IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

Lecture 37: Security II

Agenda

- Recap
- Security II
  - Buffer Overflow
  - Basic Attacks
  - Basic Defenses

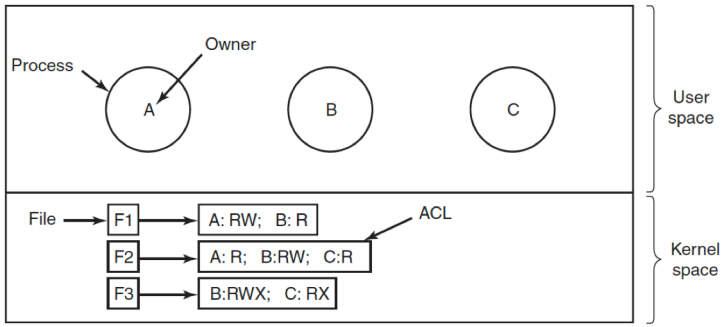
Recap

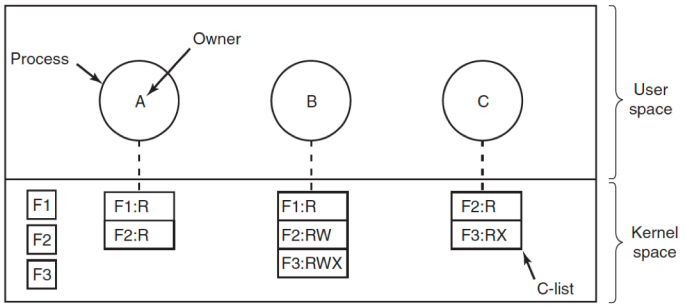
Goal	Threat
Confidentiality	Exposure of data
Integrity	Tampering with data
Availability	Denial of service

- Classic Goals & Threats
  - Confidentiality
    - Having secrete data remain secret
  - Integrity
    - Unauthorized users should not be able to modify data
  - Availability
    - Nobody can disturb the system to make it unusable
- Trusted Computing Base (TCB)
  - the (minimal) set of hardware and software necessary for enforcing all security rules

Recap

- Access Control List (ACL)
  - associate with each object an (ordered) list containing all the domains that may access the object, and how
- Capability List (C-list)
  - associated with each user/process a list of objects that may be accessed
    - individual items on a C-list are called **capabilities**





Recap

- Cryptography
  - Secret-Key Cryptography
    - Sender and receiver must both be in possession of the shared secret key
  - Public-Key Cryptography
    - Everyone picks a (public key, private key) pair and publishes the public key
      - Public key: encryption key
      - Private key: decryption key
    - User X sends a secret message to User Y
      - X encrypts the message using Y's public key
      - Y decrypts the message using Y's secret key
- Digital Signature

Recap

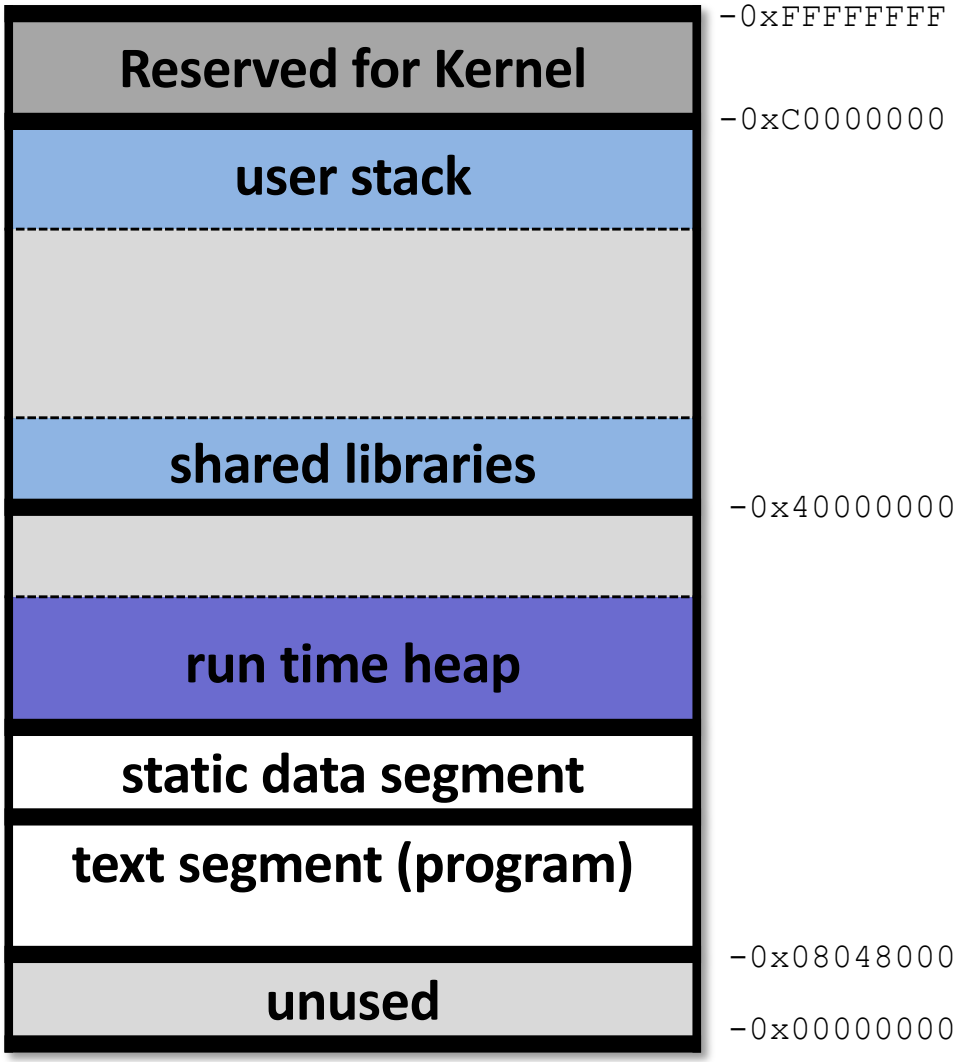
- Authentication
  - Prove the identity of a user
  - Based on three general principles:
    - Something the user knows.
    - Something the user has.
    - Something the user is
  - Common methods
    - Password
    - Hardware token
    - Software token
    - Biometrics
    - Multi-factor authentication

Agenda

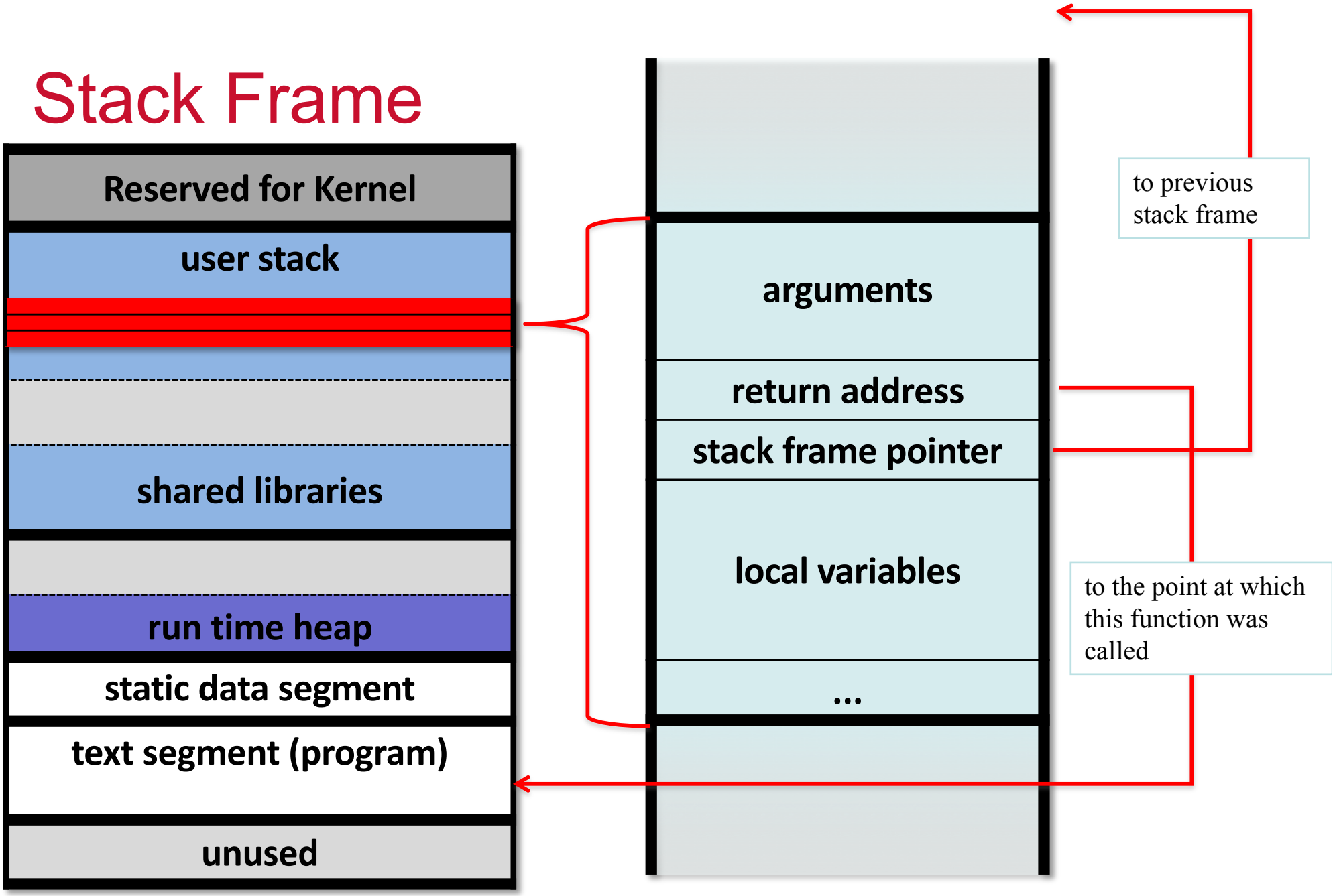
- ~~Recap~~
- Security II
  - Buffer Overflow
  - Basic Attacks
  - Basic Defenses

Linux Virtual Address Space (32-bit 86x)

- Two regions:
  - Kernel space + User space
  - configurable
    - default:
      - 1GB kernel + 3GB user



Stack Frame


Reserved for Kernel
user stack
shared libraries
run time heap
static data segment
text segment (program)
unused
arguments
return address
stack frame pointer
local variables
...
to previous stack frame
to the point at which this function was called

Buffer Overflow

- Source Code
  - “parse.c”
    - read an input file and convert upper case letters to lower case

```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}
```

```
 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
13:   if (cmd[0] == 'G')
14:     if (cmd[1] == 'E')
15:       if (cmd[2] == 'T')
16:         if (cmd[3] == ' ')
17:           header_ok = 1;
18:   if (!header_ok) return -1;
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }
```

Buffer Overflow

- How to covert to lower case: ASCII Code
  - a numerical representation of a character/symbol/action in computer
    - A: 0x41
    - a: 0x61
    - G: 0x47
    - E: 0x45
    - T: 0x54
    - Space: 0x20

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

Buffer Overflow

parse.c

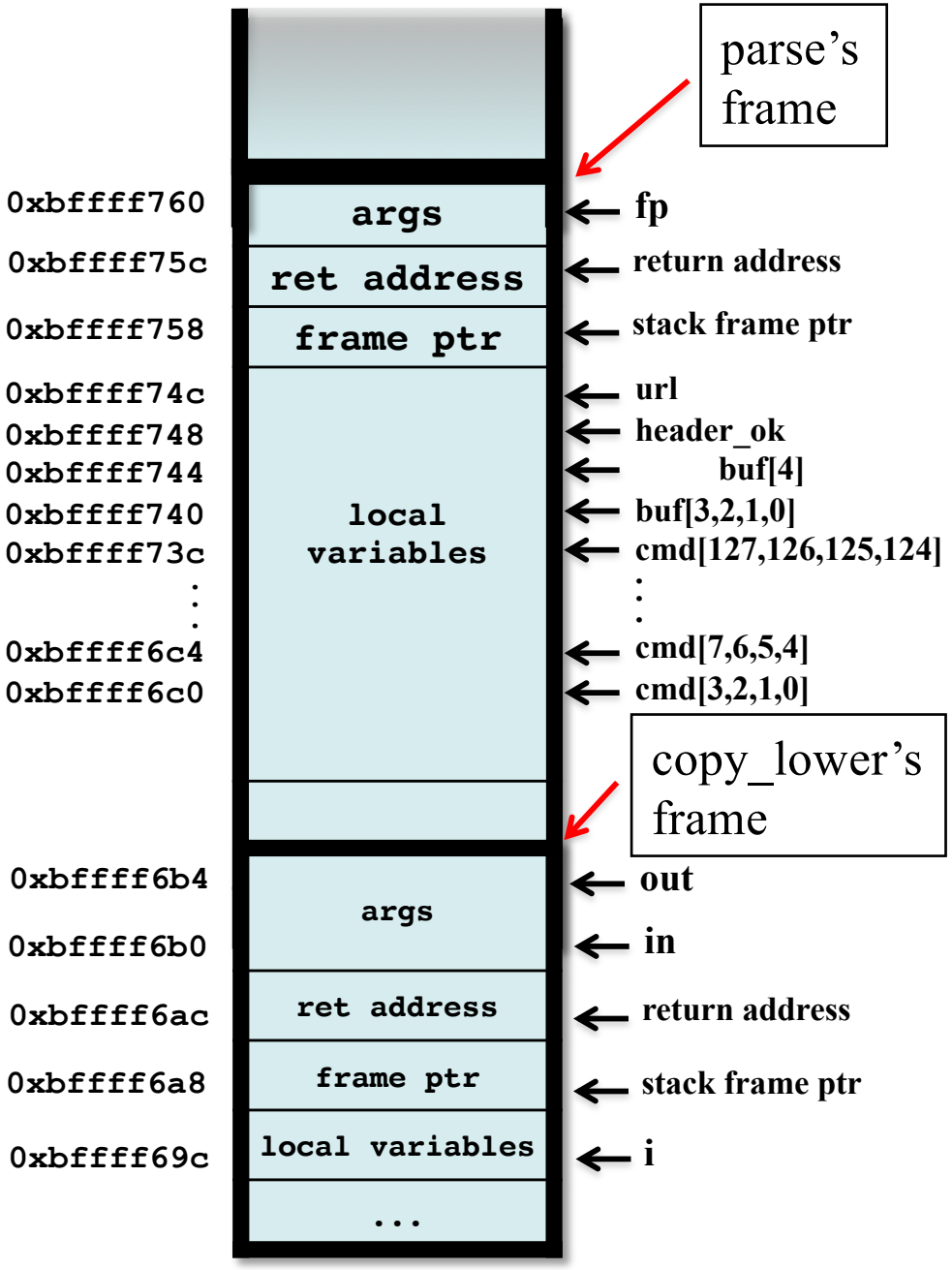
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
:
:
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

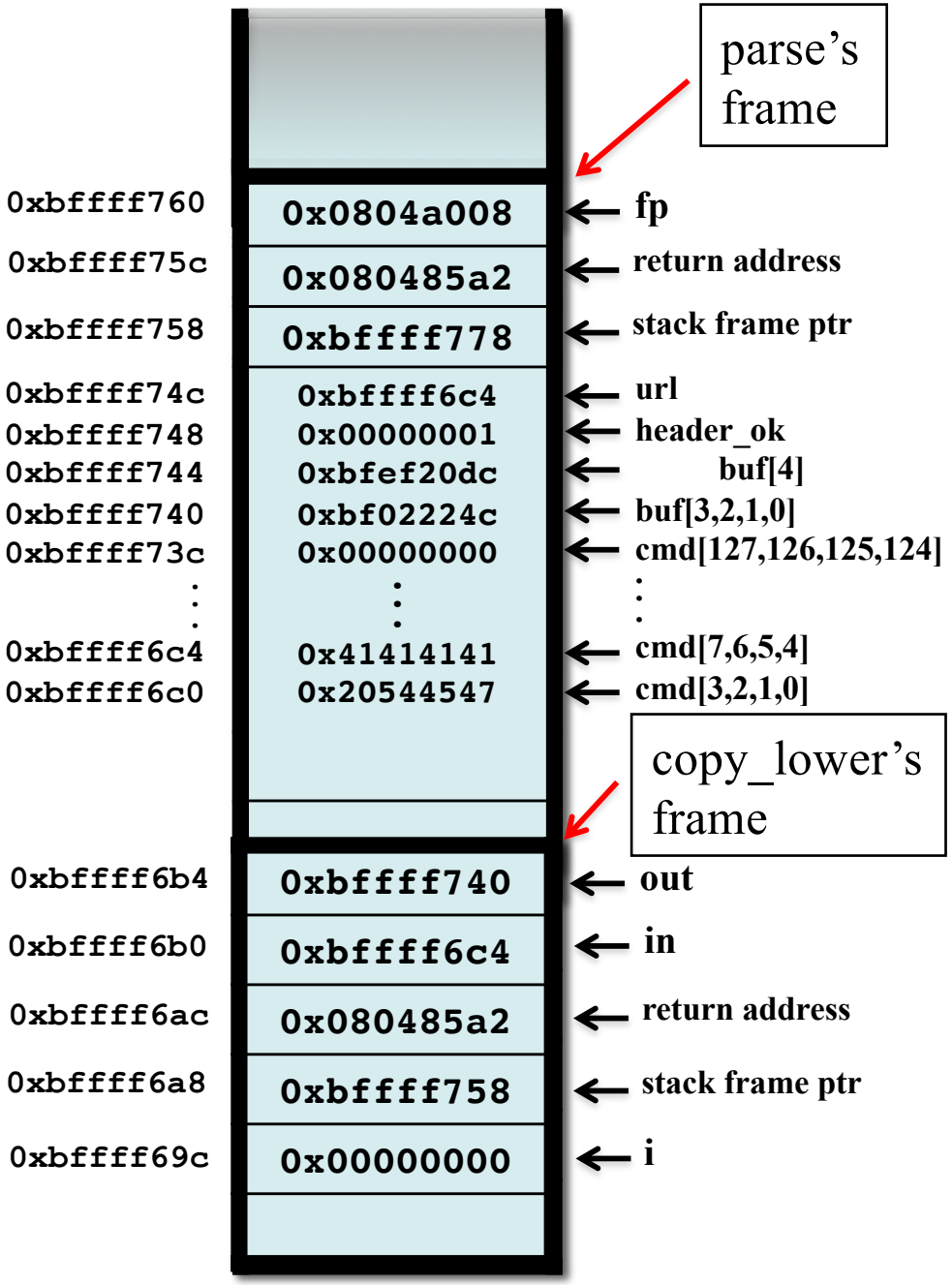
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
:
:
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

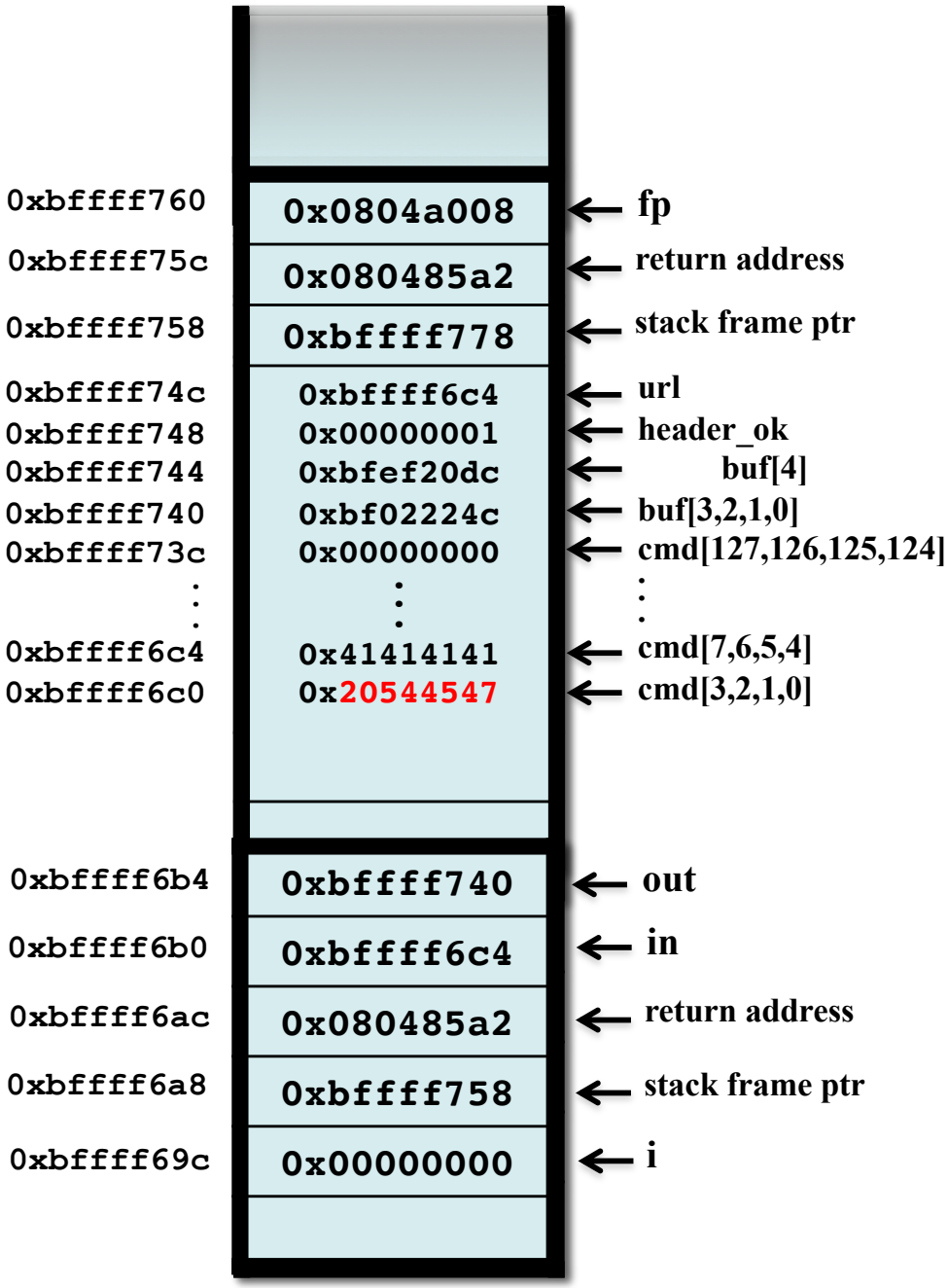
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

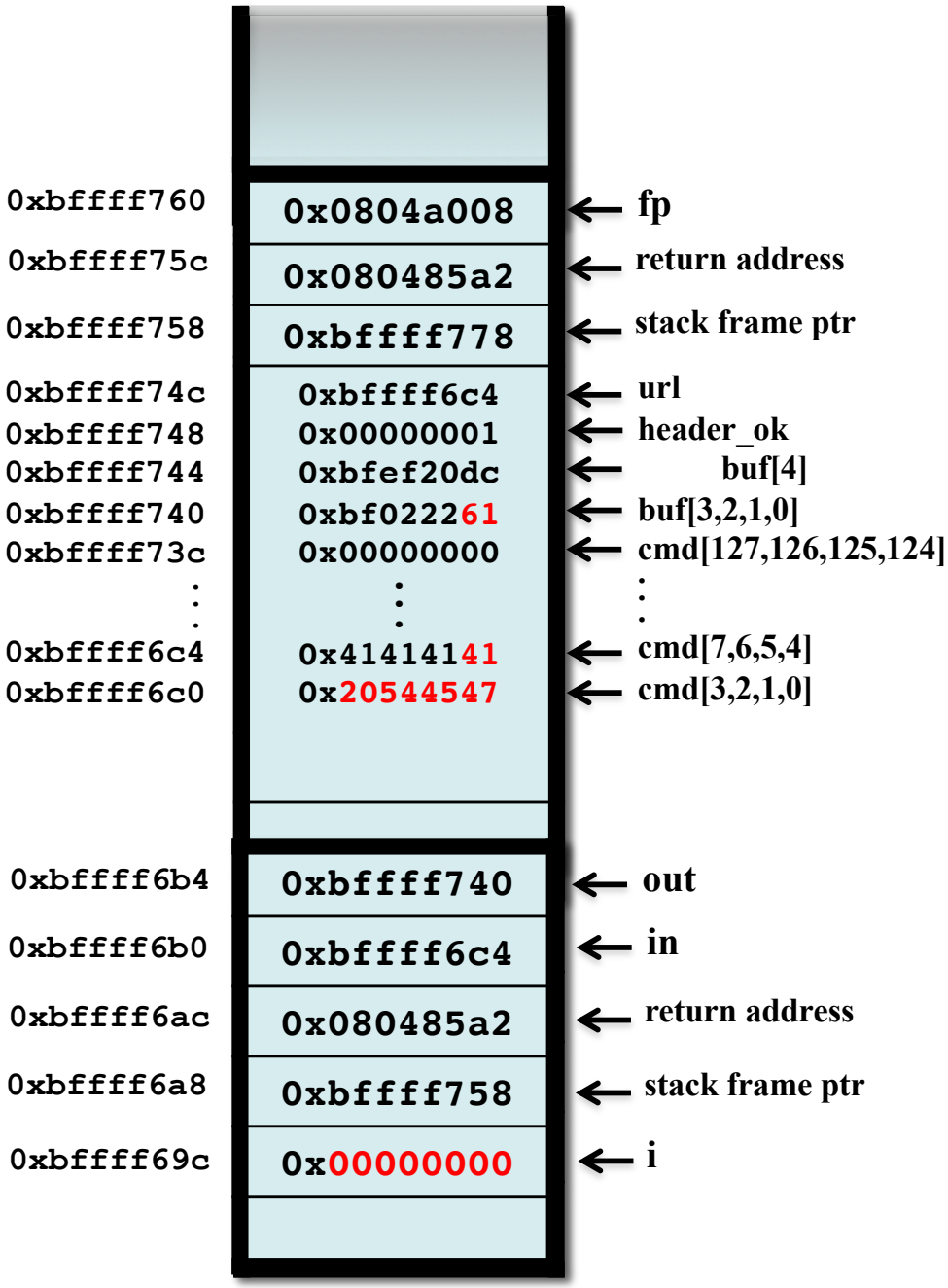
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

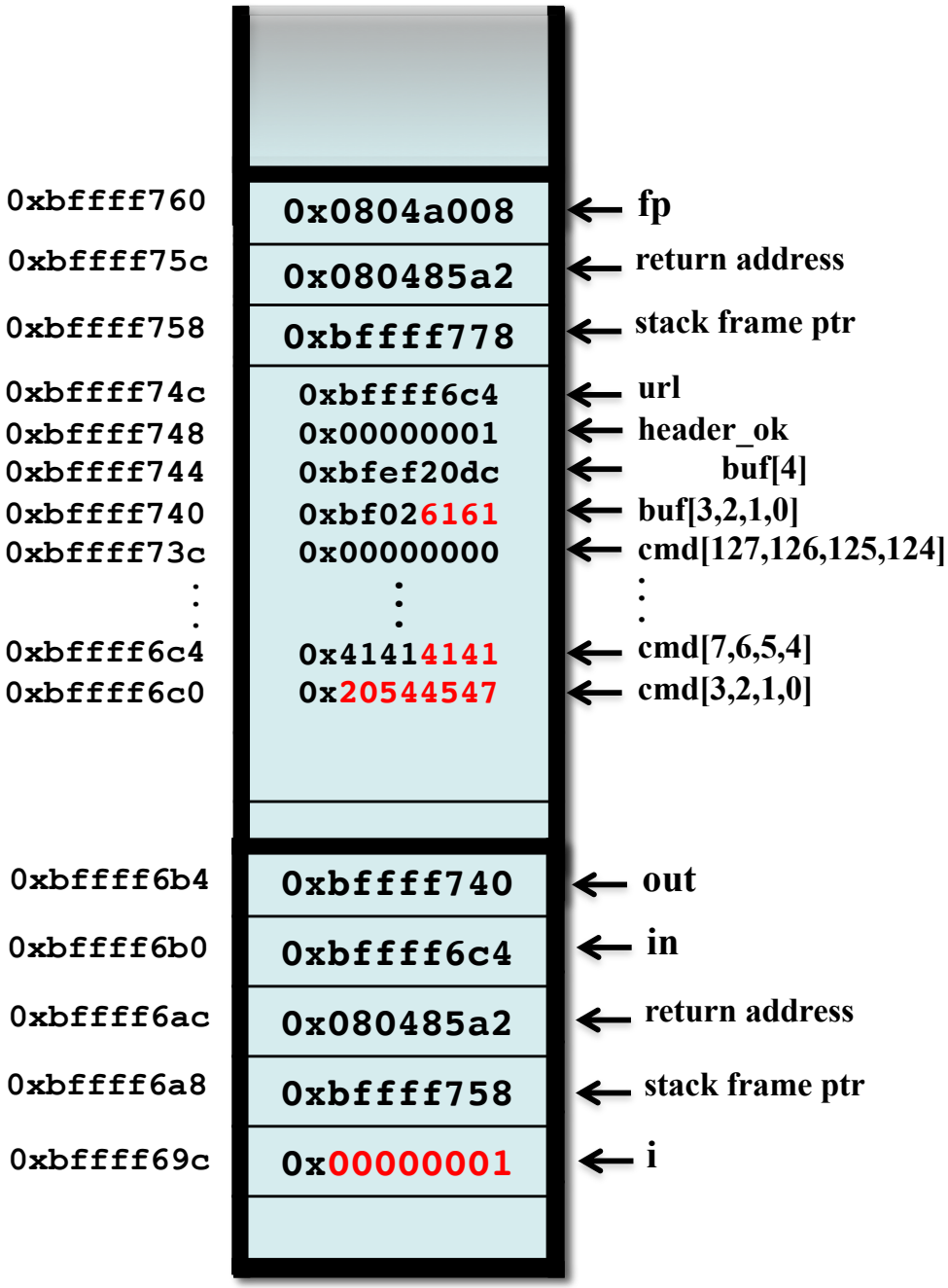
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

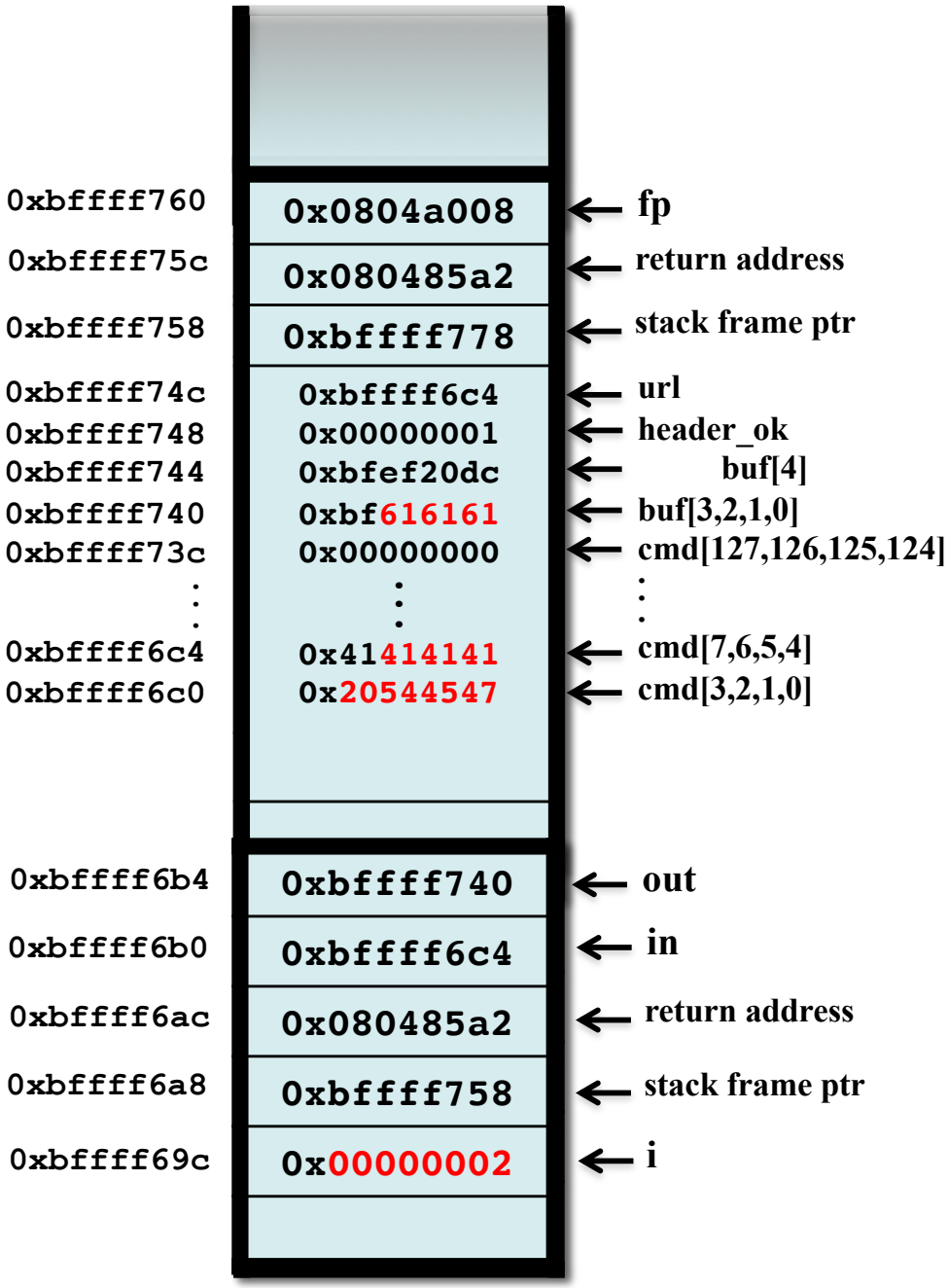
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

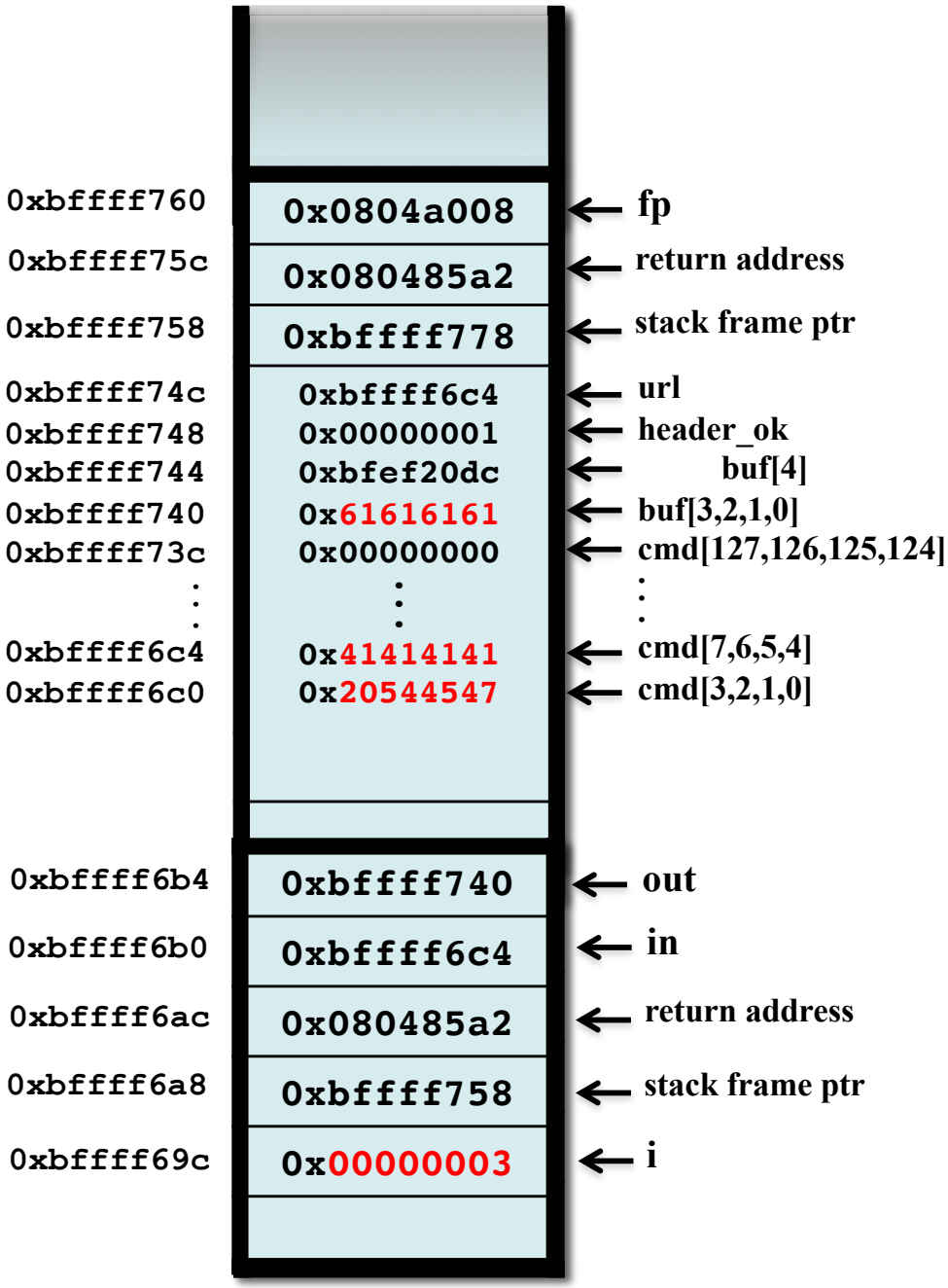
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

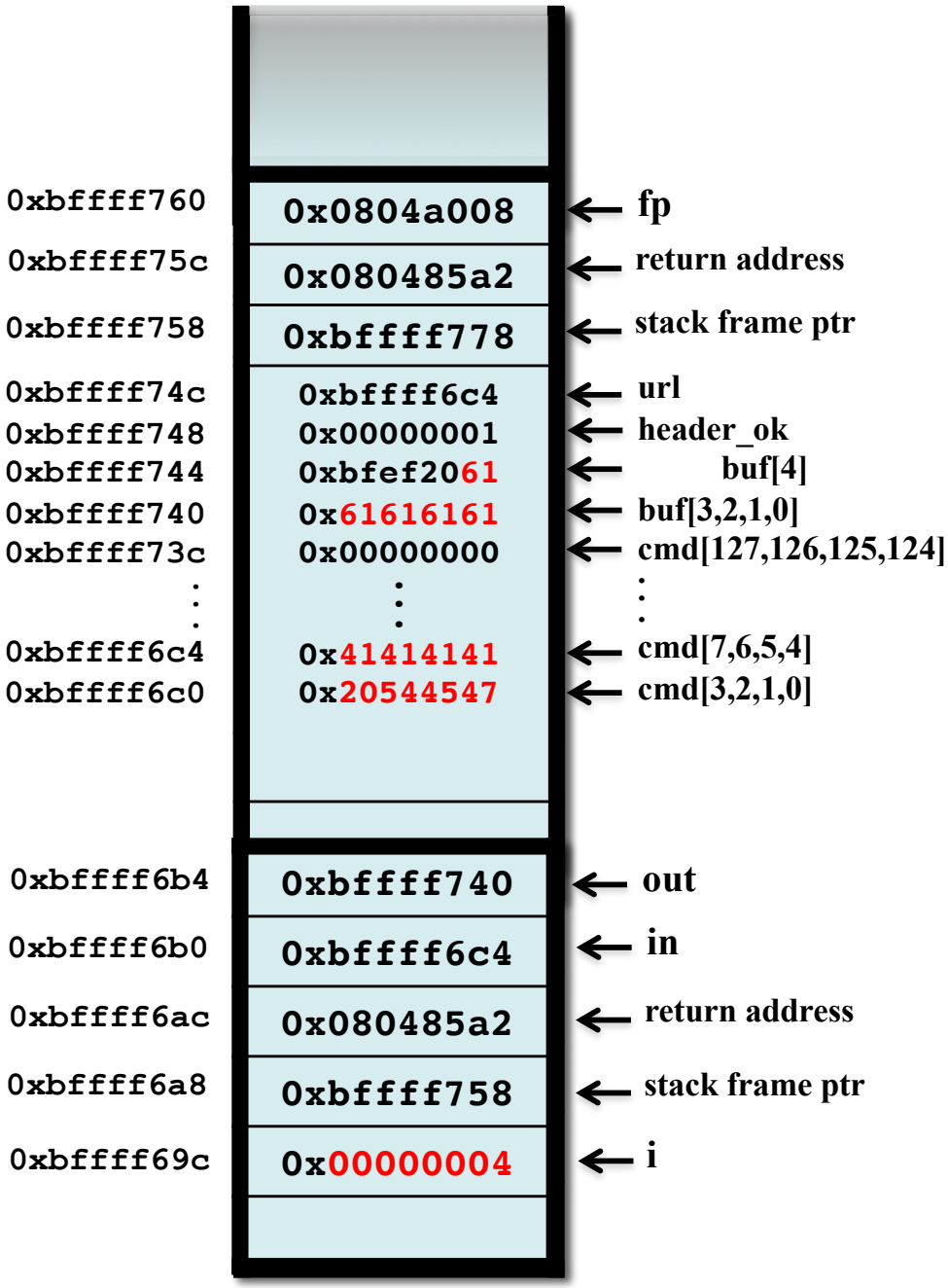
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

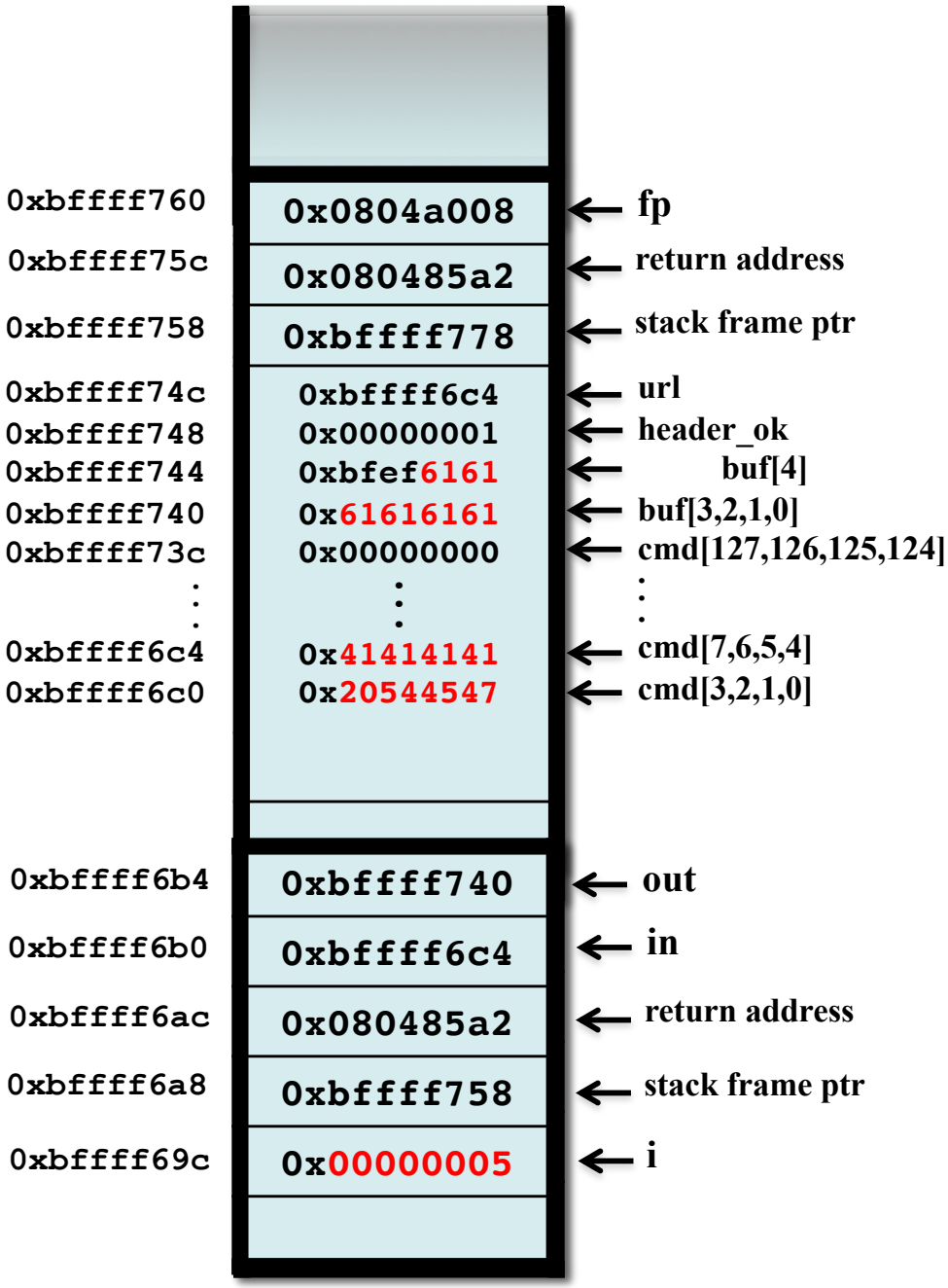
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

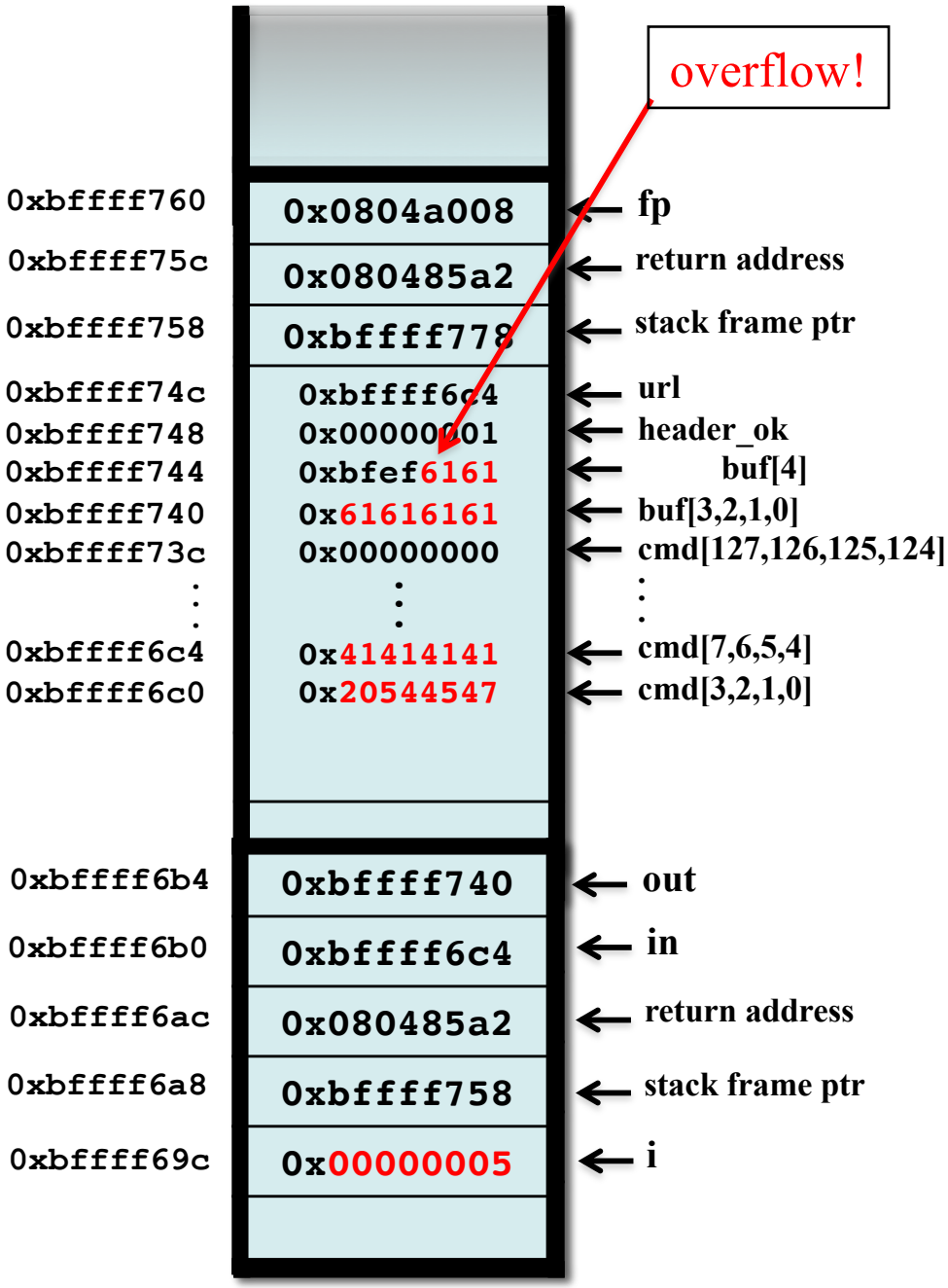
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

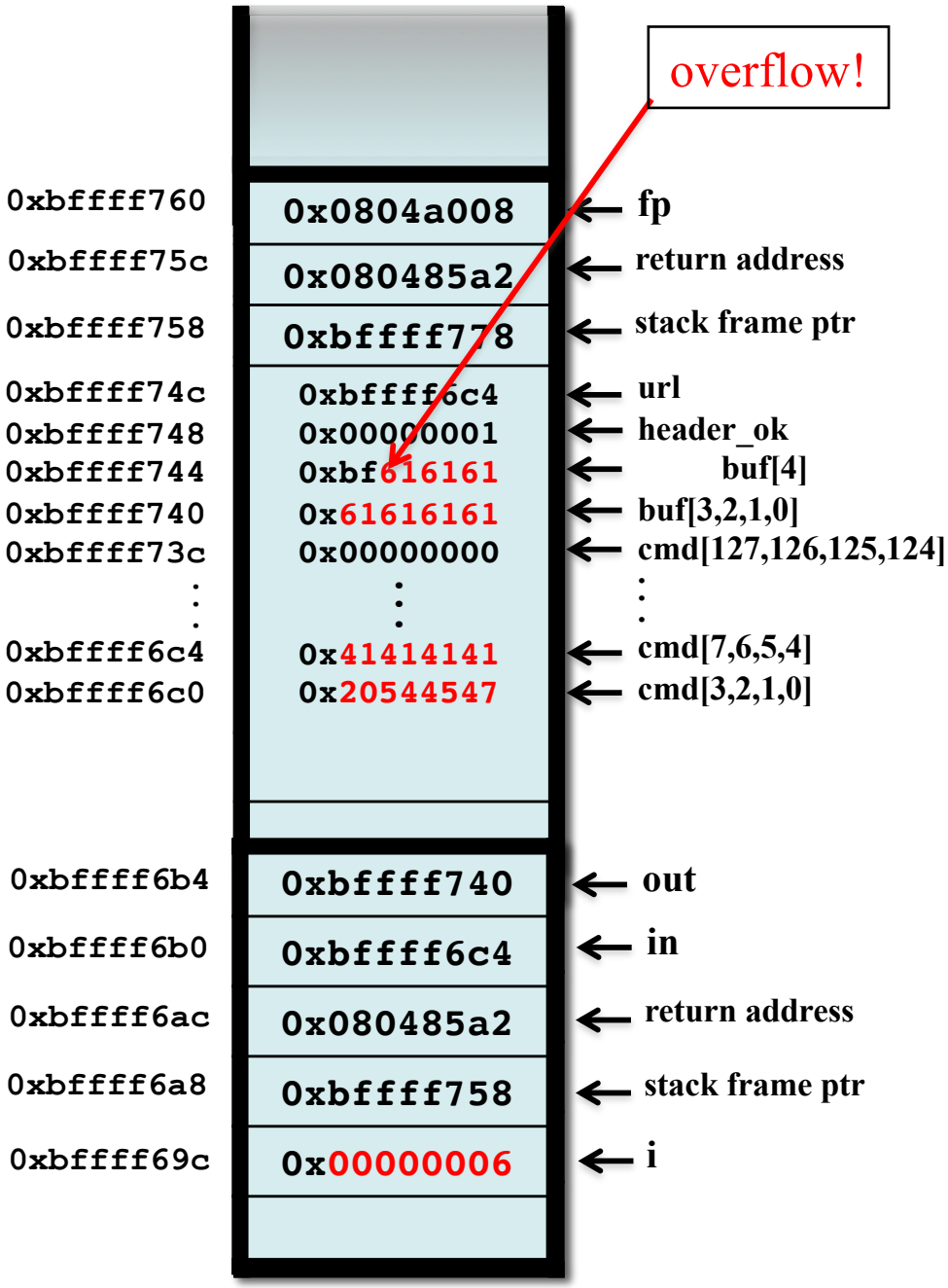
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
 .
 .
 .
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

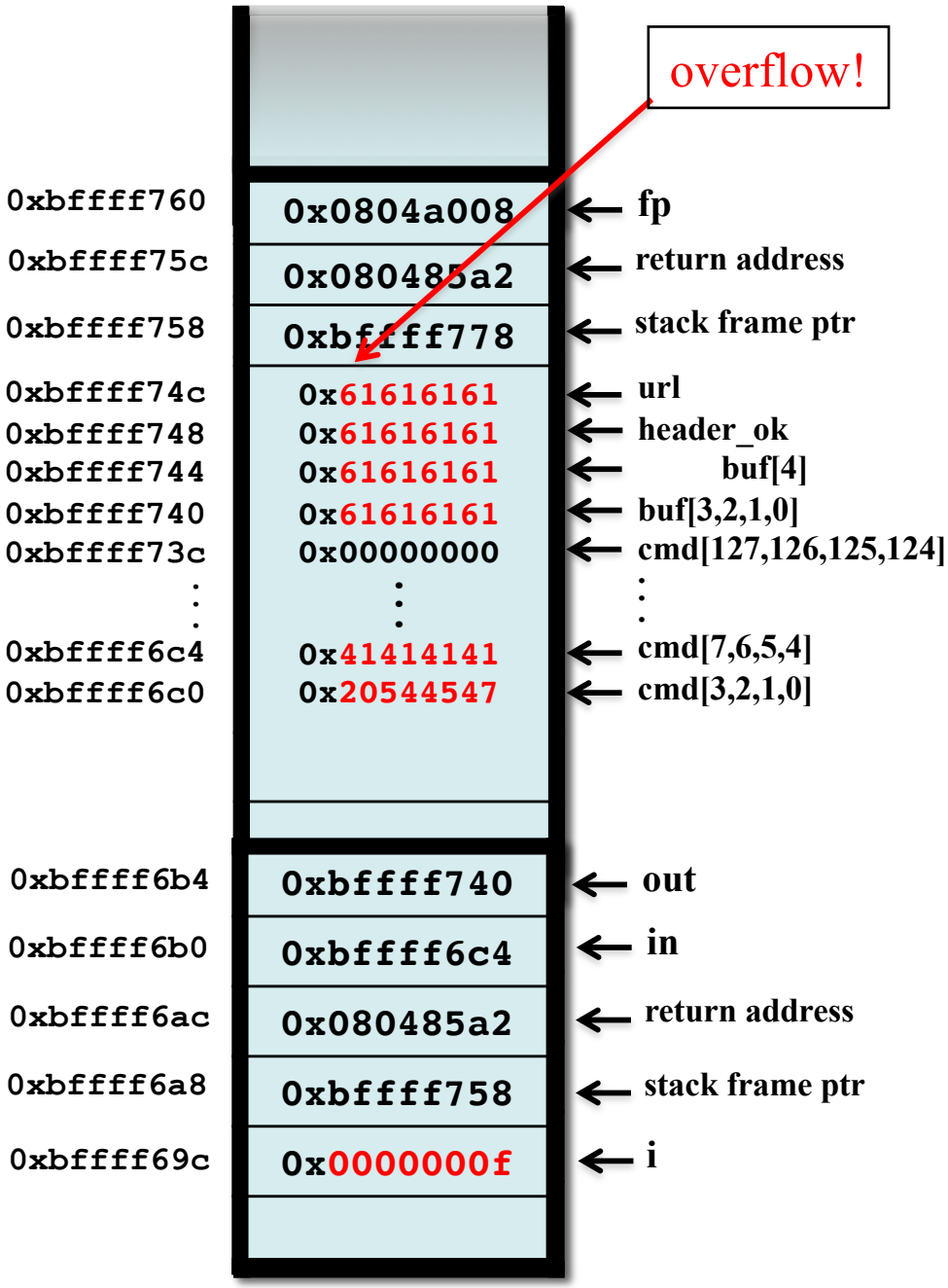
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

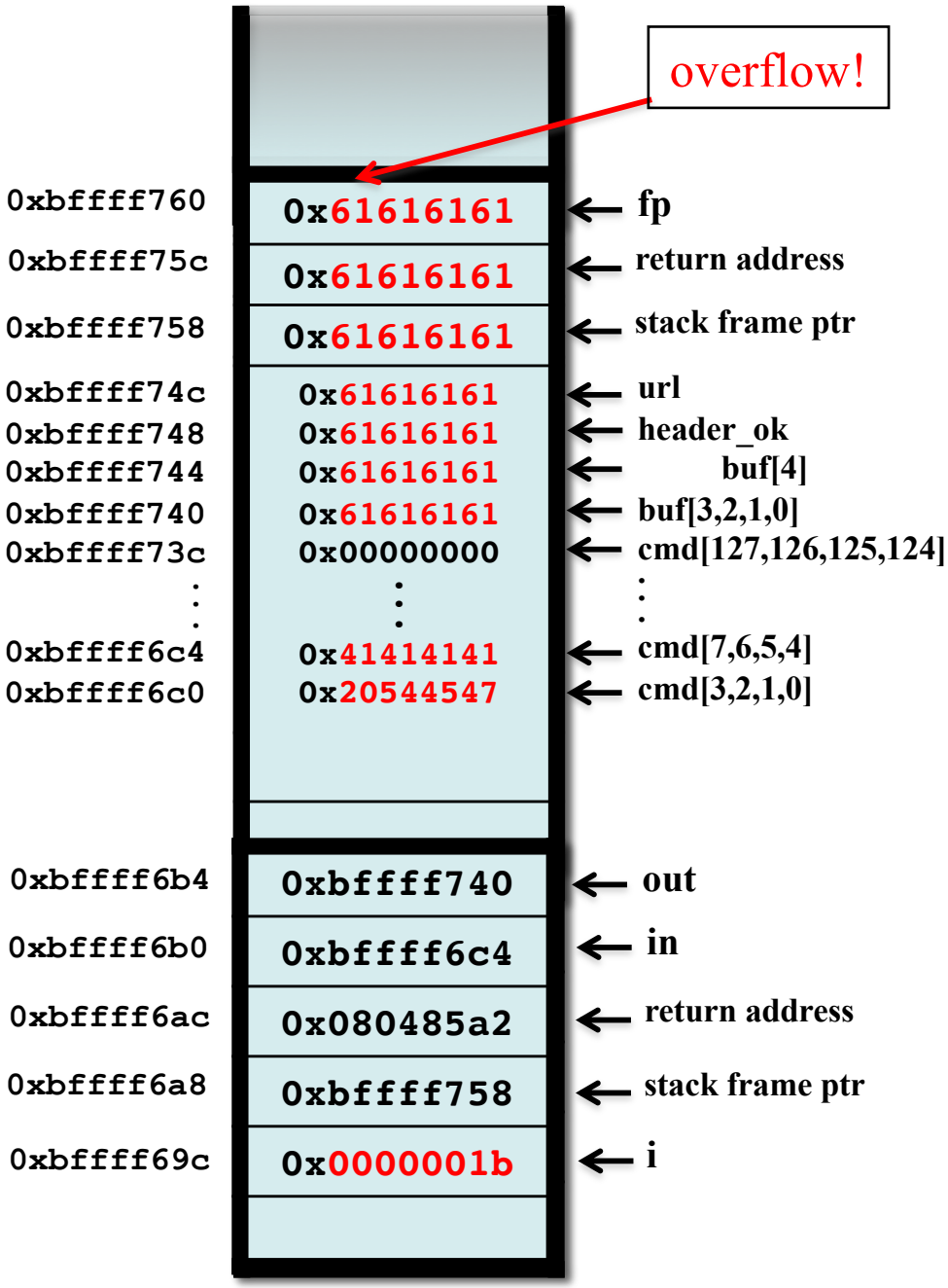
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

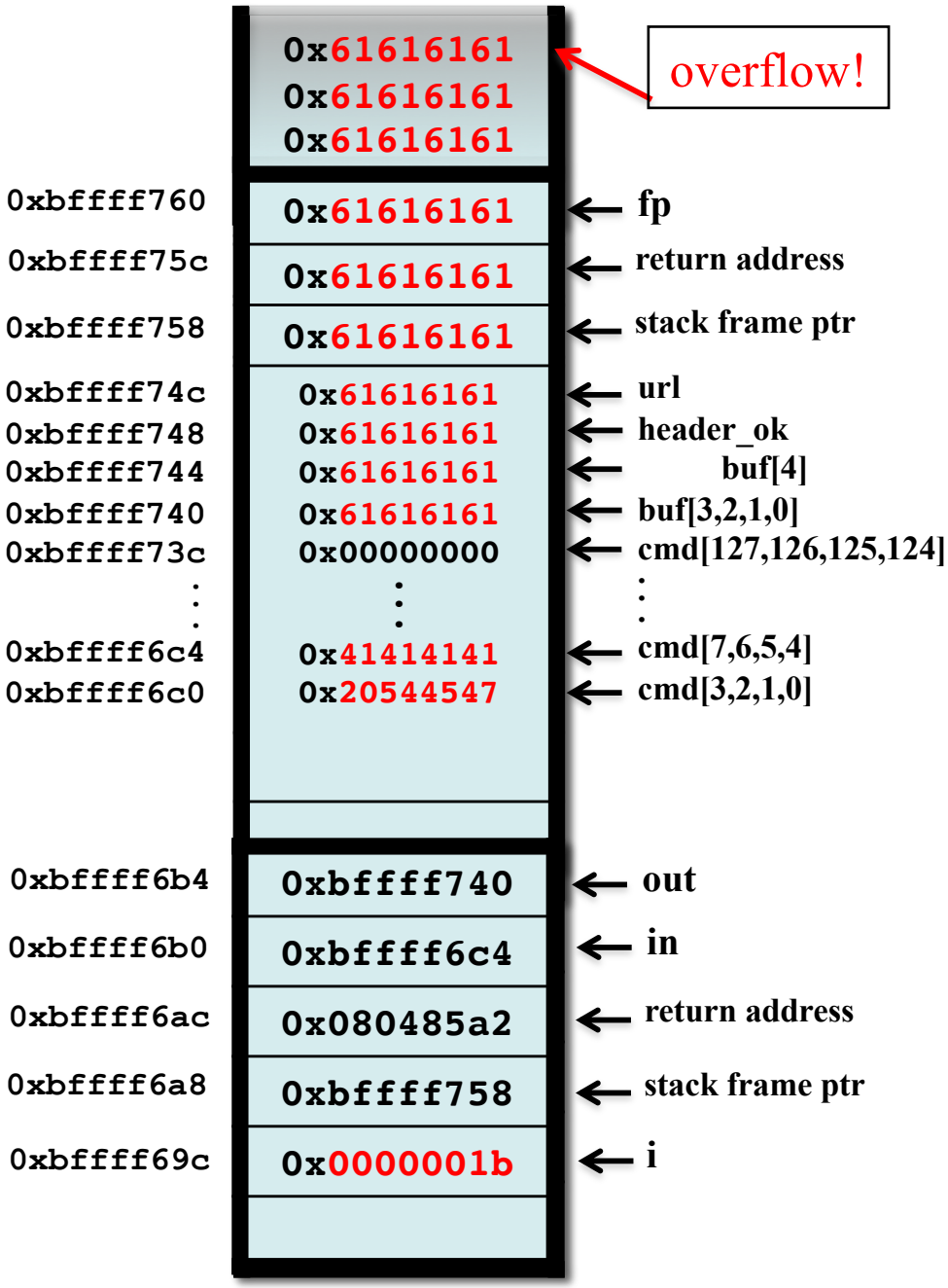
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Buffer Overflow

parse.c

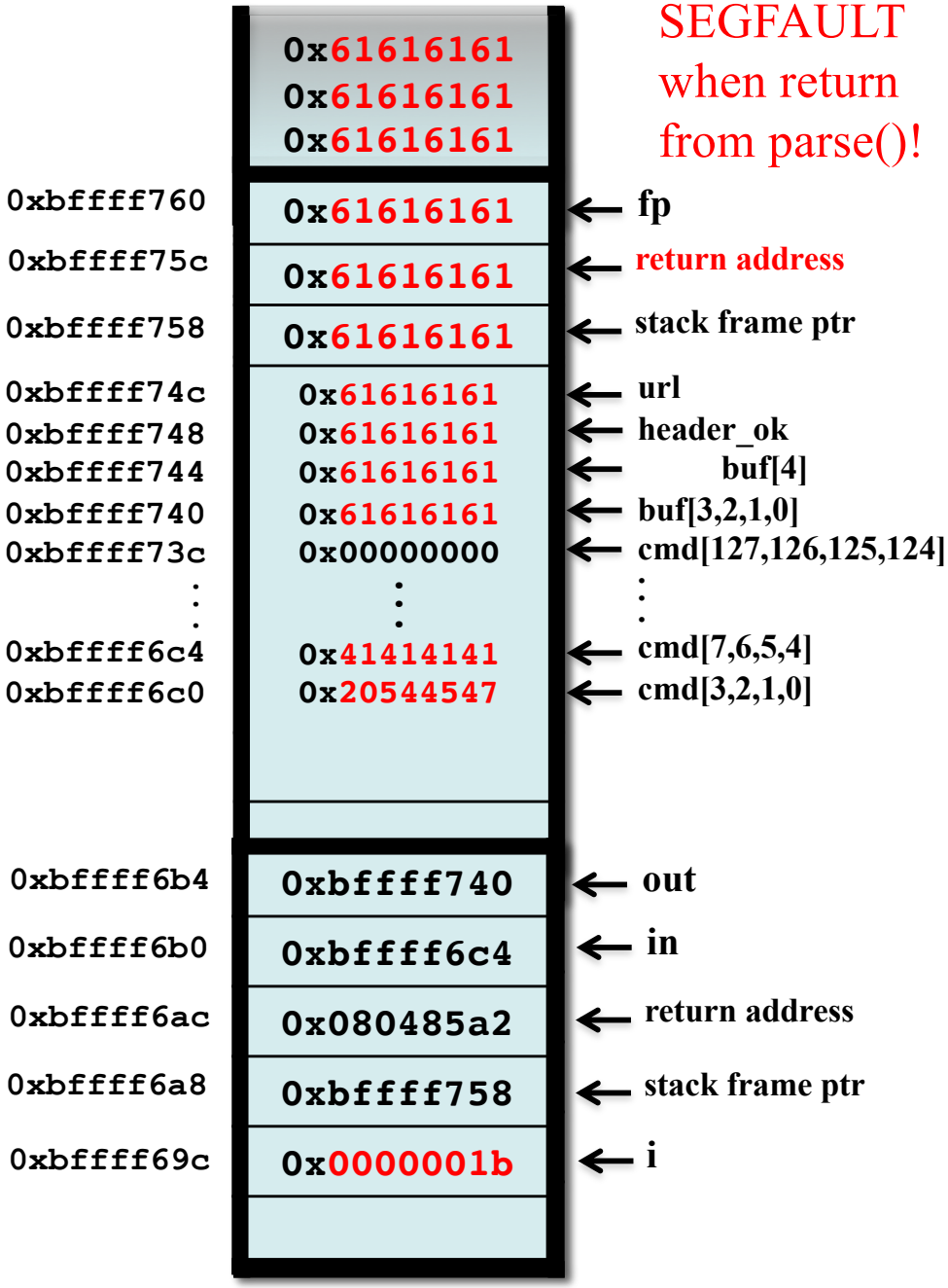
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



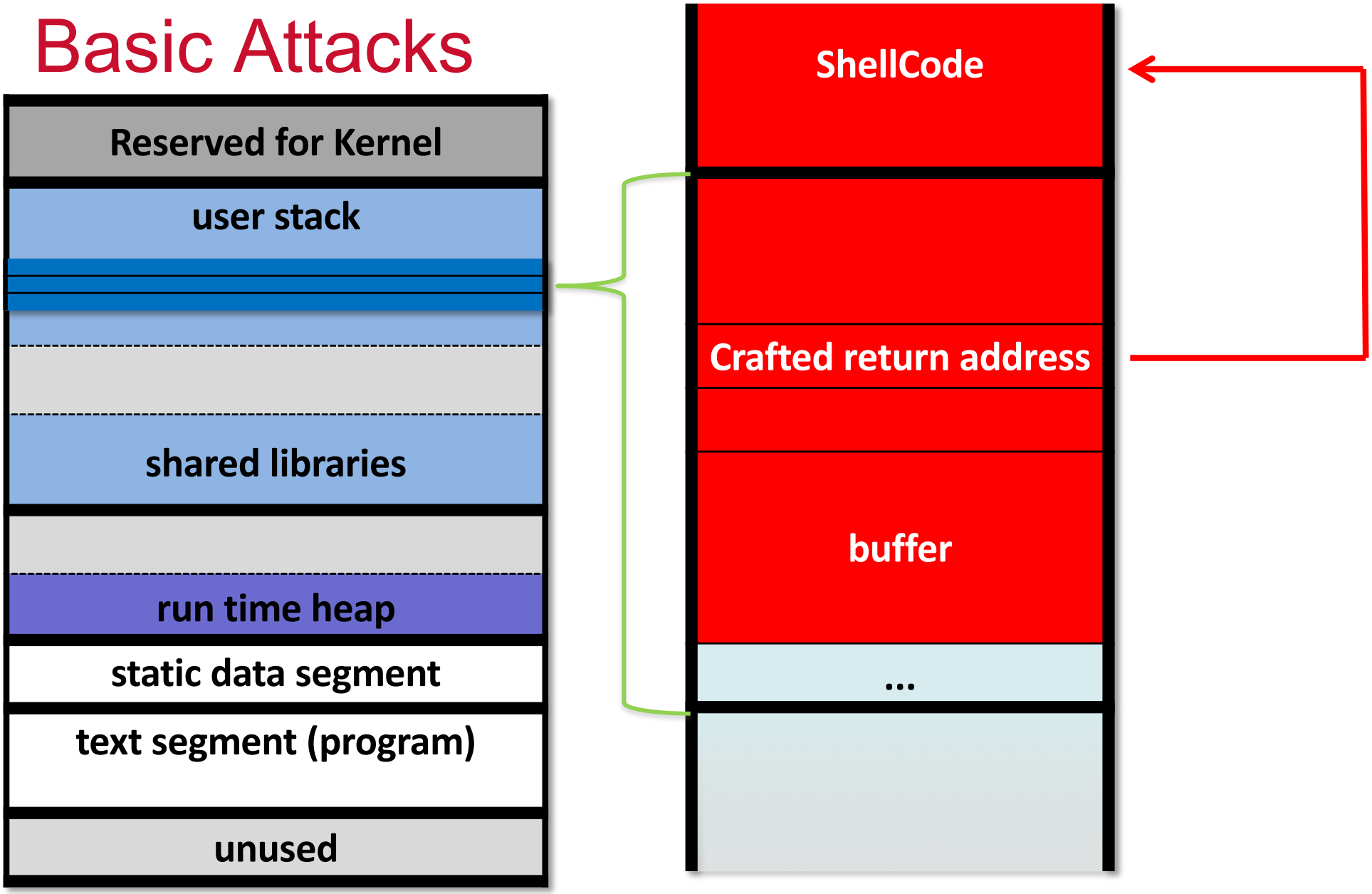
Agenda

- ~~Recap~~
- Security II
  - ~~Buffer Overflow~~
  - Basic Attacks
  - Basic Defenses

Basic Attacks

- Overwriting the return address allows an attacker to redirect the flow of program control
  - Instead of crashing, this can allow arbitrary code to be executed
    - Code segment called “shellcode”
    - E.g.,
      - the execve system call can be used to execute a binary file
      - With the correct permissions, execve(“/bin/sh”) can be used to obtain a root-level shell
  - Can reuse existing non-malicious code
    - Small code segments are called “gadgets”
    - Link them to finish a task for the attacker

Basic Attacks



Basic Attacks

parse.c

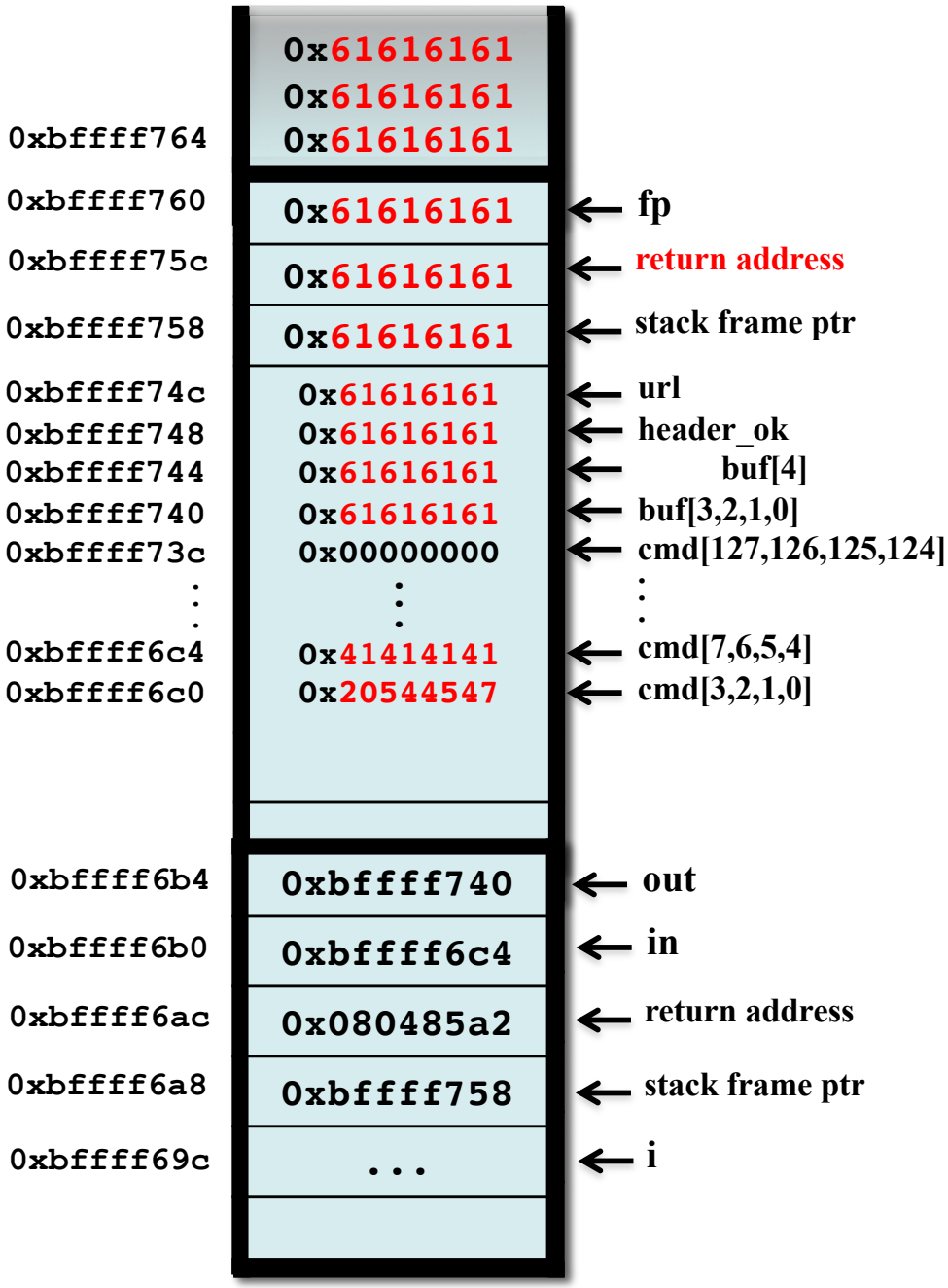
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Basic Attacks

parse.c

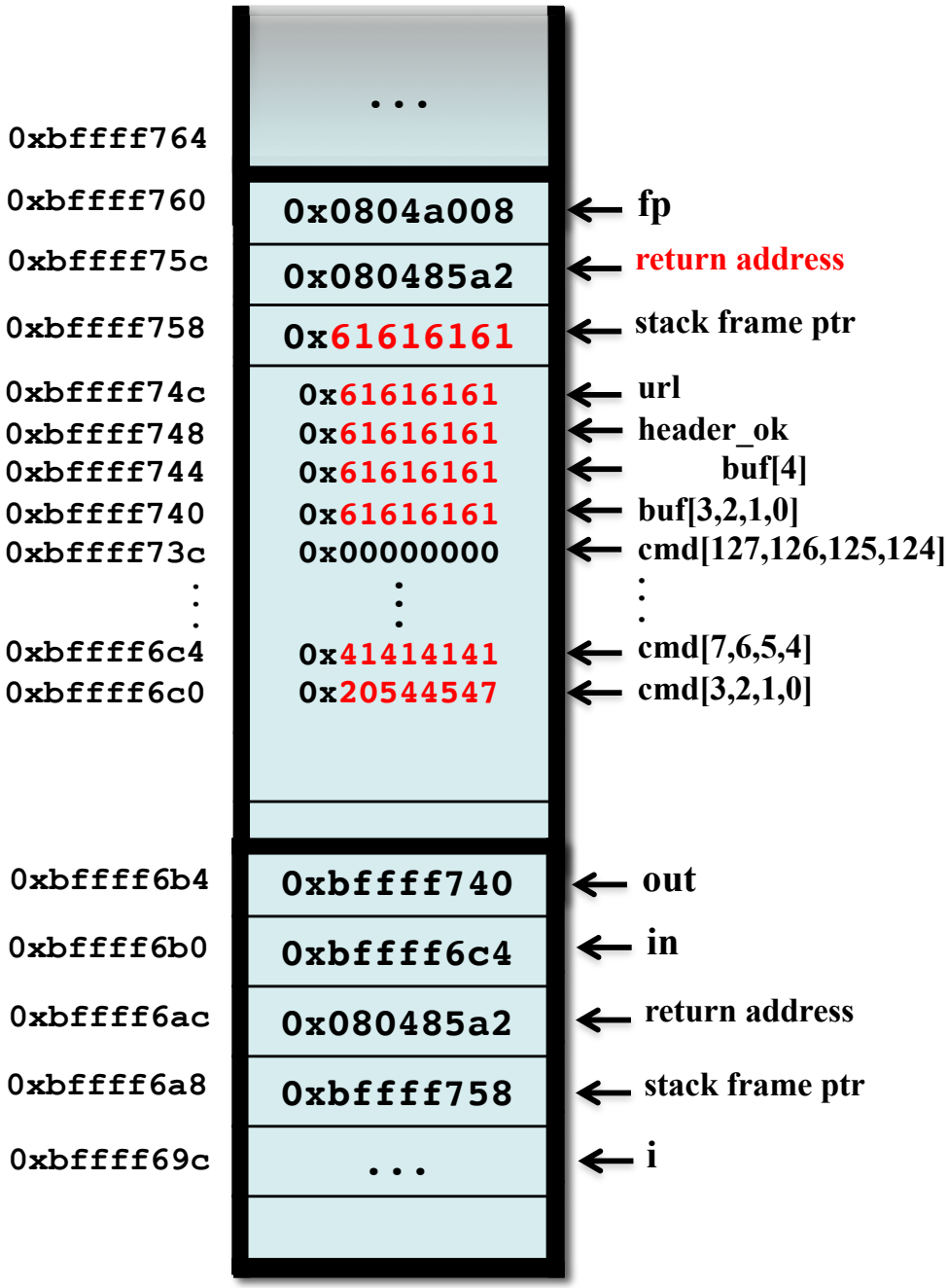
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAA\x64\xf7\xff\xffAAAA
\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\.....\xff\xff/bin/sh
```



Basic Attacks

parse.c

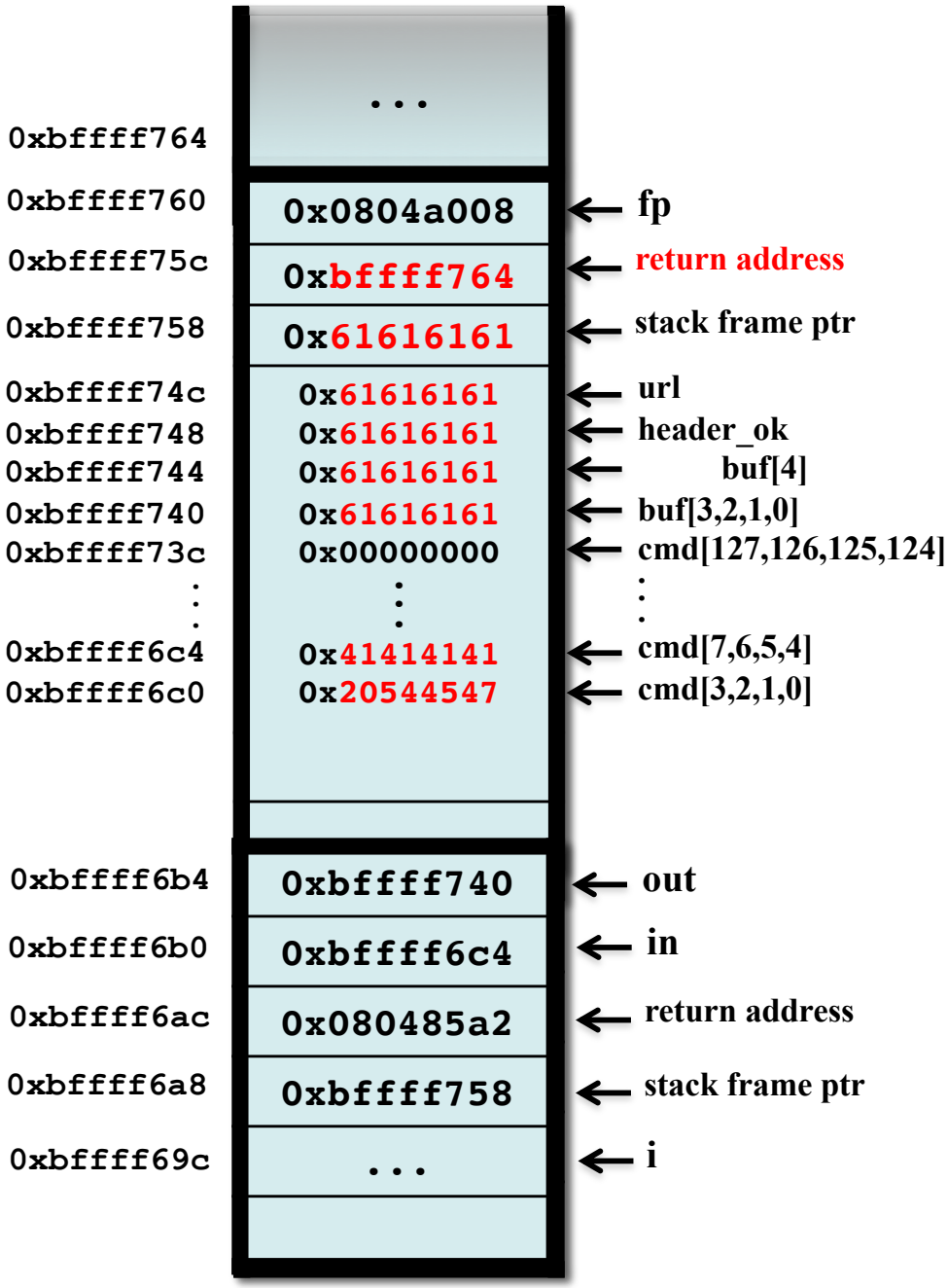
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAA\x64\xf7\xff\xbfAAAA
\xeb\x1f \x5e\x89\x76\x08\x31\xc0\x88\.....\xff\xff/bin/sh
```



Basic Attacks

parse.c

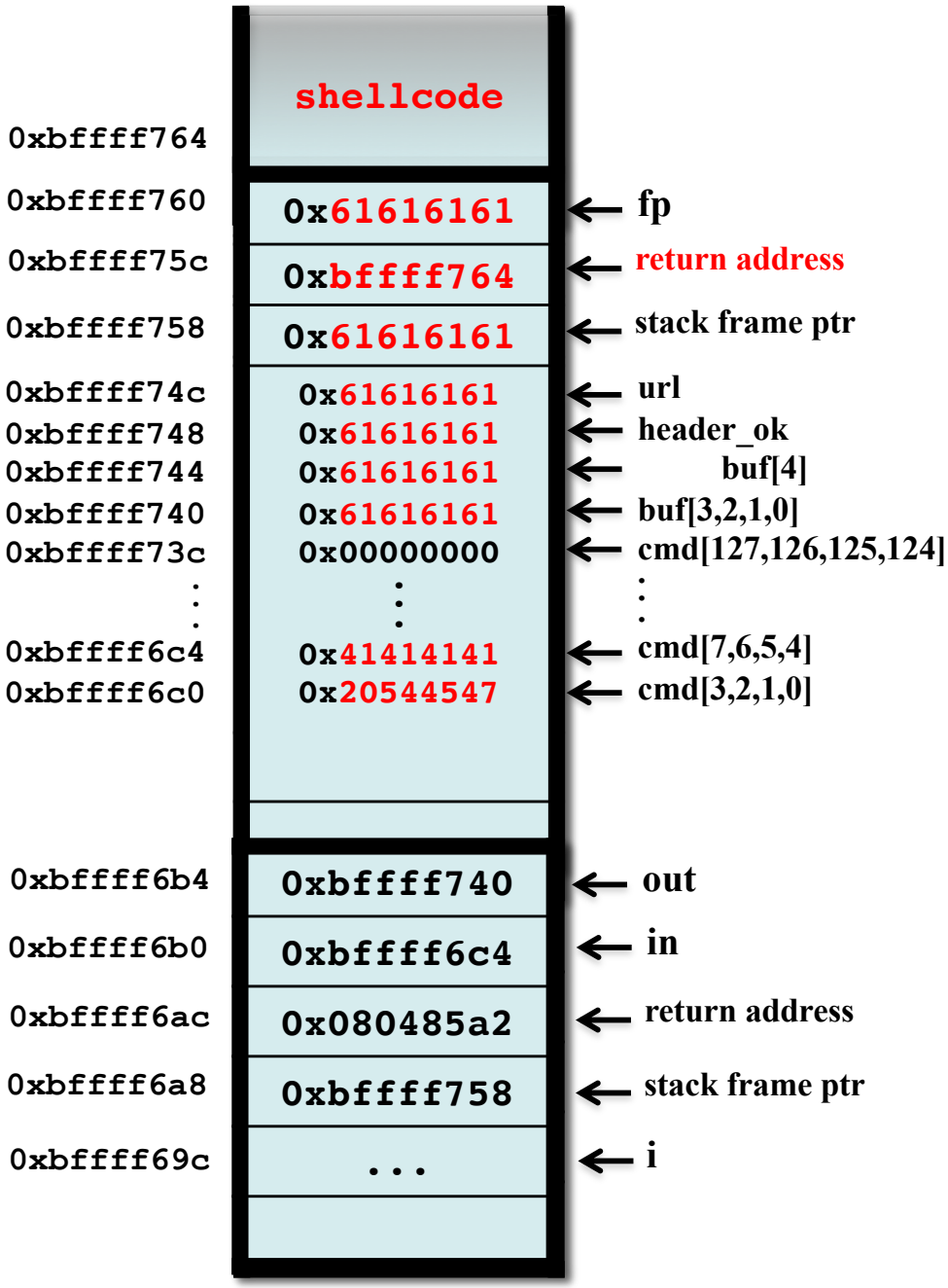
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAA\x64\xf7\xff\xbfAAAA
\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\.....\xff\xff/bin/sh
```



Basic Attacks

parse.c

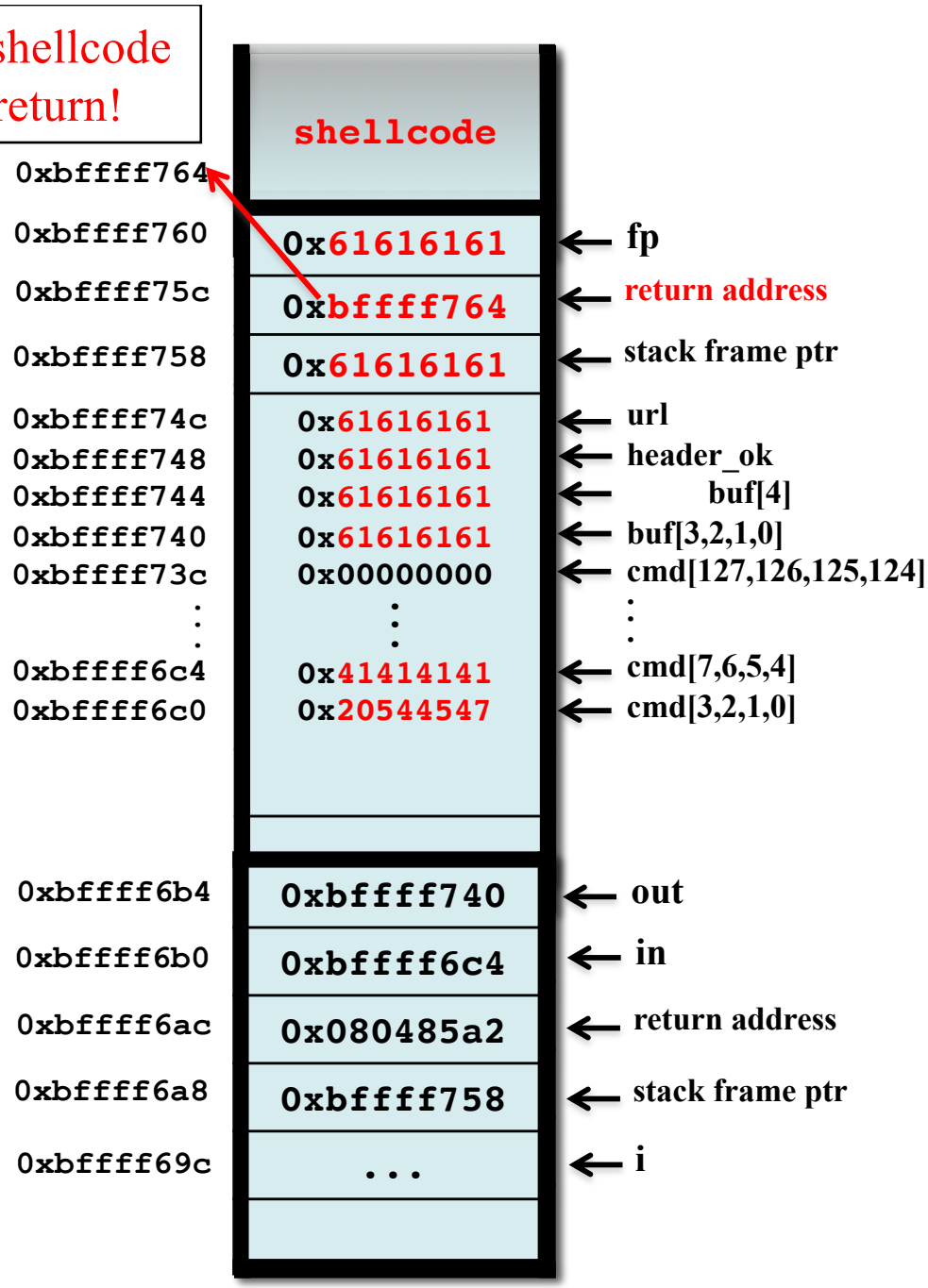
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

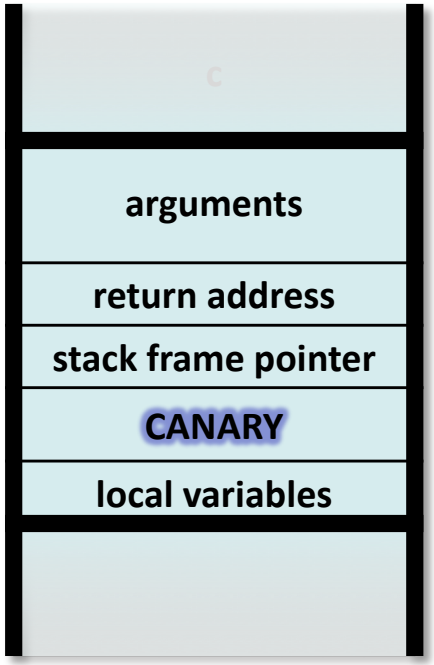
input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAA\x64\xf7\xff\xbfAAAA
\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\.....\xff\xff/bin/sh
```



Basic Defenses

- Non-execute
  - Prevent attack code execution by marking stack and heap as **non-executable**
- Address Space Layout Randomization (ASLR)
  - Start stack/heap at a random location
  - Map shared libraries to random location
    - Attacker cannot jump directly to exec function
- StackGuard
  - Embed “canary” in stack frames and verify their integrity prior to function return



Agenda

- ~~Recap~~
- ~~Security II~~
  - ~~Buffer Overflow~~
  - ~~Basic Attacks~~
  - ~~Basic Defenses~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.