Overview

- We hacked GNU cpio, a linux binary
  - Found an integer overflow bug
  - Turned buffer overflow into arbitrary code execution
  - Assigned CVE-2021-38185, severity 7.8/10 (critical)

- Goals
  - Walk through finding the bug
  - Demonstrate the exploit
  - Show that it’s not *that* hard to do
What is cpio?

- GNU binary to archive files
  - Similar to tar, zip, gzip, etc.
- Mostly fallen out of favor for replacements
- Still used in a few notable places
  - Linux kernel filesystem
Why cpio?

- Existing CVEs in klibc cpio
  - Minimal implementation of standard binaries for kernel use
- Wanted to rediscover these CVEs
  - Found all CVE bugs (and some new ones!)
  - Original authors didn’t exploit them!
- Concerns/unfamiliarity with kernel exploitation
- Searched GNU cpio for similar bugs
// ds_fgetstr
// reads a line ending in eos from file f
/* Read the input string. */
next_ch = getc (f);
while (next_ch != eos && next_ch != EOF)
{
    if (insize >= strsize - 1)
    {
        ds_resize (s, strsize * 2 + 2);
        strsize = s->ds_length;
    }
    s->ds_string[insize++] = next_ch;
    next_ch = getc (f);
}
s->ds_string[insize++] = '\0';

/* Expand dynamic string STRING, if necessary, to hold SIZE characters. */
void
ds_resize (dynamic_string *string, int size)
{
    if (size > string->ds_length)
    {
        string->ds_length = size;
        string->ds_string = (char *) xrealloc ((char *) string->ds_string, size);
    }
}
// ds_fgetstr
// reads a line ending in eos from file f
/* Read the input string. */
next_ch = getc(f);
while (next_ch != eos && next_ch != EOF)
{
    if (insize >= strsize - 1)
    {
        ds_resize(s, strsize * 2 + 2);
        strsize = s->ds_length;
    }
    s->ds_string[insize++] = next_ch;
    next_ch = getc(f);
}

s->ds_string[insize++] = '\0';

/* Expand dynamic string STRING, if necessary, to hold SIZE characters. */
void
ds_resize (dynamic_string *string, int size)
{
    if (size > string->ds_length)
    {
        string->ds_length = size;
        string->ds_string = (char *) xrealloc ((char *) string->ds_string, size);
    }
}
Stack vs Heap Overflows

- In this class, we’ve only talked about stack overflows
  - Direct access to RIP
- This exploit relies on memory exploitation
  - Exploiting malloc (usually) means heap overflow
  - Can only affect other heap memory!
- Heap exploits can get very messy
  - Fortunately, we don’t care!
Too much memory

● When you ask for too much memory from malloc, it mmaps
  ○ Creates ("maps") a new page in virtual memory
  ○ 128 kB lower limit (configurable)

● Our bug is int overflow - huge size!

● mmap pages don’t have malloc caching/ protections
  ○ Pros: don’t have to worry about them!
  ○ Cons: can’t exploit them...
House of Muney

- `mmap` exploit reliant on `munmap` (hence the name)
- 16 bytes of metadata before each `mmap` chunk
  - Implements a doubly-linked list of mapped chunks
  - First 8 bytes are previous chunk size
  - Second 8 bytes are own chunk size

Mmap chunk layout:

## Memory Mappings

```bash
pwndbg> vmmap

<table>
<thead>
<tr>
<th></th>
<th>STACK</th>
<th>HEAP</th>
<th>CODE</th>
<th>DATA</th>
<th>RWX</th>
<th>RODATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x555555555400</td>
<td>0x555555555500</td>
<td>0x555555555500</td>
<td>r-p</td>
<td>1000 0</td>
<td>/home/puzzler/vmmap</td>
<td></td>
</tr>
<tr>
<td>0x555555555500</td>
<td>0x555555555600</td>
<td>0x555555555600</td>
<td>r-xp</td>
<td>1000 1000</td>
<td>/home/puzzler/vmmap</td>
<td></td>
</tr>
<tr>
<td>0x555555555600</td>
<td>0x555555555700</td>
<td>0x555555555700</td>
<td>r-p</td>
<td>1000 2000</td>
<td>/home/puzzler/vmmap</td>
<td></td>
</tr>
<tr>
<td>0x555555555700</td>
<td>0x555555555800</td>
<td>0x555555555800</td>
<td>r-p</td>
<td>1000 2000</td>
<td>/home/puzzler/vmmap</td>
<td></td>
</tr>
<tr>
<td>0x555555555800</td>
<td>0x555555555900</td>
<td>0x555555555900</td>
<td>r-wp</td>
<td>1000 3000</td>
<td>/home/puzzler/vmmap</td>
<td></td>
</tr>
<tr>
<td>0x555555555900</td>
<td>0x7ffffff7e0000</td>
<td>0x7ffffff7e0000</td>
<td>r-p</td>
<td>26000 0</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7e0000</td>
<td>0x7ffffff7f5700</td>
<td>0x7ffffff7f5700</td>
<td>r-xp</td>
<td>149000 26000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7f5700</td>
<td>0x7ffffff7fa2000</td>
<td>0x7ffffff7fa2000</td>
<td>r-p</td>
<td>48000 16f000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7fa2000</td>
<td>0x7ffffff7fa3000</td>
<td>0x7ffffff7fa3000</td>
<td>r-p</td>
<td>1000 1ba000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7fa3000</td>
<td>0x7ffffff7fa6000</td>
<td>0x7ffffff7fa6000</td>
<td>r-p</td>
<td>3000 1ba000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7fa6000</td>
<td>0x7ffffff7f9000</td>
<td>0x7ffffff7f9000</td>
<td>r-wp</td>
<td>3000 1bd000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7f9000</td>
<td>0x7ffffff7f9a000</td>
<td>0x7ffffff7f9a000</td>
<td>r-wp</td>
<td>6000 0</td>
<td>[vvar]</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7f9a000</td>
<td>0x7ffffff7f9d000</td>
<td>0x7ffffff7f9d000</td>
<td>r-p</td>
<td>3000 0</td>
<td>[vdso]</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7f9d000</td>
<td>0x7ffffff7fd2000</td>
<td>0x7ffffff7fd2000</td>
<td>r-xp</td>
<td>2000 0</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7fd2000</td>
<td>0x7ffffff7fd3000</td>
<td>0x7ffffff7fd3000</td>
<td>r-p</td>
<td>1000 0</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7fd3000</td>
<td>0x7ffffff7ff3000</td>
<td>0x7ffffff7ff3000</td>
<td>r-wp</td>
<td>20000 1000</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7ff3000</td>
<td>0x7ffffff7ffcc000</td>
<td>0x7ffffff7ffcc000</td>
<td>r-p</td>
<td>9000 21000</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7ffcc000</td>
<td>0x7ffffff7ffd000</td>
<td>0x7ffffff7ffd000</td>
<td>r-p</td>
<td>1000 29000</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7ffffff7ffd000</td>
<td>0x7ffffff7fffd000</td>
<td>0x7ffffff7fffd000</td>
<td>r-p</td>
<td>2000 2a000</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
</tr>
<tr>
<td>0x7fffffffde000</td>
<td>0x7ffffffff0000</td>
<td>0x7ffffffff0000</td>
<td>r-wp</td>
<td>21000 0</td>
<td>[stack]</td>
<td></td>
</tr>
</tbody>
</table>
```
# House of Muney

<table>
<thead>
<tr>
<th>Lower Addresses ▲</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Higher Addresses ▼</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Own Size: 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Data: maverick</td>
</tr>
</tbody>
</table>

Target chunk (write-protected!)
## House of Muney

### Lower Addresses ↑

<table>
<thead>
<tr>
<th>Prev Size: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own Size: 8</td>
</tr>
<tr>
<td>User Data: innocent</td>
</tr>
</tbody>
</table>

### Higher Addresses ↓

<table>
<thead>
<tr>
<th>Prev Size: 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own Size: 8</td>
</tr>
<tr>
<td>User Data: maverick</td>
</tr>
</tbody>
</table>

**Innocent chunk**

**Target chunk**

(write-protected!)
House of Muney

Lower Addresses ↑

- Prev Size: 0
- Own Size: 8
- User Data: evilstuf

- Prev Size: 8
- Own Size: 8
- User Data: innocent

- Prev Size: 8
- Own Size: 8
- User Data: maverick

Higher Addresses ↓

Evil chunk (can overflow)

Innocent chunk

Target chunk (write-protected!)
## House of Muney

<table>
<thead>
<tr>
<th>Address</th>
<th>Prev Size</th>
<th>Own Size</th>
<th>User Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower Addresses ↑</td>
<td>0</td>
<td>8</td>
<td>AAAAAAAA</td>
</tr>
<tr>
<td>Innocent chunk</td>
<td>8</td>
<td>16</td>
<td>innocent</td>
</tr>
<tr>
<td>Target chunk</td>
<td>8</td>
<td>8</td>
<td>maverick</td>
</tr>
<tr>
<td>Evil chunk (can overflow)</td>
<td>0</td>
<td>8</td>
<td>AAAAAAAA</td>
</tr>
</tbody>
</table>

Higher Addresses ↓
House of Muney

Lower Addresses ↑

<table>
<thead>
<tr>
<th>Prev Size: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own Size: 8</td>
</tr>
<tr>
<td>User Data: AAAAAAAA</td>
</tr>
</tbody>
</table>

Evil chunk (can overflow)

Freed (correctly)

Freed (maliciously)

Higher Addresses ↓
House of Muney

Lower Addresses ↑

<table>
<thead>
<tr>
<th>Prev Size: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own Size: 8</td>
</tr>
<tr>
<td>User Data: AAAAAAAAA</td>
</tr>
<tr>
<td>Prev Size: 8</td>
</tr>
<tr>
<td>Own Size: 16</td>
</tr>
<tr>
<td>User Data: someusermemories</td>
</tr>
</tbody>
</table>

Evil chunk (can overflow)

New chunk
Same location as innocent+target

BUT it has r+w perms!
House of Muney

Lower Addresses ↑

<table>
<thead>
<tr>
<th>Prev Size: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own Size: 8</td>
</tr>
<tr>
<td>User Data: AAAAAAAAA</td>
</tr>
<tr>
<td>Prev Size: 8</td>
</tr>
<tr>
<td>Own Size: 16</td>
</tr>
<tr>
<td>User Data: AAAAAAAAAAsysadmin</td>
</tr>
</tbody>
</table>

Higher Addresses ↓

Evil chunk (can overflow)

New chunk
Same location as innocent+target

BUT it has r+w perms!
Memory Mappings

<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Type</th>
<th>Permissions</th>
<th>Size</th>
<th>Offset</th>
<th>File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x555555555400</td>
<td>0x555555555500</td>
<td>r--p</td>
<td>1000 0</td>
<td>/home/puzzler/vmmap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x555555555500</td>
<td>0x555555555600</td>
<td>r-xp</td>
<td>1000 1000</td>
<td>/home/puzzler/vmmap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x555555555600</td>
<td>0x555555555700</td>
<td>r--p</td>
<td>1000 2000</td>
<td>/home/puzzler/vmmap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x555555555700</td>
<td>0x555555555800</td>
<td>r--p</td>
<td>1000 2000</td>
<td>/home/puzzler/vmmap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x555555555800</td>
<td>0x555555555900</td>
<td>rwp</td>
<td>1000 3000</td>
<td>/home/puzzler/vmmap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x555555555900</td>
<td>0x555555555a00</td>
<td>rwp</td>
<td>21000 0</td>
<td>[heap]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7de6000</td>
<td>0x57fffffff7de8000</td>
<td>rwp</td>
<td>2000 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7de8000</td>
<td>0x57fffffff7ee0000</td>
<td>r--p</td>
<td>26000 0</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7e0000</td>
<td>0x57fffffff7f5700</td>
<td>r-xp</td>
<td>149000 26000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7f5700</td>
<td>0x57fffffff7fa2000</td>
<td>r--p</td>
<td>48000 16f000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fa2000</td>
<td>0x57fffffff7fa3000</td>
<td>--p</td>
<td>1000 1ba000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fa3000</td>
<td>0x57fffffff7fa6000</td>
<td>r--p</td>
<td>3000 1ba000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fa6000</td>
<td>0x57fffffff7fa9000</td>
<td>rwp</td>
<td>3000 1bd000</td>
<td>/usr/lib/x86_64-linux-gnu/libc-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fa9000</td>
<td>0x57fffffff7fae000</td>
<td>rwp</td>
<td>6000 0</td>
<td>[vvar]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fd0000</td>
<td>0x57fffffff7fd0000</td>
<td>r--p</td>
<td>3000 0</td>
<td>[vdso]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fd0000</td>
<td>0x57fffffff7fd2000</td>
<td>r-xp</td>
<td>2000 0</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fd2000</td>
<td>0x57fffffff7fd3000</td>
<td>r--p</td>
<td>1000 0</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fd3000</td>
<td>0x57fffffff7ff3000</td>
<td>rwp</td>
<td>20000 1000</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7ff3000</td>
<td>0x57fffffff7ffcc00</td>
<td>r--p</td>
<td>9000 21000</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7ffcc00</td>
<td>0x57fffffff7ffdd00</td>
<td>r--p</td>
<td>1000 29000</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7ffdd00</td>
<td>0x57fffffff7fffe00</td>
<td>rwp</td>
<td>2000 2a000</td>
<td>/usr/lib/x86_64-linux-gnu/ld-2.32.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x57fffffff7fffe00</td>
<td>0x57ffffffff0000</td>
<td>rwp</td>
<td>21000 0</td>
<td>[stack]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dynsym Table

- Dynamically linked - funcs loaded at runtime
- Two important fields
  - Name is the name of the function
  - Value is the offset from libc base address
- Overwrite value + make program search for func = control RIP!

```c
typedef struct {
    Elf64_Word    st_name;    /* Symbol name (string tbl index) */
    unsigned char st_info;    /* Symbol type and binding */
    unsigned char st_other;   /* Symbol visibility */
    Elf64_Section st_shndx;   /* Section index */
    Elf64_Addr     st_value;   /* Symbol value */
    Elf64_Xword    st_size;    /* Symbol size */
} Elf64_Sym;
```
Summary so far

- Set up memory layout
Summary so far

- Set up memory layout
- Overwrite metadata with dstring overflow
Summary so far

- Set up memory layout
- Overwrite metadata with dstring overflow
- Unmap filler (and thus libc)

<table>
<thead>
<tr>
<th>Data: dstring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prev Size: 0</td>
</tr>
<tr>
<td>Own Size: 2**30</td>
</tr>
</tbody>
</table>

*free unmapped space*
Summary so far

- Set up memory layout
- Overwrite metadata with dstring overflow
- Unmap filler (and thus libc)
- Remap a large chunk
Summary so far

- Set up memory layout
- Overwrite metadata with dstring overflow
- Unmap filler (and thus libc)
- Remap a large chunk
- Overwrite and fill with tampered libc
- Win?

<table>
<thead>
<tr>
<th>Prev Size</th>
<th>Own Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2**30</td>
</tr>
<tr>
<td>Data: dstring</td>
<td>Own Size: 16</td>
</tr>
</tbody>
</table>

Data: libc with tampered dynsym
new_save_patterns = (char **) xmalloc (max_new_patterns * sizeof (char *));
new_num_patterns = num_patterns;
ds_init (&pattern_name, 128);

pattern_fp = fopen (pattern_file_name, "r");
if (pattern_fp == NULL)
    open_fatal (pattern_file_name);
while (ds_fgetstr (pattern_fp, &pattern_name, '\n') != NULL)
{
    if (new_num_patterns >= max_new_patterns)
    {
        max_new_patterns += 1;
        new_save_patterns = (char **)
xrealloc ((char *) new_save_patterns,
        max_new_patterns * sizeof (char *));
    }
    new_save_patterns[new_num_patterns] = xstrdup (pattern_name.ds_string);
++new_num_patterns;
}
No frees?

- Lots of memory allocated
  - None (explicitly) freed?
- Does this make exploit impossible?

---

cpio:
no free() is the best security mechanism against heap exploits

me:
realloc

- Tries to extend the chunk if possible
- Otherwise, maps new chunk then munmaps old chunk
  - Can trigger exploit with this munmap!
- Causes dstring to “walk” backwards
Mind the gap!

- Walking makes a large gap
- Gap is unmapped memory
  - Can’t be written or read
  - Attempt to do so causes segfault
- Can only be fixed by filling gap by mapping memory
  - xstrdup in loop
read_pattern_file

new_save_patterns = (char **) xmalloc (max_new_patterns * sizeof (char *));
new_num_patterns = num_patterns;
ds_init (&pattern_name, 128);

pattern_fp = fopen (pattern_file_name, "r");
if (pattern_fp == NULL)
    open_fatal (pattern_file_name);
while (ds_fgetstr (pattern_fp, &pattern_name, '\n') != NULL)
{
    if (new_num_patterns >= max_new_patterns)
    {
        max_new_patterns += 1;
        new_save_patterns = (char **)
xrealloc ((char *) new_save_patterns,
                 max_new_patterns * sizeof (char *));
    }
    new_save_patterns[new_num_patterns] = xstrdup (pattern_name.ds_string);
    ++new_num_patterns;
}
xstrdup

/* Clone an object P of size S, with error checking. There's no need
for xmemdup (P, N, S), since xmemdup (P, N * S) works without any
need for an arithmetic overflow check. */

void *
xmemdup (void const *p, size_t s)
{
    return memcpy (xmalloc (s), p, s);
}

/* Clone STRING. */

char *
xstrdup (char const *string)
{
    return xmemdup (string, strlen (string) + 1);
}
Closing the gap

- We can change `strlen` by inserting null bytes
  - `ds_fgetstr` only breaks on a newline
- We can choose to malloc a single byte each loop
  - Avoids messing with mmap layout
- We can malloc the entire filename each loop
  - Lets us fill gaps in memory map
Limits of integers

- Gap grows from 1 B, to 3 B, to 7 B, to 15 B …
  - When chunk is n bytes, gap is n-1 bytes
- Our chunk is ~1 gigabyte, so our gap is ~1 gigabyte
- dstring uses signed int as index when writing
  - Signed int max is only 2 gigabytes
  - If input > 2 GB, we start writing *behind* the dstring!
new_save_patterns = (char **) xmalloc (max_new_patterns * sizeof (char *));
new_num_patterns = num_patterns;
ds_init (&pattern_name, 128);

pattern_fp = fopen (pattern_file_name, "r");
if (pattern_fp == NULL)
    open_fatal (pattern_file_name);
while (ds_fgetstr (pattern_fp, &pattern_name, '\n') != NULL)
{
    if (new_num_patterns >= max_new_patterns)
    {
        max_new_patterns += 1;
        new_save_patterns = (char **) xrealloc ((char *) new_save_patterns,
                         max_new_patterns * sizeof (char *));
    }
    new_save_patterns[new_num_patterns] = xstrdup (pattern_name.ds_string);
    ++new_num_patterns;
}
Pattern Chunk

- new_save_patterns array gets new filename each loop
  - Grows by 8 bytes with realloc
- Can grow arbitrarily large by sending 1 byte filenames
  - 1 byte name grows array but doesn’t get copied with mmap
- Can use this to shrink the gap!
- Will refer to the minimum mmap size as 1 unit
  - 1 unit = 128 kB
  - 1 GB = 8192 units
  - Need to get dstring to be 8192 units big, but gap less than 8192 units
Leapfrog!
Leapfrog!

- mmap layout can be brute-forced with Python
  - 6 units of initial size puts the dstring close enough!
- Reduce the gap by setting initial pattern chunk size
- Extra command-line args treated as pattern filenames
  - Can add up to 65530 filenames = ~7 units of initial size

- Memory layout solved … now what?
**chdir**

- We need library function not yet called
- `-D [filepath]` tries to cd to `filepath`
  - This occurs after `read_pattern_file`!
- We replace the value field of `chdir with system`
- `-D /bin/sh` will then pop a shell!

```c
typedef struct
{
  Elf64_Word     st_name;  /* Symbol name (string tbl index) */
  unsigned char  st_info;  /* Symbol type and binding */
  unsigned char  st_other;
  Elf64_Section  st_shndx; /* Symbol visibility */
  Elf64_Addr     st_value; /* Section index */
  Elf64_Xword    st_size;  /* Symbol value */
} Elf64_Sym;
```
Exploit Overview

- `cpio -iv -E [pattfile] -D /bin/sh (+ many args)`
  - `-iv` for copy-in mode (required for pattern file)
  - `-E` to include the pattern file
  - `-D` for our payload
  - Many extra args to set pattern chunk initial size
Exploit Overview

- `cpio -iv -E [pattfile] -D /bin/sh (+ many args)`
- Alternately increase the pattern chunk and dstring sizes
- Overwrite pattern chunk metadata
- Send one more filename to realloc (and thus free) the patterns
- Send a large filename to allocate a large chunk over the libc
- Overwrite this chunk with our libc

- Get shell and win!
Should you be concerned?

- I mentioned cpio is still used in a few notable places…
  - Redhat packages
  - Apple installer (brew packages)
- Arbitrary code execution on (nearly) every Linux and Mac machine in the world for the last 15 years!
Should you be concerned? No.

- It’s not networked
- Requires gigabytes of input downloaded
- Patched 13 hours (!) after bug report
Demo time! (hopefully?)

https://youtu.be/F0yKJhu7Vak