



**RIVERSIDE RESEARCH INSTITUTE**

## **IF Debugger:**

A Custom Debugger utilizing  
Microsoft Detours

Jason Raber, Team Lead - Reverse Engineer

# Overview

- The Problem: Anti-debugging
- MS Detours - How it works
- IF Debugger
- Demonstration IF Debugger
- Auto-Unpacking
- Summary

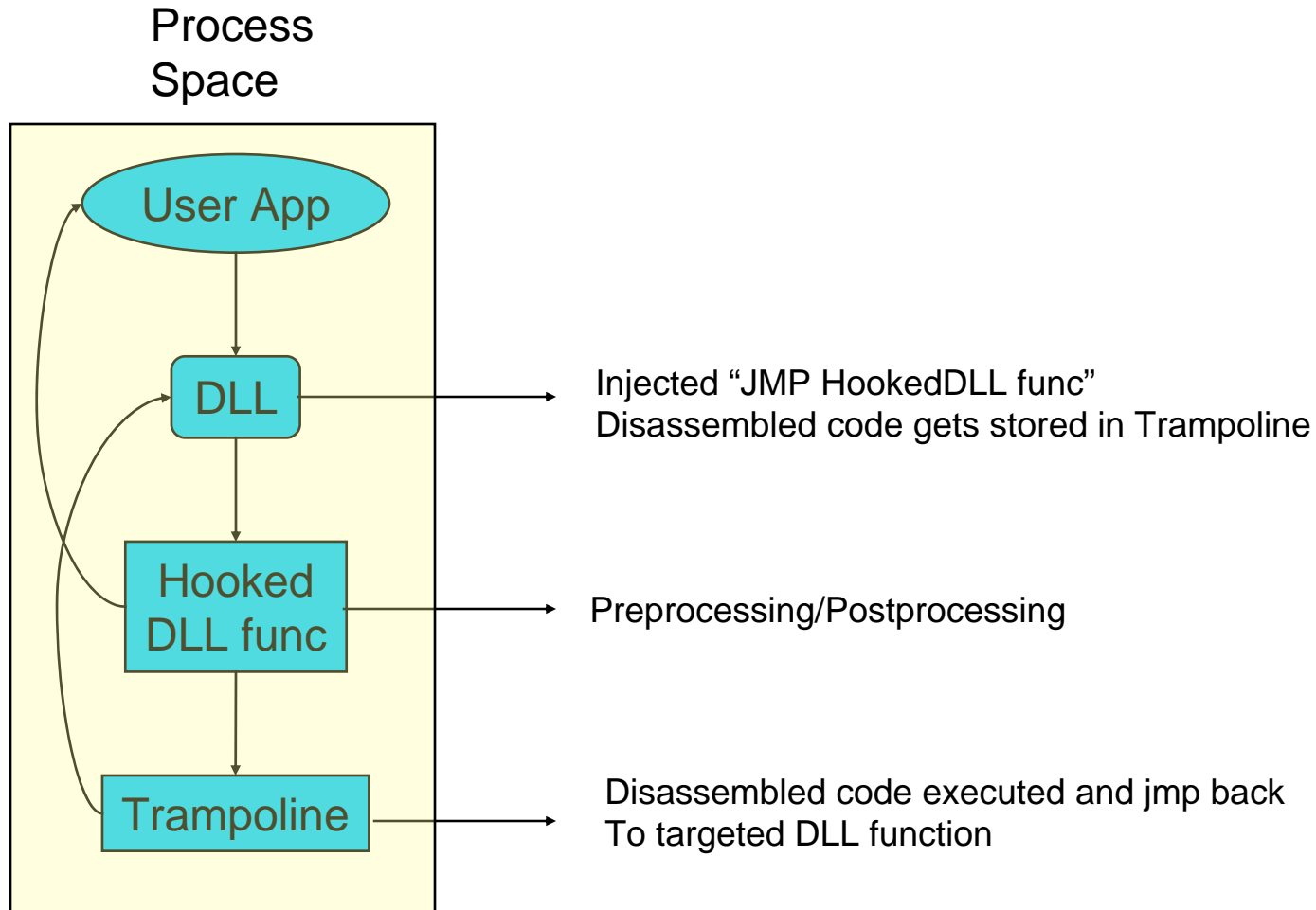
# The Problem: Anti-debugging

- 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.

# Overview

- The Problem: Anti-debugging
- MS Detours - How it works
- IF Debugger
- Demonstration IF Debugger
- Auto-Unpacking
- Summary

# Detours: How it works



# Overview

- The Problem: Anti-debugging
- MS Detours - How it works
- IF Debugger
- Demonstration IF Debugger
- Auto-Unpacking
- Summary

# Anti-debugger Checks

- IsDebuggerPresent()
- CheckRemoteDebuggerPresent()
- Timing Checks (GetTickCount())
- Checks for CC's
- Checks for hardware regs DR0-DR7
- IDT checks
- Known Debugger signatures
- Thrown exceptions

# Anti-debug Example

```
L
int main(void)
{
    if (IsDebuggerPresent())
    {
        printf("Debugger Detected - IsDebuggerPresent\n");
    }
    else printf("No Debugger present - IsDebuggerPresent\n");

    if (IsDebuggerLoaded())
    {
        printf("Debugger Detected - PEB\n");
    }
    else printf("No Debugger present - PEB\n");

    if(CheckForCCs(&main))
    {
        printf("Breakpoint detected - CC\n");
    }
    else printf("No Breakpoint detected - CC\n");

    return 0;
}
```

# PEB check

```
main.cpp* Start Page
(Global Scope) IsDebuggerLoaded()

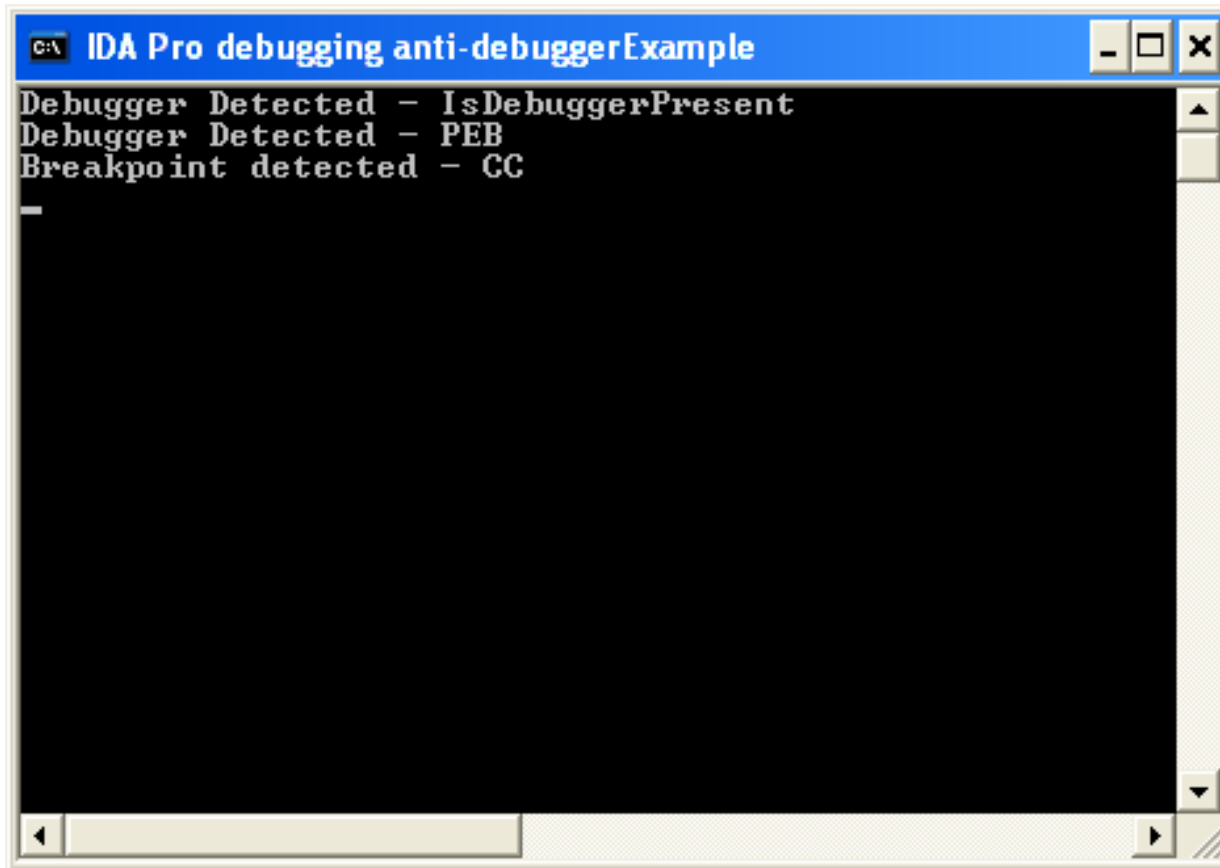
// TIB is a data structure in Win32 on x86 that stores info
// about the currently running thread
// it can be accessed through the Segment register
// FS (fs:[#])
bool IsDebuggerLoaded()
{
    _asm {
        mov eax, fs:[30h]           // pointer to PEB through
        movzx eax, byte ptr[eax+0x2] // offset to BeingDebugge
        or al,al
        jz normal_
        jmp out_
    out_:
        mov eax, 0x1
        jmp my_exit
    normal_:
        xor eax, eax

    my_exit:
        nop
    }
}
```

# INT 3 or CC's

```
__inline bool CheckForCCs(void *address) (  
    _asm (  
        mov esi, address    // load function address  
        mov al, [esi]      // load the opcode  
        cmp al, 0xCC       // check if the opcode is CCh  
        je BPXed          // yes, there is a breakpoint  
  
        // jump to return true  
        xor eax, eax       // false,  
        jmp NOBPX         // no breakpoint  
BPXed:  
    mov eax, 1            // breakpoint found  
NOBPX:  
    )  
)
```

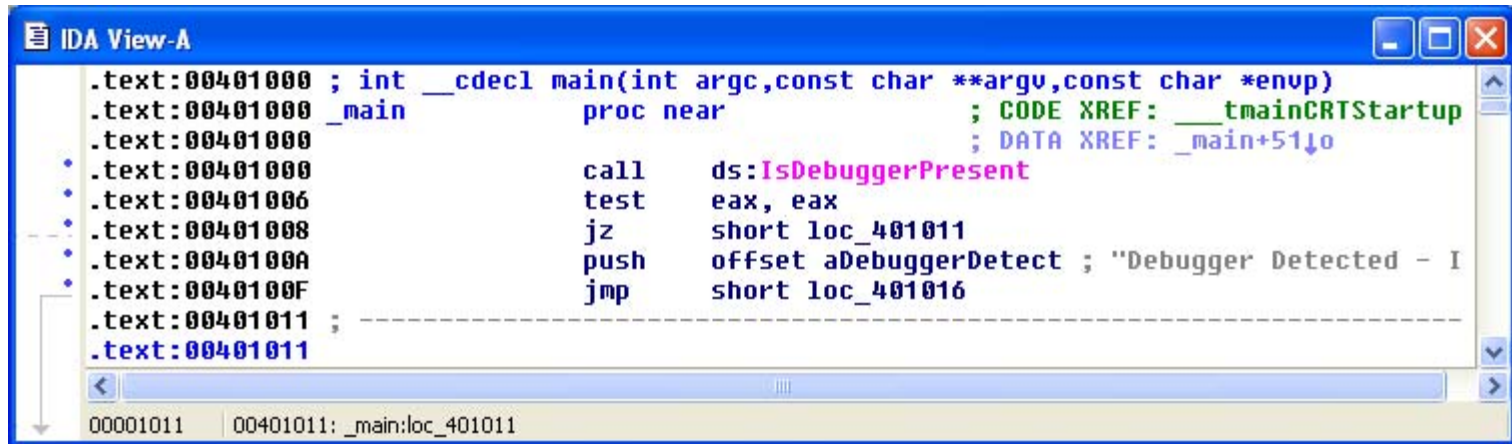
# Detected



```
C:\> IDA Pro debugging anti-debuggerExample
Debugger Detected - IsDebuggerPresent
Debugger Detected - PEB
Breakpoint detected - CC
```

Disassemble program and run through IDA debugger shows debugger detected

# IF Debugger undetected



```
IDA View-A
.text:00401000 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:00401000 _main          proc near          ; CODE XREF: __tmainCRTStartup
.text:00401000                                     ; DATA XREF: _main+51↓o
.text:00401000          call     ds:IsDebuggerPresent
.text:00401006          test    eax, eax
.text:00401008          jz     short loc_401011
.text:0040100A          push   offset aDebuggerDetect ; "Debugger Detected - I
.text:0040100F          jmp    short loc_401016
.text:00401011 ; -----
.text:00401011

00001011  00401011: _main:loc_401011
```

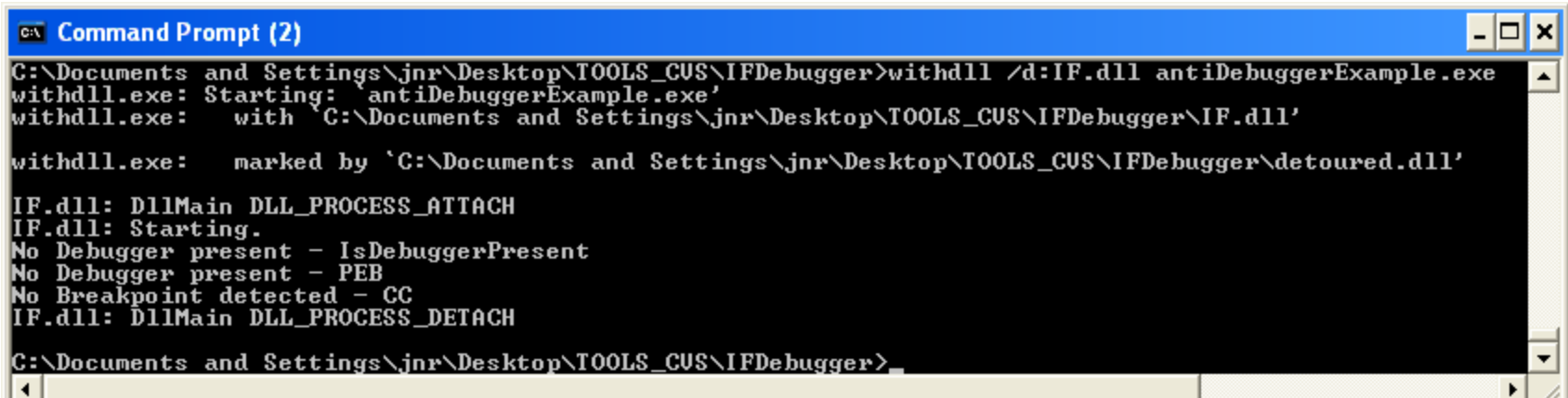
Add a breakpoint in the IF debugger @ 0x401000

# Overview

- The Problem: Anti-debugging
- MS Detours - How it works
- IF Debugger
- Demonstration IF Debugger
- Auto-Unpacking
- Summary

# Getting around anti-debug checks

- > withdll /d:IF.dll antiDebuggerExample.exe
- OK now we hit the breakpoint
- Before hitting go we can now view execution state (via registers etc)
- Continuing the run after the breakpoint “No Debugger present or CC detected” is displayed



```
C:\> Command Prompt (2)
C:\Documents and Settings\jnr\Desktop\TOOLS_CUS\IFDebugger>withdll /d:IF.dll antiDebuggerExample.exe
withdll.exe: Starting: 'antiDebuggerExample.exe'
withdll.exe:   with 'C:\Documents and Settings\jnr\Desktop\TOOLS_CUS\IFDebugger\IF.dll'
withdll.exe:   marked by 'C:\Documents and Settings\jnr\Desktop\TOOLS_CUS\IFDebugger\detoured.dll'
IF.dll: DllMain DLL_PROCESS_ATTACH
IF.dll: Starting.
No Debugger present - IsDebuggerPresent
No Debugger present - PEB
No Breakpoint detected - CC
IF.dll: DllMain DLL_PROCESS_DETACH
C:\Documents and Settings\jnr\Desktop\TOOLS_CUS\IFDebugger>
```

# Runtime BP

IF DEBUGGER

0x401000 JMP cd

BP: 0x401000 JMP cd

Disable BPs Enable BPs Clear BPs

EAX: 0x1BC1658 ST0: -1.#IND Runtime BP1: [ ]

EBX: 0x76057AFC ST1: -1.#IND Runtime BP2: [ ]

ECX: 0x1 ST2: -1.#IND Runtime BP3: [ ]

EDX: 0x77180F34 Runtime BP4: [ ]

EDI: 0x1770 Runtime BP5: [ ]

ESI: 0x2

EBP: 0x12FF60

ESP: 0x12FF54 EFLAGS: 0x246

0x50 Display DumpMemory

Stack

0x12FF54	0x401313
0x12FF58	0x1
0x12FF5C	0x1BC1630
0x12FF60	0x1BC1658
0x12FF64	0xCAC45869
0x12FF68	0x0
0x12FF6C	0x0
0x12FF70	0x7FFDD000
0x12FF74	0x12FF9C
0x12FF78	0x0
0x12FF7C	0x6
0x12FF80	0x0

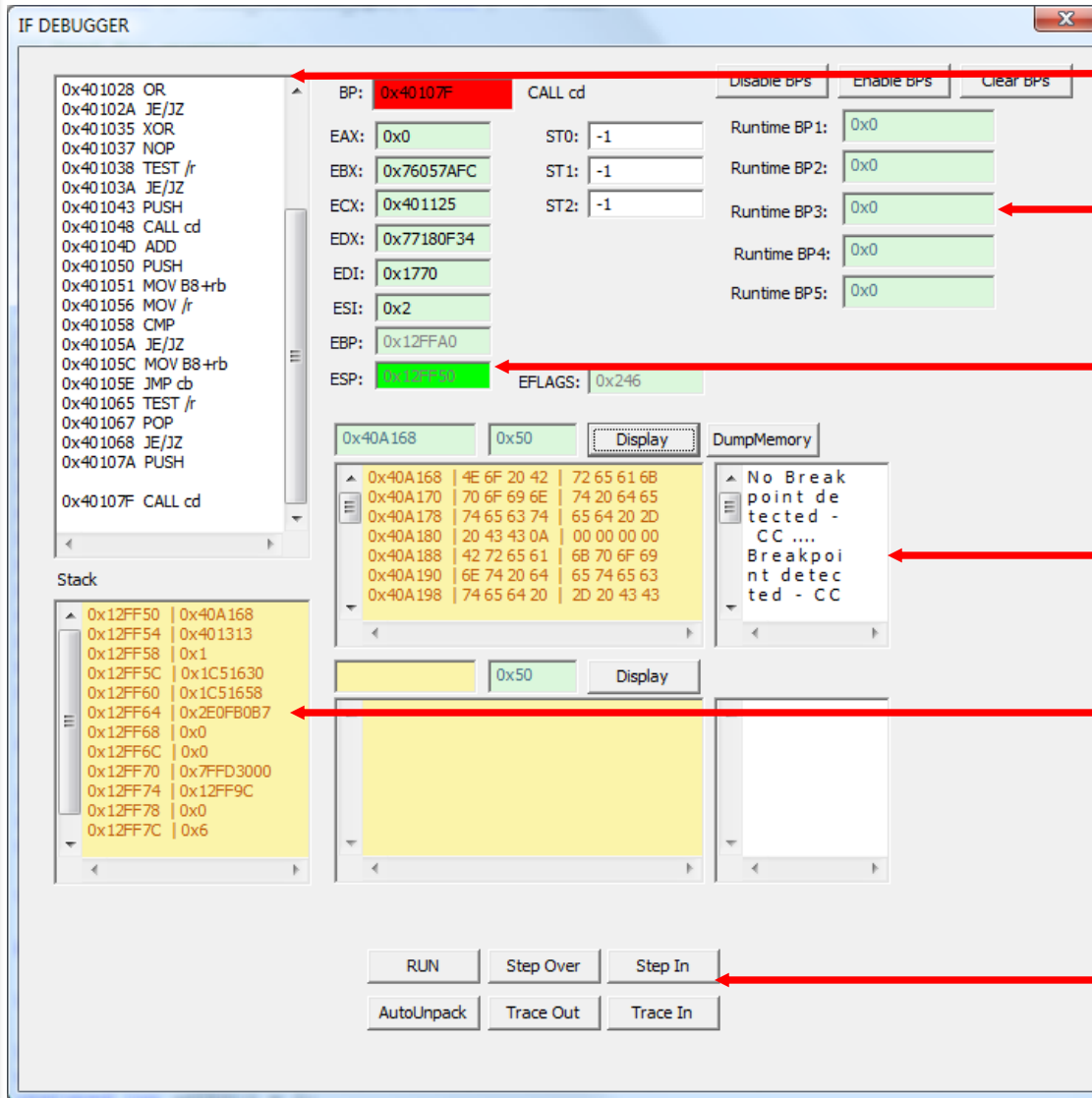
0x50 Display

RUN Step Over Step In

AutoUnpack Trace Out Trace In

Breakpoint Hit

# Debugger Primitives



Disassembly window

Runtime breakpoints

Color change reflects register contents changing by instruction 0

Memory Display

Stack

Controls

# Demo Debugger Primitives

```
int foo4(int my_var1, int my_var2)
{
    int sum = my_var1 + my_var2;

    printf("foo4\n");
    return sum;
}

void foo3(char *str) { printf("foo3 %s\n", str); }
void foo2(int my_var) { printf("foo2 %X\n", my_var); }
void foo1(void) { printf("foo1\n"); }

void nested(int my_var, char *str)
{
    printf("nested\n");

    foo2(my_var);
    foo3(str);
}

int main(void)
{
    Sleep(100);

    printf("main");
    foo1();

    _asm {
        mov eax, 0xABCD
    }

    foo2(0xBEEF);
    foo3("MyString");
    nested(0xDEAD, "nested");

    int ret = foo4(1,2);
    printf("Return value for foo4 is = %d\n", ret);

    return 0;
}
```



# IF Debugger Operations

- Step over until 0x401801
- Display 0x402118
- DumpMemory 0x40180F
- Disassemble dump.bin
- Modify Register
  - Runtime BP1 = 0x4017D4
  - Change EAX to 0x1234
  - Run

# IF vs. COTS debuggers

- Traditional Ring-3 debuggers (IDA/OLLY)
  - Register with OS
    - IsDebuggerPresent()
    - CheckRemoteDebugger()
    - PEB checks
- Ring-0 debuggers (SoftICE, WinDbg)
  - Get around IsDebuggerPresent() and CheckRemoteDebugger()
  - Requires drivers (SoftICE), or system to boot in debug-mode (WinDbg)
  - Generally need a 2<sup>nd</sup> computer
- Both types of debuggers use INT 3 and hardware registers DR0-DR7
- IDT checks
- Thrown exceptions
- Signature based detections
- The IF debugger circumvents all above anti-debug detection methods

# Done at Runtime

- Utilizing MS Detours to inject jumps to reroute code allows the IF debugger to have command of running exe
- Breakpoints on packed code
- Code remains unmodified on file

# Nuts and Bolts

- Breakpoints are emulated by injecting a jump to slack space that we own.
- Slack space then allows for control of running process (modify memory, change registers, etc)
- Replace stolen bytes
- Jump back

# Overview

- The Problem: Anti-debugging
- MS Detours - How it works
- IF Debugger
- Demonstration IF Debugger
- Auto-Unpacking
- Summary

# Auto-Unpacker

- This auto-unpacker capability lets the reverse engineer automatically unpack and decrypt an executable in a stealthy manner (i.e. bypassing all anti-debugging checks). When the auto-unpacker has stealthily unpacked and decrypted the executable, it will then notify the user that it has found the OEP (Original Entry Point).
  - Yoda Protector - took 2 seconds to unpack and decrypt to find OEP
  - UPX - 30 minute run to unpack and find OEP
- The framework is implemented... I have ideas for other heuristics for this type of automation to decrypt and unpack stealthily.

# Summary

- Software that prevents a reverse engineer from dynamic analysis through means of anti-debugging measures can be thwarted by using IF Debugger.
- This debugger will aid the engineer to do dynamic analysis quicker
- Auto-unpacker will help an engineer get a clean dump and find EOP quickly

# Contact

Jason Raber

Team Lead - Reverse Engineer

937-427-7085

[jraber@rri-usa.org](mailto:jraber@rri-usa.org)

