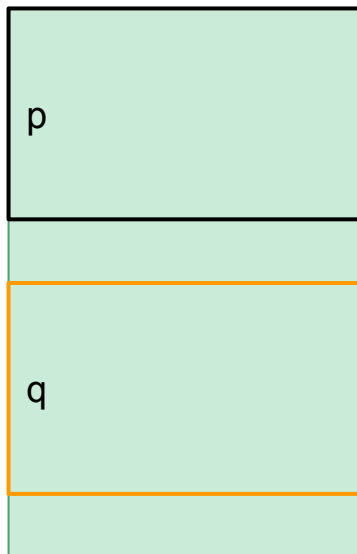


Target4: Find the Vulnerability



```
int foo(char *arg)
{
    char *p;
    char *q;

    if ( (p = tmalloc(300)) == NULL)
    {
        fprintf(stderr, "tmalloc failure\n");
        exit(EXIT_FAILURE);
    }
    if ( (q = tmalloc(325)) == NULL)
    {
        fprintf(stderr, "tmalloc failure\n");
        exit(EXIT_FAILURE);
    }

    tfree(p);
    tfree(q);

    if ( (p = tmalloc(1024)) == NULL)
    {
        fprintf(stderr, "tmalloc failure\n");
        exit(EXIT_FAILURE);
    }

    obsd_strncpy(p, arg, 1024);

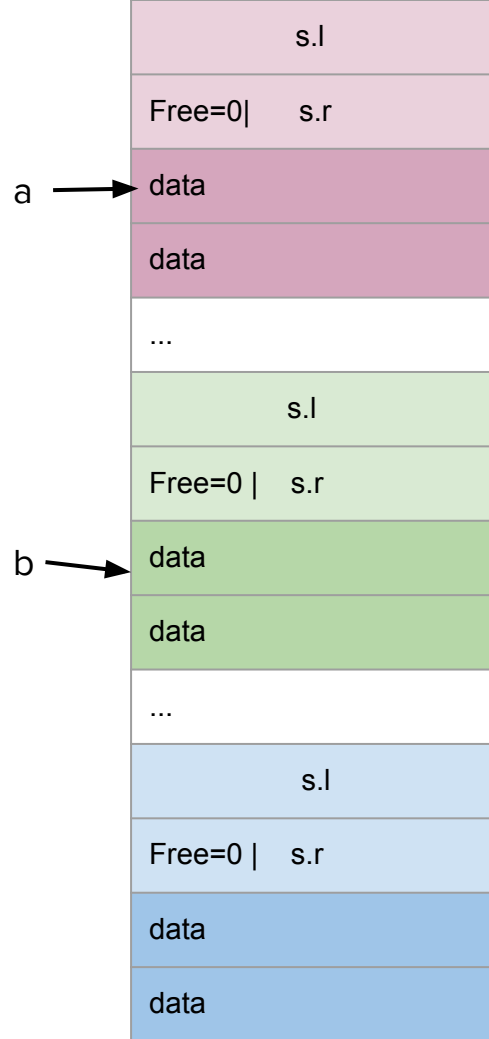
    tfree(q);

    return 0;
}
```

Heap Chunks

a = malloc(...)

b = malloc(...)



tfree()

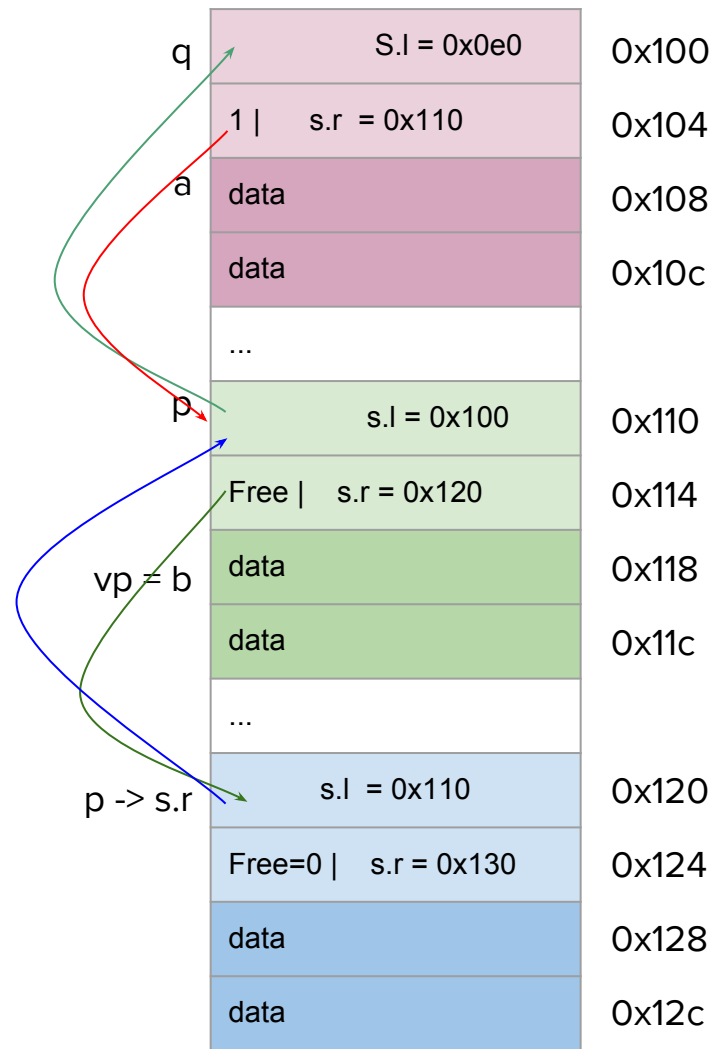
Assume a was already freed, and now we're calling tfree(b)

coalesce leftward...

```
void tfree(void *vp)
{
    CHUNK *p, *q;

    if (vp == NULL)
        return;

    p = TOCHUNK(vp);
    CLR_FREEBIT(p);
    q = p->s.l;
    if (q != NULL && GET_FREEBIT(q)) /* try to consolidate leftward */
    {
        CLR_FREEBIT(q);
        q->s.r      = p->s.r;
        p->s.r->s.l = q;
        SET_FREEBIT(q);
        p = q;
    }
}
```



tfree()

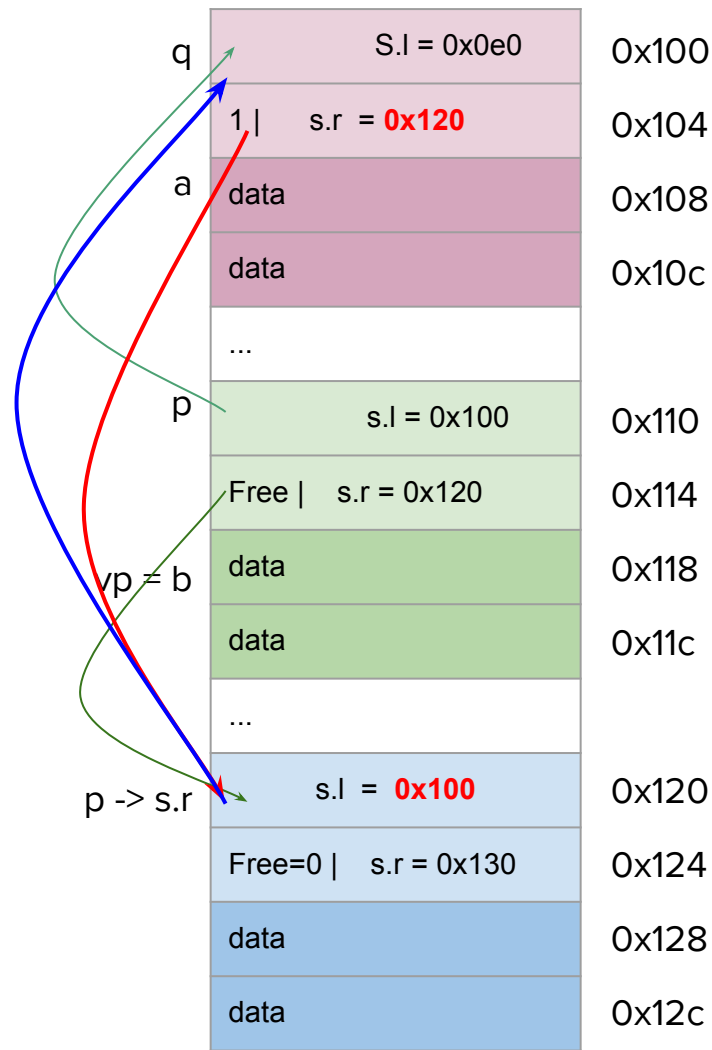
Assume a was already freed, and now we're calling tfree(b)

coalesce leftward...

```
void tfree(void *vp)
{
    CHUNK *p, *q;

    if (vp == NULL)
        return;

    p = TOCHUNK(vp);
    CLR_FREEBIT(p);
    q = p->s.l;
    if (q != NULL && GET_FREEBIT(q)) /* try to consolidate leftward */
    {
        CLR_FREEBIT(q);
        q->s.r = p->s.r;
        p->s.r->s.l = q;
        SET_FREEBIT(q);
        p = q;
    }
}
```



Aside: structs and memory

```
struct foo{
    int a;
    int b;
};
struct bar {
    struct foo * p1;
    struct foo * p2;
};
struct bar * s1 = malloc(sizeof(struct bar));
struct foo * s2 = malloc(sizeof(struct foo));
s1 -> p2 = s2;
s1 ->p2->a = 5;
```

s1 →	s1.p1	0x100
	s1.p2 = 0x10c	0x104
		0x108
s2 →	s2.a = 5	0x10c
	s2.b	0x110

	s2 = 0x10c	0xff8
	s1 = 0x100	0xffc

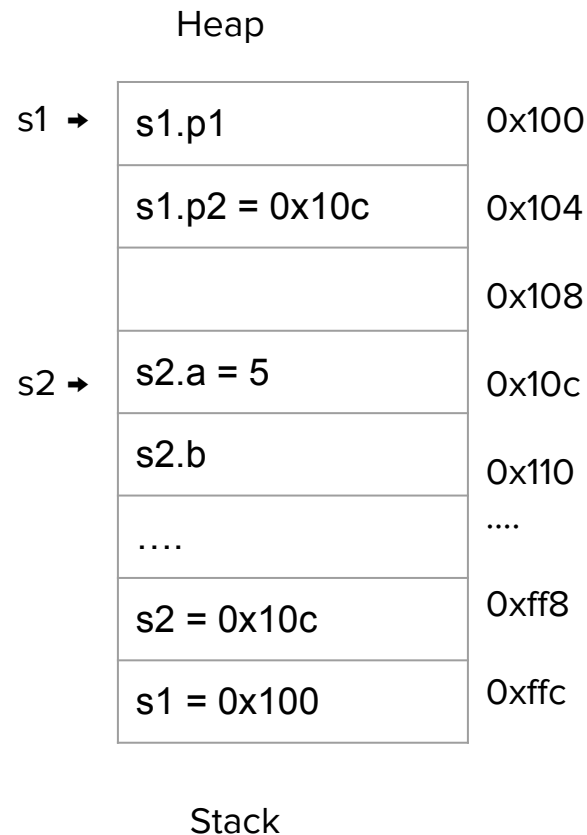
Aside: structs and memory

```
struct foo{
    int a;
    int b;
};

struct bar {
    struct foo * p1;
    struct foo * p2;
};

struct bar * s1 = malloc(sizeof(struct bar));
struct foo * s2 = malloc(sizeof(struct foo));
s1 -> p2 = s2;
s1 ->p2->a = 5;
```

Equivalent to: $*(s1 + 4 \text{ bytes}) = s2$
 $*(*(s1 + 4) + 0) = 5$



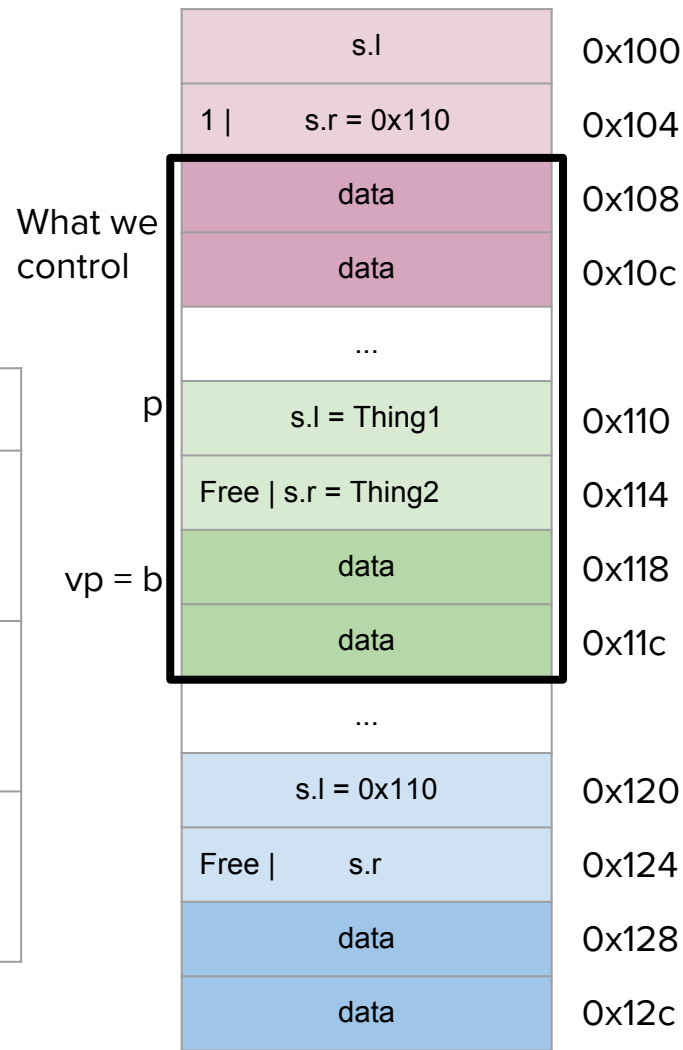
CHUNK struct

Linked List Code	Arbitrary Pointer Operations
$q = p \rightarrow s.l$	$q = *(p + 0)$ $q = *p$
$q \rightarrow s.r = p \rightarrow s.r$	$*(q + 4) = *(p + 4)$ $*(*p + 4) = *(p + 4)$
$p \rightarrow s.r \rightarrow s.l = q$	$*(*(p+4) + 0) = q$ $*(*(p+4)) = *p$

```
typedef double ALIGN;  
  
typedef union CHUNK_TAG  
{  
    struct  
    {  
        union CHUNK_TAG *l;  
        union CHUNK_TAG *r;  
    } s;  
    ALIGN x;  
} CHUNK;
```

What memory will free() change?

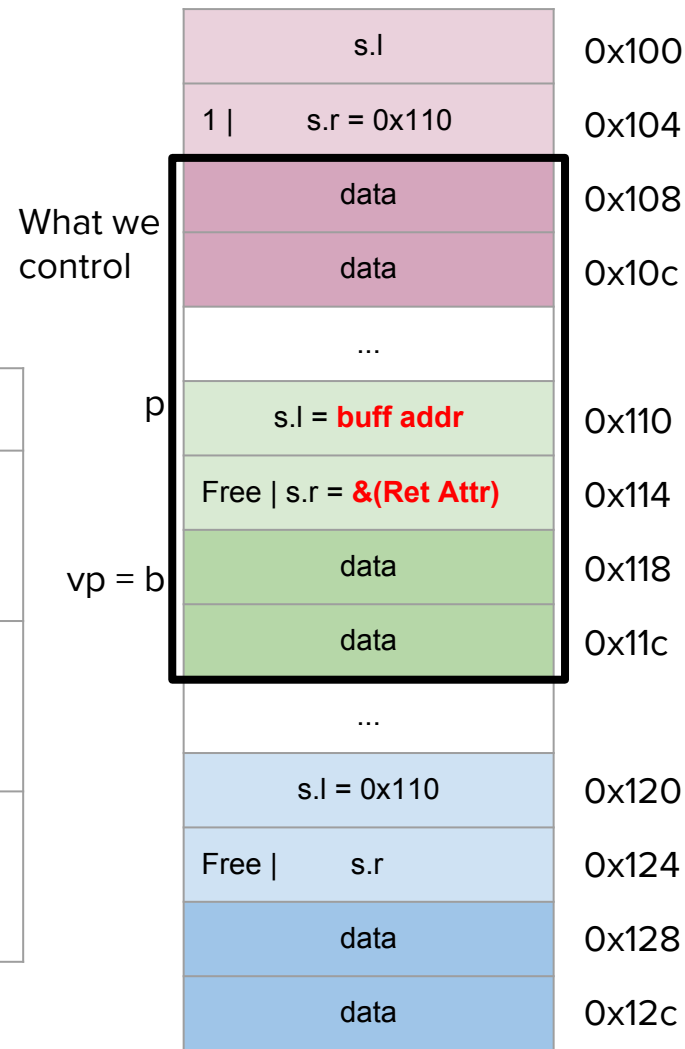
Linked List Code	Arbitrary Pointer Operations
<code>q = p -> s.l</code>	<code>q = *(p + 0)</code> <code>q = Thing1</code>
<code>q->s.r = p -> s.r</code>	<code>*(q + 4) = *(p + 4)</code> <code>Thing1[4-7] = Thing2</code>
<code>p->s.r->s.l = q</code>	<code>*(*(p+4) + 0) = q</code> <code>Thing2[0-3] = Thing1</code>



So what if in that memory we put...

The address of the Ret Addr is
\$ebp + 4

Linked List Code	Arbitrary Pointer Operations
<code>q = p -> s.l</code>	<code>q = *(p + 0)</code> <code>q = buf</code>
<code>q->s.r = p -> s.r</code>	<code>*(q + 4) = *(p + 4)</code> <code>buf[4-7] = &(ret addr)</code>
<code>p->s.r->s.l = q</code>	<code>*(*(p+4) + 0) = q</code> <code>Ret addr = buf</code>



Free Bit

In order to enter the if, we must pass GET_FREEBIT(q), so the LSb of q->s.r needs to be 1

- Remember Little Endian
- the CLR_FREEBIT operation makes sure that the original q->s.r is used for the coalescing, and SET_FREEBIT sets the bit back.

```
void tfree(void *vp)
{
    CHUNK *p, *q;

    if (vp == NULL)
        return;

    p = TOCHUNK(vp);
    CLR_FREEBIT(p);
    q = p->s.l;
    if (q != NULL && GET_FREEBIT(q)) /* try to consolidate leftward */
    {
        CLR_FREEBIT(q);
        q->s.r      = p->s.r;
        p->s.r->s.l = q;
        SET_FREEBIT(q);
        p = q;
    }
}
```

Breakdown

Linked List Code	Arbitrary Pointer Operations	Exploit result
<code>q = p -> s.l</code>	<code>q = *(p + 0)</code> <code>q = *p</code>	q = &buf
<code>q->s.r = p -> s.r</code>	<code>*(q + 4) = *(p + 4)</code> <code>*(*p + 4) = *(p + 4)</code>	buf[4-7] = ret addr
<code>p->s.r->s.l = q</code>	<code>*(*(p+4) + 0) = q</code> <code>*(*(p+4)) = *p</code>	ret addr = &buf

Side Effect: Corruption

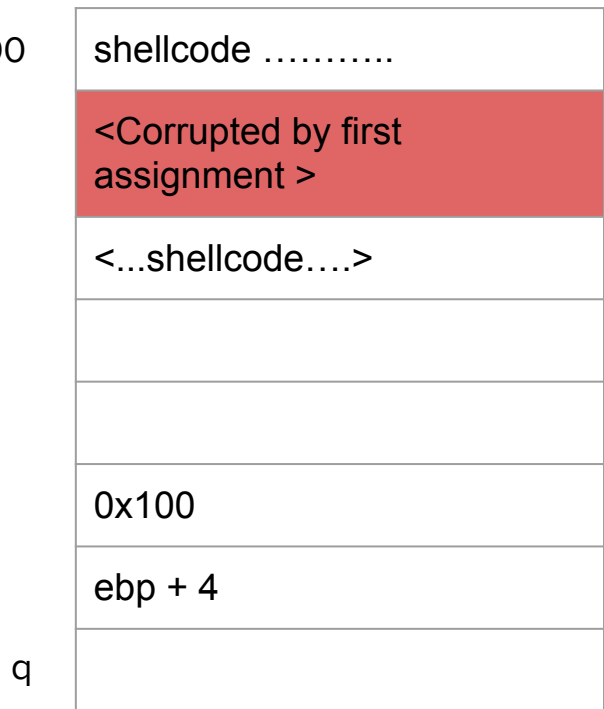
buf[4-7] = &(ret addr)

- Corrupts our buffer

Ret addr = buf

- What we want

buf = 0x100



Side Effect: Corruption

buf[4-7] = &(ret addr)

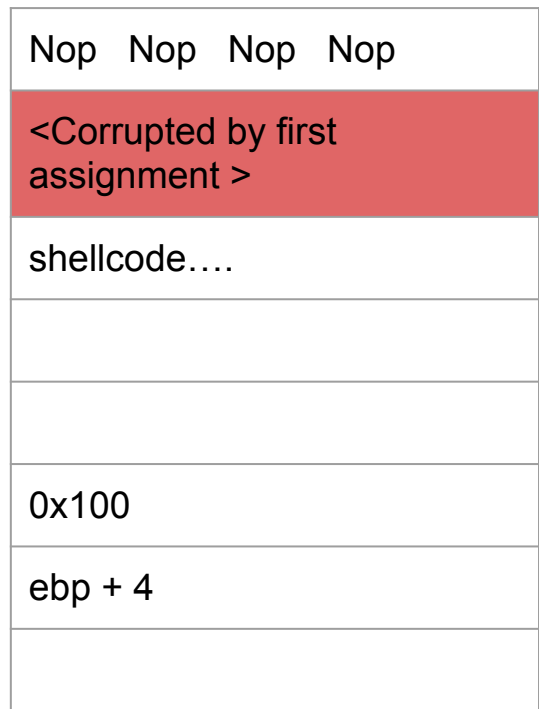
- Corrupts our buffer

Ret addr = buf

- What we want

“Solution” 1: Nops?

buf = 0x100



q

Side Effect: Corruption

$(0x108)[4-7] = \&(\text{ret addr})$

- Corrupts our buffer

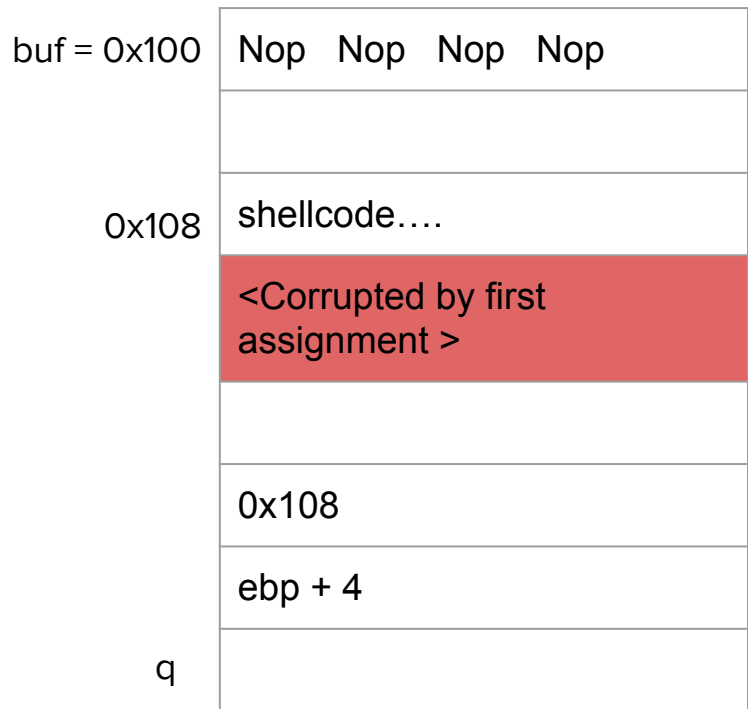
Ret addr = buf

- What we want

“Solution” 1: Nops?

- Still execute corrupted address

“Solution” 2: Choose a later address?



Side Effect: Corruption

$(0x108)[4-7] = \&(\text{ret addr})$

- Corrupts our buffer

Ret addr = buf

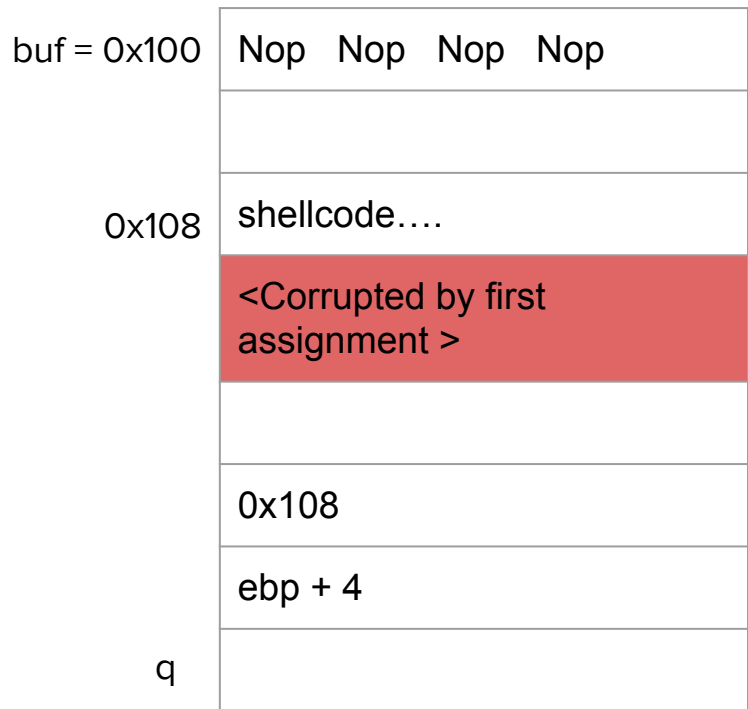
- What we want

“Solution” 1: Nops?

- Still execute corrupted address

“Solution” 2: Choose a later address?

- The corruption moves with us



Jump the corruption

buf = 0x100

buf[4-7] = &(ret addr)

- Corrupts our buffer

Ret addr = buf

- What we want

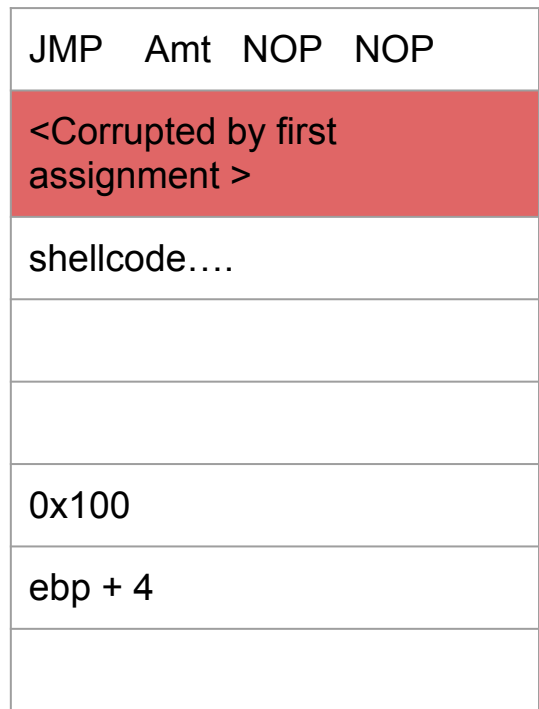
Solution 3: Jmp over the corrupted memory

JMP instruction (JMP rel16/32)

- <http://ref.x86asm.net/coder32.html>

How much to jump?

- Relative to the first byte after 'Amt'



How do we fix these vulnerabilities?

1. Buffer overflow
2. Buffer overflow (off by 1)
3. Integer
4. Double free()