Code-Inject on iOS

Created by Razzile for iOSCheaters.com

Code Inject is a great way to apply the cheats you make in IDA with the customizability and ease of use of Mobile Substrate. This Guide will help you understand the process and get set up with code Injection.

Prerequisites:

- Theos and friends set up (including SDK and perl)
- A file transfer tool of some kind (I am using WinSCP)
- The code inject header (found here)
- The code inject NIC Template (optional, found here)
- Some basic coding knowledge

The Code Explained:

If you are only interested in using the code then skip to here.

Let's start by looking at the imports:

```
13  #include <stdio.h>
14  #include <stdlib.h>
15  #include <unistd.h>
16  #include <sys/types.h>
17  #include <mach/mach.h>
```

There are a few used here but the main ones you should note however are <mach/mach.h> and <sys/types.h>.

Now we move on to the functions

Not going in list order, but instead order used. Here is a run-down of all the functions used.

Let's start with writeData() which is the function you will be calling.

writeData()

```
79  int writeData(vm_address_t offset, unsigned int data)
80  {
81  if(getType(data)) {
82  write32(offset, data);
83  }
84  else {
85  write16(offset, (unsigned short)data);
86  }
87
88  return 0;
89  }
```

getType() returns the size of the data (for 2 byte or 4 byte data) more on this later

These functions actually do the writing to memory.

What happens here is the function getType() returns the size of the data and then depending on the output of getType, calls the write function for that size (with 32 being for 4 byte and 16 being for 2 byte).

getType()

the getType() function returns gets the size of the data passed to it and returns true if 4 byte (or higher) and false if the data is 2 byte.

```
71  bool getType(unsigned int x)
72  {
73  int a = x & 0xffff8000;
74  int b = a + 0x00008000;
75
76  int c = b & 0xffff7fff;
77  return c; //true = 32bit false = 16bit
78  }
```

Here unsigned int x is the argument that holds the data. Some lovely binary arithmetic is performed on it (that is pretty hard to explain) and int c returns either 1 (true) or 0 (false) at the end.

Write()

The write functions (write16 and write32) actually do the writing to memory.

Let's take a look at write32 (write16 is the same but has unsigned short as an argument)

CFSwapInt() switches the endianness of data (because in memory everything is in reverse)

A port allows a process to have access to another process, but as we our editing our own process we only need a port for our own process mach_task_self()

```
19  int write32(vm_address_t offset, unsigned int data)
20  {
21  data = CFSwapInt32(data);
22  kern_return_t err;
23  mach_port_t port = mach_task_self();
24
25  err = vm_protect(port, (vm_address_t)offset, sizeof(data), NO,VM_PROT_READ|VM_PROT_WRITE|VM_PROT_COPY);
26  if(err != KERN_SUCCESS)
27      {
28          NSLog(@"prot error: %s \n", mach_error_string(err));
29          return 0;
30      }
31
32  vm_write(port, (vm_address_t) offset, (vm_address_t)&data, sizeof(data));
33
34  err = vm_protect(port, (vm_address_t)offset, sizeof(data), NO,VM_PROT_READ|VM_PROT_EXECUTE);
35  if(err != KERN_SUCCESS)
36      {
37        NSLog(@"prot error: %s \n", mach_error_string(err));
38          return 0;
39      }
40      NSLog(@"all is well here shipmate");
41      return 1;
42  }
```

vm_write() writes our data to memory. It writes data to offset for the size of data

Sets protections back to normal

vm_protect() changes memory protections to allow us to write to memory.

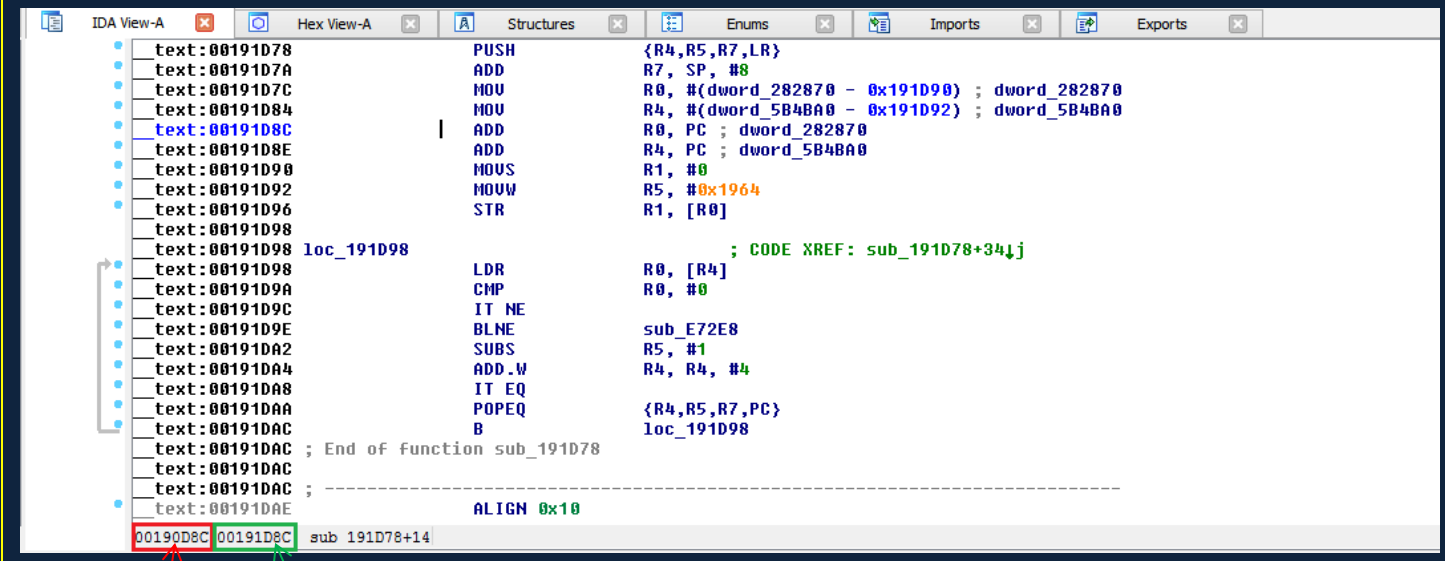
Hopefully this explanation will help you understand the process of code injection more.

For More reading please check out here. This blog has a lot of goodies in it and helped me learn a lot of cool things!

Using The Code

1) IDA

Code Inject uses different offsets from hex editing so you will need to know how to get the correct offsets.



This is the wrong offset

This is the correct offset to use for code inject but instead of writing 00012345 you can write 0x12345 instead

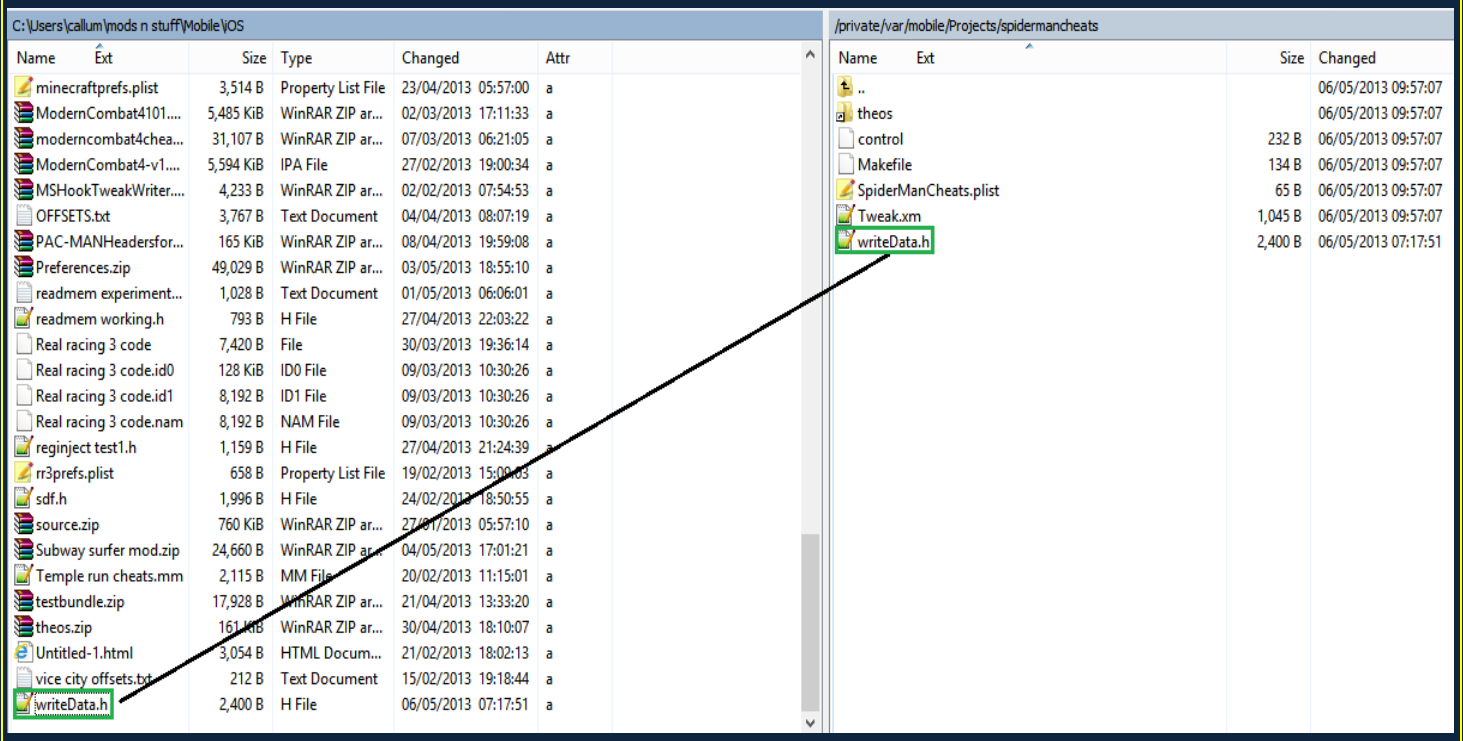
Once you have your offsets and data, you can move on to creating the tweak.

2) Creating the Tweak

First, create a new tweak project using theos:

```
Using username "root".
root@192.168.0.4's password:
Callums-jb-iPad:~ root# cd /var/mobile/Projects
Callums-jb-iPad:/var/mobile/Projects root# $THEOS/bin/nic.pl
NIC 2.0 - New Instance Creator
------------------------------
  [1.] iphone/application
  [2.] iphone/library
  [3.] iphone/preference_bundle
  [4.] iphone/tool
  [5.] iphone/tweak
Choose a Template (required): 5
Project Name (required): SpiderMan Cheats
Package Name [com.yourcompany.spidermancheats]: com.ioscheaters.spidermancheats
Author/Maintainer Name [System Administrator]: Alcatraz
[iphone/tweak] MobileSubstrate Bundle filter [com.apple.springboard]: com.gamelo
ft.AmazingSpiderMan
Instantiating iphone/tweak in spidermancheats/...
Done.
Callums-jb-iPad:/var/mobile/Projects root#
```

Now using a file tool (I will be using WinSCP) transfer the writeData.h header into your project folder



Once that is done, we can now start coding

3) Coding

Now the project is fully set up, we can begin coding.

Open up the Tweak.xm and delete its contents leaving a blank document.

Now at the top, add this:

```
#import <Foundation/Foundation.h>
#import “writeData.h”
```

Now that is done, you can write data wherever you wish.

To write data, you do this

```
writeData(0xADDRESS, 0xDATA);
```

If you only want to write when the app starts, do this:

```
%ctor {

writeData(0xADDRESS, 0xDATA);

}
```

Otherwise, put the writeData code wherever necessary.

Here is an example of a fully working tweak:

```
#import <Foundation/Foundation.h>
#import “writeData.h”


%ctor {
writeData(0x6E2F0, 0x1EFF2FE1);
}
```

The above will write 1EFF2FE1 (BX LR) at the address 0x6E2F0

4) Compilation and error solving

Compilation

To compile, go back to terminal (making sure you are still in your project directory) and type:

```
make
```

Or alternately you can compile and make a deb by typing:

```
make package
```

Add an “install” on the end of that and it will install the deb after it’s made.

```
make package install
```

After that and if it compiles, you are done. You can now spread your hack wherever you like ☺

Bug Squashing

As with all code, you can expect to see a few bugs here and there. A good way to see what’s wrong is to type:

```
make messages=yes
```

This will show you some very technical jargon behind the compilation progress, but it may help you fix your problem.

Another way is to open your makefile and on the very top line, add:

```
GO_EASY_ON_ME=1
```

This tells the compiler to ignore any warnings that occur in the compilation process

For any more help, please join the #theos channel on irc.saurik.com. the people there are real pros and will help you out in any way possible!

Examples

```
#import <Foundation/Foundation.h>
#import “writeData.h”


%ctor {
writeData(0x6E2F0, 0x1EFF2FE1);
}
```

Writes 1EFF2FE1 (BX LR in ARM) to 0x6E2F0

```
#import <Foundation/Foundation.h>
#import “writeData.h”


%hook GameLayer

-(void)loadView:(id)fp8 {

writeData(0x1E420, 0x7047);

return %orig;

}

%end
```

This example hooks on to a GameLayer class and adds custom code to it’s “loadView” method, writing 7047 (BX LR 2 byte) at 0x1E420.

Conclusion

Code inject is a wonderful way for you to apply your binary hacks over a dylib. And as it is a dylib it’s easy to install, much easier than a binary hack.

If you need any extra support, just PM me on iOSCheaters ☺

-Razzile