Glibc Heap Exploitation

CS390R - UMass Amherst

Course Information

- Project 4 still ongoing
- Homework due tonight

Today’s Content

- Browsing Malloc Source
- Heap Exploits basics
- Hooks
- System vs One Gadget
- House of Force
- Tcache Dup

Browsing glibc source

- https://www.gnu.org/software/libc/
- https://code.woboq.org/userspace/glibc/

- malloc.c
  - :3035 void *__libc_malloc(size_t bytes);
  - :3093 void __libc_free(void *mem);

Heap Exploits

- Stack Exploits -> ROP
- Heap Exploits -> ???
- Write-What-Where primitives
  - Write an arbitrary value to an arbitrary location (similar to format string writes)
- Where to write
  - Fini-Array -> Partial Relro makes it non-writable
  - GOT -> Full Relro makes it non-writable
  - Hooks -> Possible until very recent glibc 2.34
- What to write
  - system()
  - One Gadget

Hooks /1

- malloc, free, realloc, etc all have hooks in memory
- When malloc is executed it first checks if there is a listed hook. If there is, the hook is executed instead.
- Debugging feature

```
void * __libc_malloc (size_t bytes) {
    mstate ar_ptr;
    void *victim;

    _Static_assert (PTRDIFF_MAX <= SIZE_MAX / 2, "PTRDIFF_MAX is not more than half of SIZE_MAX");

    // Read the malloc hook and execute it if found
    void *(*hook) (size_t, const void *) = atomic_forced_read (__malloc_hook);
    if (__builtin_expect (hook != NULL, 0))
      return (*hook)(bytes, RETURN_ADDRESS (0));
```

Hooks /2

- We can print out the location/content of these hooks in gdb
- What if we overwrite one of these hooks using our exploit?

System()

- Goal is to call system(“/bin/sh”);
- Arg1 of function (eg. malloc/free) also becomes arg1 of corresponding hook
- Via __malloc_hook
  - malloc(<address of “/bin/sh” string>) -> system(“/bin/sh”);
- Via __free_hook
  - free(<chunk that has “/bin/sh” in its userdata>) -> system(“/bin/sh”);

One Gadget

- execve(“/bin/sh”) call located in libc
- https://github.com/david942j/one_gadget
- Usually has some constraints that need to be met to successfully spawn a shell

```
→ glibc_heap_exploitation_1 git:(main) ✗ one_gadget libc.so.6
0x4f3d5 execve("/bin/sh", rsp+0x40, environ)
constraints:
  rsp & 0xf == 0
  rcx == NULL

0x4f432 execve("/bin/sh", rsp+0x40, environ)
constraints:
  [rsp+0x40] == NULL

0x10a41c execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
```

Patching in custom libc

- Project 5, the challenges will ship with custom libc versions
- The binaries will already be patched with them since edlab does not have some of the required tools

- patchelf --set-interpreter ./ld-2.31.so ./chal
- patchelf --set-rpath ./ ./chal

House Techniques

- http://phrack.org/issues/66/10.html
- Published in 2005
- Theoretical representation of new skills representing “modern” heap exploitation

```
4 - DES-Maleficarum...

  4.1 - The House of Mind
      4.1.1 - FastBin Method
      4.1.2 - av->top Nightmare

  4.2 - The House of Prime
      4.2.1 - unsorted_chunks()

  4.3 - The House of Spirit

  4.4 - The House of Force
      4.4.1 - Mistakes

  4.5 - The House of Lore

  4.6 - The House of Underground
```

House of Force /1

- Goal: Return arbitrary pointer from malloc
- Exploits top chunk size field corruptions
- Very large allocation
  - Usually this would get mmapped to a random location since its too large for the main arena

- What if the set the top chunk size to a very large value?
  - Suddenly we can make an arbitrarily sized allocation starting at the top chunk
- With this we can move the top chunk to an arbitrary location in memory.
- This means that we can control where the next allocation is made

Memory Map

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
          0x3ff000           0x400000 rw-p     1000 0      /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/demo_bin
          0x400000           0x401000 r--p     1000 1000   /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/demo_bin
          0x401000           0x402000 r-xp     1000 2000   /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/demo_bin
          0x402000           0x403000 r--p     1000 3000   /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/demo_bin
          0x403000           0x404000 r--p     1000 3000   /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/demo_bin
          0x404000           0x405000 rw-p     1000 4000   /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/demo_bin
          0x405000           0x426000 rw-p    21000 0      [heap]
    0x7ffff79e2000     0x7ffff7bc9000 r-xp   1e7000 0      /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/libc.so.6
    0x7ffff7bc9000     0x7ffff7dc9000 ---p   200000 1e7000 /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/libc.so.6
    0x7ffff7dc9000     0x7ffff7dcd000 r--p     4000 1e7000 /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/libc.so.6
    0x7ffff7dcd000     0x7ffff7dcf000 rw-p     2000 1eb000 /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/libc.so.6
    0x7ffff7dcf000     0x7ffff7dd3000 rw-p     4000 0      [anon_7ffff7dcf]
    0x7ffff7dd3000     0x7ffff7dfc000 r-xp    29000 0      /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/ld-2.27.so
    0x7ffff7ff4000     0x7ffff7ff6000 rw-p     2000 0      [anon_7ffff7ff4]
    0x7ffff7ff6000     0x7ffff7ffa000 r--p     4000 0      [vvar]
    0x7ffff7ffa000     0x7ffff7ffc000 r-xp     2000 0      [vdso]
    0x7ffff7ffc000     0x7ffff7ffd000 r--p     1000 29000  /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/ld-2.27.so
    0x7ffff7ffd000     0x7ffff7ffe000 rw-p     1000 2a000  /home/seal/github/re-vr-course/lectures/glibc_heap_exploitation_1/ld-2.27.so
    0x7ffff7ffe000     0x7ffff7fff000 rw-p     1000 0      [anon_7ffff7ffe]
    0x7ffffffde000     0x7fffffffe000 rw-p    21000 0      [stack]
0xffffffffff600000 0xffffffffff601000 --xp     1000 0      [vsyscall]
pwndbg>
```

House of Force /2

- Technique is fully mitigated in Glibc 2.29 so we will be using glibc 2.27

Steps:

1. Overwrite topchunk with -1 (0xffffffffffffffff)
2. Calculate size of our allocation:
   a. If we need to wrap around VA: 0xffffffffffffffff - TOP + Target - 0x20
   b. if we don’t need to wrap around VA: Target - 0x20 - TOP
3. Allocate a chunk that that overlaps the target

Tcache Dup

- Goal: Allocate chunk at arbitrary address
- Multiple ways to exploit, a popular one is a double free (glibc <= 2.29)
  - Free the same chunk twice to link it into tcache bins twice
  - Next 2 allocations will return the same chunk so you can overwrite metadata of the 2nd
  - This can be used to link a fake chunk into the tcache bins
  - The third allocation will then be made at the specified location

  -> Write-What-Where vulnerability