
掘金CTF

——CTF中的内存漏洞利用技巧

杨坤 kelwya@gmail.com

清华大学网络与信息安全实验室



Outline

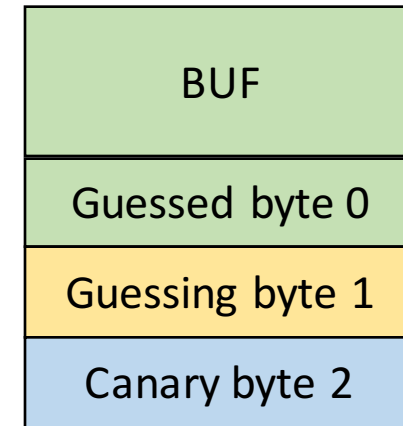
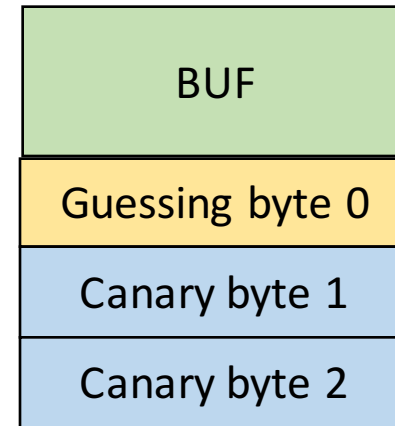
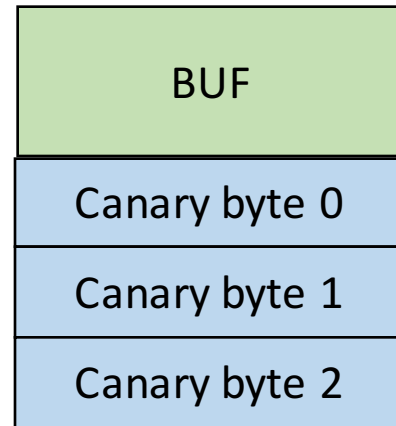


- Stack Overflow
 - Stack Canary Leakage/Overwrite
 - PC Partial Overwrite
 - FP Overwrite
- Heap Overflow
 - Fastbin
 - Unlink
 - Off-by-one
- Libc Symbol Resolution
 - Libc Database
 - Leaklib
 - Return to dl_resolve
- Useful Gadgets
 - Magic System Address
 - Universal Gadgets

Stack Canary



- Brute force canary byte-by-byte
 - 30C3 bigdata

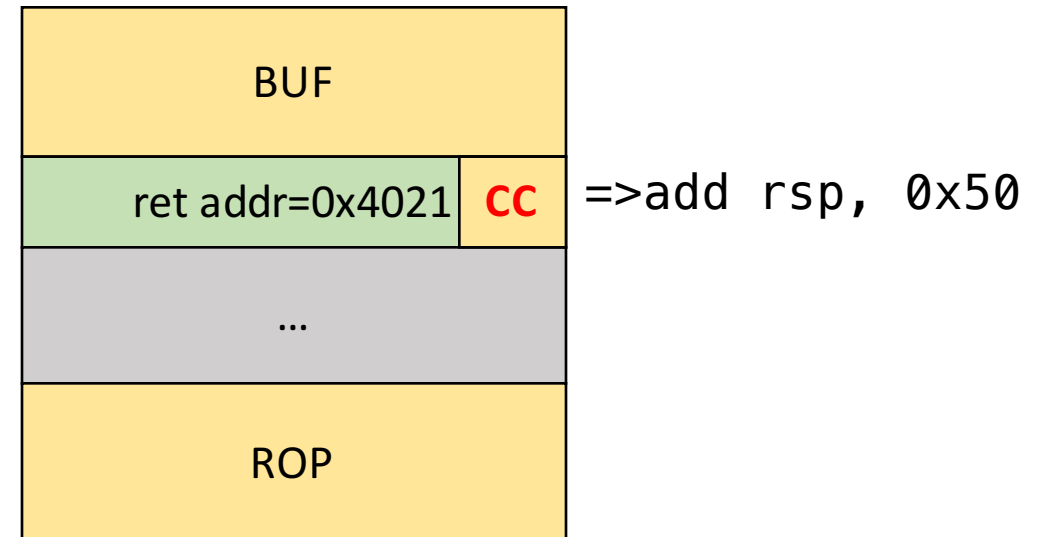


- Overwrite canary in TLS
 - Canary is stored in TLS
 - Initialized by ld.so
 - Example: ISG CTF Finals 2014, pepper, unintended solution
 - Array Index overflow, write 0 to arbitrary address



PC Partial Overwrite

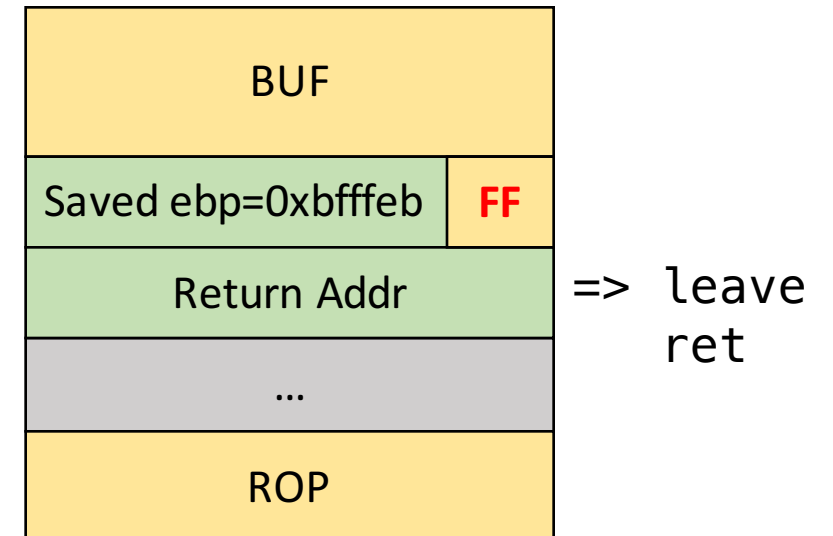
- Challenges
 - Stack payload is null terminated, and the code address starts with null byte
 - Overflow is not enough to do ROP
- Solution
 - Overwrite several bytes of return address
 - Stack Pivot
- Example
 - DEFCON Finals 2014, eliza





Stack Frame Pointer Overwrite

- Challenge
 - Overflow but cannot overrun the return address, e.g. off-by-one stack overflow
- Solution
 - Overwrite stack frame pointer register (ebp for x86)
 - ebp is restored, stack is moved with another "leave ret"
 - Do ROP in a controlled stack position
- Example
 - Codegate CTF Finals 2015, chess

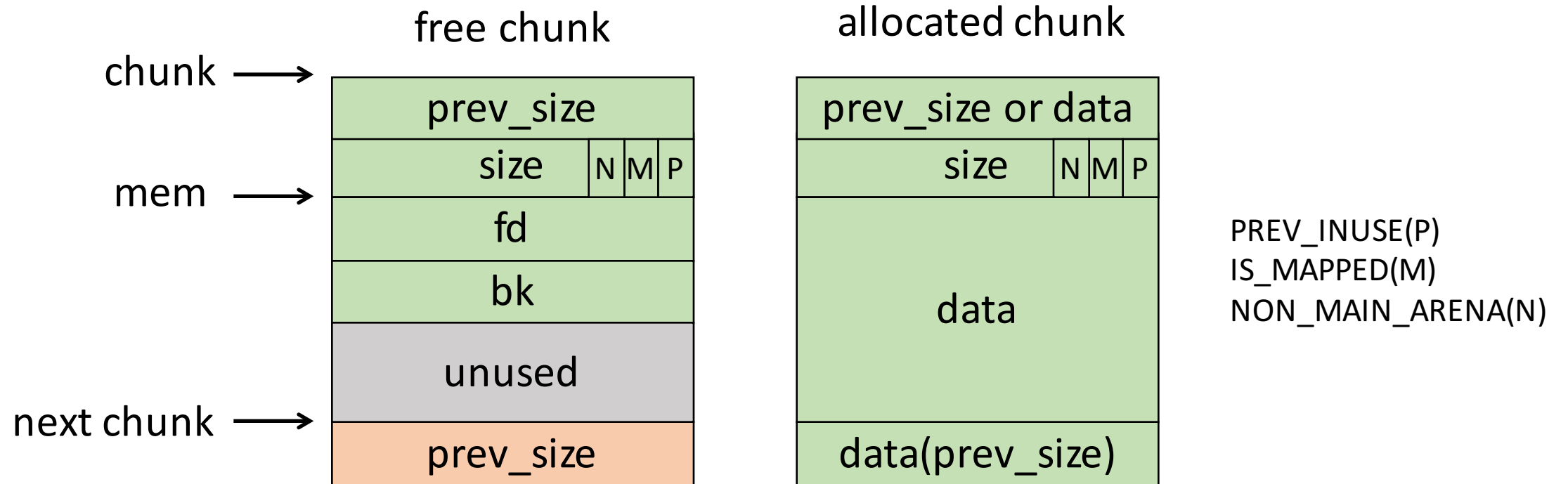




Heap

- dlmalloc – General purpose allocator
- ptmalloc2 – glibc
- jemalloc - FreeBSD and firefox
- tcmalloc – Google
- libumem - Solaris

Chunks

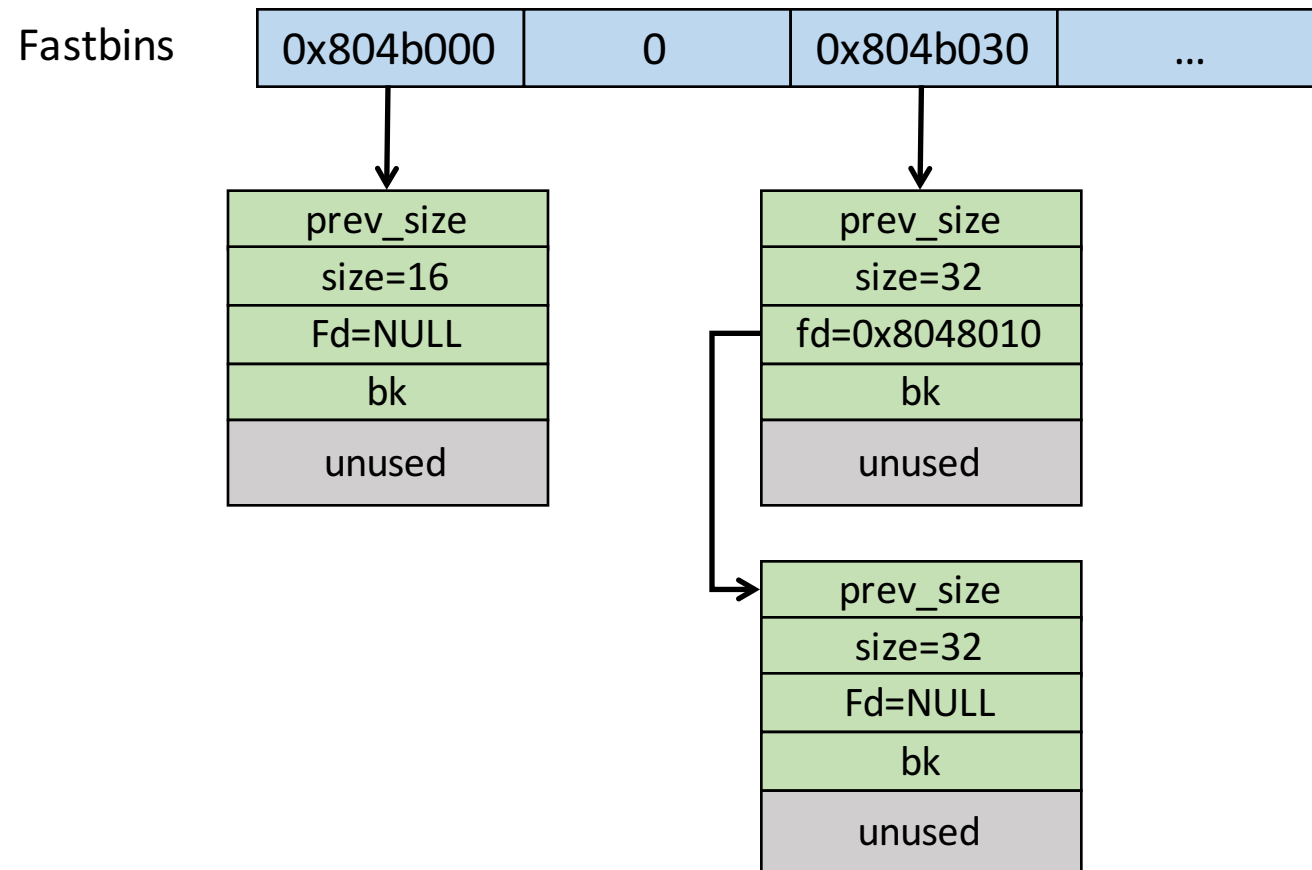


Bins

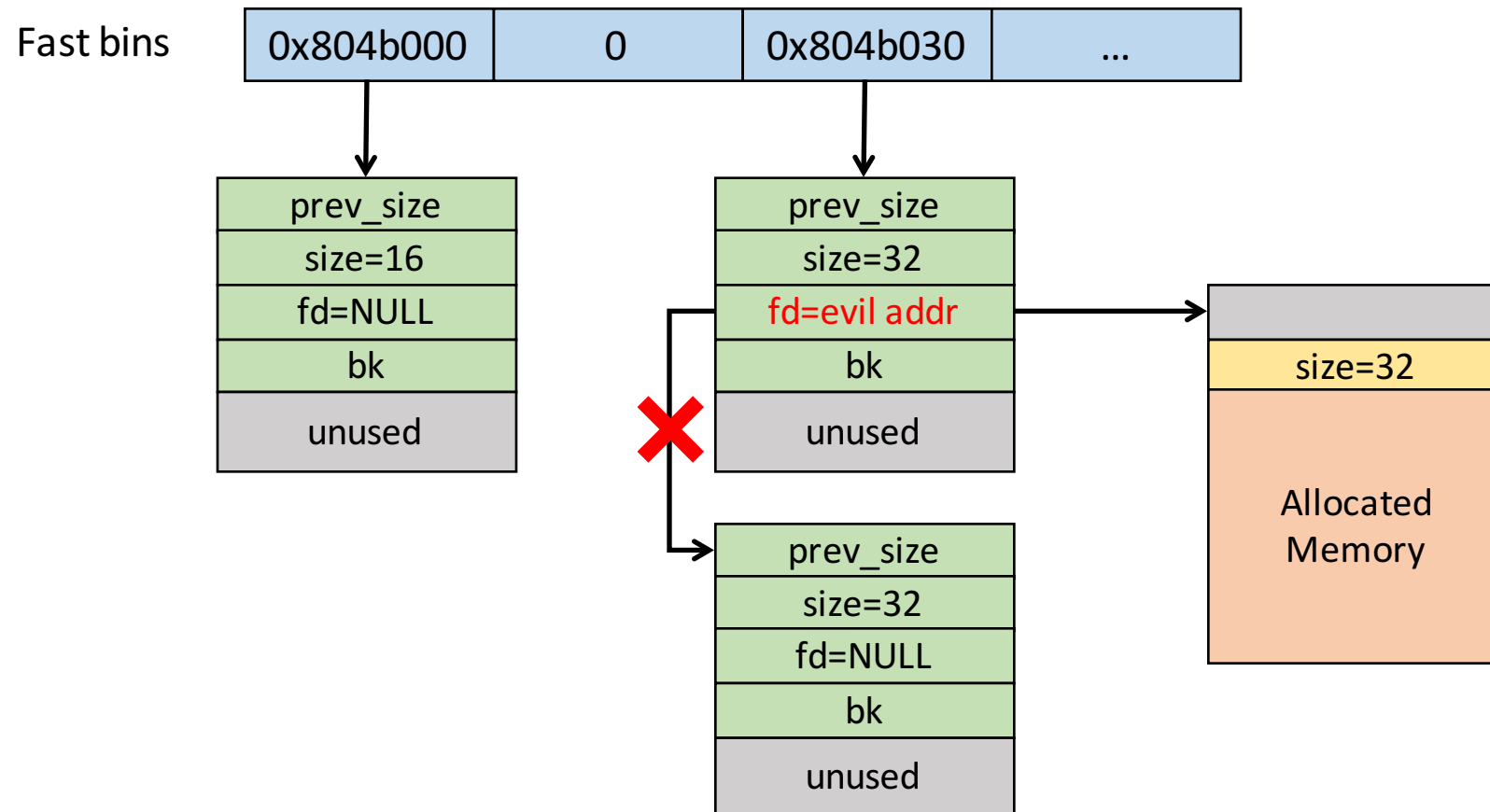


- Fast bins
 - 16 ~ 64 bytes for x86_32; 32 ~ 128 bytes for x86_64
 - LIFO: Single Linked List
 - No coalescing
- Bins
 - Circular double linked list
 - Bin 1 - Unsorted bin
 - freed small and large chunks are temporarily added to unsorted bin
 - Bin 2 to Bin 63 - Small bin
 - 16, 24, 32, ..., 508 bytes for x86_32, in each bin contains chunks with the same size
 - Bin 64 to Bin 126 - Large bin
 - ≥ 512 bytes(x86_32)

Fast bin

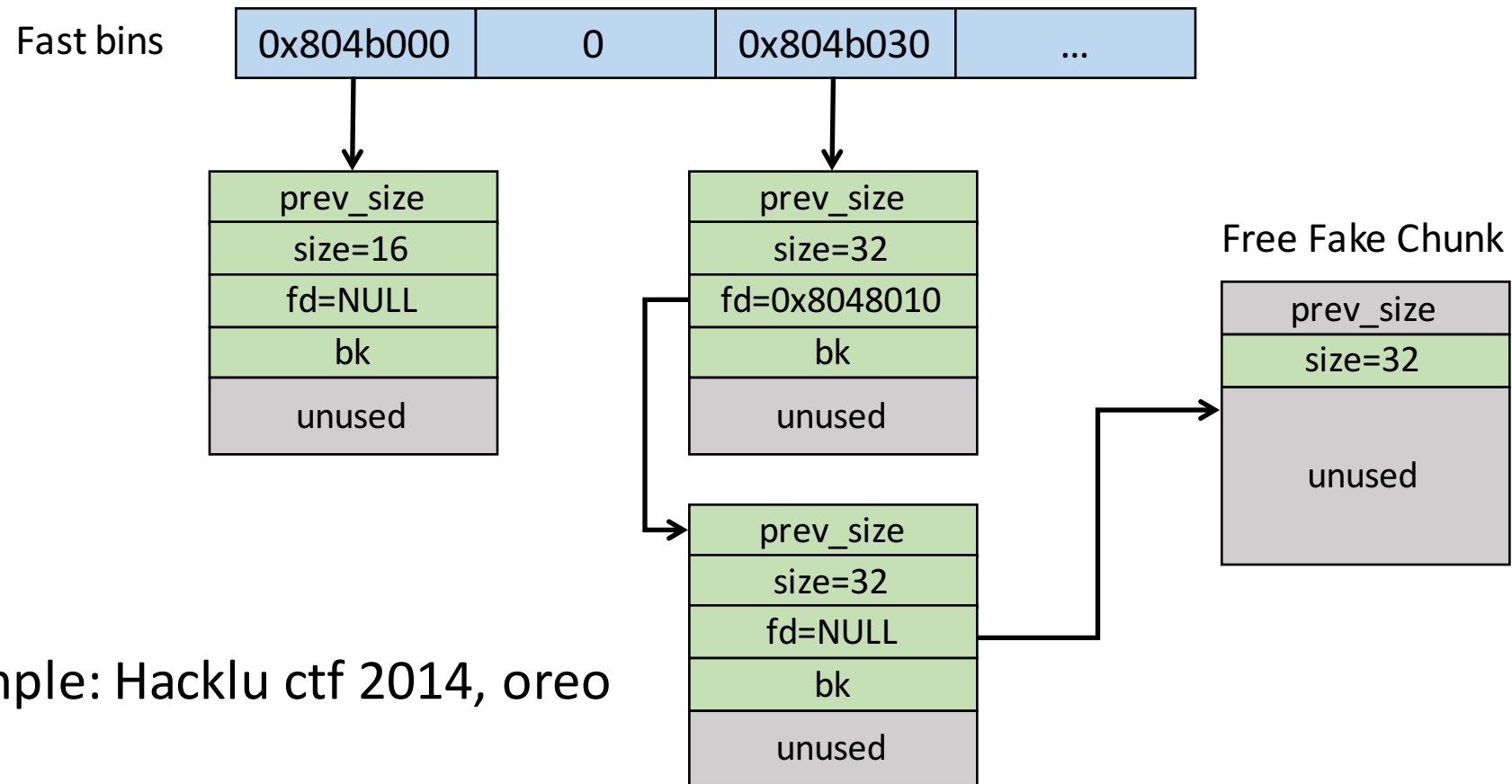


Fast bin Attack





Fast bin Attack



Example: Hacklu ctf 2014, oreo



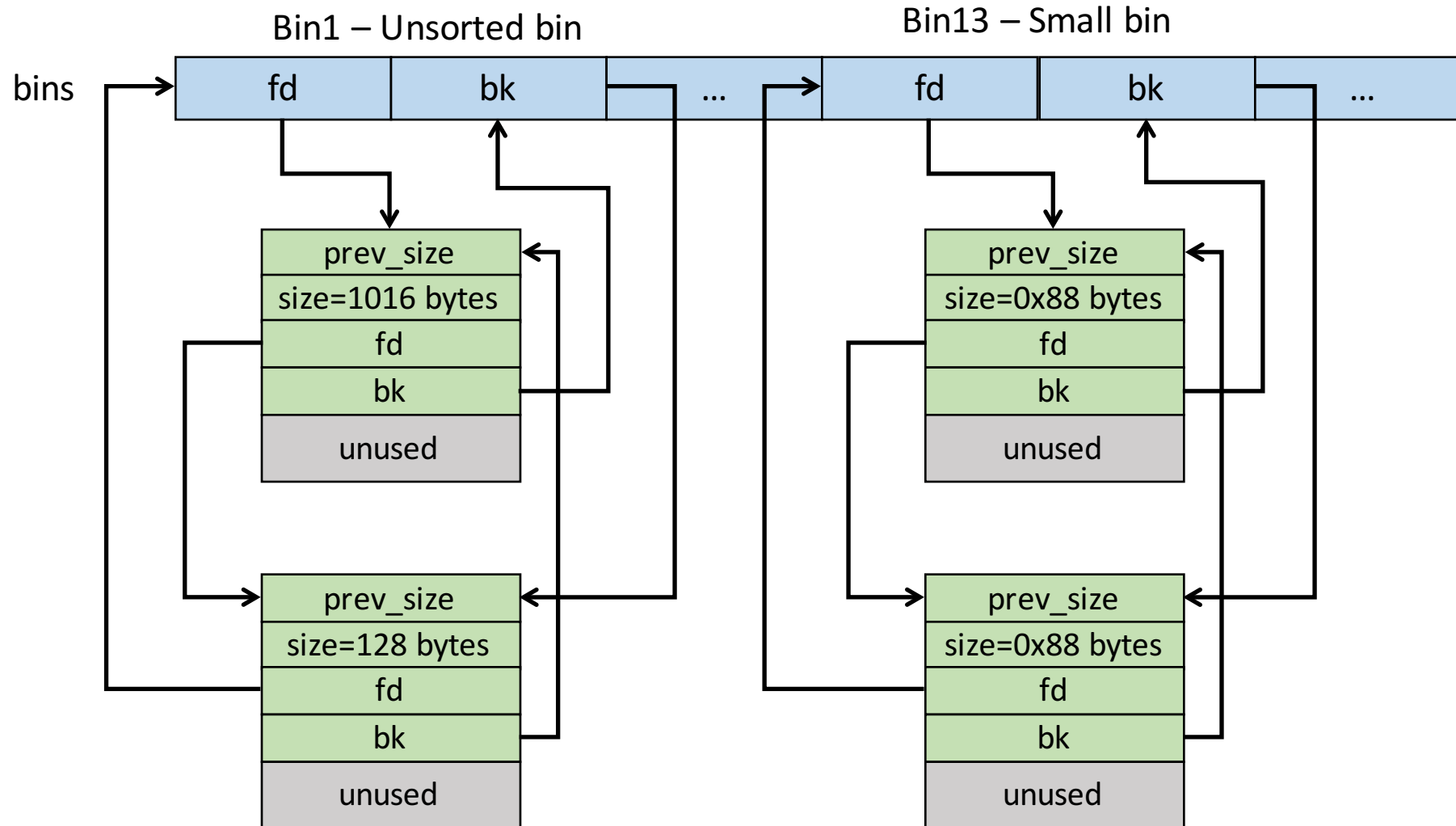
Overwrite realloc_hook in glibc

```
gdb-peda$ x/8gx 0x7ffff7dd5b08
0x7ffff7dd5b08 <_IO_wide_data_0+296>: 0x0000000000000000      0x00007ffff7dd4240
0x7ffff7dd5b18: 0x0000000000000000      0x00007ffff7ab3940
0x7ffff7dd5b28 <__realloc_hook>: 0x00007ffff7ab38e0      0x0000000000000000
0x7ffff7dd5b38: 0x0000000000000000      0x0000000100000000
```

```
gdb-peda$ x/8gx 0x7ffff7dd5b0d
0x7ffff7dd5b0d <_IO_wide_data_0+301>: 0xffff7dd424000000      0x000000000000007f
0x7ffff7dd5b1d: 0xffff7ab394000000      0xffff7ab38e00007f
0x7ffff7dd5b2d <__realloc_hook+5>: 0x000000000000007f      0x0000000000000000
0x7ffff7dd5b3d: 0x0100000000000000      0x0000000000000000
```

- Example
 - PlaidCTF 2014, datastore
 - HITCON CTF 2015, fooddb

Unsorted bin and Small bin

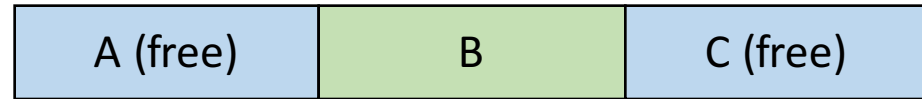




Unlink

- Condition

- Chunk A or chunk C is free
- Free(chunk B)

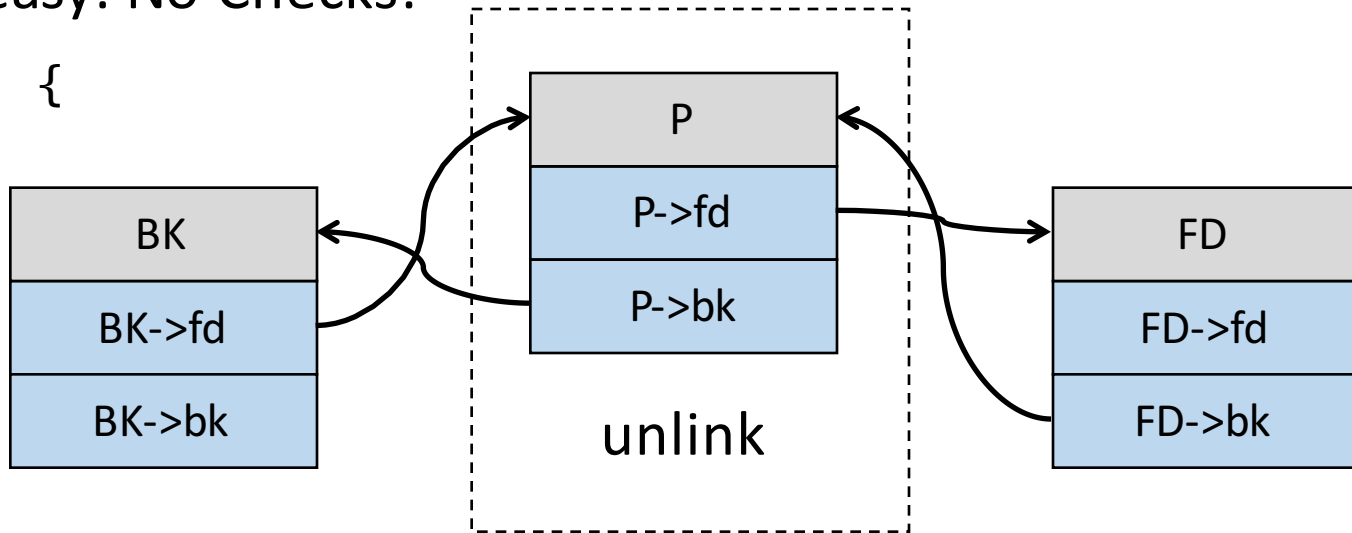


- Consolidate chunk 0 or chunk 2 with chunk 1
- Unlink chunk 0 or chunk 2

Unlink

Old school unlink technique is easy: No Checks!

```
#define unlink( P, BK, FD ) {
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```

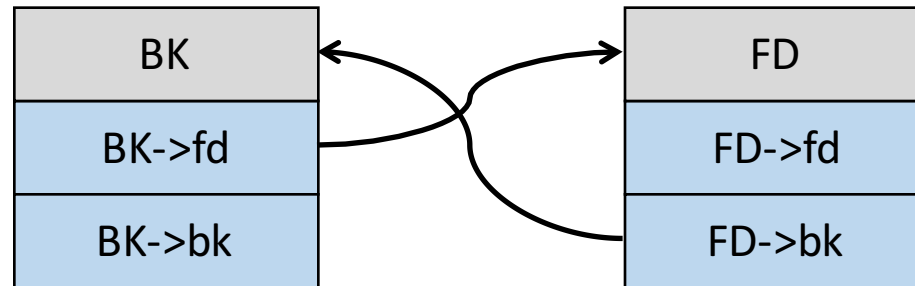


Unlink checks in modern glibc:

```
assert(P->fd->bk == P)
assert(P->bk->fd == P)
```

Bypass:

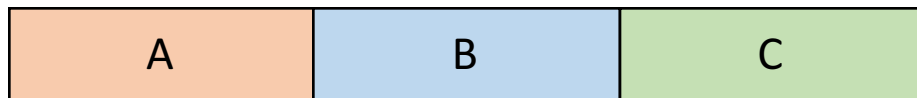
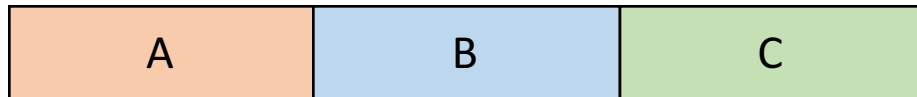
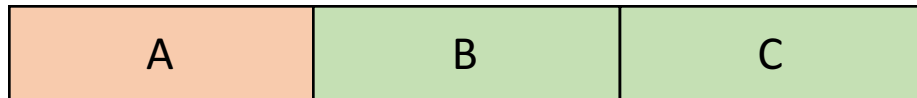
- Find a pointer X to P (*X = P)
- Set P->fd and P->bk to X
- Trigger Unlink(P)
- We have *P = X



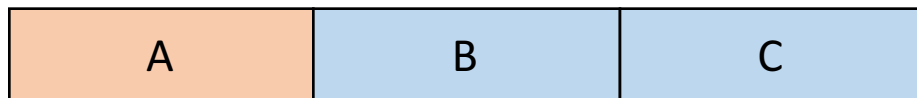


Off-by-one

- Extending Free Chunks



Overflow: $\text{size}(B) += \text{size}(C)$

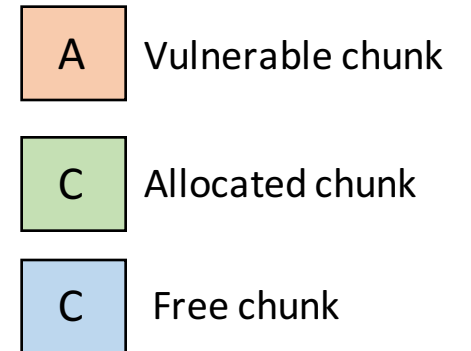


Initial State

B is freed

Overflow into B

Free chunk is extended



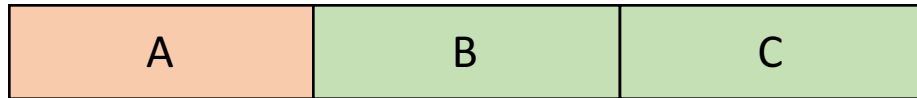
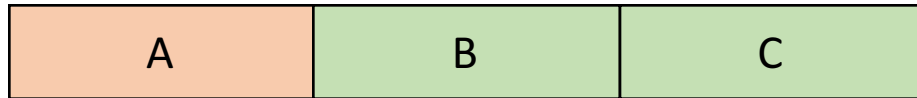
Example:

```
gethostbyname()  
heap overflow
```

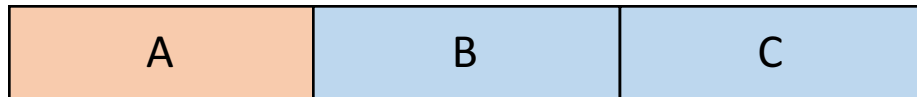



Off-by-one

- Extending Allocated Chunks



Overflow: $\text{size}(B) += \text{size}(C)$

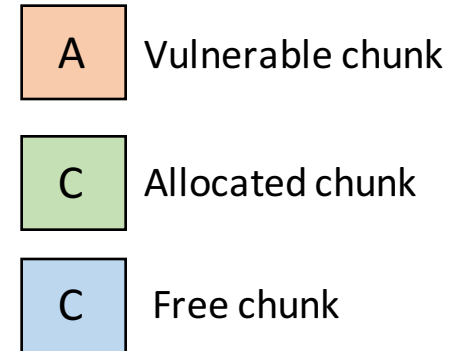


Initial State

Overflow into B

B is freed

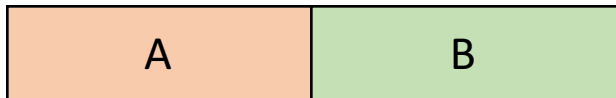
Allocate larger than B, C is overlapped



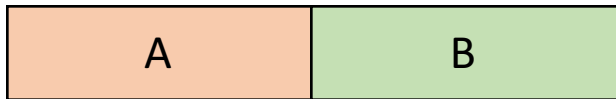
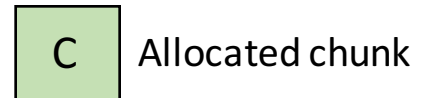


Null Byte Off-by-one

- Extending Allocated Chunks

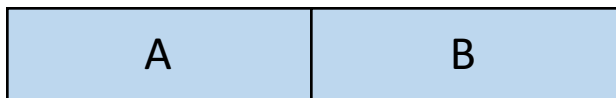


Initial State



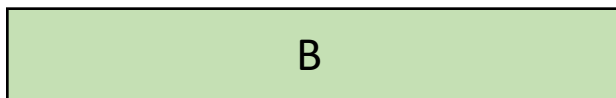
prev_inuse

Overflow: prev_inuse = 0



B is freed, A is consolidated

Example:
PlaidCTF 2015,
datastore



Allocate larger than B, A is overlapped



Libc Problem

- Challenges
 - Binary doesn't import crucial libc function, e.g. `system()`, `mprotect()`
 - We can leak libc base address, but
- Solutions
 - Libc database
 - Leaklib: Arbitrary memory read = arbitrary lib symbol resolution
 - Return-to-dl_resolve



Libc database

- Steps
 - Leak some symbol address
 - Do match in collected libc files
- Existing Resource
 - <https://github.com/niklasb/libc-database>

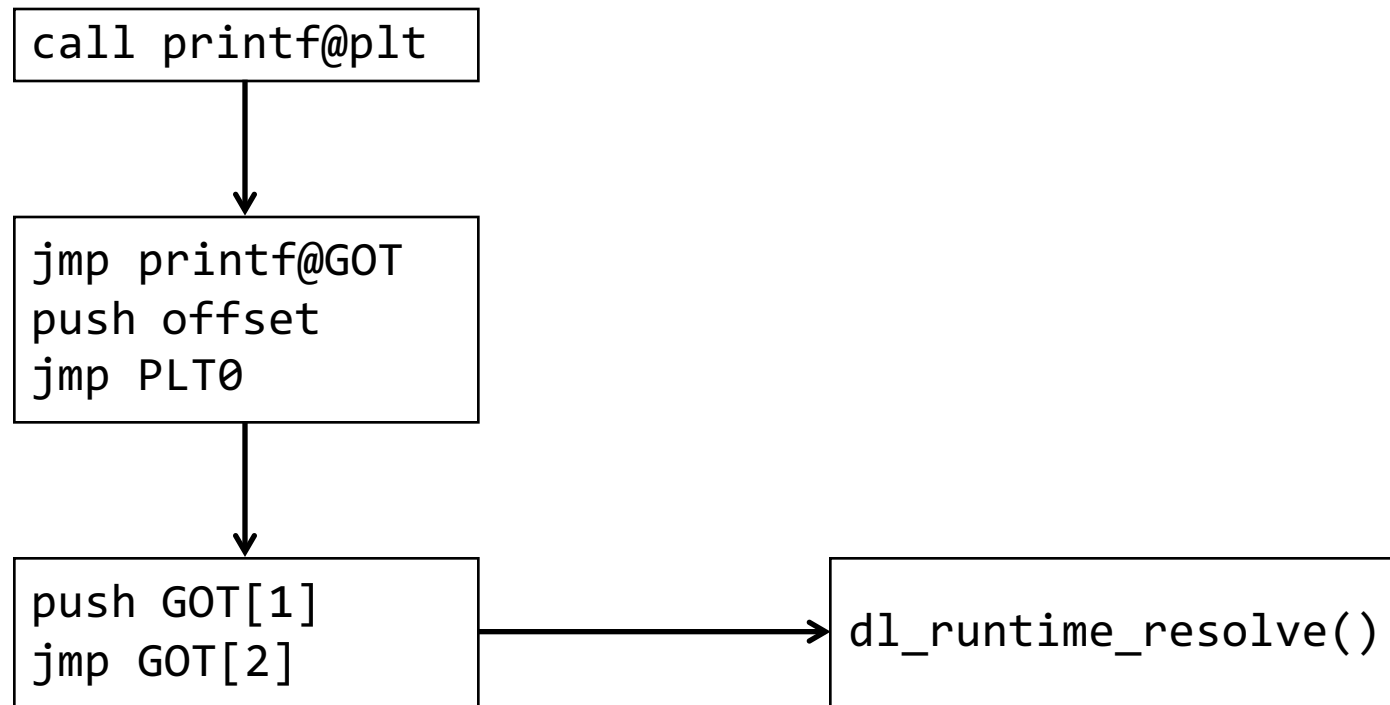
leaklib



- Precondition: Arbitrary memory leak
 - Leak link_map addr (in DT_DEBUG)
 - Leak lib base addr
 - Lookup the hash of desired symbol in hashtbr
 - Lookup in Strtab
 - Lookup in Symtab
- Tools
 - Pwntools: pwnlib.dynelf
 - Pass a memory leak handler



Libc symbol resolve





Return to dl_resolve

- Libc symbol resolving
 - Push link_map
 - Push rel_offset
 - Call `_dl_runtime_resolve(link_map, rel_offset)`
 - `rel_entry = JMPREL + rel_offset;`
 - `Elf32_Sym *sym_entry = SYMTAB[ELF32_R_SYM(rel_entry->r_info)];`
 - `char *sym_name = STRTAB + sym_entry->st_name;`
- How about we control rel_offset?
 - Construct rel_entry
 - Construct symtab
 - Construct strtab
 - Call arbitrary symbol! Win!

Example:

HITCON CTF Quals 2015, readable
Codegate CTF Finals 2015, yocto



Magic System Address

- RCE on Linux
 - System(cmd)
 - mprotect + shellcode
- One Gadget RCE
 - If stdin is redirected, controlled PC directly leads to RCE
 - Invented by Ricky from PPP?

```
000000000003F76A mov    rax, cs:environ_ptr_0
000000000003F771 lea   rdi, aBinSh      ; "/bin/sh"
000000000003F778 lea   rsi, [rsp+188h+var_158]
000000000003F77D mov    cs:lock_3, 0
000000000003F787 mov    cs:sa_refcptr, 0
000000000003F791 mov    rdx, [rax]
000000000003F794 call  execve
000000000003F799 mov    edi, 7Fh      ; status
000000000003F79E call  _exit
000000000003F79E do_system endp
000000000003F79E
```

```
00000000000D7557 mov    rax, cs:environ_ptr_0
00000000000D755E lea   rsi, [rsp+1D8h+var_168]
00000000000D7563 lea   rdi, aBinSh      ; "/bin/sh"
00000000000D756A mov    rdx, [rax]
00000000000D756D call  execve
00000000000D7572 call  abort
```




__libc_csu_init Gadgets for x64

```
.text:000000000400560 loc_400560: ; CODE XREF: __libc_csu_init+54 j
.text:000000000400560      mov     rdx, r13
.text:000000000400563      mov     rsi, r14
.text:000000000400566      mov     edi, r15d
.text:000000000400569      call   qword ptr [r12+rbx*8]
.text:00000000040056D      add     rbx, 1
.text:000000000400571      cmp     rbx, rbp
.text:000000000400574      jnz    short loc_400560
.text:000000000400576 loc_400576: ; CODE XREF: __libc_csu_init+34'j
.text:000000000400576      add     rsp, 8
.text:00000000040057A      pop     rbx
.text:00000000040057B      pop     rbp
.text:00000000040057C      pop     r12
.text:00000000040057E      pop     r13
.text:000000000400580      pop     r14
.text:000000000400582      pop     r15
.text:000000000400584      retn
.text:000000000400584 __libc_csu_init endp
```



References

- <http://phrack.org/issues/58/4.html>
- <http://acez.re/ctf-writeup-hitcon-ctf-2014-stkof-or-modern-heap-overflow/>
- Glibc Adventures: The Forgotten Chunks,
http://www.contextis.com/documents/120/Glibc_Adventures-The_Forgotten_Chunks.pdf
- <https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/>