Static hooking through predefinition

devblogs.microsoft.com/oldnewthing/20170427-00

April 27, 2017

Raymond Chen

A customer had a program that incorporated source code from two different third parties, let's call them Contoso and LitWare. These libraries were originally written for Linux, and they are trying to port them to Windows.

Contoso's library implements some useful feature that they want to use. LitWare's library implements some fancy memory management and wants to intercept all calls to `malloc`, `free`, and related functions. In particular, it wants to intercept the calls from Contoso.

The customer knew that they could use Detours to do the intercepting, but that would require them to obtain a professional license, and the cost was a concern.

Fortunately, since the customer is building all the libraries themselves, they can make changes to the code and recompile.

I suggested using this header file:

```
// interceptable.h
extern void* (*intercepted_malloc)(size_t);
#define malloc interceptable_malloc

extern void (*intercepted_free)(void*);
#define free interceptable_free

... repeat as necessary ...
```

Include this header file after `stdlib.h` so that all calls to the functions you care about are redirected to the `intercepted_...` wrappers.

The implementation file is simple:

```
// interceptable.c
#undef malloc
void* (*intercepted_malloc)(size_t) = malloc;

#undef free
void (*intercepted_free)(void*) = free;

... repeat as necessary ...
```

When the LitWare library wants to intercept the functions of interest, it does this:

```
void* (*original_malloc)(size_t);

void* replacement_malloc(size_t size)
{
 ... replacement can call original_malloc() ...
}

void install_malloc_wrapper()
{
  original_malloc = intercepted_malloc;
  intercepted_malloc = replacement_malloc;
}
```

Now, when the Contoso library calls `intercepted_malloc`, it ends up calling `replacement_malloc`, which can do whatever it wants (including calling the original `malloc`).

Raymond Chen

Follow