System Programming
Lab #1

2019-03-12

sp-tas

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)
    - Interpose calls to libc functions.
  - Behind the scenes encryption
    - Automatically encrypt otherwise unencrypted network connections.
- **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 <stdlib.h>
#include <malloc.h>

int main()
{
    free(malloc(10));
    printf("hello, world\n");
    exit(0);
}
```

hello.c

- Goal: trace the addresses and sizes of the allocated and freed blocks, 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
/* Compile-time interposition of malloc and free using C
 * preprocessor. A local malloc.h file defines malloc (free)
 * as wrappers mymalloc (myfree) respectively.
 */

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

/*
 * mymalloc - malloc wrapper function
 */
void *mymalloc(size_t size, char *file, int line)
{
    void *ptr = malloc(size);
    printf("%s:%d: malloc(%d)=%p\n", file, line, (int)size,
ptr);
    return ptr;
}
```

mymalloc.c

Compile-time Interpositioning

```
#define malloc(size) mymalloc(size, __FILE__, __LINE__ )
#define free(ptr) myfree(ptr, __FILE__, __LINE__ )

void *mymalloc(size_t size, char *file, int line);
void myfree(void *ptr, char *file, int line);
```

malloc.h

```
linux> make helloc
gcc -O2 -Wall -DCOMPILETIME -c mymalloc.c
gcc -O2 -Wall -I. -o helloc hello.c mymalloc.o
linux> make runc
./helloc
hello.c:7: malloc(10)=0x501010
hello.c:7: free(0x501010)
hello, world
```

Link-time Interpositioning

```
#ifdef LINKTIME
/* Link-time interposition of malloc and free using the
static linker's (ld) "--wrap symbol" flag. */

#include <stdio.h>

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

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

mymalloc.c

Link-time Interpositioning

```
linux> make hellol
gcc -O2 -Wall -DLINKTIME -c mymalloc.c
gcc -O2 -Wall -Wl,--wrap,malloc -Wl,--wrap,free \
-o hellol hello.c mymalloc.o
linux> make runl
./hellol
malloc(10) = 0x501010
free(0x501010)
hello, world
```

- The "`-Wl`" flag passes argument to linker
- Telling linker "`--wrap,malloc`" tells it 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
 /* Run-time interposition of malloc and free based on
 * dynamic linker's (ld-linux.so) LD_PRELOAD mechanism */
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

void *malloc(size_t size)
{
    void *(*mallocp)(size_t size);
    char *error;
    void *ptr;

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

mymalloc.c

Load/Run-time Interpositioning

```
linux> make hellor
gcc -O2 -Wall -DRUNTIME -shared -fPIC -o mymalloc.so mymalloc.c
gcc -O2 -Wall -o hellor hello.c
linux> make runr
(LD_PRELOAD="/usr/lib/x86_64-linux-gnu/libdl.so ./mymalloc.so"
./hellor)
malloc(10) = 0x559a34eca260
free(0x559a34eca260)
hello, world
```

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

- **-shared: telling linker to make output as a shared objective (.so)**
- **-fPIC: Position-Independent Code**

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

Dynamic Memory Management

`void *malloc(size_t size)`

`malloc` allocates `size` bytes of memory on the process' heap and returns a pointer to it that can subsequently be used by the process to hold up to `size` bytes. The contents of the memory are undefined.

`void *calloc(size_t nmemb, size_t size)`

`calloc` allocates `nmemb*size` bytes of memory on the process' heap and returns a pointer to it that can subsequently be used by the process to hold up to `size` bytes. The contents of the memory are set to zero.

`void *realloc(void *ptr, size_t size)`

`realloc` changes the size of the memory block pointed to by `ptr` to `size` bytes. The contents are copied up to `min(size, old size)`, the rest is undefined.

`void free(void *ptr)`

`free` explicitly frees a previously allocated block of memory.

Dynamic Memory Management

```c
#include <stdlib.h>

void main(void) {
  void *p;
  char *str;
  int  *A;

  // allocated 1024 bytes of memory
  p = malloc(1024);

  // allocated an integer array with 500 integer
  A = (int*)calloc(500, sizeof(int));

  // allocate a string with 16 characters…
  str = (char*)malloc(16*sizeof(char));

  // ...then resize that string to hold 512 characters
  str = (char*)realloc(str, 512*sizeof(char));

  // finally, free all allocated memory
  free(p);
  free(A);
  free(str);
}
```

example1.c

Shared library loading interfaces

- `#include <dlfcn.h>`

- `void *dlopen(const char *filename, int flags)`
  - `dlopen` loads the dynamic shared object (shared library) file named by the null-terminated string filename and returns an opaque "handle" for the loaded object.
  - **`RTLD_LAZY`** – perform lazy binding
  - **`RTLD_NOW`** – all undefined symbols in the shared object are resolved before `dlopen` returns

- `void *dlsym(void *handle, const char *symbol)`
  - `dlsym` takes a "handle" of a dynamic loaded shared object returned by `dlopen` and returns the address where that symbol is loaded into memory
  - `RTLD_DEFAULT` – find the first occurrence of the desired symbol
  - **`RTLD_NEXT`** – find the next occurrence of the desired symbol in the search order after the current object

- `int dlclose(void *handle)`
  - `dlclose` decrements the reference count on the dynamically loaded shared object referred to by `handle`

Lab Assignment #1 – Linker Lab

- Download skeleton code from eTL

- Hand In
  - Upload your files **eTL**
  - A tarball of your implementation (20/20/20/+10 pts for each part)
  - A report (10 pts)

- PLEASE, READ the Hand-out!!!

- Assigned: Mar. 12
- Deadline: Mar. 26, 11:59:59 PM

- Lab #2 (3/19) will be Q&A session

(Part 1) Tracing dynamic memory allocation

test1.c

```
#include <stdlib.h>

void main(void) {
  void *a;

  a = malloc(1024);
  a = malloc(32);
  free(malloc(1));
  free(a);
}
```

output

```
stuXXX@spN ~/linklab/part1 $ make run test1
[0001] Memory tracer started.
[0002]            (nil) : malloc( 1024 ) = 0xb87010
[0003]            (nil) : malloc( 32 ) = 0xb87420
[0004]            (nil) : malloc( 1 ) = 0xb87450
[0005]            (nil) : free( 0xb87450 )
[0006]            (nil) : free( 0xb87420 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          0
[0012]
[0013] Memory tracer stopped.
stuXXX@spN ~/linklab/part1 $
```

(Part 2) Tracing unfreed memory

test1.c

```
#include <stdlib.h>

void main(void) {
  void *a;

  a = malloc(1024);
  a = malloc(32);
  free(malloc(1));
  free(a);
}
```

output

```
stuXXX@spN ~/linklab/part2 $ make run test1
[0001] Memory tracer started.
[0002]            (nil) : malloc( 1024 ) = 0x2415060
[0003]            (nil) : malloc( 32 ) = 0x24154c0
[0004]            (nil) : malloc( 1 ) = 0x2415540
[0005]            (nil) : free( 0x2415540 )
[0006]            (nil) : free( 0x24154c0 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          33
[0012]
[0013] Non-deallocated memory blocks
[0014]   block                size        ref cnt   caller
[0015]   0x2415060            1024        1         ???:0
[0016]
[0017] Memory tracer stopped.
stuXXX@spN ~/linklab/part2 $
```

(Part 3) Pinpointing call locations

```
//
// return the PC of the callsite to the dynamic memory management function
//
//   fname      pointer to character array to hold function name
//   fnlen      length of character array
//   ofs        pointer to offset to hold PC offset into function
//
// returns
//    0         on success
//   <0         on error
//
int get_callinfo(char *fname, size_t fnlen, unsigned long long *ofs);
```

~/linklab/part3/callinfo.h

```
int get_callinfo(char *fname, size_t fnlen,
                 unsigned long long *ofs)
{
  return -1;
}
```

~/linklab/part3/callinfo.c

(Part 3) Pinpointing call locations

```
stuXXX@spN ~/linklab/part3 $ make run test1
[0001] Memory tracer started.
[0002]          main:6  : malloc( 1024 ) = 0x14f0060
[0003]          main:10 : malloc( 32 ) = 0x14f04c0
[0004]          main:1d : malloc( 1 ) = 0x14f0540
[0005]          main:25 : free( 0x14f0540 )
[0006]          main:2d : free( 0x14f04c0 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          33
[0012]
[0013] Non-deallocated memory blocks
[0014]   block              size        ref cnt    caller
[0015]   0x14f0060          1024        1          main:6
[0016]
[0017] Memory tracer stopped.
stuXXX@spN ~/linklab/part3 $
```

(Part 3) Backtracing library

- `#include <libunwind.h>`

- `int unw_get_proc_name(unw_cursor_t *cp, char *bufp, size_t len, unw_word_t *offp);`
  - Get name of current procedure
  - Returns the name of the procedure that created the stack frame identified by argument cp

- `int unw_getcontext(unw_context_t *ucp);`
- `int unw_init_local(unw_cursor_t *c, unw_context_t *ctxt);`

- `int unw_step(unw_cursor_t *cp);`

(Part 3) Backtracing library

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#define UNW_LOCAL_ONLY
#include <libunwind.h>

static void print_backtrace(void)
{
    //
    // having fun with functions of unwind library =)
    //
}

void dummy(void)
{
    print_backtrace();
}

int main(void)
{
    dummy();

    return 0;
}
```

output

```
root@sp3:/home/ta/hkim# ./test
ip = 0x55c7345cdc82 (dummy), sp = 0x7ffca4c637e0 off=0x4
ip = 0x55c7345cdc8e (main), sp = 0x7ffca4c637f0 off=0x4
ip = 0x7f8c3c09fb97 (__libc_start_main), sp = 0x7ffca4c63800 off=0xe2
ip = 0x55c7345cdala (_start), sp = 0x7ffca4c638c0 off=0x25
```

objdump -d test

```
0000000000000c79 <dummy>:
 c79:   55                      push   %rbp
 c7a:   48 89 e5                mov    %rsp,%rbp
 c7d:   e8 78 fe ff ff          callq  afa <print_backtrace>
 c82:   90                      nop
 c83:   5d                      pop    %rbp
 c84:   c3                      retq

0000000000000c85 <main>:
 c85:   55                      push   %rbp
 c86:   48 89 e5                mov    %rsp,%rbp
 c89:   e8 eb ff ff ff          callq  c79 <dummy>
 c8e:   b8 00 00 00 00          mov    $0x0,%eax
 c93:   5d                      pop    %rbp
 c94:   c3                      retq
 c95:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 c9c:   00 00 00
 c9f:   90                      nop
```

(Bonus) Detect and ignore illegal deallocations

- Detect double- free / illegal free

test4.c – test case for bonus part

```
#include <stdlib.h>

void main(void) {
  void *a;

  a = malloc(1024);
  free(a);
  free(a);
  free((void*)0x1706e90);
}
```

output

```
stuXXX@spN ~/linklab/bonus $ make run test4
[0001] Memory tracer started.
[0002]          main:6  : malloc( 1024 ) = 0x1b30060
[0003]          main:11 : free( 0x1b30060 )
[0004]          main:19 : free( 0x1b30060 )
[0005]              *** DOUBLE_FREE  *** (ignoring)
[0006]          main:23 : free( 0x1706e90 )
[0007]              *** ILLEGAL_FREE *** (ignoring)
[0008]
[0009] Statistics
[0010]   allocated_total      1024
[0011]   allocated_avg        1024
[0012]   freed_total          1024
[0013]
[0014] Memory tracer stopped.
stuXXX@spN ~/linklab/bonus $
```

Skeleton code snippet

```c
//
// init - this function is called once when the shared library is loaded
//
__attribute__((constructor))
void init(void)
{
  char *error;

  LOG_START();

  // initialize a new list to keep track of all memory (de-)allocations
  // (not needed for part 1)
  list = new_list();

  // ...
}
```

```c
//
// fini - this function is called once when the shared library is unloaded
//
__attribute__((destructor))
void fini(void)
{
  // ...

  LOG_STATISTICS(0L, 0L, 0L);

  LOG_STOP();

  // free list (not needed for part 1)
  free_list(list);
}
```

Utilities

- Read someone else's code

- `memlog.c`
  - `int mlog(int pc, const char* fmt, …)`

- `memlist.c`
  - `item *new_list(void)`
  - `void free_list(void)`
  - `item *alloc(item *list, void *ptr, size_t size)`
  - `item *dealloc(item *list, void *ptr)`
  - `item *find(item *list, void *ptr)`
  - `void dump_list(item *list)`