RIVERSIDE RESEARCH INSTITUTE

QuietRIATT

Rebuilding the Import Address Table Using Hooked DLL Calls

Jason Raber - Team Lead, Reverse Engineer

Brian Krumheuer – Reverse Engineer

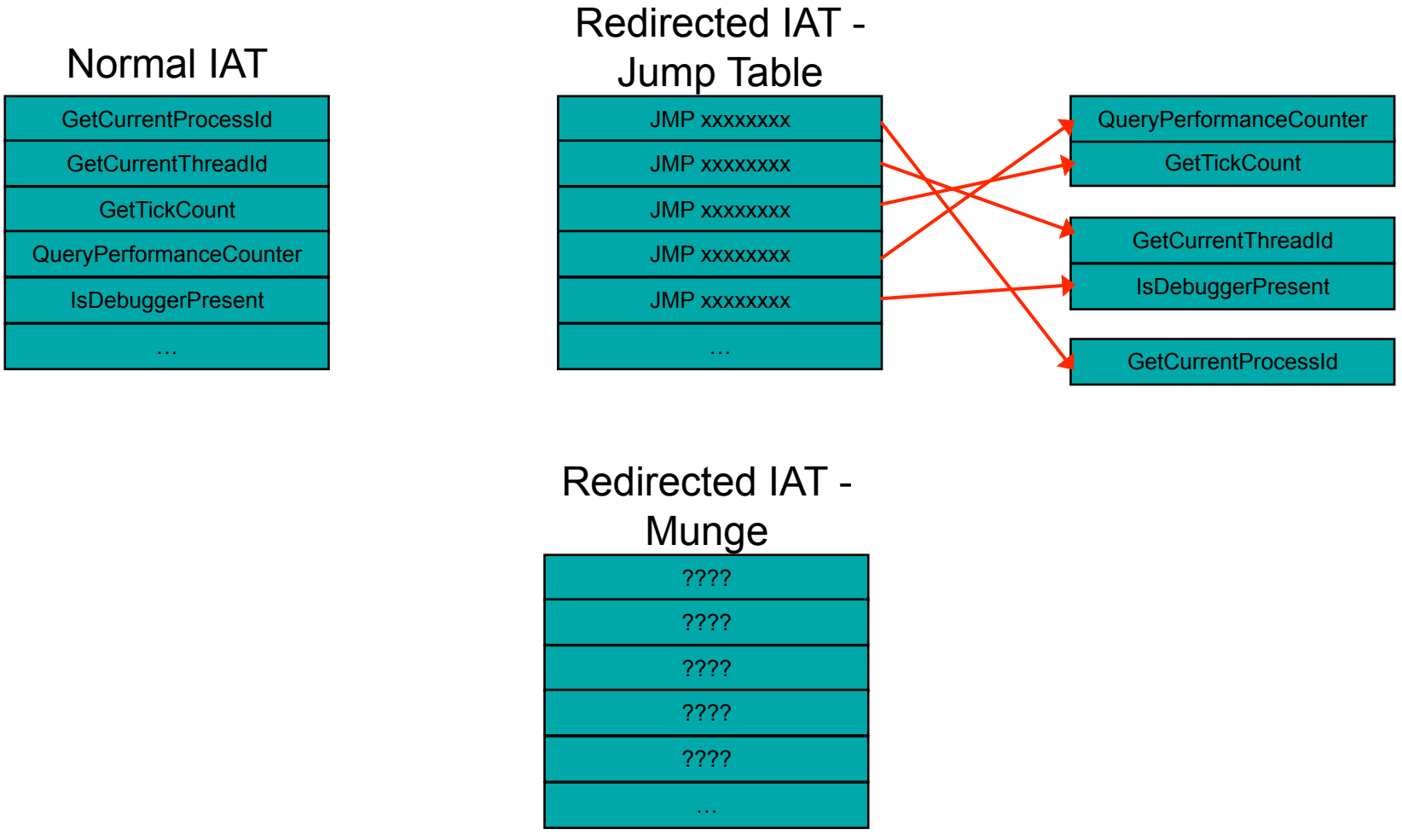
Overview

- The Problem: An EXE without an IAT
- How QuietRIATT Works
- Detours
- QuietRIATT
- Demonstration
- Summary
- Contact Info / Q&A

The Problem: An EXE without IAT

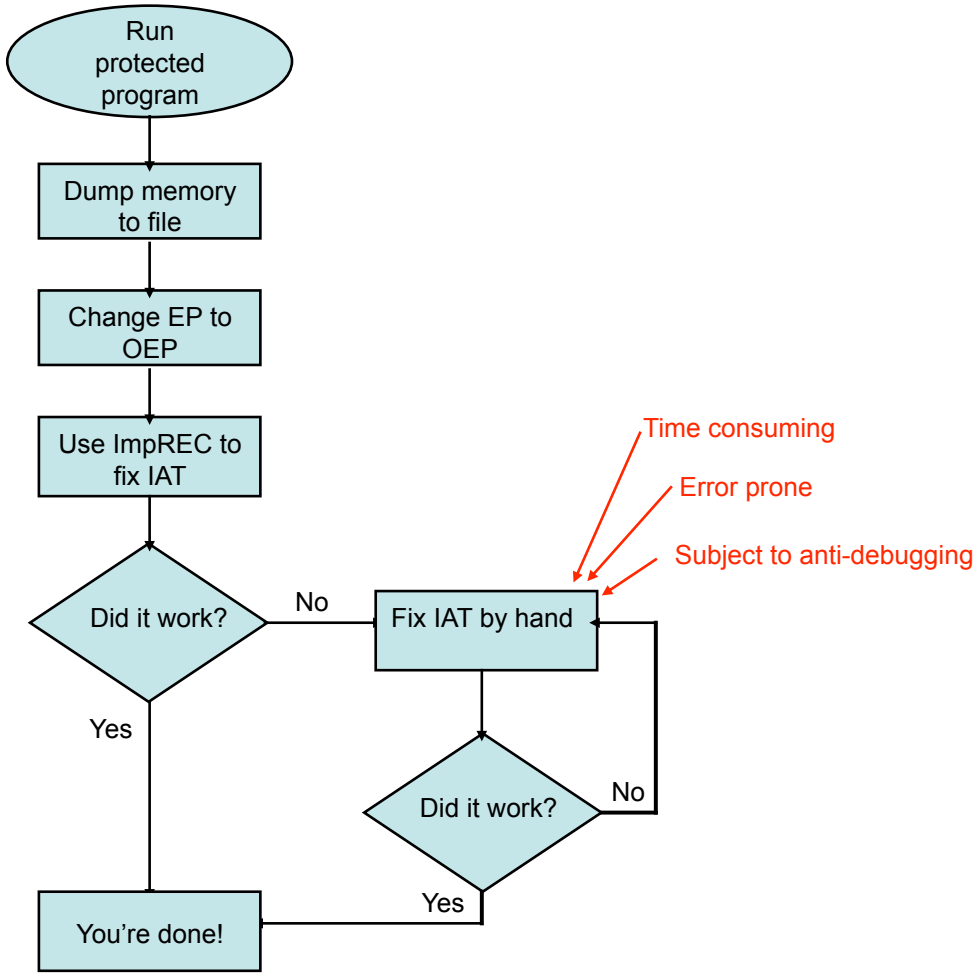
- Some malware employ protections that redirect the IAT, some completely destroy it
- When ImpREC falls short, QuietRIATT to the rescue!
- Lengthy manual labor now takes seconds
- Uses Detours to record DLL calls and assist in rebuilding the IAT

The Problem: An EXE without IAT



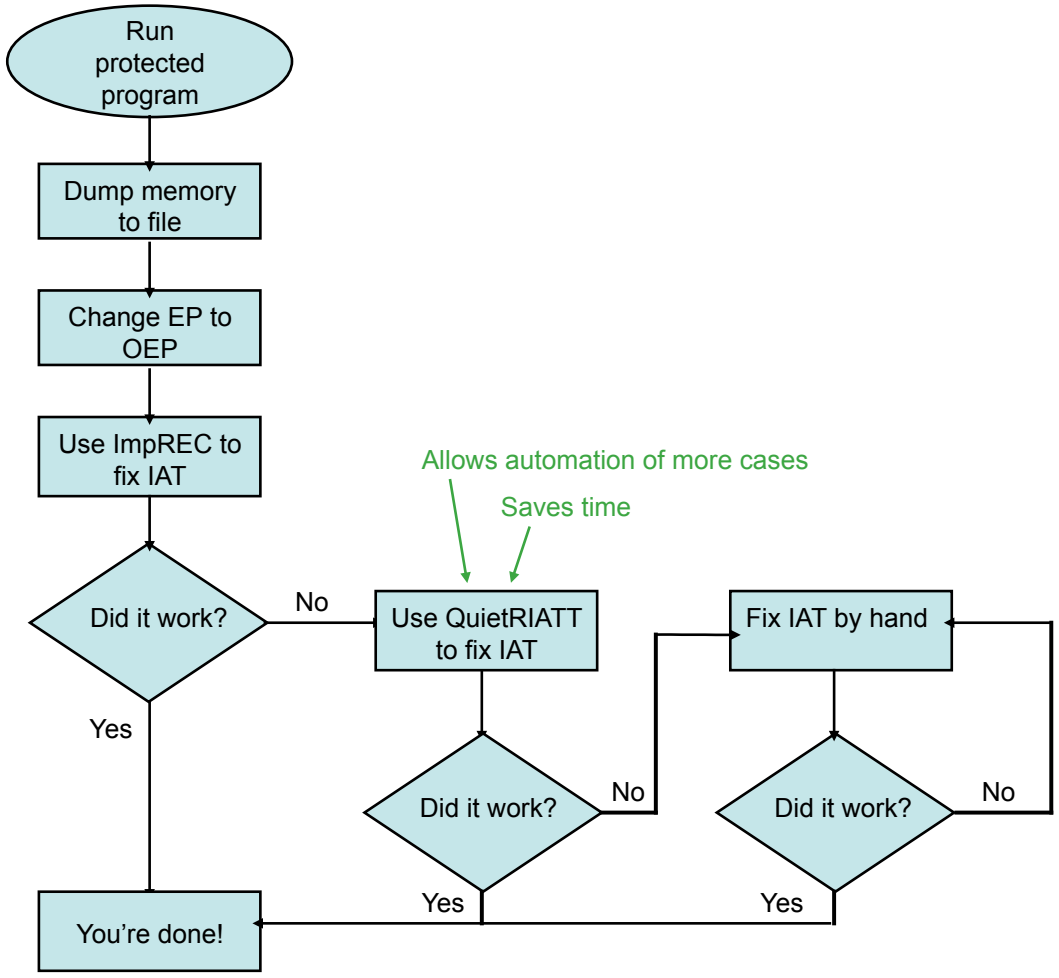
The Problem: An EXE without IAT

Removing Malware Wrapper-Style Protections



The Problem: An EXE without IAT

Removing Malware Wrapper-Style Protections



Overview

- The Problem: An EXE without an IAT
- How QuietRIATT Works
- Detours
- QuietRIATT
- Demonstration
- Summary
- Contact Info / Q&A

How QuietRIATT Works

1) Hook DLL calls using modified MS Detours

2) Detours ‘traceapi’ generates a log file of DLL calls

3) QuietRIATT annotates the IDAPro database

4) QuietRIATT generates a tree file with IAT info

5) Import tree file into ImpREC

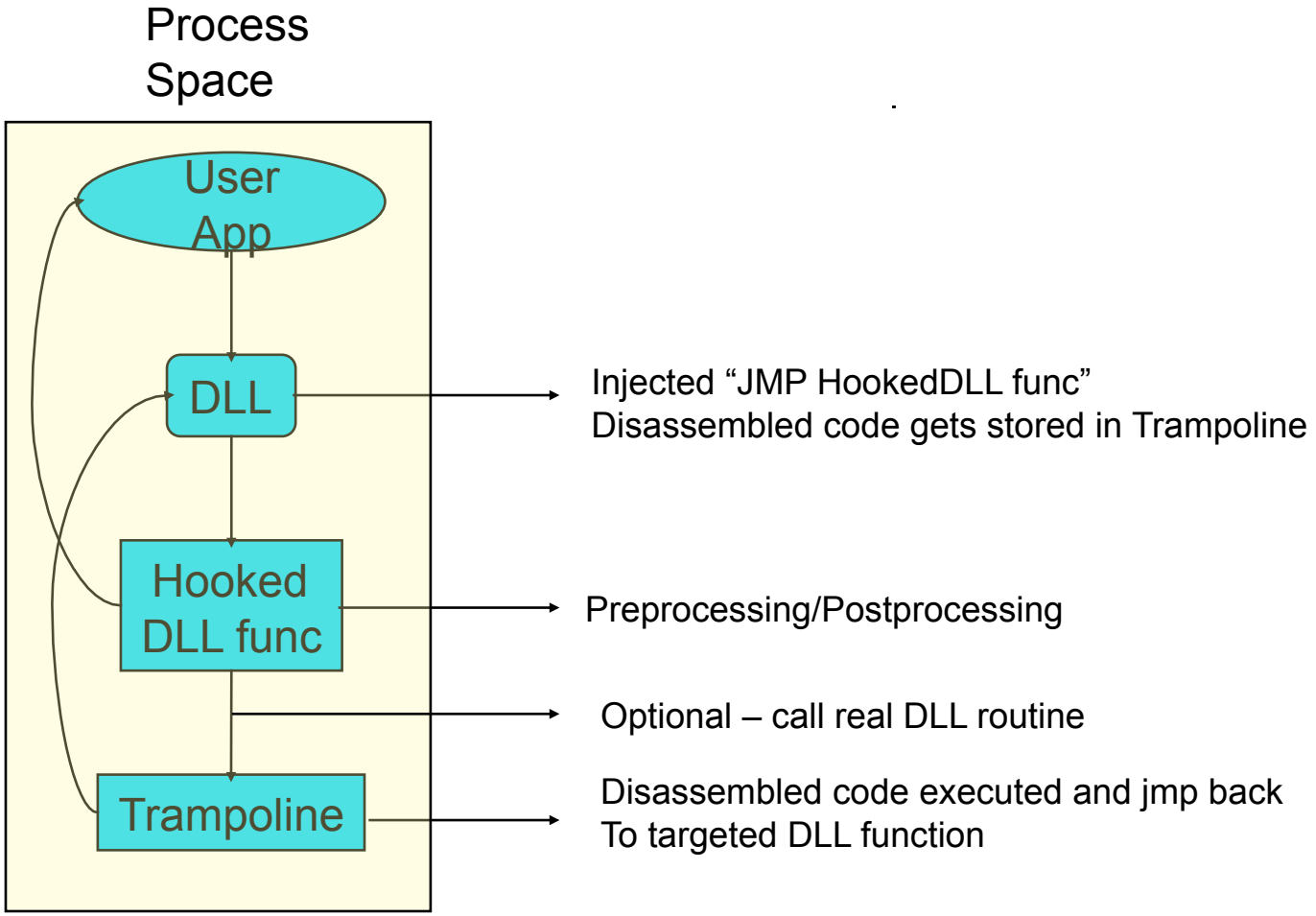
Overview

- The Problem: An EXE without an IAT
- How QuietRIATT Works
- Detours
- QuietRIATT
- Demonstration
- Summary
- Contact Info / Q&A

Why Detours

- Wanted something like Linux ‘strace’
  - Traces system calls
- Detours – ‘traceapi’
  - Similar to strace but traces DLL calls
  - Outputs parameters and return values
  - Helps see ‘real’ DLL calls from kernel32, user32, etc
    - This helps us when rebuilding IAT

Detours


Process
Space
User
App
DLL
Hooked
DLL func
Trampoline
Injected “JMP HookedDLL func”
Disassembled code gets stored in Trampoline
Preprocessing/Postprocessing
Optional – call real DLL routine
Disassembled code executed and jmp back
To targeted DLL function

Detours Macro

_win32.cpp is found in TRACEAPI found in detours under samples

```
void __stdcall Mine_Sleep(DWORD dwMilliseconds)
{
    PRINT_CALLER; _PrintEnter("Sleep (0x%X)\n", dwMilliseconds);

    __try {
        Real_Sleep(dwMilliseconds);
    } __finally {
        _PrintExit("Sleep  () ->\n");
    };
}
```

Inject macro

Macro Code:

```
#define GET_CALLER_ADDR  \
{                        \
    __asm mov eax, ebp   \
    __asm add eax, 4     \
    __asm mov pStack, eax \
}

#define PRINT_CALLER \
{                    \
    int *pStack = 0;  \
    GET_CALLER_ADDR   \
    _Print("[[[ %X ]]]\n", *pStack ); \
}
```

Detours in action

- Kernel32 Sleep API call is rerouted to trampoline space
- Return address is pushed on the stack

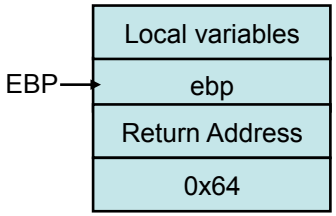
```
004017F3                push    ecx
004017F4                push    64h
004017F6                call    ds:Sleep
004017FC                push    offset aMain
00401801                call    ds:printf
00401807                add     esp, 4
0040180A                call    foo1
0040180F                mov     eax, 0ABCDh
00401814                push    0BFFFFh
```

```
7C802440 90              nop
7C802441 90              nop
7C802442 E9 E9 53 86 93  jmp     Mine_Sleep (10067830h)
7C802447 6A 00           push    0
7C802449 FF 75 08        push    dword ptr [ebp+8]
7C80244C E8 4B FF FF FF  call    7C80239C
7C802451 5D              pop     ebp
```

```
void __stdcall Mine_Sleep(DWORD dwMilliseconds)
{
10067830 push        ebp
10067831 mov         ebp,esp
10067833 push        0FFFFFFFFh
10067835 push        101332B0h
1006783A push        offset _except_handler3 (10096970h)
1006783F mov         eax,dword ptr fs:[00000000h]
10067845 push        eax
10067846 mov         dword ptr fs:[0],esp
1006784D add         esp,0FFFFFFF4h
10067850 push        ebx
10067851 push        esi
10067852 push        edi
    PRINT_CALLER; _PrintEnter("Sleep (0x%X)\n", dwMilliseconds);
10067853 mov         dword ptr [pStack],0
1006785A mov         eax,ebp
1006785C add         eax,4
1006785F mov         dword ptr [pStack],eax
10067862 mov         eax,dword ptr [pStack]
10067865 mov         ecx,dword ptr [eax]
10067867 push        ecx
```

Prolog stuff
Note: SP is assigned to BP to set stack frame

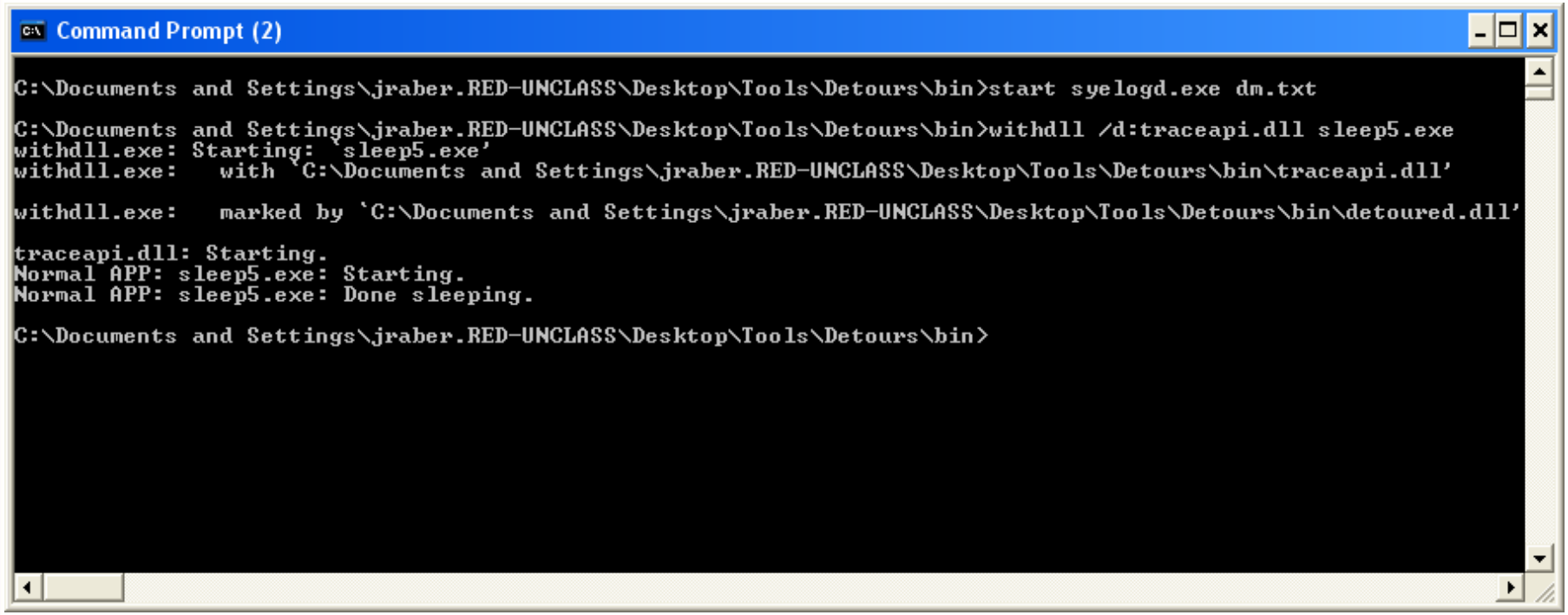
Stack – Grows High to Low



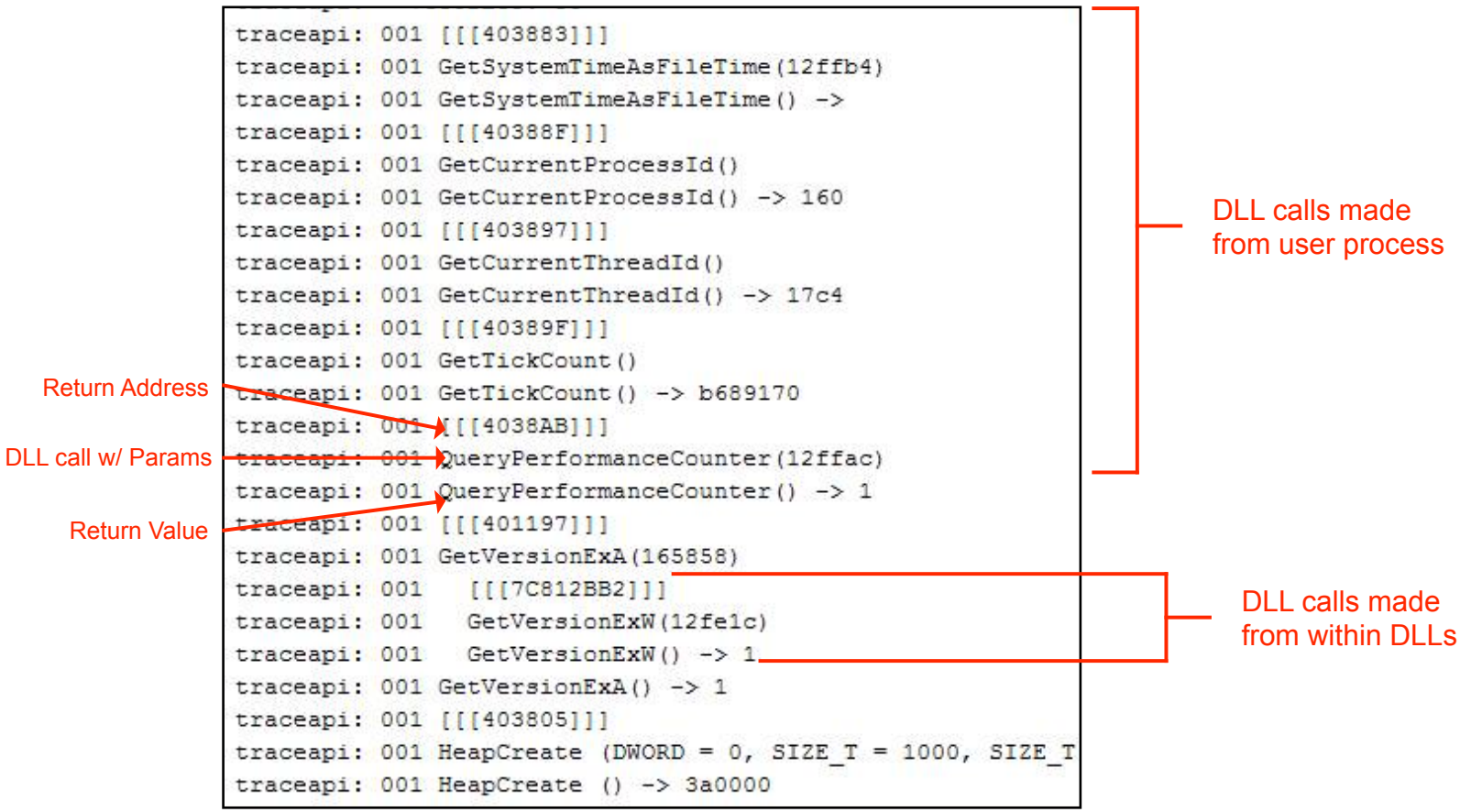
Since BP as saved due to saving stack frame
We can move down 4 bytes to ref
the return address

Running Traceapi

- **syelogd.exe** – system event logging. Use this utility to set up a pipe
- **withdll.exe** – load the detour traceapi.dll and detoured.dll into process sleep5.exe all done at runtime



Detours – User Process



Special Cases

In case ‘traceapi’ attach fails, don’t worry, the function is too small to trampoline.

```
20090211143519891  352 50.60: traceapi: ###
20090211143519891  352 50.60: traceapi: ### Env= 00165c88 [3d3a3a3d 005c3a3a]
20090211143519901  352 50.50: traceapi: Attach failed: `CoFreeAllLibraries': error 9
20090211143519901  352 50.60: traceapi:   77503507: 8b c0
20090211143519901  352 50.60: traceapi:   77503509: c3
20090211143519901  352 50.60: traceapi:   7750350A: 90
20090211143519911  352 50.50: traceapi: Attach failed: `GetCurrentProcess': error 9
20090211143519911  352 50.60: traceapi:   7C80DE85: 83 c8 ff
20090211143519911  352 50.60: traceapi:   7C80DE88: c3
20090211143519911  352 50.60: traceapi:   7C80DE89: 90
20090211143519921  352 50.60: traceapi: 001 [[[403883]]]
20090211143519921  352 50.60: traceapi: 001 GetSystemTimeAsFileTime(12ffb4)
20090211143519921  352 50.60: traceapi: 001 GetSystemTimeAsFileTime() ->
```

Overview

- The Problem: An EXE without an IAT
- How QuietRIATT Works
- Detours
- QuietRIATT
- Demonstration
- Summary
- Contact Info / Q&A

QuietRIATT

Quiet = Stealthy

R = Riverside

I = Import

A = Address

T = Table

T = Tool

QuietRIATT Steps

- Preparation:
  - Make DLL Function List
- Plug-In:
  - Read Detours output file
  - Find return address
  - Match ‘real’ call
  - Annotate IDA Pro
  - Create input file to ImpREC
  - Rebuild it

DLL Function List

- In order for QuietRIATT to know which DLL each function comes from, it is necessary to disassemble each DLL beforehand and make a list of the functions. This list is read into QuietRIATT during initialization. IDA makes this easy.

kernel32.dll export list from IDA disassembly

Name	Address	Ordinal
ActivateActCtx	7C80A644	1
AddAtomA	7C8354ED	2
AddAtomW	7C8326C1	3
AddConsoleAliasA	7C870CCF	4
AddConsoleAliasW	7C870C91	5
AddLocalAlternateComputerNameA	7C858F26	6
AddLocalAlternateComputerNameW	7C858E0A	7
AddRefActCtx	7C82BF01	8
AddVectoredExceptionHandler	7C808F63	9
AllocConsole	7C871321	10
AllocateUserPhysicalPages	7C85E712	11
AreFileApisANSI	7C83594F	12
AssignProcessToJobObject	7C82E44A	13
AttachConsole	7C871509	14
BackupRead	7C856DDF	15

- This is machine specific, so it has to be done on the same machine where the target program is run.

Create Function List

- Disassemble DLLs used in target application (e.g. kernel32, user32, …)
- Copy and paste export list into a text editor

QuietRIATT_liblist.txt

```
ActivateActCtx                          7C80A644 1    kernel32.dll
AddAtomA                                7C8354ED 2    kernel32.dll
AddAtomW                                7C8326C1 3    kernel32.dll
AddConsoleAliasA                        7C870CBF 4    kernel32.dll
AddConsoleAliasW                        7C870C81 5    kernel32.dll
AddLocalAlternateComputerNameA          7C858F26 6    kernel32.dll
AddLocalAlternateComputerNameW          7C858E0A 7    kernel32.dll
AddRefActCtx                            7C82BF01 8    kernel32.dll
AddVectoredExceptionHandler             7C808F63 9    kernel32.dll
AllocConsole                            7C871311 10   kernel32.dll
AllocateUserPhysicalPages               7C85E712 11   kernel32.dll
AreFileApisANSI                         7C83594F 12   kernel32.dll
```

Add DLL name to end (next to ordinal)

QuietRIATT and the 6 Degrees of Abe Simpson

Detours output file:

```
001 [[[ 4016BF ]]]
001 GetSystemTimeAsFileTime (13ffb4)
001 GetSystemTimeAsFileTime  () ->
001 [[[ 4016CB ]]]
001 GetCurrentProcessId ()
001 GetCurrentProcessId () -> e88
```

```
.text:004016B8                push    eax
.text:004016B9                call    ds:dword_402030
.text:004016BF                mov     esi, [ebp+var_4]
.text:004016C2                xor     esi, [ebp+var_8]
```

```
.rdata:00402028 dword_402028    dd 0CCCCCCCCh
.rdata:0040202C dword_40202C    dd 0CCCCCCCCh
.rdata:00402030 dword_402030    dd 0CCCCCCCCh
.rdata:00402034                 align 8
.rdata:00402038 dword_402038    dd 0CCCCCCCCh
.rdata:0040203C dword_40203C    dd 0CCCCCCCCh
```

```
.rdata:00402030 ; void __stdcall GetSystemTimeAsFileTime(LPFILETIME lpSystemTimeAsFileTime)
.rdata:00402030 GetSystemTimeAsFileTime dd 0CCCCCCCCh   ; DATA XREF: ___security_init_cookie+35↑r
.rdata:00402034                 align 8
```

ImpREC tree file:

```
16   1   0000202C   kernel32.dll   021B   InterlockedExchange
17   1   00002030   kernel32.dll   01BE   GetSystemTimeAsFileTime
```

Finding Return Address – 5 Byte Calls

```
.text:004016B8                  push    eax
.text:004016B9                  call    ds:dword_402030
.text:004016BF                  mov     esi, [ebp+var_4]
.text:004016C2                  xor     esi, [ebp+var_8]
```

```
call_addr = decode_prev_insn(ret_addr);
```

```
.text:004016B8                  push    eax
.text:004016B9                  call    ds:dword_402030
.text:004016BF                  mov     esi, [ebp+var_4]
.text:004016C2                  xor     esi, [ebp+var_8]
```

```
ua_ana0(call_addr);
set_name(cmd.Operands[0].addr, func_name);
```

```
.text:004016B8                  push    eax
.text:004016B9                  call    ds:GetSystemTimeAsFileTime
.text:004016BF                  mov     esi, [ebp+var_4]
.text:004016C2                  xor     esi, [ebp+var_8]
```

Finding Return Address – 2 Byte Calls

```
001 [[[ 40100E ]]]
001 printf (hello world!)
```

```
.text:00401001                  mov     esi, ds:dword_40209C
.text:00401007                  push    offset aHelloWorld ; "hello world!\n"
.text:0040100C                  call    esi
.text:0040100E                  add     esp, 8
```

```
// Check previous instructions until finding one with our reg in destination
for (int i=0; i<32; i++)
{
    prev_inst = decode_prev_insn(prev_inst);

    if (prev_inst == BADADDR)
        break;

    ua_ana0(prev_inst);

    if (cmd.itype == NN_mov &&
        cmd.Operands[0].reg == callReg &&
        cmd.Operands[1].type == o_mem)
    {
        set_name(cmd.Operands[1].addr, func_name);
        break;
    }
}
```

Special cases

- Unanalyzed Code
- IAT Redirection
- Jump Tables
- Addr Not Found
- Unknown Calls

Special Cases - Unanalyzed Code

Return Address in Unanalyzed Code

Detours Output

```
001 PeekMessageA (,,,,) -> 1
001 [[[ 4544F5 ]]]
001 GetMessageA (13fd5c,0,0,0)
001   [[[ 42DA28 ]]]
```

IDA Disassembly

```
.text:004544A7
.text:004544A7 ; ---------------------------------------------------------------------------
.text:004544A8 dword_4544A8    dd 8BEC4D8Bh, 1850FF01h, 6A006Ah, 7D250E8h, 0CCCCCC00h
.text:004544A8                 dd 0CCCCCCCCh, 0A164h, 0FF6A0000h, 4E8DDE68h, 89645000h
.text:004544A8                 dd 25h, 1CEC8300h, 8BF98B57h, 1450FF07h, 1A74C084h, 6A006Ah
.text:004544A8                 dd 4C8D006Ah, 0FF511024h, 4F338415h, 74C08500h, 0FFF88305h
.text:004544A8                 dd 0C0321275h, 244C8B5Fh, 0D89641Ch, 0
.text:0045450C ; ---------------------------------------------------------------------------
.text:0045450C                 add     esp, 28h
```

IDA SDK Functions

```
do_unknown(0x4544F5, true);
ua_code(0x4544F5);
```

Special Cases – Unanalyzed Code

```
.text:004544EF     db 0FFh
.text:004544F0     db  15h
.text:004544F1     db  84h ; ä
.text:004544F2     db  33h ; 3
.text:004544F3     db  4Fh ; O
.text:004544F4     db    0
.text:004544F5 ; ---------------------------------------
.text:004544F5     test    eax, eax
.text:004544F7     jz      short near ptr unk_4544FE
.text:004544F9     cmp     eax, 0FFFFFFFFh
```

```
.text:004544EF     call    ds:dword_4F3384
.text:004544F5     test    eax, eax
.text:004544F7     jz      short loc_4544FE
.text:004544F9     cmp     eax, 0FFFFFFFFh
```

```
//look backwards at bytes until finding a call
for(int i=2; i<=7; i++)
{
    do_unknown(ret_addr - i, true);

    if (ua_code(ret_addr - i))
    {
        if( cmd.itype == NN_call ||
            cmd.itype == NN_callfi || // Indirect Call Far
            cmd.itype == NN_callni)   // Indirect Call Near
        {
            result = cmd.ea;
            break;
        }
    }

    do_unknown(ret_addr - i, true);
}
```

Special Cases – IAT Redirection

Detours Output

```
001 [[[ 4D85B8 ]]]
001 TlsGetValue (DWORD = f)
001 TlsGetValue () -> b91e90
```

Call to a memory address that's not in the IAT

```
.text:004D85B2 call    dword_5733BC
.text:004D85B8 mov     esi, eax
```

No data at the address, so check the xrefs

```
.data:005733BC 00 00 00 00        dword_5733BC    dd 0
```

We find an IAT entry being moved into the address

```
.text:004D87C1                    mov     eax, ds:TlsGetValue
.text:004D87C6                    mov     dword_5733BC, eax
```

Special Cases – IAT Redirection

```
// For all cross references of addr
for (bool ok=xb.first_to(addr, XREF_DATA); ok; ok=xb.next_to())
{
    // If addr is being written to
    if (xb.type == dr_W)
    {
        ua_ana0(xb.from);

        // See what value is being written.  ex: mov addr, reg.
        if (cmd.Operands[1].type == o_reg)
        {
            ushort myReg = cmd.Operands[0].reg;

            // See if previous instruction is setting reg
            prev_inst = decode_prev_insn(xb.from);
            ua_ana0(prev_inst);

            if (cmd.itype == NN_mov &&
                cmd.Operands[0].reg == myReg &&
                (cmd.Operands[1].type == o_mem ||
                 cmd.Operands[1].type == o_near ||
                 cmd.Operands[1].type == o_far))
            {
                set_name(cmd.Operands[1].addr, name);
                found = true;
                break;
            }
        }
    }
}
```

Could add a check to see if the addr is in the IAT, and if not, make a recursive call.

Special Cases

- Jump Tables

```
if (!addr_is_in_iat(addr))
{
    // Check to see if addr is a jump to IAT addr
    ua_ana0(addr);
    if (cmd.itype >= NN_ja && cmd.itype <= NN_jmpshort)
    {
        if (addr_is_in_iat(cmd.Operands[0].addr))
        {
            set_name(cmd.Operands[0].addr, name);
        }
    }
}
```

Special Cases – Addr Not Found

IDA Pro Message Window

```
0045020D: Couldn't find address of call GetClassNameA.  Have to fix manually.
```

IDA Pro Disassembly

```
.text:004501EB 53                      push    ebx
.text:004501EC 8B 1D 8C 33 4F 00       mov     ebx, ds:dword_4F338C
.text:004501F2 57                      push    edi
.text:004501F3 BE 00 01 00 00          mov     esi, 100h
.text:004501F8 EB 06                   jmp     short loc_450200
.text:004501F8                 ; ---------------------------------------------------------
.text:004501FA 8D 9B 00 00 00 00       align 10h
.text:00450200
.text:00450200                 loc_450200:                             ; CODE
.text:00450200 56                      push    esi
.text:00450201 8D 4C 24 14             lea     ecx, [esp+28h+var_14]
.text:00450205 E8 86 6A FB FF          call    sub_406C90
.text:0045020A 56                      push    esi
.text:0045020B 50                      push    eax
.text:0045020C 55                      push    ebp
.text:0045020D FF D3                   call    ebx
.text:0045020F 8D 4C 24 10             lea     ecx, [esp+30h+var_20]
```

Addr of GetClassNameA being moved into EBX.

decode_prev_insn() won't work past unanalyzed data.

Call being made through EBX.

Special Cases – Addr Not Found

IDA Pro Message Window

```
004D9172: Couldn't find address of call IsProcessorFeaturePresent.  Have to fix manually.
```

IDA Pro Disassembly

```
.text:004D9151 __ms_p5_mp_test_fdiv proc near          ; CODE XREF: __fpmath+5↑p
.text:004D9151                 push    offset aKernel32 ; "KERNEL32"
.text:004D9156                 call    ds:GetModuleHandleA
.text:004D915C                 test    eax, eax
.text:004D915E                 jz      short loc_4D9175
.text:004D9160                 push    offset aIsprocessorfea ; "IsProcessorFeaturePresent"
.text:004D9165                 push    eax
.text:004D9166                 call    ds:GetProcAddress
.text:004D916C                 test    eax, eax
.text:004D916E                 jz      short loc_4D9175
.text:004D9170                 push    0
.text:004D9172                 call    eax
.text:004D9174                 retn
```

Special Cases – Unknown Calls

- If not every call is used during execution (which is likely), QuietRIATT won't know what the call is, so defaults have to be chosen as placeholders.

support.h

```
DEFAULT_DLL_FUNC default_funcs[] = {
    {"kernel32.dll", "AddAtomA", 2},
    {"user32.dll",   "MessageBoxA", 477},
    {"shell32.dll",  "GetFileNameFromBrowse", 63},
    {"advapi32.dll", "ReportEventA", 523},
    {"comdlg32.dll", "CommDlgExtendedError", 105},
    {"gdi32.dll",    "TextOutA", 591},
    {"msvcr80.dll",  "__winitenv", 255},
    {"ole32.dll",    "CreateErrorInfo", 138},
    {"comctl32.dll", "CreateStatusWindowA", 6},
    {NULL, NULL, NULL}
};
```

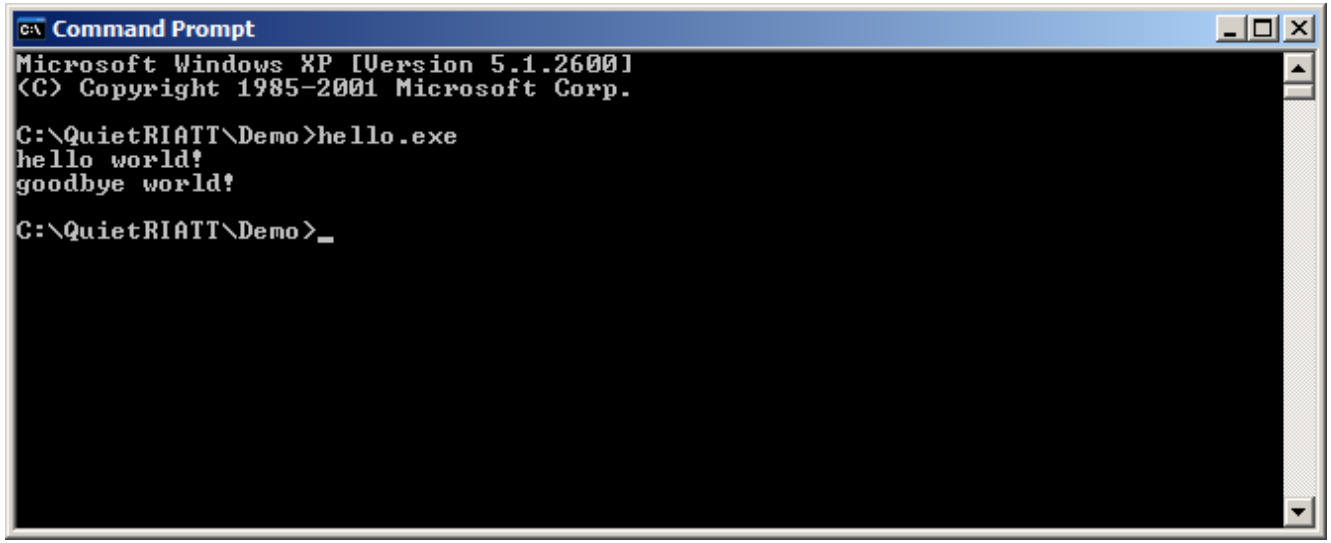
- When new functionality is discovered in the program, re-run Detours and QuietRIATT and the new functions will be added.

Overview

- The Problem: An EXE without an IAT
- How QuietRIATT Works
- Detours
- QuietRIATT
- Demonstration
- Summary
- Contact Info / Q&A

Demonstration

- Sample “Hello World” with IAT removed



Overview

- The Problem: An EXE without an IAT
- How QuietRIATT Works
- Detours
- QuietRIATT
- Demonstration
- Summary
- Contact Info / Q&A

Summary

- Not an ImpREC replacement, QuietRIATT fills a gap that ImpREC doesn't cover
- A stealthy solution
- Can save many hours of tedious, error prone manual labor

Future Work

- Add ability for QuietRIATT to fix binary directly (no need for ImpREC).
- In cases where IAT is dynamic, keep internal list of entries
- Feed QuietRIATT run trace from stealthy debugger to fix case where “address not found”

Overview

- The Problem: An EXE without an IAT
- How QuietRIATT Works
- Detours
- QuietRIATT
- Demonstration
- Summary
- Contact Info / Q&A

Contact Info / Q&A

Riverside Research Institute Software Security Team
http://www.rri-usa.org/isrsoftware.html

For binary and source code, contact us at:

Jason Raber
Team Lead, Reverse Engineer
937-427-7085
jraber@rri-usa.org

Brian Krumheuer
Reverse Engineer
937-427-7087
bkrumheuer@rri-usa.org