

Emulated Breakpoint Debugger and Data Mining using Detours

Jason Raber and Eric Laspe
Riverside Research Institute
jraber@rri-usa.org

Abstract

The ability to do dynamic analysis is a powerful tool in the arsenal of a reverse engineer. Sometimes a piece of code such as malware can employ anti-debugging or packing measures to make dynamic analysis difficult. We have instrumented Microsoft Detours into a stealthy debugger to emulate a breakpoint rather than using “INT 3” or DR0-DR7 hardware registers.

Understanding code and data flow at a functional level can now be achieved by using an IDA Pro plug-in and the data mining feature that has been extended to Detours. ‘IF’ is the tool that incorporates the emulated breakpoints and data mining capabilities.

1. Introduction

Microsoft has released a library called Detours that includes functionality to intercept Win32 system calls. Detours used with the Linux ‘strace’ utility gives the reverse engineer the ability to understand a running executable by recovering the system calls executed. Leveraging the functionality of the Detours library to hook system calls, we created a tool (IF) that will allow the reverse engineer to create a data mining utility and a debugger that will break on an address without using ‘INT 3’ or hardware breakpoints.

2. Internal Function Data Mining

Static analysis using IDA Pro can be a tedious process of running code through a debugger and annotating the disassembly in IDA. The IF tool helps reduce the time it takes to annotate the IDA database. Given the address of one or more functions, IF will hook the function(s) and print a summary detailing the program’s execution, including:

- Data flow
- Code flow
- Order of execution

- Parameters and return values of each function

We have created an IDA Pro plug-in that will allow the engineer to automate the dumping of function names and addresses to be fed into the IF utility.

2.1 IDA Pro Plug-in

The IDA Pro plug-in allows the user to automatically generate a detailed list, in the form of a .cpp file, of all the functions in a disassembly characterized by IDA Pro. The plug-in uses IDA’s database structures to extract and parse names, addresses, parameter types, declaration types, and return types from functions called in a given binary. This information is used to create the file `_attach_detach_internal.cpp`, a compilation of hook instructions used by Detours to intercept calls to those functions. Fig. 1 shows a set of function hook calls generated by the plug-in.

```
// Plug-in generates this declaration and
// set of calls for every function
static void (__cdecl * Real_func)(int,
int) = NULL;
...
Real_func = (void (__cdecl *) (int,
int)) 0x401750;
ATTACH(&(PVOID&)Real_func, Mine_func);
...
DETACH(&(PVOID&)Real_func, Mine_func);
```

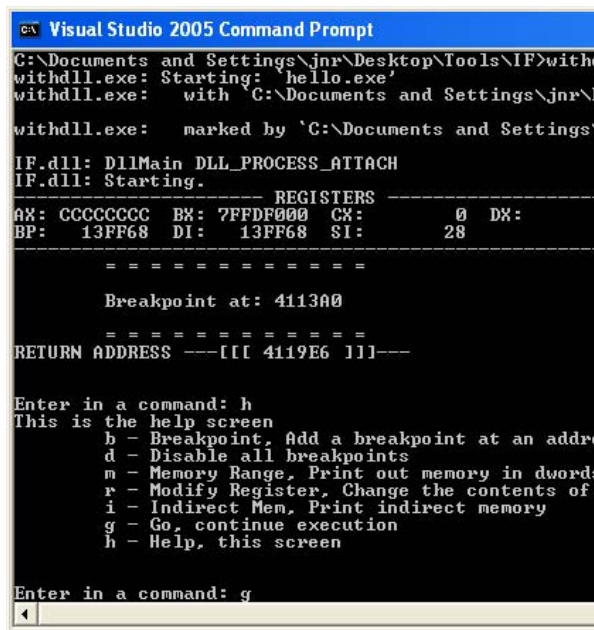
Figure 1: Function hook

2.2 IF Interface

Once the `_attach_detach_internal.cpp` file has been generated, all one needs to do is recompile the IF utility with this generated file. This will set up function pointers to be used by IF that will allow for the dumping of the hooked functions and their parameters.

3. IF Debugger

Using a custom debugger with Detours involves many of the same techniques that are employed to hook a function for data mining. After static analysis, the engineer may witness an area he would like to debug. The IF debugger will add an emulated breakpoint to the address the user specifies at runtime. This is done by hooking the targeted address and injecting an unconditional jmp instruction in place of the instruction the engineer wants to analyze. The destination address of the injected jmp will be code that he controls—such as code for printing out register contents—and allow the user to debug the subsequent code. Fig. 2 shows the output after hitting a designated breakpoint.



```
Visual Studio 2005 Command Prompt
C:\Documents and Settings\jnr\Desktop\Tools\IF\withdll.exe: Starting: 'hello.exe'
withdll.exe: with 'C:\Documents and Settings\jnr\Desktop\Tools\IF\withdll.exe'
withdll.exe: marked by 'C:\Documents and Settings\jnr\Desktop\Tools\IF\withdll.exe'
IF.dll: DllMain DLL_PROCESS_ATTACH
IF.dll: Starting.
-----
REGISTERS
-----
AX: CCCCCC BX: 7FFDF000 CX: 0 DX: 28
BP: 13FF68 DI: 13FF68 SI: 28
-----
Breakpoint at: 4113A0
-----
RETURN ADDRESS ---[[[ 4119E6 ]]]---
Enter in a command: h
This is the help screen
b - Breakpoint, Add a breakpoint at an address
d - Disable all breakpoints
m - Memory Range, Print out memory in dword
r - Modify Register, Change the contents of
i - Indirect Mem, Print indirect memory
g - Go, continue execution
h - Help, this screen
Enter in a command: g
```

Figure 2: Debugger output

As you can see, the user can view the state of the registers upon hitting a designated breakpoint. The IF debugger allows the engineer to now change registers, add another runtime breakpoint, change memory, and more.

3.1 All Done at Runtime

Using Detours to inject jumps to reroute code allows the IF debugger to have command of the running executable at that address. This also allows the reverse engineer to halt the debugger even with code that is packed. Since all IF debugger injections are dynamic, the code remains unmodified after its run is completed. Once Detours has hooked an address the

engineer would like to analyze, the IF debugger will transfer control to the user. If the reverser wants to run to the next breakpoint, the IF debugger will redirect the code back through an indirect jump to the original address of the running executable via an asm statement:

```
_asm { jmp [Real_address] }
```

3.2 Effectiveness vs. COTS Debuggers

Traditional Ring-3 debuggers (OllyDbg and IDA Pro) need to register with the OS to begin debugging a user application. Ring-0 kernel-level debuggers circumvent registration-based debugger checks (e.g. IsDebuggerPresent), but still use INT 3's and hardware debug registers. They also require installing device drivers (SoftICE), or require the system to boot in debug-mode (WinDbg). WinDbg also needs a second PC to control the system being debugged.

Using the IF Debugger, registration with the OS is not required. Checks for IsDebuggerPresent, CheckRemoteDebuggerPresent, GetTickCount, INT 3's, and debug register use are circumvented. IF Debugger does not require any drivers, kernel modifications, or an additional PC to control the system being debugged.

4. Conclusion

Software that prevents a reverse engineer from dynamic analysis through means of anti-debugging or packing measures can be thwarted by using IF Debugger. Data mining through the IF utility can aid the reverse engineer to construct a quick data and code flow analysis after just one run of the executable.

References

- [1] Hunt, G., and Brubacher, D., "Detours: Binary Interception of Win32 Functions," *Proceedings of the 3rd USENIX Windows NT Symposium*, Seattle, WA, July 1999, pp. 135-143.
- [2] Pedram, "Process Stalker vs. MS05-030." Internet: OpenRCE.org, www.openrce.org/articles/full_view/12, accessed August 6, 2007.
- [3] Aho, Sethi, and Ullman, *Compilers - Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA, 1985.
- [4] Høglund, G., and Bulter, J., *Rootkits: Subverting the Windows Kernel*, Addison-Wesley, Upper Saddle River, NJ, 2006.