

Linking:

Case study: Library interpositioning

Case Study: Library Interpositioning

- **Library interpositioning : powerful linking technique that allows programmers to intercept calls to arbitrary functions**
- **Interpositioning can occur at:**
 - Compile time: When the source code is compiled
 - Link time: When the relocatable object files are statically linked to form an executable object file
 - Load/run time: When an executable object file is loaded into memory, dynamically linked, and then executed.

Some Interpositioning Applications

■ Security

- Confinement (sandboxing)
- Behind the scenes encryption

■ Debugging

- In 2014, two Facebook engineers debugged a treacherous 1-year old bug in their iPhone app using interpositioning
- Code in the SPDY networking stack was writing to the wrong location
- Solved by intercepting calls to Posix write functions (write, writev, pwrite)

Source: Facebook engineering blog post at
<https://code.facebook.com/posts/313033472212144/debugging-file-corruption-on-ios/>

Some Interpositioning Applications

■ Monitoring and Profiling

- Count number of calls to functions
- Characterize call sites and arguments to functions
- Malloc tracing
 - Detecting memory leaks
 - **Generating address traces**

Example program

```
#include <stdio.h>
#include <malloc.h>

int main()
{
    int *p = malloc(32);
    free(p);
    return(0);
}                                     int.c
```

- Goal: trace the addresses and sizes of the allocated and freed blocks, without breaking the program, and without modifying the source code.
- Three solutions: interpose on the `lib malloc` and `free` functions at compile time, link time, and load/run time.

Compile-time Interpositioning

```
#ifdef COMPILETIME
#include <stdio.h>
#include <malloc.h>

/* malloc wrapper function */
void *mymalloc(size_t size)
{
    void *ptr = malloc(size);
    printf("malloc(%d)=%p\n",
        (int)size, ptr);
    return ptr;
}

/* free wrapper function */
void myfree(void *ptr)
{
    free(ptr);
    printf("free(%p)\n", ptr);
}
#endif
```

mymalloc.c

Compile-time Interpositioning

```
#define malloc(size) mymalloc(size)
#define free(ptr) myfree(ptr)

void *mymalloc(size_t size);
void myfree(void *ptr);
```

malloc.h

```
linux> make intc
gcc -Wall -DCOMPILETIME -c mymalloc.c
gcc -Wall -I. -o intc int.c mymalloc.o
linux> make runc
./intc
malloc(32)=0x1edc010
free(0x1edc010)
linux>
```

Link-time Interpositioning

```
#ifdef LINKTIME
#include <stdio.h>

void *__real_malloc(size_t size);
void __real_free(void *ptr);

/* malloc wrapper function */
void *__wrap_malloc(size_t size)
{
    void *ptr = __real_malloc(size); /* Call libc malloc */
    printf("malloc(%d) = %p\n", (int)size, ptr);
    return ptr;
}

/* free wrapper function */
void __wrap_free(void *ptr)
{
    __real_free(ptr); /* Call libc free */
    printf("free(%p)\n", ptr);
}
#endif
```

mymalloc.c

Link-time Interpositioning

```
linux> make intl
gcc -Wall -DLINKTIME -c mymalloc.c
gcc -Wall -c int.c
gcc -Wall -Wl,--wrap,malloc -Wl,--wrap,free -o intl
int.o mymalloc.o
linux> make runl
./intl
malloc(32) = 0x1aa0010
free(0x1aa0010)
linux>
```

- The “-Wl” flag passes argument to linker, replacing each comma with a space.
- The “--wrap,malloc” arg instructs linker to resolve references in a special way:
 - Refs to `malloc` should be resolved as `__wrap_malloc`
 - Refs to `__real_malloc` should be resolved as `malloc`

Load/Run-time Interpositioning

```
#ifdef RUNTIME
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

/* malloc wrapper function */
void *malloc(size_t size)
{
    void *(*mallocp)(size_t size);
    char *error;

    mallocp = dlsym(RTLD_NEXT, "malloc"); /* Get addr of libc malloc */
    if ((error = dlerror()) != NULL) {
        fputs(error, stderr);
        exit(1);
    }
    char *ptr = mallocp(size); /* Call libc malloc */
    printf("malloc(%d) = %p\n", (int)size, ptr);
    return ptr;
}
```

mymalloc.c

Load/Run-time Interpositioning

```
/* free wrapper function */
void free(void *ptr)
{
    void (*freep)(void *) = NULL;
    char *error;

    if (!ptr)
        return;

    freep = dlsym(RTLD_NEXT, "free"); /* Get address of libc free */
    if ((error = dlerror()) != NULL) {
        fputs(error, stderr);
        exit(1);
    }
    freep(ptr); /* Call libc free */
    printf("free(%p)\n", ptr);
}
#endif
```

mymalloc.c

Load/Run-time Interpositioning

```
linux> make intr
gcc -Wall -DRUNTIME -shared -fpic -o mymalloc.so mymalloc.c -ldl
gcc -Wall -o intr int.c
linux> make runr
(LD_PRELOAD="./mymalloc.so" ./intr)
malloc(32) = 0xe60010
free(0xe60010)
linux>
```

- The **LD_PRELOAD** environment variable tells the dynamic linker to resolve unresolved refs (e.g., to `malloc`) by looking in `mymalloc.so` first.

Interpositioning Recap

■ Compile Time

- Apparent calls to malloc/free get macro-expanded into calls to mymalloc/myfree

■ Link Time

- Use linker trick to have special name resolutions
 - malloc → __wrap_malloc
 - __real_malloc → malloc

■ Load/Run Time

- Implement custom version of malloc/free that use dynamic linking to load library malloc/free under different names