Lander Brandt
Opsys
HW3 Writeup

syscallout

Executing the syscalls binary:

```
1507    0.000000 execve("./syscalls", ["./syscalls"], [/* 20 vars */]) = 0
1507    0.001650 brk(0)            = 0x8ab7000
```

do some memory mapping for the program to get it all set up, load some libraries, etc.

```
1507    0.000576 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
1507    0.000319 mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0xb7744000
```

this access/open finds shared libraries that can be loaded pretty quickly, opens them, then reads it into this application's memory space

```
1507    0.000343 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
1507    0.000347 open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
1507    0.000338 fstat64(3, {st_mode=S_IFREG|0644, st_size=18572, ...}) = 0
1507    0.000315 mmap2(NULL, 18572, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb773f000
```

close the shared library cache

```
1507    0.000276 close(3)          = 0
1507    0.000351 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
```

need to get access to library functions somehow

```
1507    0.000259 open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

read the shared libs into this application memory space

```
1507    0.000249 read(3, "\177ELF
\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\226\1\0004\0\0\0"..., 512) = 512
1507    0.000236 fstat64(3, {st_mode=S_IFREG|0755, st_size=1713640, ...}) = 0
1507    0.000214 mmap2(NULL, 1723100, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0xb759a000
1507    0.000189 mmap2(0xb7739000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x19f) = 0xb7739000
1507    0.000199 mmap2(0xb773c000, 10972, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb773c000
```

close the libc library

```
1507    0.000297 close(3)          = 0
1507    0.000224 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0xb7599000
1507    0.000193 set_thread_area({entry_number:-1 -> 6, base_addr:0xb7599900, limit:
1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0,
useable:1}) = 0
```

set read-only portions of memory

Lander Brandt
Opsys
HW3 Writeup

```
1507      0.000246 mprotect(0xb7739000, 8192, PROT_READ) = 0
1507      0.000193 mprotect(0x8049000, 4096, PROT_READ) = 0
1507      0.000202 mprotect(0xb7767000, 4096, PROT_READ) = 0
1507      0.000186 munmap(0xb773f000, 18572) = 0
1507      0.000260 brk(0)             = 0x8ab7000
1507      0.000185 brk(0x8ad8000)      = 0x8ad8000
```

open the input file for read access, load data about it, etc.

```
1507      0.000211 open("test.txt", O_RDONLY) = 3
1507      0.000828 fstat64(3, {st_mode=S_IFREG|0755, st_size=194, ...}) = 0
```

start reading the input file

```
1507      0.000419 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0xb7743000
1507      0.000286 read(3, "b\r\nbabble\r\nbabe\r\nbabel\r\nbaboon\r\n"..., 4096) = 194
1507      0.000403 fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
1507      0.000358 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0xb7742000
```

start writing the buffer to stdout. looks like many write calls are made because internally it detects the linefeed and flushes stdout

```
1507      0.000236 write(1, "b\r\n", 3) = 3
1507      0.000694 write(1, "babble\r\n", 8) = 8
1507      0.000785 write(1, "babe\r\n", 6) = 6
1507      0.000750 write(1, "babel\r\n", 7) = 7
1507      0.000651 write(1, "baboon\r\n", 8) = 8
1507      0.000565 write(1, "babushka\r\n", 10) = 10
1507      0.000590 write(1, "baby\r\n", 6) = 6
1507      0.000770 write(1, "baccalaureate\r\n", 15) = 15
1507      0.000610 write(1, "bacchanalia\r\n", 13) = 13
1507      0.000870 write(1, "bachelor\r\n", 10) = 10
1507      0.000634 write(1, "bacillus\r\n", 10) = 10
1507      0.000639 write(1, "back\r\n", 6) = 6
1507      0.000944 write(1, "a\r\n", 3) = 3
1507      0.000693 write(1, "aardvark\r\n", 10) = 10
1507      0.000712 write(1, "aback\r\n", 7) = 7
1507      0.000841 write(1, "abacus\r\n", 8) = 8
1507      0.000573 write(1, "abaft\r\n", 7) = 7
1507      0.000809 write(1, "abalone\r\n", 9) = 9
1507      0.000772 write(1, "abandon\r\n", 9) = 9
1507      0.000664 write(1, "abandoned\r\n", 11) = 11
1507      0.000676 write(1, "abase\r\n", 7) = 7
1507      0.000726 write(1, "abash\r\n", 7) = 7
1507      0.000861 write(1, "abate\r\n", 7) = 7
1507      0.000597 write(1, "azure\r\n", 7) = 7
```

Lander Brandt
Opsys
HW3 Writeup

perform one last read/write since there was an error in my condition which caused it to still read at eof

```
1507       0.000731 read(3, "", 4096)   = 0
1507       0.000403 write(1, "azure\r\n", 7) = 7
```

close the input file

```
1507       0.000801 close(3)            = 0
```

clean up allocated memory and exit the application

```
1507       0.000466 munmap(0xb7743000, 4096) = 0
1507       0.000311 exit_group(0)       = ?
```

syscallout2

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  -nan    0.000000           0         3           read
  -nan    0.000000           0        24           write
  -nan    0.000000           0         3           open
  -nan    0.000000           0         3           close
  -nan    0.000000           0         1           execve
  -nan    0.000000           0         3         3 access
  -nan    0.000000           0         3           brk
  -nan    0.000000           0         2           munmap
  -nan    0.000000           0         3           mprotect
  -nan    0.000000           0         8           mmap2
  -nan    0.000000           0         4           fstat64
  -nan    0.000000           0         1           set_thread_area
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    58         3 total
```

syscalloutjava

The Java application unlike the C application needs to load both the C runtime /and/ the Java runtime. It loads everything necessary to run the barebones Java VM/runtime, the .class files, then all libraries necessary for those .class files to operate. It then does JIT compilation to translate the compiled Java class into native code (a lot of memory allocation/deallocation), and then performs the similar operations done in the C appliation.

ls comparison

Lander Brandt
Opsys
HW3 Writeup
The actual LS command appears to load a configuration file somewhere along the lines (/etc/nsswitch.conf). It also makes use of the lgetxattr system call, which my application doe not. This call will usually return -1 (operation not supported), so I'm not sure what the purpose of this is. Finally, the ls command makes use of something in coreutils that my application doesn't. Something to do with date formatting it seems?

4a:

I actually got a compilation error and had to comment out the #include <unistd.h> line, but nothing prints because the "write" method overrides the sys call in this scope.

4b:

Goodbye did *not* print for me, but if it's supposed to I'd suppose it's because sys call #1 is the "write" function, so the "syscall" function just calls "write" with the provided varargs.

4c:

The sys call table for 32-bit architecture is be different, so on 32-bit sys call 1 is "exit".

4d:

syscall 1 is write on 64-bit, "exit" on 32-bit.

4e:

The program doesn't output anything because we override the sys call function so that it doesn't actually do anything.

4f:

The assembly is hardcoded to use syscall 4, which is the write syscall, so it now succeeds.

4g:

No, it has to match exactly. I guess it's because the prototypes are exactly the same, so the linker uses the declaration provided in application code instead of library code.

sys32callout.txt

Lander Brandt
Opsys
HW3 Writeup

executing application, loading libraries, etc.

```
22561 execve("./a.out", ["./a.out"], [/* 20 vars */]) = 0
22561 brk(0)                          = 0x9215000
22561 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
22561 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0xfffffffff779f000
22561 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
22561 open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
22561 fstat64(3, {st_mode=S_IFREG|0644, st_size=40453, ...}) = 0
22561 mmap2(NULL, 40453, PROT_READ, MAP_PRIVATE, 3, 0) = 0xfffffffff7795000
22561 close(3)                        = 0
22561 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
22561 open("/lib32/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
22561 read(3, "\17ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\20\226\1\0004\0\0\0"..., 512) = 512
22561 fstat64(3, {st_mode=S_IFREG|0755, st_size=1717736, ...}) = 0
```

library mapping?

```
22561 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0xfffffffff7794000
22561 mmap2(NULL, 1731292, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0xfffffffff75ed000
22561 mprotect(0xf778d000, 4096, PROT_NONE) = 0
22561 mmap2(0xf778e000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1a0) = 0xfffffffff778e000
22561 mmap2(0xf7791000, 10972, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0xfffffffff7791000
22561 close(3)                        = 0
22561 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0xfffffffff75ec000
```

right here appears to be where the context switches

```
22561 set_thread_area(0xffc54560)     = 0
22561 mprotect(0xf778e000, 8192, PROT_READ) = 0
22561 mprotect(0x8049000, 4096, PROT_READ) = 0
22561 mprotect(0xf77c1000, 4096, PROT_READ) = 0
```

unmap the memory that was mapped to kernel space (?)

```
22561 munmap(0xf7795000, 40453)       = 0
```

executing the syscall

```
22561 write(1, "goodbye\n", 8)        = 8
```

exiting the process

Lander Brandt
Opsys
HW3 Writeup

```
22561 exit_group(0)                    = ?
```

comparing with “sigcall”?

I’m not sure exactly what this means but the actual sys call function and the sys call assembly do exactly the same thing (if that’s the second part of this question?).