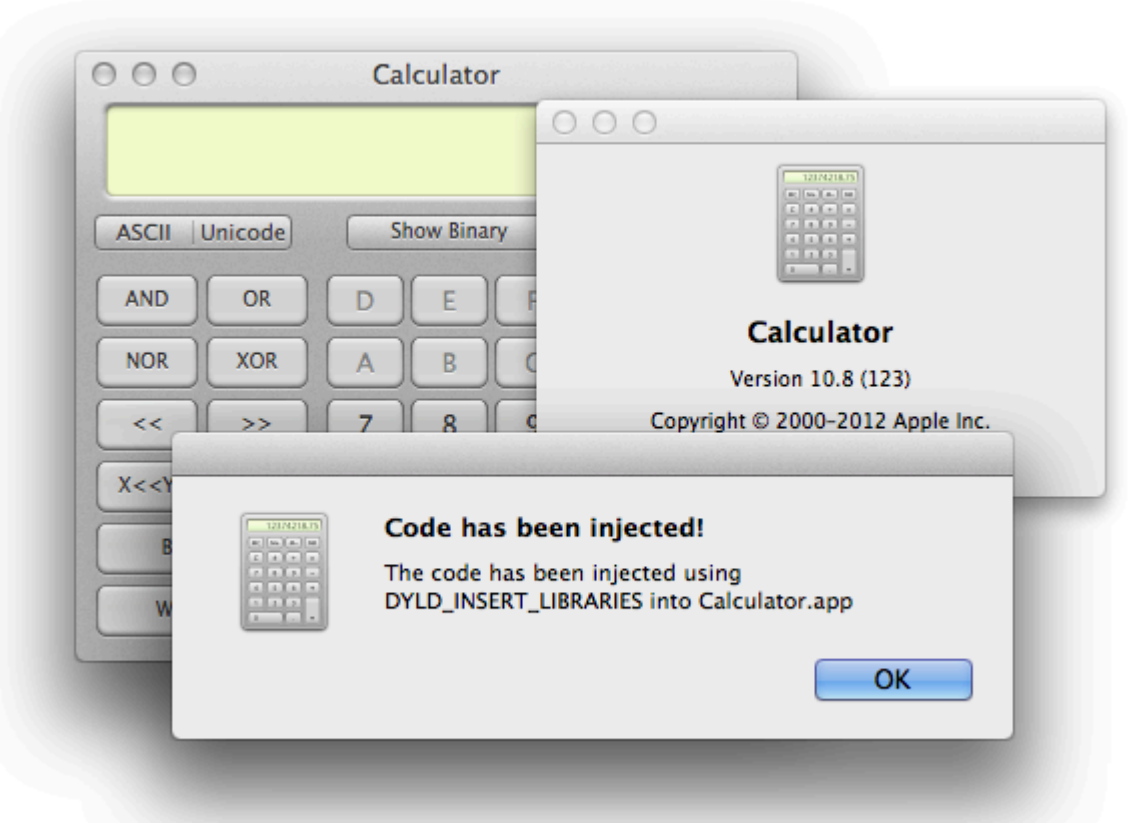
Simple code injection using DYLD_INSERT_LIBRARIES

Published: 2012-12-18 · Archived: 2026-04-05 21:44:42 UTC

In the following article I will describe a simple method to inject code into executables on Mac OS X 10.8 using the DYLD_INSERT_LIBRARIES environment variable.

I also wrote a simple launcher that starts Calculator.app and injects code to modify the About Box of Calculator.app. When bringing the About Box window of Calculator.app, a custom alert will be displayed:



Note: Code injection should be used with care. You should probably not ship applications using code injection and the discussed environment variable DYLD_INSERT_LIBRARIES.

On Mac OS X 10.8, there are several ways to inject code into an arbitrary 64-bit process:

- *writing a plugin* if the targeted application supports plugins. Such a solution is possible for applications like Safari, Mail, Xcode… but not for the Calculator.app.
- Injecting code through *Scripting Additions*: you create a daemon to watch the launch of the targeted application and this daemon sends a custom Apple event to trigger the code injection. This solution is apparently used by 1Password 3.
- you can inject code using *Mach ports* by using mach_override and mach_inject but it has some downsides (you need to be in the procmod group or root).
- Injecting code through a *kernel extension*: This is really powerful but the code runs in the kernel.
- *Modifying the binary* of the application but this can't be reused and you need to reapply the changes for each new version of the application.
- Injecting code using the `DYLD_INSERT_LIBRARIES` environment variable: this is a really simple solution to implement but you can only inject code in the application you launch.

Now that we know the different ways for injecting code, let's look more in details at the `DYLD_INSERT_LIBRARIES` environment variable.

Using DYLD_INSERT_LIBRARIES has several advantages:

- It is simple to implement.

- All the code runs in userland and you don't need to be root or run with the procmod group.
- You don't rely on complex third party code.
- It can be used to inject code in any application you start.
- You only inject code in a specific application.

It has some downsides:

- You can only inject code in the applications you start.

On Mac OS X, the dynamic linker (dyld) can be used to load a dynamic library specified in the DYLD_INSERT_LIBRARIES environment variable into an executable. I created a simple dynamic library that replaces the `-[CalculatorController showAbout:]` method of Calculator.app with a custom implementation. Following is the code of the dynamic library (ACCalculatorOverrides.m). Here is what it does:

- I created an object called ACCalculatorOverrides with a `+(void)load` class method. This code is invoked really early by the runtime. It exchanges the implementation of the method `-[CalculatorController showAbout:]` by the implementation of `-[ACCalculatorOverrides patchedShowAbout:]`.
- The method `-[ACCalculatorOverrides patchedShowAbout:]` will call the original method to display the original About Box and then display a custom alert.

```
 #import "ACCalculatorOverrides.h"

#include <stdio.h>
#include <objc/runtime.h>
#include <Foundation/Foundation.h>
#include <AppKit/AppKit.h>

static IMP sOriginalImp = NULL;

@implementation ACCalculatorOverrides

+(void)load
{
    // We replace the method -[CalculatorController showAbout:] with the method -[ACCalculatorOverrides patchedS
    Class originalClass = NSClassFromString(@"CalculatorController");
    Method originalMeth = class_getInstanceMethod(originalClass, @selector(showAbout:));
    sOriginalImp = method_getImplementation(originalMeth);

    Method replacementMeth = class_getInstanceMethod(NSClassFromString(@"ACCalculatorOverrides"), @selector(patc
    method_exchangeImplementations(originalMeth, replacementMeth);
}

-(void)patchedShowAbout:(id)sender
{
    // We first call the original method to display the original About Box
    sOriginalImp(self, @selector(showAbout:), self);
```

```
    // Run our custom code which simply display an alert
    NSAlert *alert = [NSAlert alertWithMessageText:@"Code has been injected!" defaultButton:@"OK" alternateButt
    [alert runModal];
}

@end
```

It is possible to manually build this dynamic library by running in the Terminal:

```
gcc -framework AppKit -framework Foundation -o CalculatorOverrides.dylib -dynamiclib ACCalculatorOverrides.m
```

and manually injecting it into Calculator.app by running in the Terminal:

```
DYLD_INSERT_LIBRARIES=/PATH_TO/CalculatorOverrides.dylib /Applications/Calculator.app/Contents/MacOS/Calculato
```

But to make it simpler to use, let’s write a launcher. The launcher will be a background-only application (LSUIElement set to YES) that will execute the previously mentioned command line and then quit itself. Here is the code of the launcher:

```
#import "AppDelegate.h"

@implementation AppDelegate

- (void)dealloc
{
    [super dealloc];
}

-(void)bringToFrontApplicationWithBundleIdentifier:(NSString*)inBundleIdentifier
{
    // Try to bring the application to front
    NSArray* appsArray = [NSRunningApplication runningApplicationsWithBundleIdentifier:inBundleIdentifier];
    if([appsArray count] > 0)
    {
        [[appsArray objectAtIndex:0] activateWithOptions:NSApplicationActivateIgnoringOtherApps];
    }

    // Quit ourself
    [[NSApplication sharedApplication] terminate:self];
}

-(void)launchApplicationWithPath:(NSString*)inPath andBundleIdentifier:(NSString*)inBundleIdentifier
{
    if(inPath != nil)
```

```
    {
        // Run Calculator.app and inject our dynamic library
        NSString *dyldLibrary = [[NSBundle bundleForClass:[self class]] pathForResource:@"CalculatorOverrides" c
        NSString *launcherString = [NSString stringWithFormat:@"DYLD_INSERT_LIBRARIES=\"%@\" \"%@\" &", dyldLibr
        system([launcherString UTF8String]);

        // Bring it to front after a delay
        [self performSelector:@selector(bringToFrontApplicationWithBundleIdentifier:) withObject:inBundleIdentif
    }
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    NSString *calculatorPath = @"/Applications/Calculator.app/Contents/MacOS/Calculator";
    if([[NSFileManager defaultManager] fileExistsAtPath:calculatorPath])
        [self launchApplicationWithPath:calculatorPath andBundleIdentifier:@"com.apple.calculator"];
}

@end
```

Download: You can download here the compiled Calculator Launcher. If you launch it, it will launch the Calculator.app and inject the code to display an alert when you display the About Box of Calculator.app. If you are interested by the source code, the full sources are available here.

Source: https://blog.timac.org/2012/1218-simple-code-injection-using-dyld_insert_libraries/