Target4: Find the Vulnerability

```c
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_strlcpy(p, arg, 1024);
    tfree(p);
    tfree(q);
    return 0;
}
```
Heap Chunks

\[ a = \text{malloc}(\ldots) \]

\[ b = \text{malloc}(\ldots) \]
tfree()

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

```c
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; // cancel
        p->s.r->s.l = q;
        SET_FREEBIT(q);
        p = q;
    }
}
```
tfree()

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

coaalesce leftward...

```c
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

```c
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;
```
Aside: structs and memory

```c
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 bytes) = s2
*( *(s1 + 4) + 0) = 5
CHUNK struct

<table>
<thead>
<tr>
<th>Linked List Code</th>
<th>Arbitrary Pointer Operations</th>
</tr>
</thead>
</table>
| \( q = p -> s.l \) | \( q = *(p + 0) \)  
|                  | \( q = *p \)           |
| \( q->s.r = p -> s.r \) | \( *(q + 4) = *(p + 4) \)  
|                  | \( *(p + 4) = *(p + 4) \) |
| \( p->s.r->s.l = q \) | \( *( *(p+4) + 0 ) = q \)  
|                  | \( *( *(p+4)) = *p \)  

```c
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?

<table>
<thead>
<tr>
<th>Linked List Code</th>
<th>Arbitrary Pointer Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>q = p -&gt; s.l</td>
<td>q = *(p + 0)</td>
</tr>
<tr>
<td></td>
<td>q = Thing1</td>
</tr>
<tr>
<td>q-&gt;s.r = p -&gt; s.r</td>
<td>*(q + 4) = *(p + 4)</td>
</tr>
<tr>
<td></td>
<td>Thing1[4-7] = Thing2</td>
</tr>
<tr>
<td>p-&gt;s.r-&gt;s.l = q</td>
<td>*( *(p+4) + 0 ) = q</td>
</tr>
<tr>
<td></td>
<td>Thing2[0-3] = Thing1</td>
</tr>
</tbody>
</table>

What we control:

```
vp = b
s.l = Thing1
Free | s.r = Thing2
```

```
p
s.l = 0x110
Free | s.r
```

```
q  = p -> s.l
q = *(p + 0)
q = Thing1
q->s.r = p -> s.r
*(q + 4) = *(p + 4)
Thing1[4-7] = Thing2
p->s.r->s.l = q
*( *(p+4) + 0 ) =  q
Thing2[0-3] = Thing1
```
So what if in that memory we put...

The address of the Ret Addr is $ebp +4

<table>
<thead>
<tr>
<th>Linked List Code</th>
<th>Arbitrary Pointer Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>q = p -&gt; s.l</td>
<td>q = *(p + 0)</td>
</tr>
<tr>
<td></td>
<td>q = buf</td>
</tr>
<tr>
<td>q-&gt;s.r = p -&gt; s.r</td>
<td>*(q + 4) = *(p + 4)</td>
</tr>
<tr>
<td></td>
<td>buf[4-7] = &amp;(ret addr)</td>
</tr>
<tr>
<td>p-&gt;s.r-&gt;s.l = q</td>
<td>*( *(p+4) + 0 ) = q</td>
</tr>
<tr>
<td></td>
<td>Ret addr = buf</td>
</tr>
</tbody>
</table>

![Diagram showing linked list and data structures with addresses and pointers]
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.
## Breakdown

<table>
<thead>
<tr>
<th>Linked List Code</th>
<th>Arbitrary Pointer Operations</th>
<th>Exploit result</th>
</tr>
</thead>
</table>
| q = p -> s.l     | q = *(p + 0)  
|                  | q = *p         | q = &buf       |
| q->s.r = p -> s.r | *(q + 4) = *(p + 4)  
|                  | *(p + 4) = *(p + 4) | buf[4-7] = ret addr |
| p->s.r->s.l = q  | *( *(p+4) + 0 ) = q  
|                  | *( *(p+4)) = *p   | ret addr = &buf |
Side Effect: Corruption

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

Ret addr = buf
- What we want

buf = 0x100

shellcode ...........

<Corrupted by first assignment>

<...shellcode....>

0x100

ebp + 4

q
Side Effect: Corruption

buf[4-7] = &(ret addr)
  - Corrupts our buffer
Ret addr = buf
  - What we want

“Solution” 1: Nops?

```
buf = 0x100
Nop  Nop  Nop  Nop
<Corrupted by first assignment >
shellcode....
```

```
0x100
ebp + 4
q
```
Side Effect: Corruption

(0x108)[4-7] = &(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] = &(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[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()