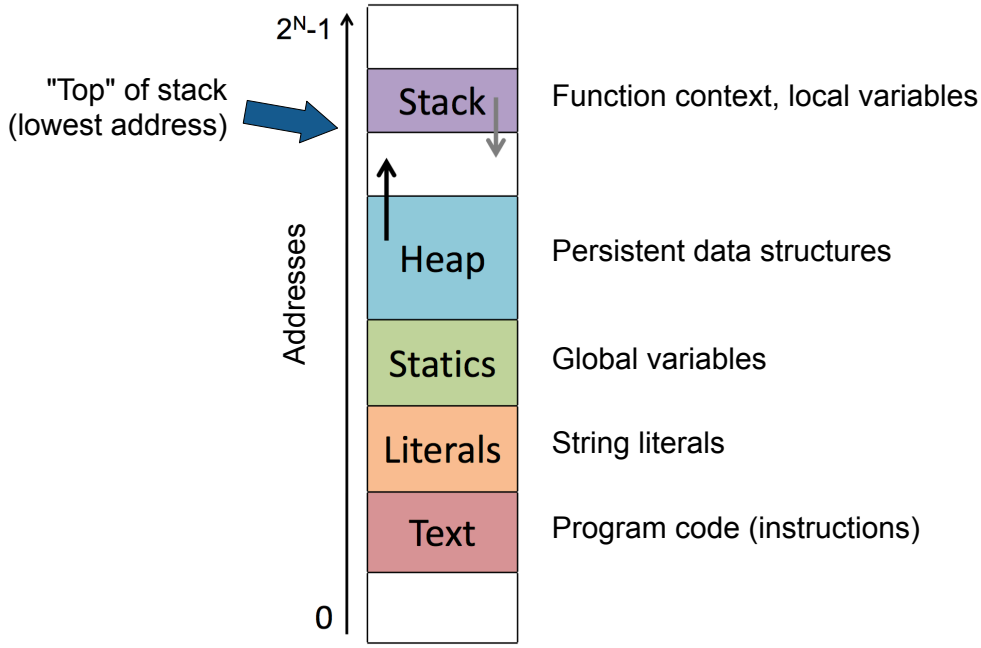
```
/* Get string input, store to str */
char* gets(char* str) {
    int c = getchar();
    char* p = str;
    while (c != '\n' && c != EOF) {
        *p++ = c;
        c = getchar();
    }
    *p = '\0';
    return str;
}
```

end-of-file (input) marker

```
char* strcpy(char* dest, char* src);  // copy string
// see also: strcat, scanf, fscanf, sscanf, ...
```

```
char* fgets(char* str, int n, FILE* stream);
char* strncpy(char* dest, char* src, size_t n);
```

Memory Layout (Segments, redux)



Vulnerable Buffer Code

```
/* Echo Line */
void echo() {
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
void call_echo() {
    echo();
}
```

```
unix>./buftest
Type a string:0123456789012345678901234
0123456789012345678901234
```

```
unix>./buftest
Type a string:012345678901234567890123
Segmentation Fault
```

Buffer Overflow Assembly

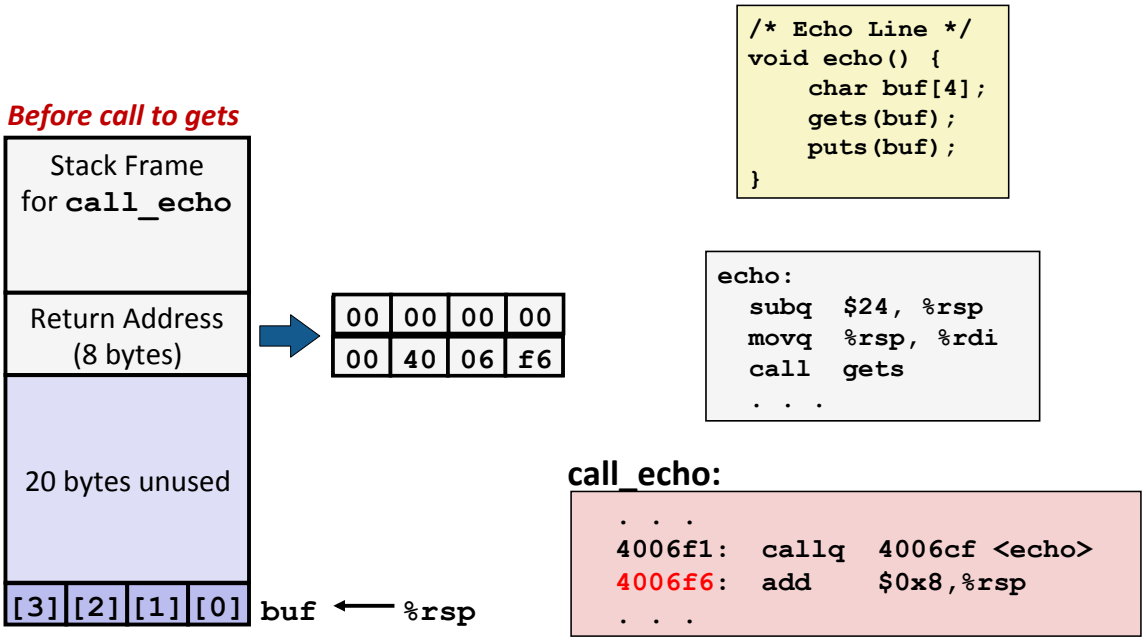
echo:

```
00000000004006cf <echo>:
 4006cf:  48 83 ec 18           sub    $0x18,%rsp
 4006d3:  48 89 e7              mov    %rsp,%rdi
 4006d6:  e8 a5 ff ff ff        callq  400680 <gets>
 4006db:  48 89 e7              mov    %rsp,%rdi
 4006de:  e8 3d fe ff ff        callq  400520 <puts@plt>
 4006e3:  48 83 c4 18           add    $0x18,%rsp
 4006e7:  c3                    retq
```

call_echo:

```
 4006e8:   48 83 ec 08          sub    $0x8,%rsp
  4006ec:  b8 00 00 00 00       mov    $0x0,%eax
  4006f1:  e8 d9 ff ff ff       callq  4006cf <echo>
  4006f6:  48 83 c4 08          add    $0x8,%rsp
  4006fa:  c3                   retq
```


Buffer Overflow Stack



Buffer Overflow Examples

After call to gets

Stack Frame for `call_echo`

00	00	00	00
00	40	06	f6
00	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

`buf` ← `%rsp`

```
unix>./buftest
Type a string:012345678901234567890123
012345678901234567890123
```

Overflowed, but did not corrupt state

After call to gets

Stack Frame for `call_echo`

00	00	00	00
00	40	00	34
33	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

`buf` ← `%rsp`

```
unix>./buftest
Type a string:0123456789012345678901234
Segmentation Fault
```

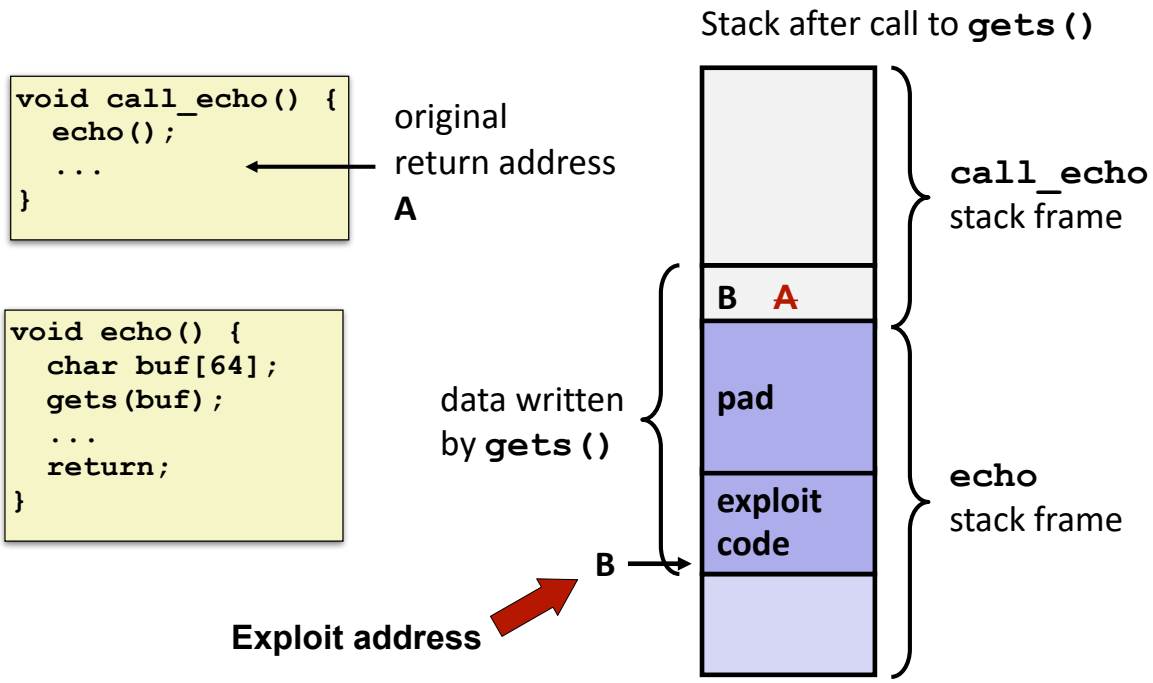
Overflowed and corrupted return pointer

Character Representations (redux)

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Code Injection Attacks



Best Practices

Use length-limited string routines (**fgets**, **strncpy**, ...)

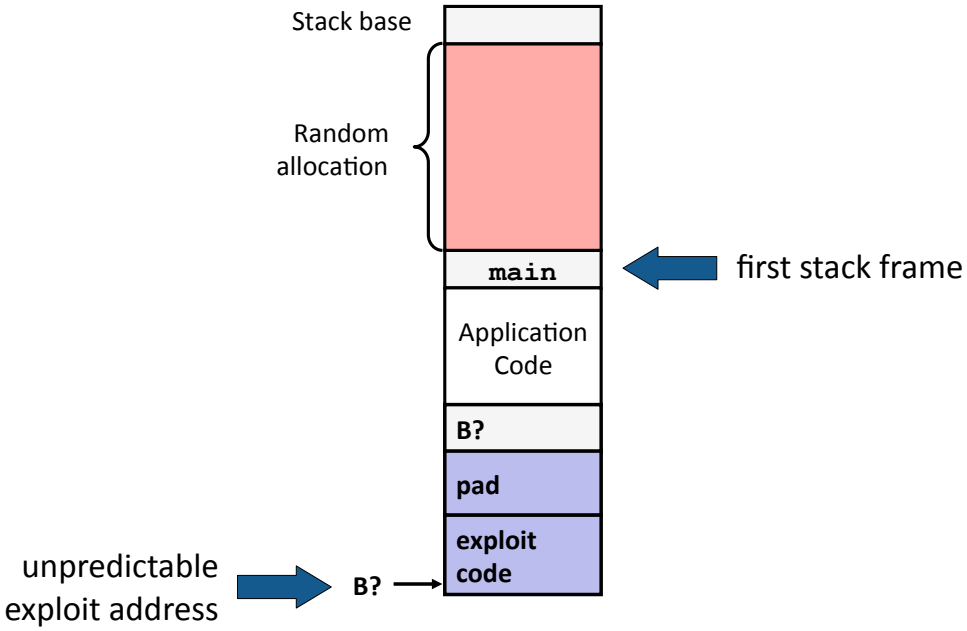
No **%s** in **scanf**

... etc ...

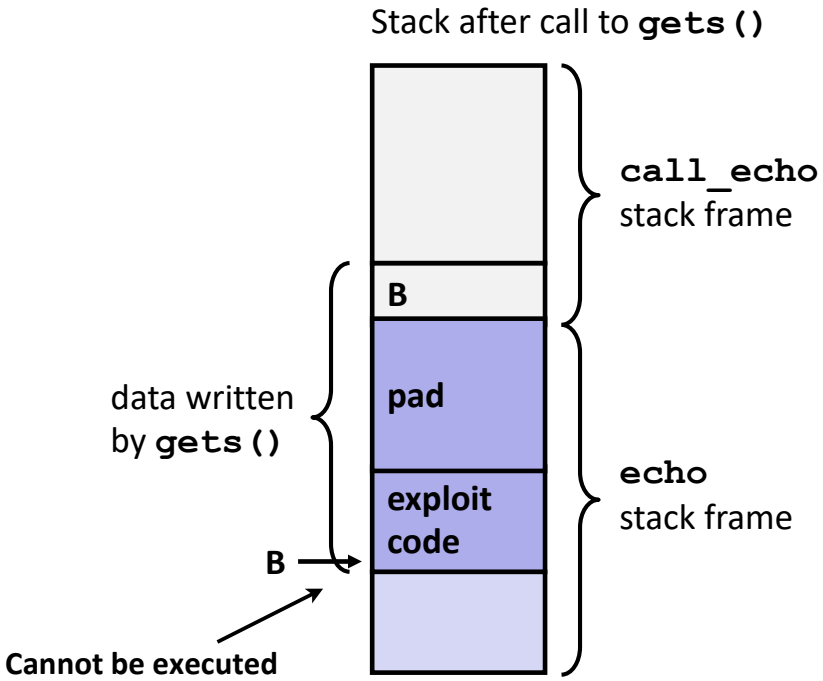
Don't make any mistakes...

```
/* Echo Line */
void echo() {
    char buf[64];
    fgets(buf, 64, stdin); // safe
    puts(buf);
}
```

Stack Randomization (ASLR)

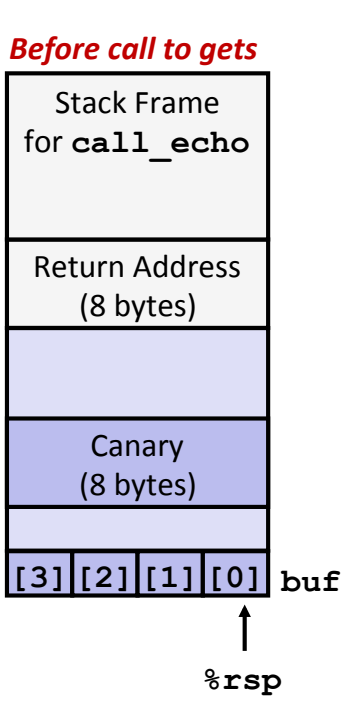


Non-Executable Memory Segments



Stack Canaries

Stack Canary Example

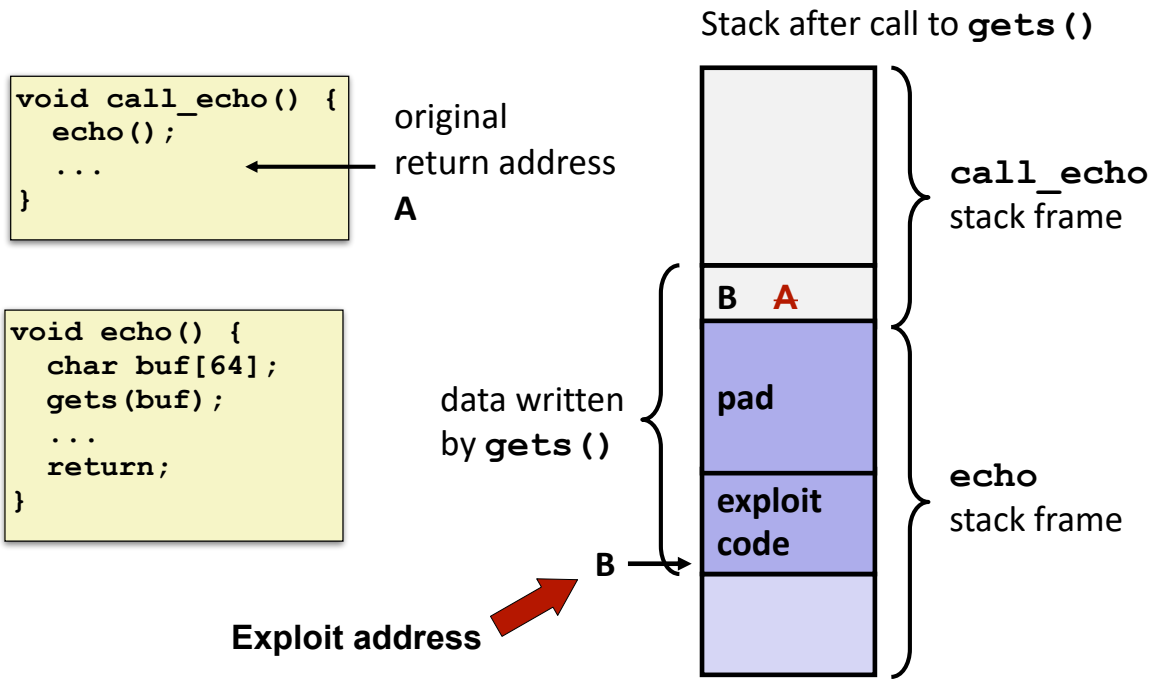



```
/* Echo Line */
void echo() {
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

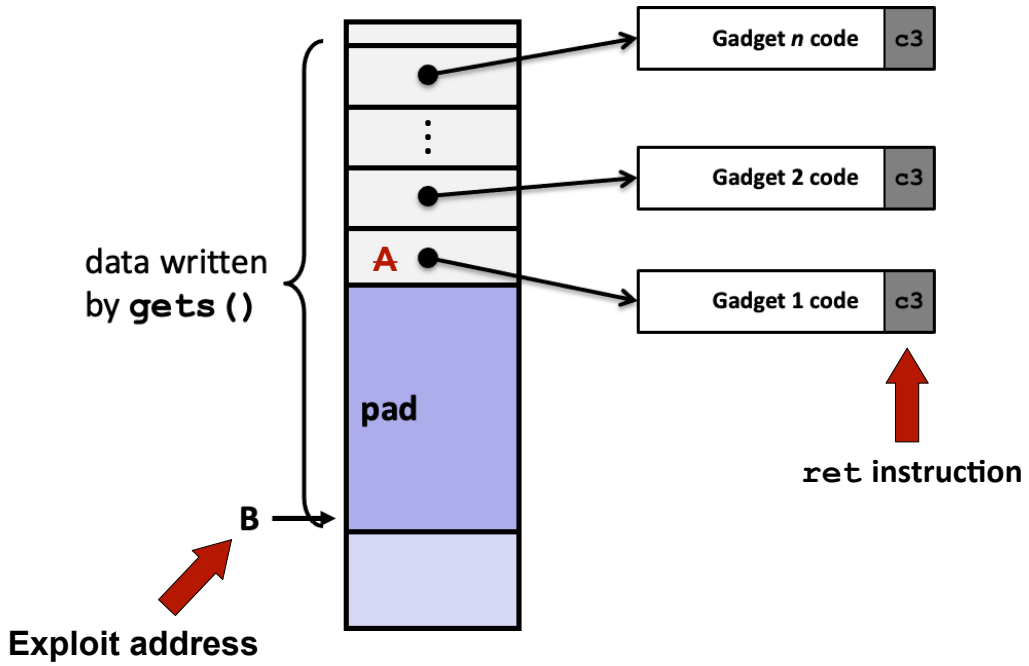
echo:

```
  40072f:  sub    $0x18,%rsp
  400733:  mov    %fs:0x28,%rax   # Get canary
  40073c:  mov    %rax,0x8(%rsp)  # Put on stack
  400741:  xor    %eax,%eax
  400743:  mov    %rsp,%rdi
  400746:  callq  4006e0 <gets>
  40074b:  mov    %rsp,%rdi
  40074e:  callq  400570 <puts@plt>
  400753:  mov    0x8(%rsp),%rax  # Retrieve canary
  400758:  xor    %fs:0x28,%rax   # Check canary
  400761:  je     400768 <echo+0x39>
  400763:  callq  400580 <__stack_chk_fail@plt>
  400768:  add    $0x18,%rsp
  40076c:  retq
```

Code Injection Attacks (redux)



Return-Oriented Programming (ROP)



Partial Functions as Gadgets

```
long ab_plus_c(long a, long b, long c) {
   return a * b + c;
}
```

```
00000000004004d0 <ab_plus_c>:
  4004d0:  48 0f af fe  imul %rsi,%rdi
  4004d4:  48 8d 04 17  lea (%rdi,%rdx,1),%rax
  4004d8:  c3           retq
```

rax ← rdi + rdx

Gadget address = 0x4004d4

Partial Instructions as Gadgets

```
void setval(unsigned *p) {
    *p = 3347663060u;
}
```

Encodes `movq %rax, %rdi`

```
<setval>:
  4004d9:  c7 07 d4 48 89 c7  movl  $0xc78948d4,(%rdi)
  4004df:  c3                 retq
```

rdi ← rax

Gadget address = 0x4004dc