Debugging Mac OSX Applications with IDA Pro

Copyright 2020 Hex-Rays SA

Overview

IDA Pro fully supports debugging native OSX applications.

However, this task is riddled with gotchas and often times it demands precise workarounds that are not required for other platforms. In this tutorial we will purposefully throw ourselves into the various pitfalls of debugging on a Mac, in the hopes that learning things the hard way will ultimately lead to a smoother experience overall.

Begin by downloading <u>samples.zip</u> which contains the sample applications used in this writeup.

Supported OSX Versions

It is Apple's unofficial policy to support only the 3 latest versions of OSX. Any older versions will usually stop receiving security updates. Thus, Hex-Rays has adopted the same policy. We can only guarantee mac debugging support on versions that Apple is actively maintaining.

Note however that our mac debugger has been perfectly functional since it was first developed for OSX 10.5 Leopard - so debugging on old unsupported OSX versions will probably work, but we cannot promise to fix any issues that arise. Doing so will leave us vulnerable to old unfixed security flaws in the OS.

Codesigning & Permissions

It is important to note that a debugger running on OSX requires <u>special permissions</u> in order to function properly. This means that the debugger itself must be codesigned in such a way that MacOS allows it to inspect other processes.

The main IDA Pro application is not codesigned in this way. Later on we'll discuss why.

To quickly demonstrate this, let's open a binary in IDA Pro and try to debug it. In this example we'll be debugging the **helloworld** app from <u>samples.zip</u> on MacOSX 10.15 Catalina using IDA 7.5. Begin by loading the file in IDA:

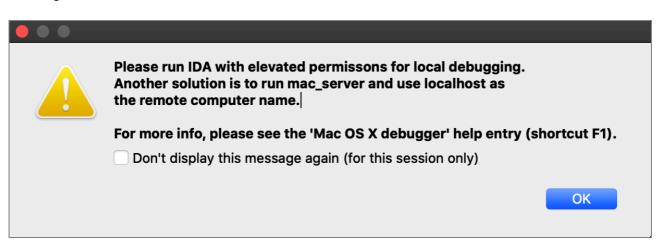
Now go to menu Debugger>Select debugger and select Local Mac OS X Debugger:



Immediately IDA should print a warning message to the Output window:

This program must either be codesigned or run as root to debug mac applications.

This is because IDA is aware that it is not codesigned, and is warning you that attempting to debug the target application will likely fail. Try launching the application with shortcut **F9**. You will likely get this error message:



Codesigning IDA Pro might resolve this issue, but we have purposefully decided not to do this. Doing so would require refactoring IDA's internal plugin directory structure so that it abides by Apple's bundle structure guidelines. This would potentially break existing plugins as well as third-party plugins written by users. We have no plans to inconvenience our users in such a way.

Also note that running IDA as root will allow you to use the Local Mac OS X Debugger without issue, but this is not advisable.

A much better option is to use IDA's mac debug server - discussed in detail in the next section.

Using the Mac Debug Server

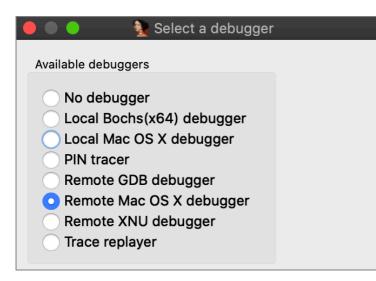
A good workaround for the debugging restrictions on OSX is to use IDA's debug server - even when debugging local apps on your mac machine. The mac debug server is a standalone application that communicates with IDA Pro via IPC, so we can ship it pre-codesigned and ready for debugging right out of the box:

\$ codesign -dvv /Applications/IDA\ Pro\ 7.5/idabin/dbgsrv/mac_server64 Executable=/Applications/IDA Pro 7.5/ida.app/Contents/MacOS/dbgsrv/mac_server64 Identifier=com.hexrays.mac_serverx64 Format=Mach-0 thin (x86_64) CodeDirectory v=20100 size=6090 flags=0x0(none) hashes=186+2 location=embedded Signature size=9002 Authority=Developer ID Application: Hex-Rays SA (ZP7XF62S2M) Authority=Developer ID Certification Authority Authority=Apple Root CA Timestamp=May 19, 2020 at 4:13:31 AM

Let's try launching the server:

\$ /Applications/IDA\ Pro\ 7.5/idabin/dbgsrv/mac_server64 IDA Mac OS X 64-bit remote debug server(MT) v7.5.26. Hex-Rays (c) 2004-2020 Listening on 0.0.0.0:23946...

Now go back to IDA and use menu Debugger>Switch debugger to switch to remote debugging:



Now use Debugger>Process options to set the Hostname and Port fields to localhost and 23946.

(Note that the port number was printed by mac_server64 after launching it):

	🐁 Debug application setup: macosx
<u>Application</u>	/Users/troy/helloworld
<u>I</u> nput file	/Users/troy/helloworld
<u>D</u> irectory	/Users/troy
<u>P</u> arameters	
<u>H</u> ostname	localhost Port 23946

Also be sure to check the option **Save network settings as default** so IDA will remember this configuration.

Now go to _main in the helloworld disassembly, press F2 to set a breakpoint, then F9 to launch the process. Upon launching the debugger you might receive this prompt from the OS:

Developer Tools Access is trying to take control of another process. Enter your password to allow this.
Username:
Password:
Cancel Take Control

OSX is picky about debugging permissions, and despite the fact that mac_server is properly codesigned you still must explicitly grant it permission to take control of another process. Thankfully this only needs to be done once per login session, so OSX should shut up until the next time you log out (we discuss how to disable this prompt entirely in the **Debugging Over SSH** section below).

After providing your credentials the debugger should start up without issue:

🔋 IDA View-RIP 🗆 👩 📢	3 🕱	🕻 General registers
_text:000000100000F50	R	AX 00000010000F50 👞 main
_text:000000100000F50 ; ======= S U B R O U T I N E =================================		BX 000000000000000 👗
text:00000010000F50		CX 00007FFEEFBFFD48 🖕 debug017:
text:000000100000F50 ; Attributes: bp-based frame		
		DX 00007FFEEFBFFCF0 🖕 debug017:
<pre>text:0000000100000F50 ; intcdecl main(int argc, const char **argv, const char **envp) text:000000100000F50 public main</pre>	R	SI 00007FFEEFBFFCE0 🖕 debug017:
text:000000100000F50 main proc near	R	DI 000000000000001 🖕
	R	BP 00007FFEEFBFFCD0 🖕 debug017:
text:000000100000550 var 8= dword ptr -8		SP 00007FFEEFBFFCC8 🖕 debug017:
text:000000100000F50 var 4= dword ptr -4		IP 000000100000F50 🖕 main
RAX. text:000000100000F50		B 000000000000000 L
RIP text:0000000100000F50 push rbp		the second s
text:00000000000051 mov rbp, rsp		9 00000000000000 🖕
text:000000100000F54 sub rsp, 10h	R	10 000000000000000 🖕
text:000000100000F58 mov [rbp+var_4], 0	R	11 000000000000000 👞
text:000000100000FFF lea rdi, aHelloWorld ; "hello, world!\n"	R	12 00000000000000 🖕
text:000000100000F66 mov al, 0 text:000000100000F68 call printf		13 00000000000000 🛴
- text:000000000000000 call printi		19 0000000000000

Using a Launch Agent

To simplify using the mac server, save the following XML as com.hexrays.mac_server64.plist in ~/Library/LaunchAgents/:

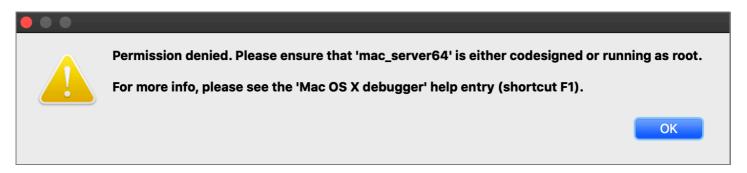
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist>
<plist version="1.0">
<dict>
   <key>Label</key>
    <string>com.hexrays.mac_server64</string>
    <key>ProgramArguments</key>
        <string>/Applications/IDA Pro 7.5/dbgsrv/mac_server64</string>
        <string>-i</string>
        <string>localhost</string>
    </array>
    <key>StandardOutPath</key>
    <string>/tmp/mac_server64.log</string>
    <key>StandardErrorPath</key>
    <string>/tmp/mac_server64.log</string>
    <key>KeepAlive</key>
    <true/>
</dict>
</plist>
```

Now mac_server64 will be launched in the background whenever you log in. You can connect to it from IDA at any time using the **Remote Mac OS X Debugger** option. Hopefully this will make local debugging on OSX almost as easy as other platforms.

Debugging System Applications

There are some applications that OSX will refuse to allow IDA to debug.

For example, load /System/Applications/Calculator.app/Contents/MacOS/Calculator in IDA and try launching the debugger. You will likely get this error message:



Despite the fact that mac_server64 is codesigned, it *still* failed to retrieve permission from the OS to debug the target app. This is because Calculator.app and all other apps in /System/Applications/ are protected by <u>System Integrity Protection</u> and they cannot be debugged until SIP is <u>disabled</u>. Note that the error message is a bit misleading because it implies that running mac_server64 as root will resolve the issue - it will not. Not even root can debug apps protected by SIP.

Disabling SIP allows IDA to debug applications like Calculator without issue:

📑 IDA View-RIP	□ @ ⊗		🕻 General registers
text:000000100009F82 text:000000100009F82; Att text:00000010009F82; voi	<pre>LculatorController_applicationDidFinishLaunching proc near ; DATA XREF:objc_const:0000</pre>	R R R R R R	AX 00000010023A9B0
RIP -text:000000100009F82 R11 -text:000000100009F82 push text:000000100009F83 push text:000000100009F86 push text:000000100009F87 push text:000000100009F87 mov text:000000100009F97 mov text:000000100009F976 mov text:000000100009F76 mov	rbp rbp, rsp rl5 rl4 rl2 rbx rsp, 28h rbx, rdi rl4, cs:selRef_window rl3, cs:selRef_window rl3, cs:selRef_window rl3, cs:selRef_window	R R R R R R	B 00000000000026 9 00000000000000 10 0000000000000 10 0000000000
text:000000100009FA7 call text:000000100009FAA mov text:000000100009FAD mov 00009F82 000000100009F82:	<pre>rl3; _objc_msgSend rl5, rax rdi, cs:classRef_NSMutableDictionary ; Class -[CalculatorController applicationD (Synchronized with RIP)</pre>	P	집 Modules ath 집 /System/Applications/Calculator.app/Contents/MacOS/Calculator
Threads] /usr/lib/dyld] /usr/lib/libobjc-trampolines.dylib
Decimal Hex State 12547 3103 Ready 20217 5000 Decide	Name Calculator	•	System/Library/Frameworks/Accelerate.framework/Versions/A/Acc System/Library/Frameworks/Accelerate.framework/Versions/A/Fra System/Library/Frameworks/Accelerate.framework/Versions/A/Framework/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/A/Framework/Versions/Framework/Versions/Framework/Framework/Framework/Framework/Framework/Framework/Framework/Framework/F
🔄 20747 510B Ready	FFFFFFFFFFFFFF		/System/Library/Frameworks/Accelerate.framework/Versions/A/Framework/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Versions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/A/Frameworks/Nersions/Ne

The effects of SIP are also apparent when attaching to an existing process. Try using menu **Debugger>Attach to process**, with SIP enabled there will likely only be a handful of apps that IDA can debug:

	Choose process to attach to
ID	Name
2371	[64] Wireshark
1136	[64] Vim
1135	[64] MacVim
888	[64] ida64
Line 1 of 4	
Help	Search Cancel OK

Disabling SIP makes all system apps available for attach:

	Ch	oose process to attach to
ID	Name	
1269	[64]	cloudphotod
1268	[64]	mdworker
1248	[64]	ScopedBookmarkAg
1244	[64]	photolibraryd
1237	[64]	followupd
1235	[64]	photoanalysisd
1136	[64]	keyboardservices
1135	[64]	AppleSpell
1134	[64]	ida64
1133	[64]	Spotlight
1127	[64]	appstoreagent
876	[64]	mdworker_shared
750	[64]	com.apple.hiserv
749	[64]	Calculator
623	[64]	SafariBookmarksS
620	[64]	mdworker_shared
585	[64]	spindump_agent
Line 14 of	131	
	Help	Search Cancel OK

It is unfortunate that such drastic measures are required to inspect system processes running on your own machine, but this is the reality of MacOS. We advise that you only disable System Integrity Protection when absolutely necessary, or use a virtual machine that can be compromised with impunity.

Debugging System Libraries

With IDA you can debug any system library in /usr/lib/ or any framework in /System/Library/.

This functionality is fully supported, but surprisingly it is one of the hardest problems the mac debugger must handle. To demonstrate this, let's try debugging the **_getaddrinfo** function in libsystem_info.dylib.

Consider the getaddrinfo application from <u>samples.zip</u>:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
int main(int argc, char **argv)
  if ( argc != 2 )
  {
    fprintf(stderr, "usage: %s <hostname>\n", argv[0]);
    return 1;
  struct addrinfo hints;
  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_INET;
  hints.ai_flags |= AI_CANONNAME;
  struct addrinfo *result:
  int code = getaddrinfo(argv[1], NULL, &hints, &result);
  if ( code != 0 )
  {
    fprintf(stderr, "failed: %d\n", code);
    return 2;
  struct sockaddr_in *addr_in = (struct sockaddr_in *)result->ai_addr;
  char *ipstr = inet_ntoa(addr_in->sin_addr);
  printf("IP address: %s\n", ipstr);
  return ∅;
}
```

Try testing it out with a few hostnames:

<pre>\$./getaddrinfo localhost</pre>
IP address: 127.0.0.1
<pre>\$./getaddrinfo hex-rays.com</pre>
IP address: 104.26.10.224
<pre>\$./getaddrinfo foobar</pre>
failed: 8

Now load libsystem_info.dylib in IDA and set a breakpoint at _getaddrinfo:

\$ ida64 -o/tmp/libsystem_info /usr/lib/system/libsystem_info.dylib

text:000000000008F30		
text:000000000008F30	; ====== S U B F	R O U T I N E =================================
text:000000000008F30		
text:000000000008F30	; Attributes: bp-based fr	rame
text:000000000008F30		
text:000000000008F30	; int cdecl getaddrinfo	co(const char *, const char *, const addrinfo *, addrinfo **)
text:000000000008F30	public o	getaddrinfo
text:000000000008F30		
text:000000000008F30		; ruserok+57 \downarrow p
text:000000000008F30		
text:000000000008F30	var 28 = gword g	ptr -28h
text:000000000008F30	var 20 = gword p	ptr -20h
text:000000000008F30	var 18 = gword r	ptr -18h
text:000000000008F30	$var_{10} = qword r$	ptr -10h
text:000000000008F30	var 8 = gword r	ptr -8
text:000000000008F30		-
text:000000000008F30	push 1	ada
text:000000000008F31		rbp, rsp
text:000000000008F34		rsp, 30h
text:000000000008F38		eax, eax
text:000000000008F3A		r8d, eax
		•

Choose **Remote Mac OS X Debugger** from the Debugger menu and under **Debugger>Process options** be sure to provide a hostname in the **Parameters** field. IDA will pass this argument to the executable when launching it:

	🦹 Debug application setup: macosx
<u>Application</u>	/Users/troy/getaddrinfo
Input file	/usr/lib/system/libsystem_info.dylib
<u>D</u> irectory	/Users/troy 🛛 🔛
<u>P</u> arameters	hex-rays.com
<u>H</u> ostname	localhost Port 23946

Before launching the process, use **Ctrl+S** to pull up the segment list for libsystem_info.dylib. Pay special attention to the __**eh_frame** and __**nl_symbol_ptr** segments. Note that they appear to be next to each other in memory:

Name	Start	End	R	W	x
😛 HEADER	000000000000000000000000000000000000000	0000000000011D0	R	•	Х
😝text	0000000000011D0	00000000004980A	R		Х
😝stubs	00000000004980A	000000000049CF0	R		Х
🛟stub_helper	0000000000049CF0	00000000004A52A	R		Х
🛟const	00000000004A530	00000000004A6FC	R		Х
😝cstring	00000000004A6FC	00000000004CA70	R		Х
😝oslogstring	00000000004CA70	00000000004DE06	R		Х
😝unwind_info	000000000004DE08	00000000004dfa0	R	•	Х
😝eh_frame	000000000004DFA0	00000000004DFF8	R		Х
😝nl_symbol_ptr	000000000004E000	00000000004E008	R	W	
😝got	00000000004E008	00000000004E090	R	W	•
😝la_symbol_ptr	00000000004E090	00000000004E718	R	W	
🛟const	00000000004E720	00000000004F3B0	R	W	
😝data	000000000004F3B0	00000000004FB6C	R	W	
😝common	000000000004FB70	00000000004FF80	R	W	
😝bss	00000000004FF80	000000000050578	R	W	•

This will be important later.

Finally, use **F9** to launch the debugger and wait for our breakpoint at **_getaddrinfo** to be hit. We can now start stepping through the logic:

🖪 IDA View-RIP			0 0	3 1	🕻 General registers
text:00007FFF6D2 text:00007FFF6D2 text:00007FFF6D2 text:00007FFF6D2 text:00007FFF6D2 text:00007FFF6D2 text:00007FFF6D2 text:00007FFF6D2	11F30 11F30 var_28= qword ptr -28h 11F30 var_20= qword ptr -20h 11F30 var_18= qword ptr -18h 11F30 var_10= qword ptr -10h 11F30 var_8= qword ptr -8 11F30		E XREF: serok+5	R R R R R	AX 00000000000000 BX 0000000000000 CX 0000000000000 DX 00007FFEEFBFFC60 GI 00000000000000 DI 00007FFEEFBFFC10 BP 00007FFEEFBFFC10 C debug307:00007FFEEFBFFC10 C debug307:00007FFEEFBFFFEFFFEFFFFFEFFFFFFFFFFFFFFF
text:00007FFF6D2 text:00007FFF6D2	11F31 mov rbp, rsp 11F34 sub rsp, 30h 11F38 xor eax, eax 11F30 mov rbd, eax 11F31 mov rbd, eax 11F30 mov [rbd+var_8], rd 11F41 mov [rbd+var_10], r 11F45 mov [rbd+var_10], r 11F45 mov [rbd+var_20], r 11F45 mov [rbd+var_20], r 11F55 mov rdi, [rbd+var_1 11F59 mov rdx, [rbd+var_2] 11F59 mov rcx, [rbd+var_2] 11F50 mov [rbd+var_2], r	rsi rdx rcx 3] 10] 18] 20]		R R R R R R R R	SP 00007FFEEFBFFBE0 debug307:00007FFEEFBFFBE0 IP 00007FFEED211F68 getaddrinfo+38 00007FFEEFBFFC58 debug307:00007FFEEFBFFC58 getaddrinfo u2 000000000000 getaddrinfo u2 0000000000000 u 10 00007FFEEFBFFD90 setaddrinfo u 00000000000000 setaddrinfo u 00000000000000 setaddrinfo u 00000000000000 setaddrinfo u 0000000000000 setaddrinfo u 0000000000000 setaddrinfo u u
text:00007FFF6D211F64 mov r8, [rbp+var_28] Text:00007FFF6D211F68 call _getaddrinfo_internal				5	B Modules
text:00007FFF6D2 00008D98 00007FFF	11F6D add rsp, 30h 6D211F68: getaddrinfo+38	(Synchronized with RI	P)		Path
					追 /Users/troy/getaddrinfo 過 /usr/lib/dyld
🗾 Call Stack			0 0		🖞 /usr/lib/libSystem.B.dylib
Address	Module Fun	nction			🔟 /usr/lib/libc++.1.dylib
500007FFF6D211F68 5	libsystem_info.dylib _g	getaddrinfo+0x38			J] /usr/lib/libc++abi.dylib
≣ 0000000100000E99	getaddrinfo _m	nain+99			J] /usr/lib/libobjc.A.dylib
00007FFF6D116CC9	libdyld.dylib _s	tart+1			}] /usr/lib/system/libcache.dylib }] /usr/lib/system/libcommonCrypto.dylib

Everything appears to be working normally, but use **Ctrl+S** to pull up the segment information again. We can still see **___eh_frame**, but it looks like **___nl_symbol_ptr** has gone missing:

# HEADER	00007FFF6D209000	00007FFF6D20A1D0	R		Х
<pre>text</pre>	00007FFF6D20A1D0	00007FFF6D25280A	R		Х
🖶stubs	00007FFF6D25280A	00007FFF6D252CF0	R		х
<pre>stub_helper</pre>	00007FFF6D252CF0	00007fff6d25352A	R		х
😝 debug384	00007fff6d25352A	00007FFF6D253530	R		х
-const	00007FFF6D253530	00007FFF6D2536FC	R		х
<pre>cstring</pre>	00007FFF6D2536FC	00007FFF6D255A70	R		х
<pre>oslogstring</pre>	00007FFF6D255A70	00007FFF6D256E06	R		х
😝 debug385	00007FFF6D256E06	00007FFF6D256E08	R		х
<pre>unwind_info</pre>	00007FFF6D256E08	00007FFF6D256FA0	R		х
<pre>eh_frame</pre>	00007FFF6D256FA0	00007FFF6D256FF8	R		х
🖶 debug386	00007FFF6D256FF8	00007FFF6D257000	R		х
🚌 libsystem_kernel.dylib:HEADER	00007FFF6D257000	00007FFF6D257B00	R	•	
<pre>libsystem_kernel.dylib:text</pre>	00007FFF6D257B00	00007FFF6D279A14	R		Х

It is actually still present, but we find it at a much higher address:

😝 libsystem_featureflags.dylib:	00007FFF93C39710	00007FFF93C39729	R	•	•
🛟 debug548	00007FFF93C39729	00007FFF93C39730	R	W	.
<pre>nl_symbol_ptr</pre>	00007FFF93C39730	00007FFF93C39738	R	W	•
🖶got	00007FFF93C39738	00007FFF93C397C0	R	W	•
😝la_symbol_ptr	00007FFF93C397C0	00007FFF93C39E48	R	W	.
😝 debug549	00007FFF93C39E48	00007FFF93C39E50	R	W	.
😝const	00007FFF93C39E50	00007fff93C3AAE0	R	W	.
🜐data	00007FFF93C3AAE0	00007FFF93C3B29C	R	W	.
😝 debug550	00007FFF93C3B29C	00007FFF93C3B2A0	R	W	.
🖶common	00007FFF93C3B2A0	00007FFF93C3B6B0	R	W	.
🛟bss	00007FFF93C3B6B0	00007FFF93C3BCA8	R	W	.
😝 debug551	00007FFF93C3BCA8	00007FFF93C3BCB0	R	W	.
UNDEF	00007FFF93C3BCB0	00007FFF93C3C3C0	?	?	?
😝 libsystem_kernel.dylib:const	00007FFF93C3C3C0	00007FFF93C3E0B0	R		.

Recall that we opened the file directly from the filesystem (/usr/lib/system/libsystem_info.dylib). However this is actually *not* the file that OSX loaded into memory. The libsystem_info image in process memory was mapped in from the <u>dyld_shared_cache</u>, and the library's segment mappings were modified before it was inserted into the cache.

IDA was able to detect his situation and adjust the database so that it matches the layout in process memory. This functionality is fully supported, but it is not trivial. Essentially the debugger must split your database in half, rebase all code segments to one address, then rebase all data segments to a completely different address.

It is worth noting there is another approach that achieves the same result, but without so much complexity.

Debugging Modules in dyld_shared_cache

As an alternative for the above example, note that you can load any module directly from a dyld_shared_cache file and debug it. For example, open the shared cache in IDA:

\$ ida64 -o/tmp/libsystem_info2 /var/db/dyld/dyld_shared_cache_x86_64h

When prompted, select the "single module" option:

Load file /private/var/db/dyld/dyld_shared_cache_x86_64h as			
Apple DYLD cache for x86_64h (complete image) [macho64.dylib]			
Apple DYLD cache for x86_64h (single module) [macho64.dylib]			
Apple DYLD cache for x86_64h (single module plus dependencies) [macho64.dylib] Binary file			

Then choose the libsystem_info module:

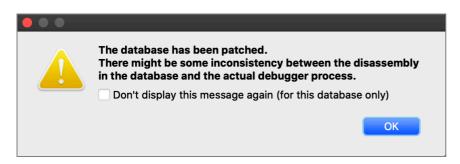
Choose a module to load	
File name	Address
/usr/lib/system/libsystem_coreservices.dylib	0x7FFF671E3000
/usr/lib/system/libsystem_darwin.dylib	0x7FFF671E7000
/usr/lib/system/libsystem_dnssd.dylib	0x7FFF671F0000
/usr/lib/system/libsystem_featureflags.dylib	0x7FFF671F8000
/usr/lib/system/libsystem_info.dylib	0x7FFF671FA000
/usr/lib/system/libsystem_kernel.dylib	0x7FFF67248000
/usr/lib/system/libsystem_m.dylib	0x7FFF67275000
/usr/lib/system/libsystem_malloc.dylib	0x7FFF672BD000
/usr/lib/system/libsystem_networkextension.dylib	0x7FFF672E5000
/usr/lib/system/libsystem_notify.dylib	0x7FFF672F3000
/usr/lib/system/libsystem_platform.dylib	0x7FFF672FD000
/usr/lib/system/libsystem_pthread.dylib	0x7FFF67306000
Line 1634 of 1810	
Help Search Cancel	ОК

Select the **Remote Mac OS X Debugger** and for **Debugger>Process options** use the exact same options as before:

	🧕 Debug application setup: macosx
<u>Application</u>	/Users/troy/getaddrinfo
<u>I</u> nput file	/usr/lib/system/libsystem_info.dylib
<u>D</u> irectory	/Users/troy
<u>P</u> arameters	hex-rays.com
<u>H</u> ostname	localhost v Port 23946 v

Now set a breakpoint at _getaddrinfo and launch the process with F9.

After launching the debugger you might see this warning:



This is normal. Modules from the dyld_shared_cache will contain tagged pointers, and IDA patched the pointers when loading the file so that analysis would not be hindered by the tags. IDA is warning us that the patches might cause a discrepancy between the database and the process, but in this case we know it's ok. Check **Don't display this message again** and don't worry about it.

Launching the process should work just like before, and we can start stepping through the function in the shared cache:

IDA View-RIP		Ø	8	👿 General registers
libsystem_info:text:00007FFF6D211F30 libsystem_info:_text:00007FFF6D211F30 libsystem_info:_text:00007FFF6D211F30 libsystem_info:_text:00007FFF6D211F30 libsystem_info:_text:00007FFF6D211F30 libsystem_info:_text:00007FFF6D211F30 libsystem_info:_text:00007FFF6D211F30 libsystem_info:_text:00007FFF6D211F30	_getaddrinfo proc near var_28= qword ptr -28h var_20= qword ptr -20h var_18= qword ptr -18h var_10= qword ptr -10h var_8= qword ptr -8			RAX 00000000000000 RBX 0000000000000 RCX 000000000000 RDX 00007FFEFBFFC60 RSI 00000000000000 RDI 00007FFEFBFFC10 RDP 00007FFEFBFFC10 debug309:0007FFEFBFFC10
Ibsystem info: text:00007FFF6211F30 libsystem info: text:00007FFF6211F34 libsystem info: text:00007FFF6211F41 libsystem info: text:00007FFF6211F49 libsystem info: text:00007FFF6211F49 libsystem info: text:00007FFF6211F49 libsystem info: text:00007FFF6211F59 libsystem info: text:00007FFF6211F59	mov rbp, rsp sub rsp, 30h ixor eax, eax mov r8d, eax mov [rbp+var_10], rsi mov [rbp+var_10], rsi mov [rbp+var_18], rdx mov [rbp+var_18], rdx mov [rbp+var_20], rcx mov rdip+var_8] mov rsi, [rbp+var_18] mov rcx, [rbp+var_18] mov rcx, [rbp+var_20] mov rcx, [rbp+var_20]		0	RSP 00007FFEEFBFFBE0 debug309:00007FFEEFBFFBE0 RIP 00007FFEEFBFF6211F68 _getaddrinfo+38 R8 00007FFEEFBFFC58 debug309:00007FFEEFBFFC58 R9 00000000000000 _getaddrinfo R10 0007FFEEFBFFD90 debug309:00007FFEEFBFFD90 R11 00007FFEEFBFFD91 _getaddrinfo R12 000000000000000000000000000000000000
RIP libsystem info: text:00007FFF6D211F64 libsystem info: text:00007FFF6D211F68	mov r8, [rbp+var_28] callgetaddrinfo_internal			Modules
libsystem_info:text:00007FFF6D211F6D 00008F68 00007FFF6D211F68: getaddr	••	P1	-	Path
		- ,		/Users/troy/getaddrinfo /usr/lib/dyld
Jall Stack		٥	ß	B /usr/lib/libSystem.B.dylib
		•	ω	<pre>//usr/lib/libc++.1.dylib</pre>
Address Module	Function			/usr/lib/libc++abi.dylib
00007FFF6D211F68 libsystem_info.dy				/usr/lib/libobjc.A.dylib
5 000000100000E99 getaddrinfo	_main+99			/usr/lib/system/libcache.dylib
Jibdyld.dylib	_start+1			<pre>/usr/lib/system/libcommonCrypto.dylib</pre>

This time there was no special logic to map the database into process memory. Since we loaded the module directly from the cache, segment mappings already match what's expected in the process. Thus only one rebasing operation was required (as apposed to the segment scattering discussed in the previous example).

Both techniques are perfectly viable and IDA goes out of its way to fully support both of them. In the end having multiple solutions to a complex problem is a good thing.

Debugging Objective-C Applications

When debugging OSX applications it is easy to get lost in some obscure Objective-C framework. IDA's mac debugger provides tools to make debugging Objective-C code a bit less painful.

Consider the **bluetooth** application from <u>samples.zip</u>:

```
#import <IOBluetooth/IOBluetooth.h>
int main(void)
{
    NSArray *devices = [IOBluetoothDevice pairedDevices];
    int count = [devices count];
    for ( int i = 0; i < count; i++ )
    {
        IOBluetoothDevice *device = [devices objectAtIndex:i];
        NSLog(@"%@:\n", [device name]);
        NSLog(@" paired: %d\n", [device isPaired]);
        NSLog(@" connected: %d\n", [device isConnected]);
    }
    return 0;
}</pre>
```

The app will print all devices that have been paired with your host via Bluetooth. Try running it:

<pre>\$./bluetooth</pre>			
	bluetooth[17025:15645888]	č	
2020-05-22 16:27:14.443 ł	bluetooth[17025:15645888]	paired: 1	
2020-05-22 16:27:14.443 k	bluetooth[17025:15645888]	connected: 1	
2020-05-22 16:27:14.443 k	bluetooth[17025:15645888]	Apple Magic Mouse	:
2020-05-22 16:27:14.443 k	bluetooth[17025:15645888]	paired: 1	
2020-05-22 16:27:14.443 k	bluetooth[17025:15645888]	connected: 1	
2020-05-22 16:27:14.443 k	bluetooth[17025:15645888]	iPhone SE:	
2020-05-22 16:27:14.443 k	bluetooth[17025:15645888]	paired: 0	
2020-05-22 16:27:14.443 k	bluetooth[17025:15645888]	connected: 0	

Let's try debugging this app. First consider the call to method +[IOBluetoothDevice pairedDevices]:

	text:00000010000E30 push	rbp
	text:0000000100000E31 mov	rbp, rsp
	text:0000000100000E34 sub	rsp, 20h
	text:0000000100000E38 mov	[rbp+var_4], 0
	text:0000000100000E3F mov	rax, cs:classRef_IOBluetoothDevice
	text:0000000100000E46 mov	<pre>rsi, cs:selRef_pairedDevices ; SEL</pre>
	text:0000000100000E4D mov	rdi, rax ; id
RIP	text:0000000100000E50 call	cs:_objc_msgSend_ptr
	text:0000000100000E56 mov	[rbp+var_10], rax

If we execute a regular instruction step with **F7**, IDA will step into the **_objc_msgSend** function in libobjc.A.dylib, which is probably not what we want here. Instead use shortcut **Shift+O**. IDA will automatically detect the address of the Objective-C method that is being invoked and break at it:

	IOBluetooth: text:00007FFF358F7D60 ;
	IOBluetooth: text:00007FFF358F7D60
	IOBluetooth: text:00007FFF358F7D60 ; +[IOBluetoothDevice pairedDevices]
RIP	IOBluetooth: text:00007FFF358F7D60 IOBluetoothDevice pairedDevices:
R11	IOBluetooth: text:00007FFF358F7D60 push rbp
	IOBluetooth:text:00007FFF358F7D61 mov rbp, rsp
	IOBluetooth:text:00007FFF358F7D64 sub rsp, 150h
	IOBluetooth:text:00007FFF358F7D6B mov rax, cs:off_7FFF8AF8EF68
	IOBluetooth:text:00007FFF358F7D72 mov rax, [rax]
	IOBluetooth:text:00007FFF358F7D75 mov [rbp-8], rax

This module appears to be Objective-C heavy, so it might be a good idea to extract Objective-C type info from the module using right click -> **Load debug symbols** in the Modules window:

Modules				
Path				
System/Library/Frameworks/Foundation.framework/Versions/C	/Foundation			
/System/Library/Frameworks/GSS.framework/Versions/A/GSS				
/System/Library/Frameworks/IOBluetooth.framework/Versions/A	A/IOBluetooth			
System/Library/Frameworks/IOK Copy #C	-			
System/Library/Frameworks/IOS Copy all 수이	Surface			
System/Library/Frameworks/Ima	reIO			
System/Library/Frameworks/Ima Value filter ^F	purces/libGIF.			
Modify filters へひF	purces/libJP2.			
/System/Library/Frameworks/Ima	urces/libJPEG			
/System/Library/Frameworks/Ima	ources/libPng.			
/System/Library/Frameworks/Ima	purces/libRadi			
System/Library/Frameworks/Ima Break on access	purces/libTIFF			
System/Library/Frameworks/Ker	beros			

This operation will extract any Objective-C types encoded in the module, which should give us some nice prototypes for the methods we're stepping in:

IOBluetooth: text:00007FFF358F7D60 IOBluetooth: text:00007FFF358F7D60	; ====================================
IOBluetooth: text:00007FFF358F7D60	
IOBluetooth: text:00007FFF358F7D60 IOBluetooth: text:00007FFF358F7D60	; Attributes: bp-based frame
IOBluetooth: text:00007FFF358F7D60	
IOBluetooth:text:00007FFF358F7D60	IOBluetoothDevice_pairedDevices_ proc near
IOBluetooth:text:00007FFF358F7D60 IOBluetooth:text:00007FFF358F7D60	; DATA XREF: IOBluetooth:objc_const:

Let's continue to another method call - but this time the code invokes a stub for **_objc_msgSend** that IDA has not analyzed yet, so its name has not been properly resolved:

```
IOBluetooth:text:00007FFF358F7D91mov[rbp+var_A0], raxIOBluetooth:text:00007FFF358F7D98movrax, cs:classRef_IOBluetoothPreferencesIOBluetooth:text:00007FFF358F7D9Fmovrsi, cs:selRef_sharedPreferencesIOBluetooth:text:00007FFF358F7DA6movrdi, raxIOBluetooth:text:00007FFF358F7DA9callcs:off_7FFF8AF8F058IOBluetooth:text:00007FFF358F7DAFmovrsi, cs:selRef_pairedDevices_0IOBluetooth:text:00007FFF358F7DB6movrdi, rax
```

In this case Shift+O should still work:

	IOBluetooth: text:00007FFF358EDDE0	
	IOBluetooth: text:00007FFF358EDDE0	; ====== S U B R O U T I N E =================================
	IOBluetooth: text:00007FFF358EDDE0	
	IOBluetooth: text:00007FFF358EDDE0	; Attributes: bp-based frame
	IOBluetooth: text:00007FFF358EDDE0	
	IOBluetooth: text:00007FFF358EDDE0	; idcdecl +[IOBluetoothPreferences sharedPreferences](id, SEL)
	IOBluetooth: text:00007FFF358EDDE0	
	IOBluetooth: text:00007FFF358EDDE0	; DATA XREF: IOBluetooth: _ objc_const:
	IOBluetooth: text:00007FFF358EDDE0	
	IOBluetooth:text:00007FFF358EDDE0	
	IOBluetooth: text:00007FFF358EDDE0	
	IOBluetooth:text:00007FFF358EDDE0	
	IOBluetooth: text:00007FFF358EDDE0	
	IOBluetooth:text:00007FFF358EDDE0	
RIP	IOBluetooth:text:00007FFF358EDDE0	
R11	IOBluetooth:text:00007FFF358EDDE0	
	IOBluetooth:text:00007FFF358EDDE1	mov rbp, rsp

Shift+O is purposefully flexible so that it can be invoked at any point before a direct or indirect call to _**objc_msgSend**. It will simply intercept execution at the function in libobjc.A.dylib and use the arguments to calculate the target method address.

However, you must be careful. If you use this action in a process that does *not* call **_objc_msgSend**, you will lose control of the process. It is best to only use it when you're certain the code is compiled from Objective-C and an **_objc_msgSend** call is imminent.

Decompiling Objective-C at Runtime

The Objective-C runtime analysis performed by Load debug symbols will also improve decompilation.

Consider the method -[IOBluetoothDevice isConnected]:

	IOBluetooth:text:00007FFF35901BB0	
	IOBluetooth: text:00007FFF35901BB0	; ====================================
	IOBluetooth: text:00007FFF35901BB0	
	IOBluetooth: text:00007FFF35901BB0	; Attributes: bp-based frame
	IOBluetooth: text:00007FFF35901BB0	
	IOBluetooth: text:00007FFF35901BB0	; BOOL cdecl -[IOBluetoothDevice isConnected](IOBluetoothDevice *self, SEL)
	IOBluetooth: text:00007FFF35901BB0	IOBluetoothDevice isConnected proc near
	IOBluetooth: text:00007FFF35901BB0	; DATA XREF: IOBluetooth: objc const:
	IOBluetooth: text:00007FFF35901BB0	
	IOBluetooth: text:00007FFF35901BB0	var 52= byte ptr -52h
	IOBluetooth: text:00007FFF35901BB0	var 51= byte ptr -51h
	IOBluetooth: text:00007FFF35901BB0	var 50= gword ptr -50h
	IOBluetooth: text:00007FFF35901BB0	var 48= gword ptr -48h
	IOBluetooth: text:00007FFF35901BB0	var 39= byte ptr -39h
	IOBluetooth: text:00007FFF35901BB0	var 38= gword ptr -38h
	IOBluetooth: text:00007FFF35901BB0	var 2C= dword ptr -2Ch
	IOBluetooth: text:00007FFF35901BB0	var 28= gword ptr -28h
	IOBluetooth: text:00007FFF35901BB0	var 20= gword ptr -20h
	IOBluetooth: text:00007FFF35901BB0	var 18= qword ptr -18h
	IOBluetooth: text:00007FFF35901BB0	var 10= byte ptr -10h
	IOBluetooth: text:00007FFF35901BB0	var 8= gword ptr -8
RIP	IOBluetooth: text:00007FFF35901BB0	
R11	IOBluetooth: text:00007FFF35901BB0	push rbp
	IOBluetooth: text:00007FFF35901BB1	mov rbp, rsp

Before we start stepping through this method we might want to peek at the pseudocode to get a sense of how it works. Note that the Objective-C analysis created local types for the **IOBluetoothDevice** class, as well as many other classes:

	🗴 📴 IDA View-RIP 🛛 🗞 Local Types
Ordinal	Name Size Sync Description
43	IOBluetoothDeviceExpansion 00000088 struct {NSObject super;IOBlu
44	SDPQueryCallbackDispatcher 00000018 struct {NSObject super;id mT
45	IOBluetoothObject 🕒 🕒 🔵 🔵 👷 🙀 Please edit the type declaration
46	IOBluetoothDevice
47	IOBluetoothHCIUnifiedInqui Offset Size struct IOBluetoothDevice
48	BTClient (IOBluetoothObject super;
49	BluetoothDeviceManager 0018 0008 id mServerDevice;
50	IOBluetoothL2CAPChannelExp 0024 0006 BluetoothDeviceAddress mAddress;
51	IOBluetoothL2CAPChannelDe 0030 0008 NSString *mName;
52	IOBluetoothL2CAPChannel 0038 0008 NSDate *mLastNameUpdate; 0040 0004 unsigned int mClassOfDevice;
53	\$431E0FFFF5EECFE295EE5EFA{ 0044 0001 unsignedint8 mPageScanRepetitionMode; 10ElustesthuserMeasureEls
54	IOBIuetoothUserMessageBloc 0046 0001 unsignedint8 mPageScanMode;
55	\$600350704F50506D3FAE14398 0050 0008 NSDate *mLastInguiryUpdate;
56	IOBluetoothRFCOMMChannel 0058 0002 unsignedint16 mConnectionHandle;
57	IOBluetoothRFCOMMConnectic 005A 0001 unsigned int8 mLinkType; 005B 0001 unsigned int8 mEncryptionMode;
58	IOBluetoothSDPServiceReco: 0060 0008 NSArray *mServiceArray;
59	IOBIuetoothSerialPort 0070 0008 IOBluetoothRFCOMMConnection *mRFCOMMConnection;
60	IOBluetoothSerialPortManac 0078 0008 id _mReserved;
61	Notificationinio
62	IOBluetoothNotification

This type info results in some sensible pseudocode:

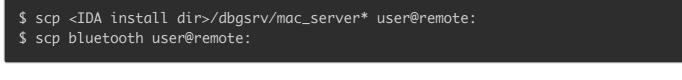
```
📑 Pseudocode-A
  1 BOOL ____cdecl -[IOBluetoothDevice isConnected](IOBluetoothDevice *self, SEL a2)
  2 {
      // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
  3
  4
  5
      if ( self->super.mIOService )
   6
      {
        state = 0xAAAAAAAAAAAAAAAAA;
  7
        v3 = j_IOServiceGetState(self->super.mIOService, &state);
  8
        if (\sqrt{v3} | | (state \& 1) != 0)
  9
 10
        {
          if...
 11
                                                       // logging
          objc_msgSend(self, "setIOService:", OLL);
 12
 13
        }
      }
if...
  14
 15
                                                      // __stack_chk_guard
      return self->super.mIOService != 0;
 16
• 17 }
```

We knew nothing about this method going in - but it's immediately clear that device connectivity is determined by the state of an **io_service_t** handle in the **IOBIuetoothObject** superclass, and we're well on our way.

Debugging Over SSH

In this section we will discuss how to remotely debug an app on a mac machine using only an SSH connection. Naturally, this task introduces some unique complications.

To start, copy the mac_server binaries and the **bluetooth** app from <u>samples.zip</u> to the target machine:



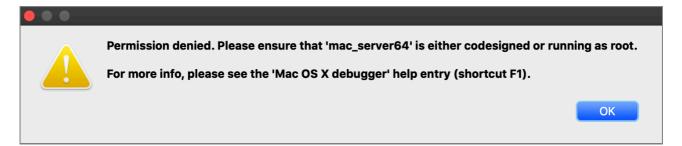
Now ssh to the target machine and launch the mac_server:



Now open the **bluetooth** binary on the machine with your IDA installation, select **Remote Mac OS X Debugger** from the debugger menu, and for **Debugger>Process options** set the debugging parameters. Be sure to replace **<remote user>** and **<remote ip>** with the username and ip address of the target machine:

	🧕 Debug application setup: macosx
<u>Application</u>	/Users/ <remote user="">/bluetooth</remote>
Input file	/Users/ <remote user="">/bluetooth</remote>
Directory	/Users/ <remote user=""></remote>
<u>P</u> arameters	
<u>H</u> ostname	<remote ip=""> Port 23946 V</remote>

Try launching the debugger with F9. You might get the following error message:



This happened because debugging requires manual authentication from the user for every login session (via the **Take Control** prompt discussed under **Using the Mac Debug Server**, above). But since we're logged into the mac via SSH, the OS has no way of prompting you with the authentication window and thus debugging permissions are refused.

Note that mac_server64 might have printed this workaround:

WARNING: The debugger could not acquire the necessary permissions from the OS to debug mac applications. You will likely have to specify the proper credentials at process start. To avoid this, you can set the MAC_DEBMOD_USER and MAC_DEBMOD_PASS environment variables.

But this is an extreme measure. As an absolute last resort you can launch the mac_server with your credentials in the environment variables, which should take care of authentication without requiring any interaction with the OS. However there is a more secure workaround.

In your SSH session, terminate the mac_server process and run the following command:

\$ security authorizationdb read system.privilege.taskport > taskport.plist

Edit taskport.plist and change the **authenticate-user** option to **false**:

```
<key>authenticate-user</key>
<false/>
```

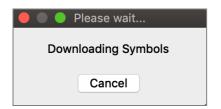
Then apply the changes:

```
$ sudo security authorizationdb write system.privilege.taskport < taskport.plist</pre>
```

This will completely disable the debugging authentication prompt (even across reboots), which should allow you to use the debug server over SSH without OSX bothering you about permissions.

Dealing With Slow Connections

When debugging over SSH you might experience some slowdowns. For example you might see this dialog appear for several seconds when starting the debugger:

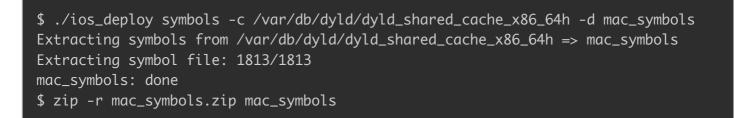


During this operation IDA is fetching function names from the symbol tables for all dylibs that have been loaded in the target process. It is a critical task (after all we want our stack traces to look nice), but it is made complicated by the sheer volume of dylibs loaded in a typical OSX process due to the dyld_shared_cache. This results in several megabytes of raw symbol names that mac_server must transmit over the wire every time the debugger is launched.

We can fix this by using the same trick that IDA's <u>Remote iOS Debugger</u> uses to speed up debugging - by extracting symbol files from the dyld cache and parsing them locally. Start by downloading the <u>ios_deploy</u> utility from our downloads page, and copy it to the remote mac:

```
$ scp ios_deploy user@remote:
```

Then SSH to the remote mac and run it:



Copy mac_symbols.zip from the remote machine to your host machine and unzip it. Then open **Debugger>Debugger options>Set specific options** and set the **Symbol path** field:

😣 🔵 🕂	🍡 Mac OSX Debugger Options	
<u>S</u> ymbol path	/Users/troy/mac_symbols	~
	Cancel OK	

Now try launching the debugger again, it should start up much faster.

Also keep the following in mind:

- Use /var/db/dyld/dyld_shared_cache_i386 if debugging 32-bit apps
- You must perform this operation after every OSX update. Updating the OS will likely update the dyld_shared_cache, which invalidates the extracted symbol files.
- The ios_deploy utility simply invokes dyld_shared_cache_extract_dylibs_progress from the dsc_extractor.bundle library in Xcode. If you don't want to use ios_deploy there are likely other thirdparty tools that do something similar.

Support

If you have any questions about this writeup or encounter any issues with the debugger itself in your environment, don't hesitate to contact us at support@hex-rays.com.

Our Mac support team has years of experience keeping the debugger functional through rapid changes in the Apple developer ecosystem. It is likely that we can resolve your issue quickly.