Automating RE with Python

A Gentle Introduction

Who are you

- I assume some experience with
  - Binary reversing?
  - Debugging?
  - Exploitation?
  - Python?

WHOAMI

- NERD (ZOMBIES, CYLONS...)
- GEEK (REVERSING, PYTHON...)
- CONSULTANT :)

WE ARE HIRING!

SHAMELESS PLUG!

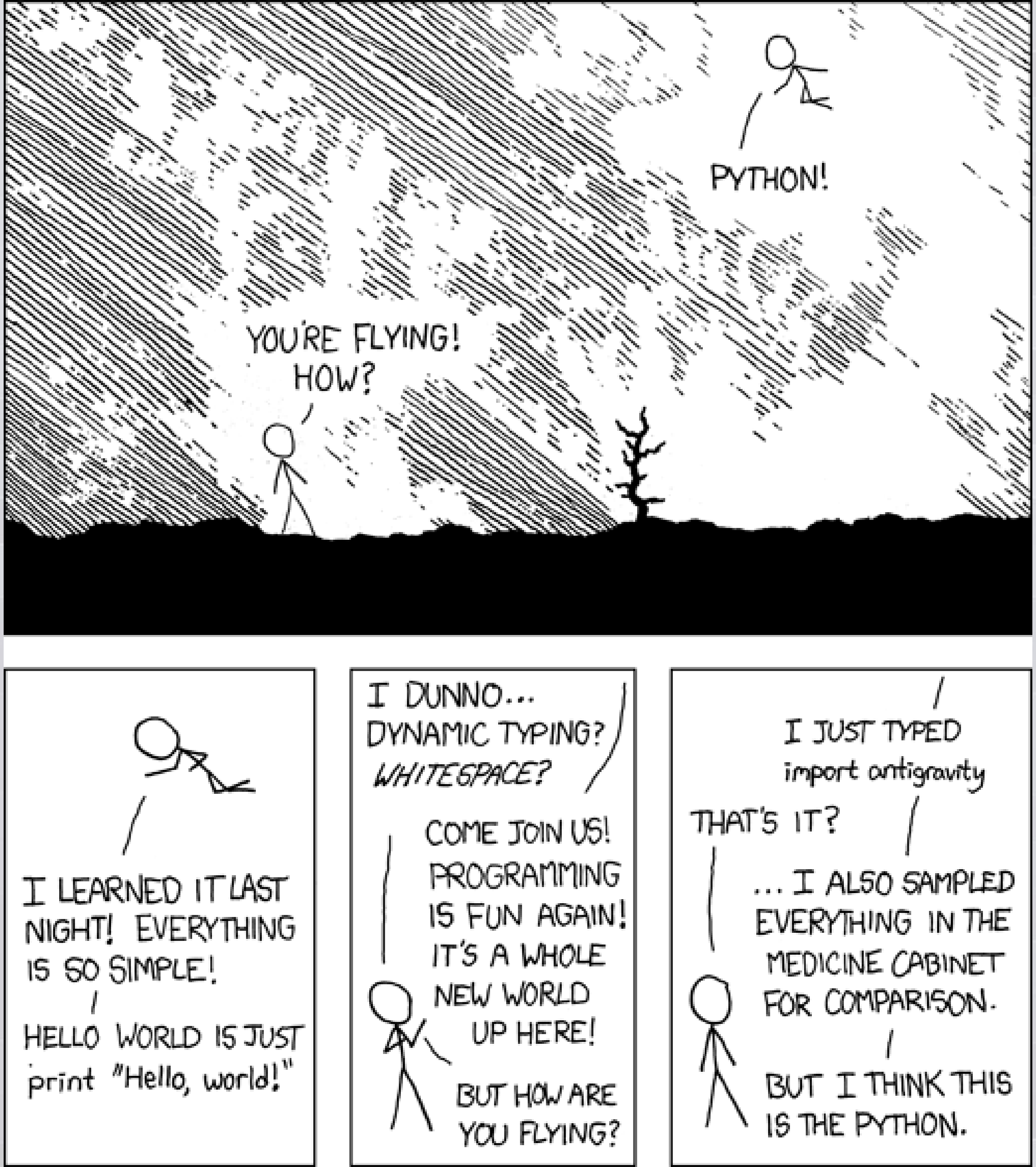
- SECURITY PEOPLE
- HIGHLY SKILLED TEAM
- HARDWARE, MOBILE, BINARY, WEB, SOURCE CODE, NETWORK …
- FUN, RESEARCH, CONS, ETC.
- CONSULTANCY / BUSINESS ORIENTED
- TALK TO ME!

n runs professionals

AGENDA

- CHECKING OUT THE APPLICATION
  - STATIC ANALYSIS
  - WINAPPDBG
  - INTEL PIN
  - VDB / VTRACE
- FREEDOM

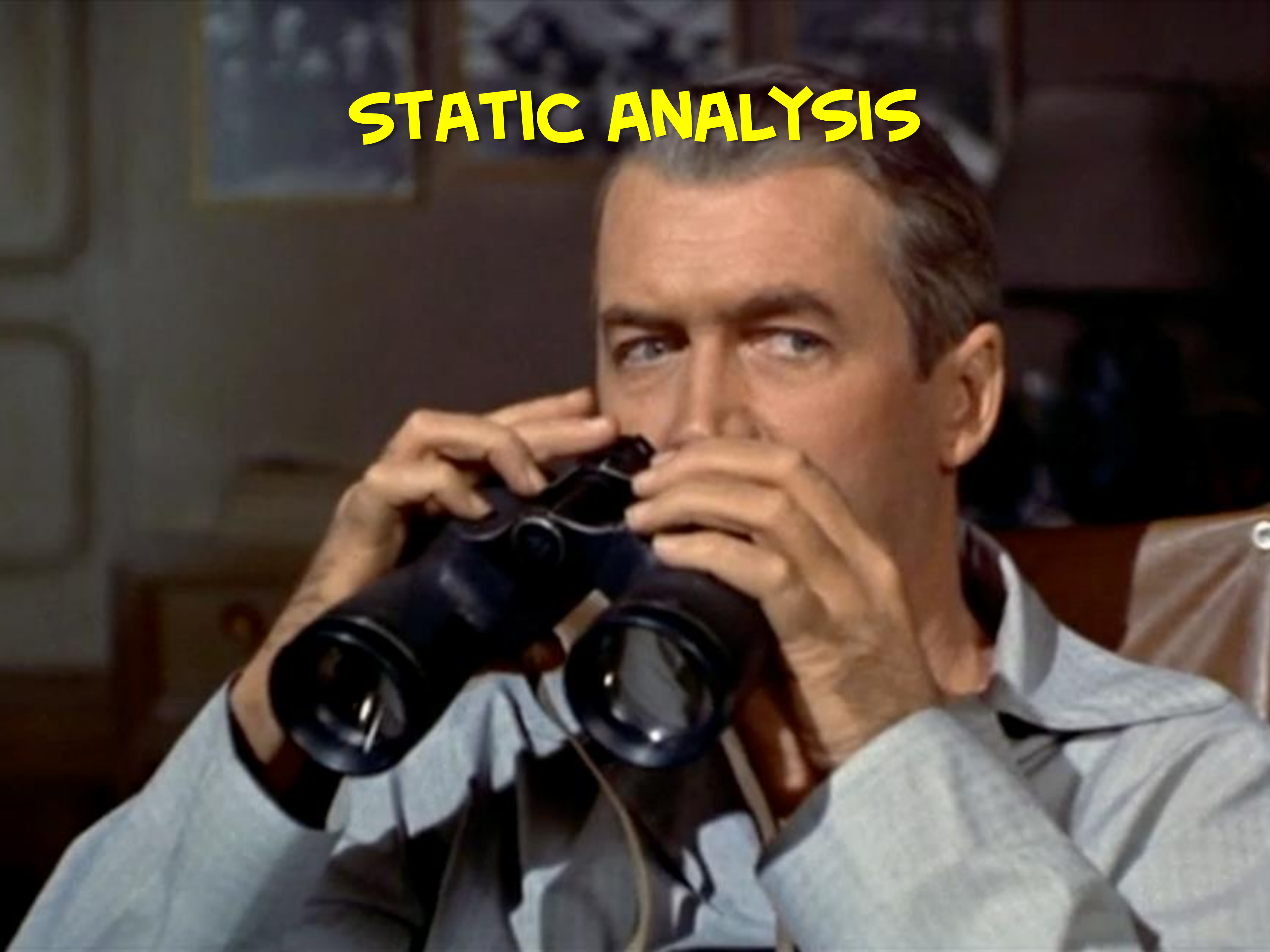
IN
TWO HOURS!

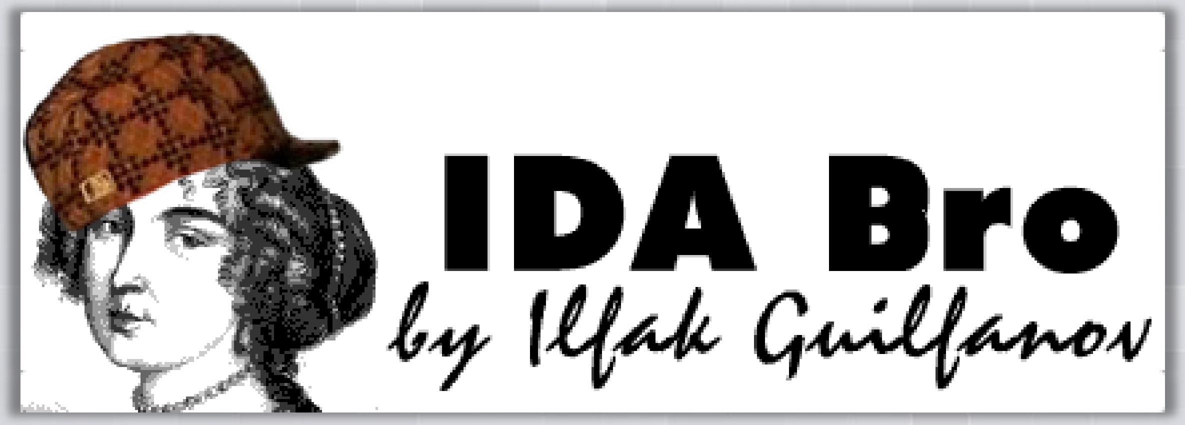

PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Setting the Scope

- 2 hours is not very long
- Just an introduction
- Basically a compilation of things very intelligent people did
- A nice overview if you don't have a deep knowledge of this topic

Static Analysis



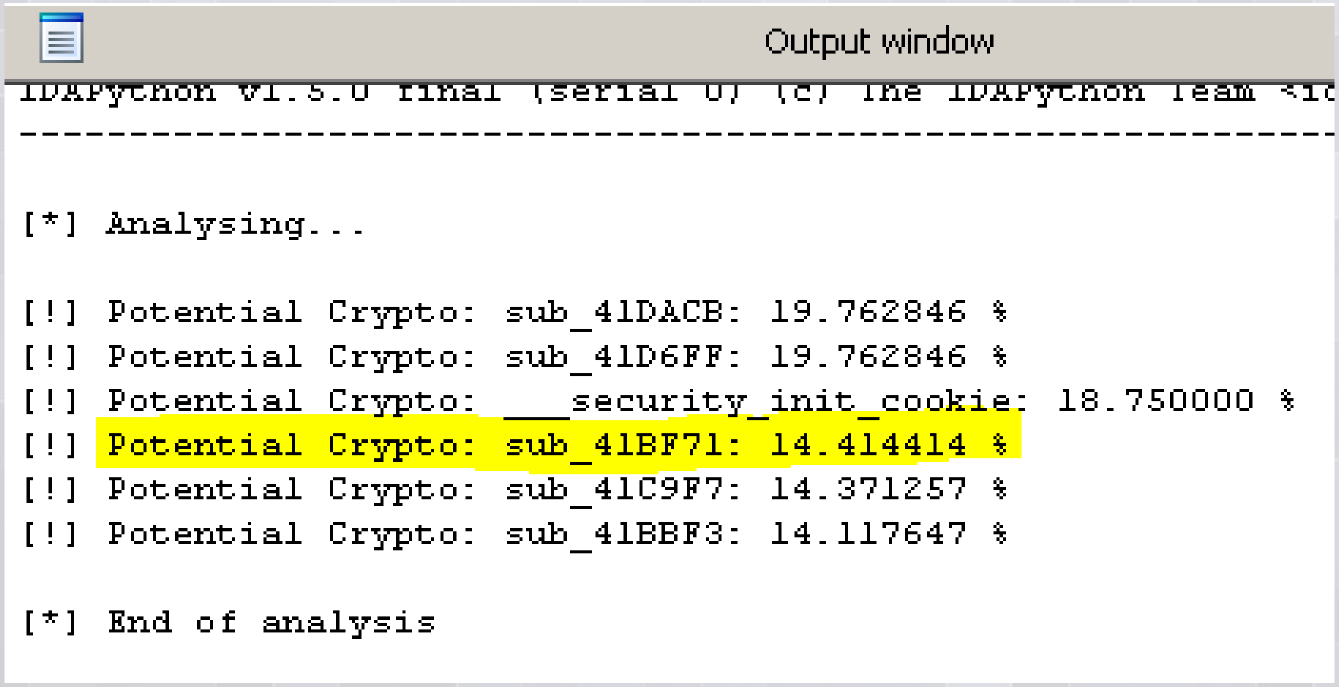

IDA Bro
by Ilfak Guilfanov

STATIC ANALYSIS

IDA PRO SCRIPTS

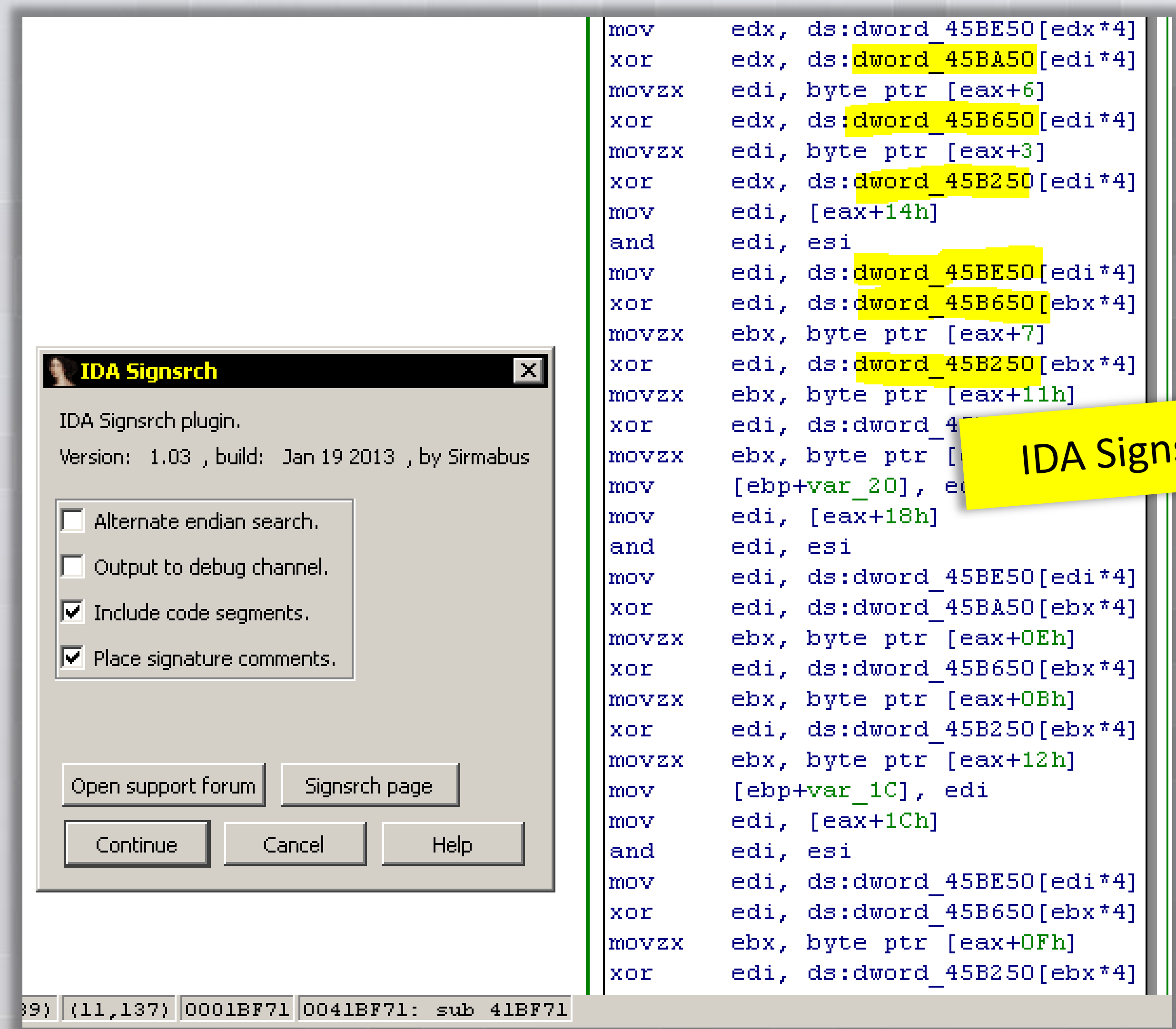
- IDC
  - IN C LANGUAGE
  - MUST RECOMPILE EVERY TIME
- IDAPYTHON
  - PYTHON BINDINGS
  - ME GUSTA...

NAIVE CRYPTO SEARCH



```
IDAPython v1.5.0 final (serial 0) (c) The IDAPython Team <id
-----------------------------------------------------------------------


[*] Analysing...

[!] Potential Crypto: sub_41DACB: 19.762846 %
[!] Potential Crypto: sub_41D6FF: 19.762846 %
[!] Potential Crypto: ___security_init_cookie: 18.750000 %
[!] Potential Crypto: sub_41BF71: 14.414414 %
[!] Potential Crypto: sub_41C9F7: 14.371257 %
[!] Potential Crypto: sub_41BBF3: 14.117647 %

[*] End of analysis
```

NAIVE CRYPTO SEARCH



IDA Signsrch

IDA Signsrch plugin.
Version: 1.03 , build: Jan 19 2013 , by Sirmabus

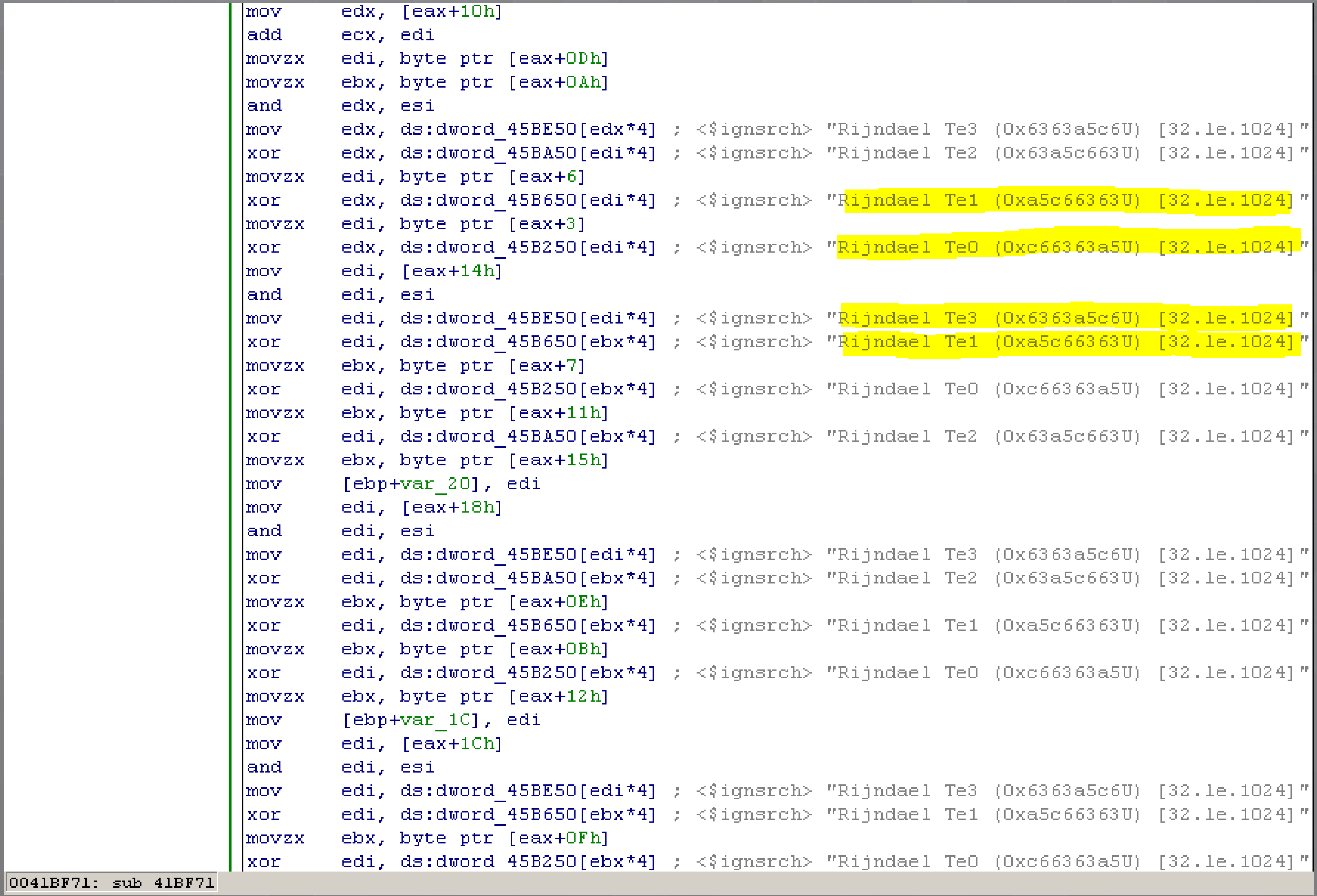
- [ ] Alternate endian search.
- [ ] Output to debug channel.
- [x] Include code segments.
- [x] Place signature comments.

```
mov     edx, ds:dword_45BE50[edx*4]
xor     edx, ds:dword_45BA50[edi*4]
movzx   edi, byte ptr [eax+6]
xor     edx, ds:dword_45B650[edi*4]
movzx   edi, byte ptr [eax+3]
xor     edx, ds:dword_45B250[edi*4]
mov     edi, [eax+14h]
and     edi, esi
mov     edi, ds:dword_45BE50[edi*4]
xor     edi, ds:dword_45B650[ebx*4]
movzx   ebx, byte ptr [eax+7]
xor     edi, ds:dword_45B250[ebx*4]
movzx   ebx, byte ptr [eax+11h]
xor     edi, ds:dword_4
movzx   ebx, byte ptr [
mov     [ebp+var_20], ed
mov     edi, [eax+18h]
and     edi, esi
mov     edi, ds:dword_45BE50[edi*4]
xor     edi, ds:dword_45BA50[ebx*4]
movzx   ebx, byte ptr [eax+0Eh]
xor     edi, ds:dword_45B650[ebx*4]
movzx   ebx, byte ptr [eax+0Bh]
xor     edi, ds:dword_45B250[ebx*4]
movzx   ebx, byte ptr [eax+12h]
mov     [ebp+var_1C], edi
mov     edi, [eax+1Ch]
and     edi, esi
mov     edi, ds:dword_45BE50[edi*4]
xor     edi, ds:dword_45B650[ebx*4]
movzx   ebx, byte ptr [eax+0Fh]
xor     edi, ds:dword_45B250[ebx*4]
```

```
39) (11,137) 0001BF71 0041BF71: sub 41BF71
```

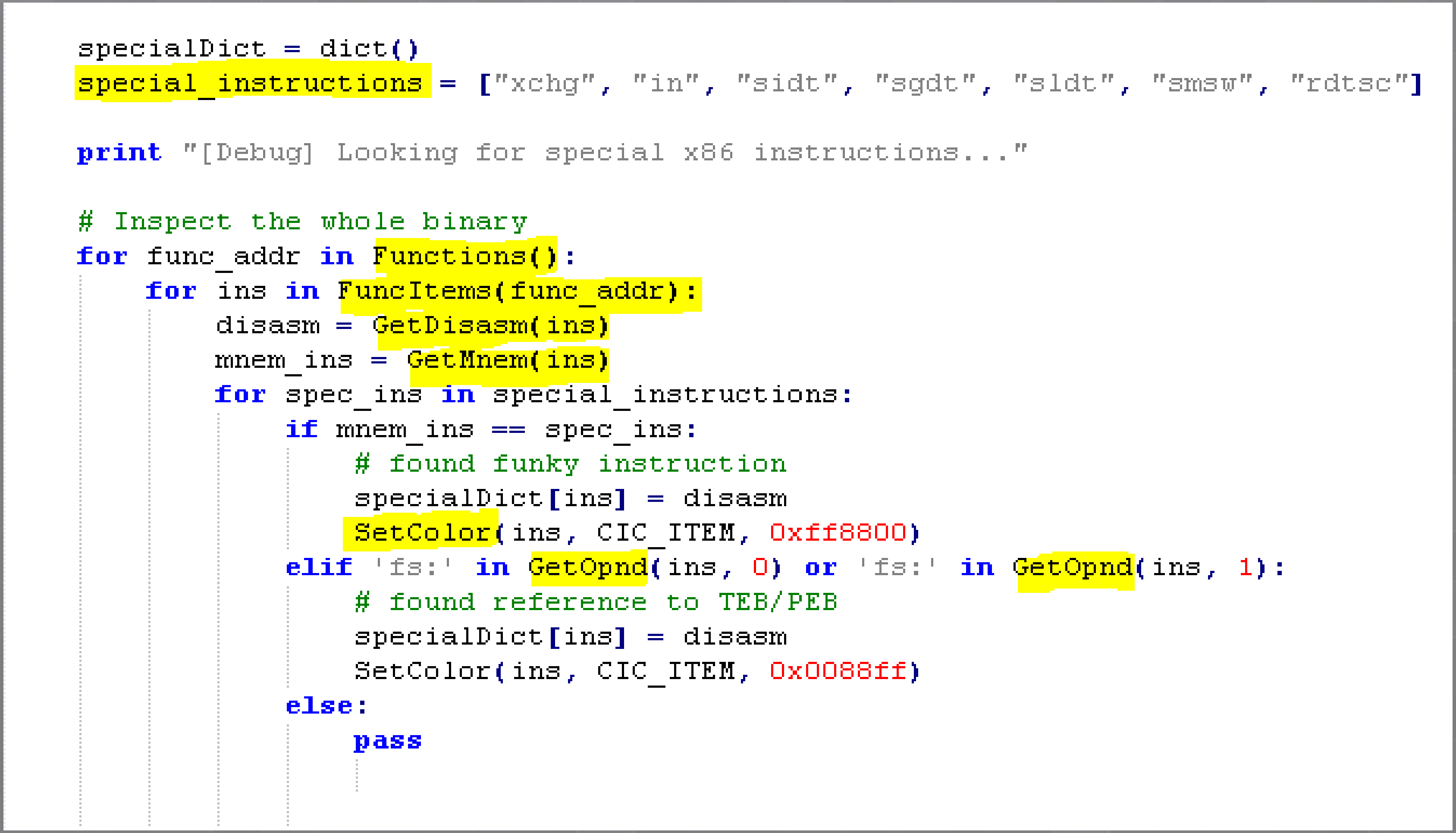
IDA Signsrch from Luigi Auriemma

NAIVE CRYPTO SEARCH



```
mov     edx, [eax+10h]
add     ecx, edi
movzx   edi, byte ptr [eax+0Dh]
movzx   ebx, byte ptr [eax+0Ah]
and     edx, esi
mov     edx, ds:dword_45BE50[edx*4] ; <$ignsrch> "Rijndael Te3 (0x6363a5c6U) [32.le.1024]"
xor     edx, ds:dword_45BA50[edi*4] ; <$ignsrch> "Rijndael Te2 (0x63a5c663U) [32.le.1024]"
movzx   edi, byte ptr [eax+6]
xor     edx, ds:dword_45B650[edi*4] ; <$ignsrch> "Rijndael Te1 (0xa5c66363U) [32.le.1024]"
movzx   edi, byte ptr [eax+3]
xor     edx, ds:dword_45B250[edi*4] ; <$ignsrch> "Rijndael Te0 (0xc66363a5U) [32.le.1024]"
mov     edi, [eax+14h]
and     edi, esi
mov     edi, ds:dword_45BE50[edi*4] ; <$ignsrch> "Rijndael Te3 (0x6363a5c6U) [32.le.1024]"
xor     edi, ds:dword_45B650[ebx*4] ; <$ignsrch> "Rijndael Te1 (0xa5c66363U) [32.le.1024]"
movzx   ebx, byte ptr [eax+7]
xor     edi, ds:dword_45B250[ebx*4] ; <$ignsrch> "Rijndael Te0 (0xc66363a5U) [32.le.1024]"
movzx   ebx, byte ptr [eax+11h]
xor     edi, ds:dword_45BA50[ebx*4] ; <$ignsrch> "Rijndael Te2 (0x63a5c663U) [32.le.1024]"
movzx   ebx, byte ptr [eax+15h]
mov     [ebp+var_20], edi
mov     edi, [eax+18h]
and     edi, esi
mov     edi, ds:dword_45BE50[edi*4] ; <$ignsrch> "Rijndael Te3 (0x6363a5c6U) [32.le.1024]"
xor     edi, ds:dword_45BA50[ebx*4] ; <$ignsrch> "Rijndael Te2 (0x63a5c663U) [32.le.1024]"
movzx   ebx, byte ptr [eax+0Eh]
xor     edi, ds:dword_45B650[ebx*4] ; <$ignsrch> "Rijndael Te1 (0xa5c66363U) [32.le.1024]"
movzx   ebx, byte ptr [eax+0Bh]
xor     edi, ds:dword_45B250[ebx*4] ; <$ignsrch> "Rijndael Te0 (0xc66363a5U) [32.le.1024]"
movzx   ebx, byte ptr [eax+12h]
mov     [ebp+var_1C], edi
mov     edi, [eax+1Ch]
and     edi, esi
mov     edi, ds:dword_45BE50[edi*4] ; <$ignsrch> "Rijndael Te3 (0x6363a5c6U) [32.le.1024]"
xor     edi, ds:dword_45B650[ebx*4] ; <$ignsrch> "Rijndael Te1 (0xa5c66363U) [32.le.1024]"
movzx   ebx, byte ptr [eax+0Fh]
xor     edi, ds:dword_45B250[ebx*4] ; <$ignsrch> "Rijndael Te0 (0xc66363a5U) [32.le.1024]"
0041BF71: sub_41BF71
```

FIND SPECIAL X86 INSTRUCTIONS



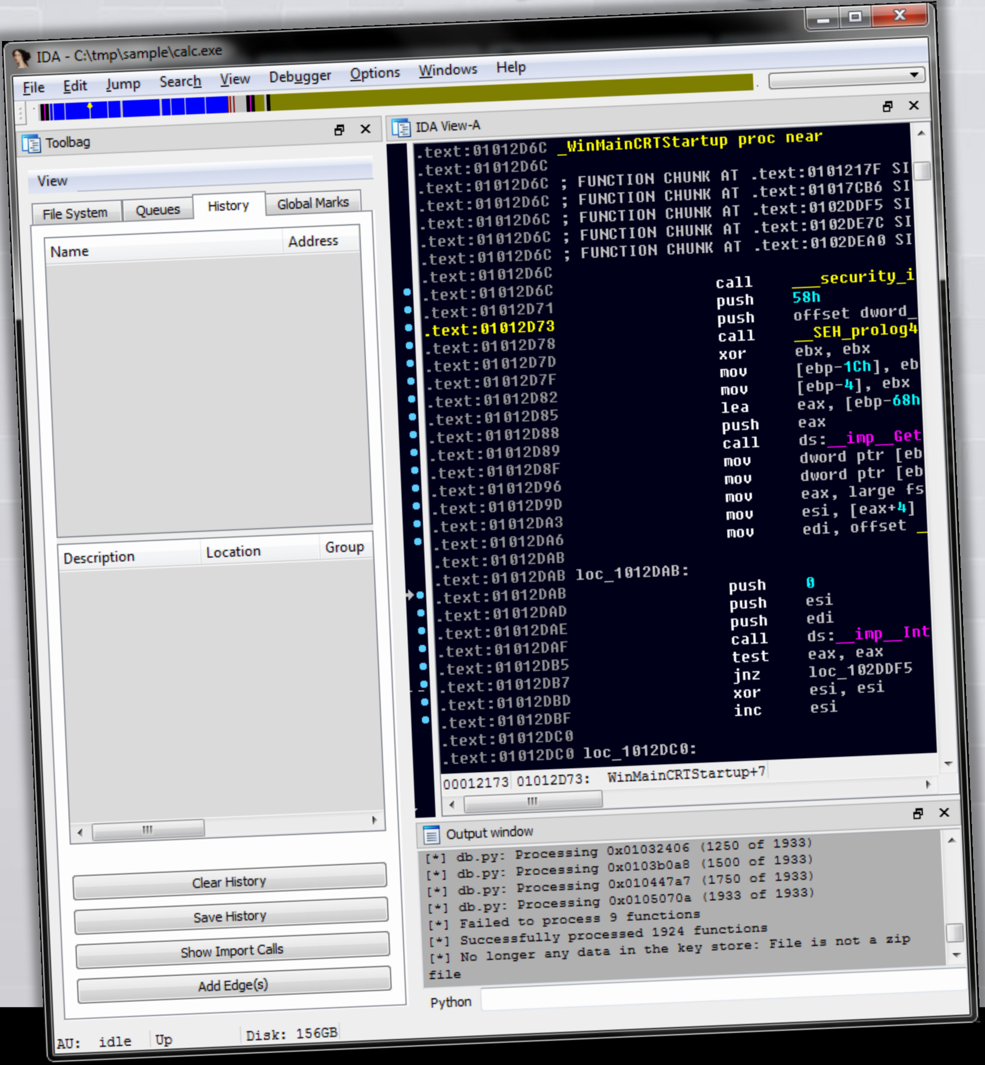
```
specialDict = dict()
special_instructions = ["xchg", "in", "sidt", "sgdt", "sldt", "smsw", "rdtsc"]

print "[Debug] Looking for special x86 instructions..."

# Inspect the whole binary
for func_addr in Functions():
    for ins in FuncItems(func_addr):
        disasm = GetDisasm(ins)
        mnem_ins = GetMnem(ins)
        for spec_ins in special_instructions:
            if mnem_ins == spec_ins:
                # found funky instruction
                specialDict[ins] = disasm
                SetColor(ins, CIC_ITEM, 0xff8800)
            elif 'fs:' in GetOpnd(ins, 0) or 'fs:' in GetOpnd(ins, 1):
                # found reference to TEB/PEB
                specialDict[ins] = disasm
                SetColor(ins, CIC_ITEM, 0x0088ff)
            else:
                pass
```

LET'S DO IT!!!

M.I.L.F. PLUGIN

IDA Pro Plugin

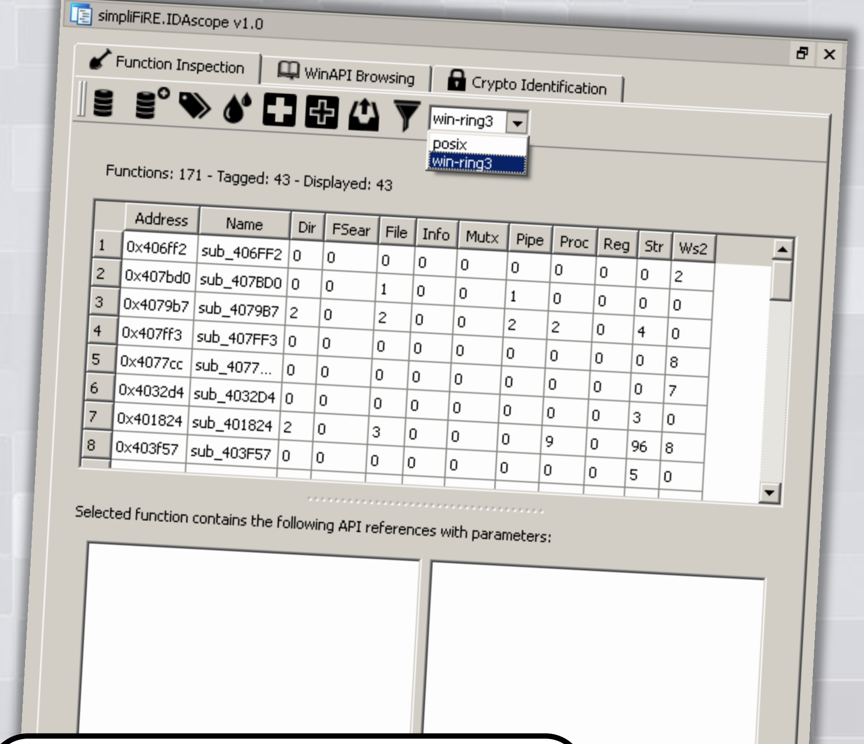
SERIOUS PLUGINS


IDA - C:\tmp\sample\calc.exe
File Edit Jump Search View Debugger Options Windows Help
Toolbag
View
File System Queues History Global Marks
Name Address
Description Location Group
Clear History
Save History
Show Import Calls
Add Edge(s)
IDA View-A
Output window
Python

```
.text:01012D6C _WinMainCRTStartup proc near
.text:01012D6C
.text:01012D6C ; FUNCTION CHUNK AT .text:0101217F SI
.text:01012D6C ; FUNCTION CHUNK AT .text:01017CB6 SI
.text:01012D6C ; FUNCTION CHUNK AT .text:0102DDF5 SI
.text:01012D6C ; FUNCTION CHUNK AT .text:0102DE7C SI
.text:01012D6C ; FUNCTION CHUNK AT .text:0102DEA0 SI
.text:01012D6C
.text:01012D6C                  call    ___security_i
.text:01012D71                  push    58h
.text:01012D73                  push    offset dword_
.text:01012D78                  call    __SEH_prolog4
.text:01012D7D                  xor     ebx, ebx
.text:01012D7F                  mov     [ebp-1Ch], eb
.text:01012D82                  mov     [ebp-4], ebx
.text:01012D85                  lea     eax, [ebp-68h
.text:01012D88                  push    eax
.text:01012D89                  call    ds:__imp__Get
.text:01012D8F                  mov     dword ptr [eb
.text:01012D96                  mov     dword ptr [eb
.text:01012D9D                  mov     eax, large fs
.text:01012DA3                  mov     esi, [eax+4]
.text:01012DA6                  mov     edi, offset _
.text:01012DAB
.text:01012DAB loc_1012DAB:
.text:01012DAB                  push    0
.text:01012DAD                  push    esi
.text:01012DAE                  push    edi
.text:01012DAF                  call    ds:__imp__Int
.text:01012DB5                  test    eax, eax
.text:01012DB7                  jnz     loc_102DDF5
.text:01012DBD                  xor     esi, esi
.text:01012DBF                  inc     esi
.text:01012DC0
.text:01012DC0 loc_1012DC0:
00012173 01012D73: WinMainCRTStartup+7
```

```
[*] db.py: Processing 0x01032406 (1250 of 1933)
[*] db.py: Processing 0x0103b0a8 (1500 of 1933)
[*] db.py: Processing 0x010447a7 (1750 of 1933)
[*] db.py: Processing 0x0105070a (1933 of 1933)
[*] Failed to process 9 functions
[*] Successfully processed 1924 functions
[*] No longer any data in the key store: File is not a zip
file
```

```
AU:  idle   Up      Disk: 156GB
```



Functions: 171 - Tagged: 43 - Displayed: 43

	Address	Name	Dir	FSear	File	Info	Mutx	Pipe	Proc	Reg	Str	Ws2
1	0x406ff2	sub_406FF2	0	0	0	0	0	0	0	0	0	2
2	0x407bd0	sub_407BD0	0	0	1	0	0	1	0	0	0	0
3	0x4079b7	sub_4079B7	2	0	2	0	0	2	2	0	4	0
4	0x407ff3	sub_407FF3	0	0	0	0	0	0	0	0	0	8
5	0x4077cc	sub_4077...	0	0	0	0	0	0	0	0	0	7
6	0x4032d4	sub_4032D4	0	0	0	0	0	0	0	0	3	0
7	0x401824	sub_401824	2	0	3	0	0	0	9	0	96	8
8	0x403f57	sub_403F57	0	0	0	0	0	0	0	0	5	0

Selected function contains the following API references with parameters:

IDA Pro Plugins

HOW'S EVERYONE DOING?

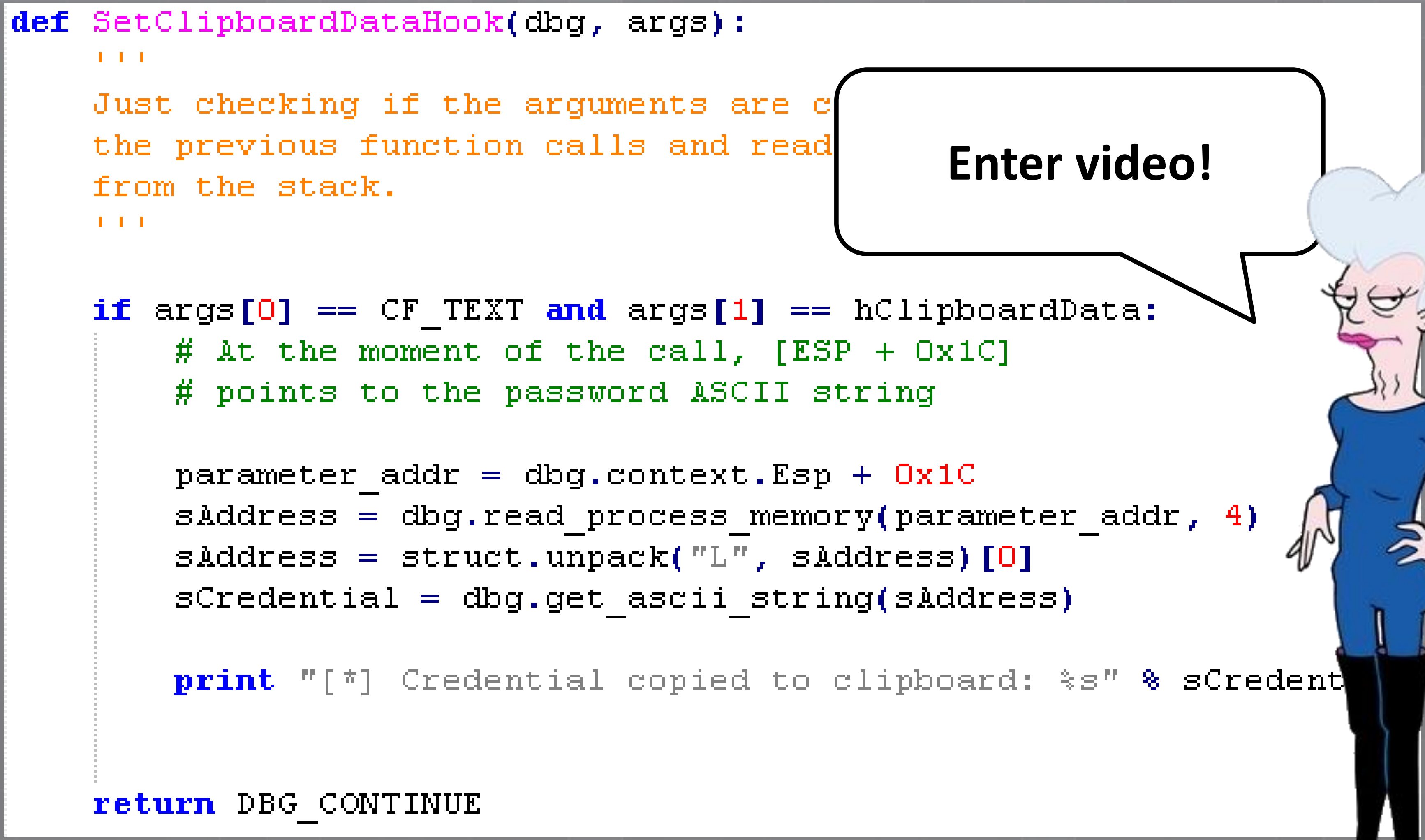
WINAPPDBG

KEEPASSADA

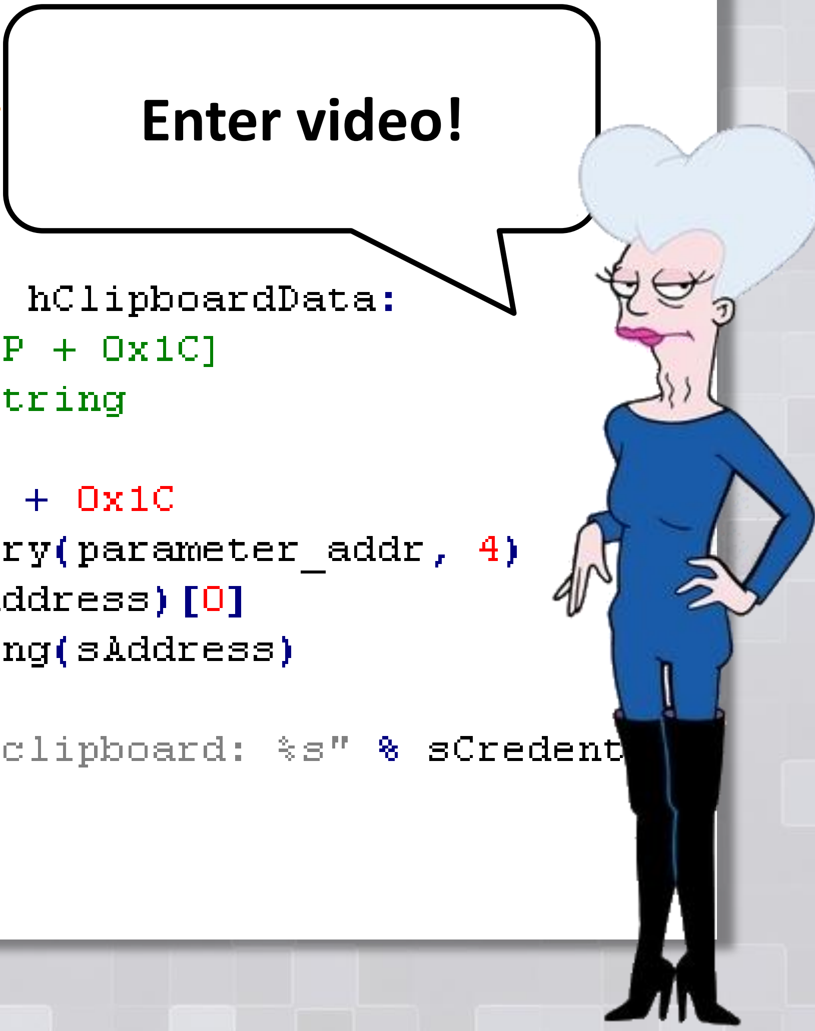
Copying data to the clipboard:

- OpenClipboard()
- EmptyClipboard()
- **hClipboardData** = *GlobalAlloc*()   // Get RetValue
- pchData = (char*)GlobalLock(hClipboardData)
- strcpy(pchData, LPCSTR(strData))
- GlobalUnlock(hClipboardData)
- *SetClipboardData*(CF_TEXT, **hClipboardData**)   // Hook this
- CloseClipboard()

KEEPASSADA



```
def SetClipboardDataHook(dbg, args):
    '''
    Just checking if the arguments are c
    the previous function calls and read
    from the stack.
    '''

    if args[0] == CF_TEXT and args[1] == hClipboardData:
        # At the moment of the call, [ESP + 0x1C]
        # points to the password ASCII string

        parameter_addr = dbg.context.Esp + 0x1C
        sAddress = dbg.read_process_memory(parameter_addr, 4)
        sAddress = struct.unpack("L", sAddress)[0]
        sCredential = dbg.get_ascii_string(sAddress)

        print "[*] Credential copied to clipboard: %s" % sCredent


    return DBG_CONTINUE
```



HOMEWORK!

WinAppDbg

- Win32 API wrapper
  - Fuck Yeah Python!™
- Written by Mario Vilas
  - This is Mario
  - Buy him a beer if you meet him

WINAPPDBG

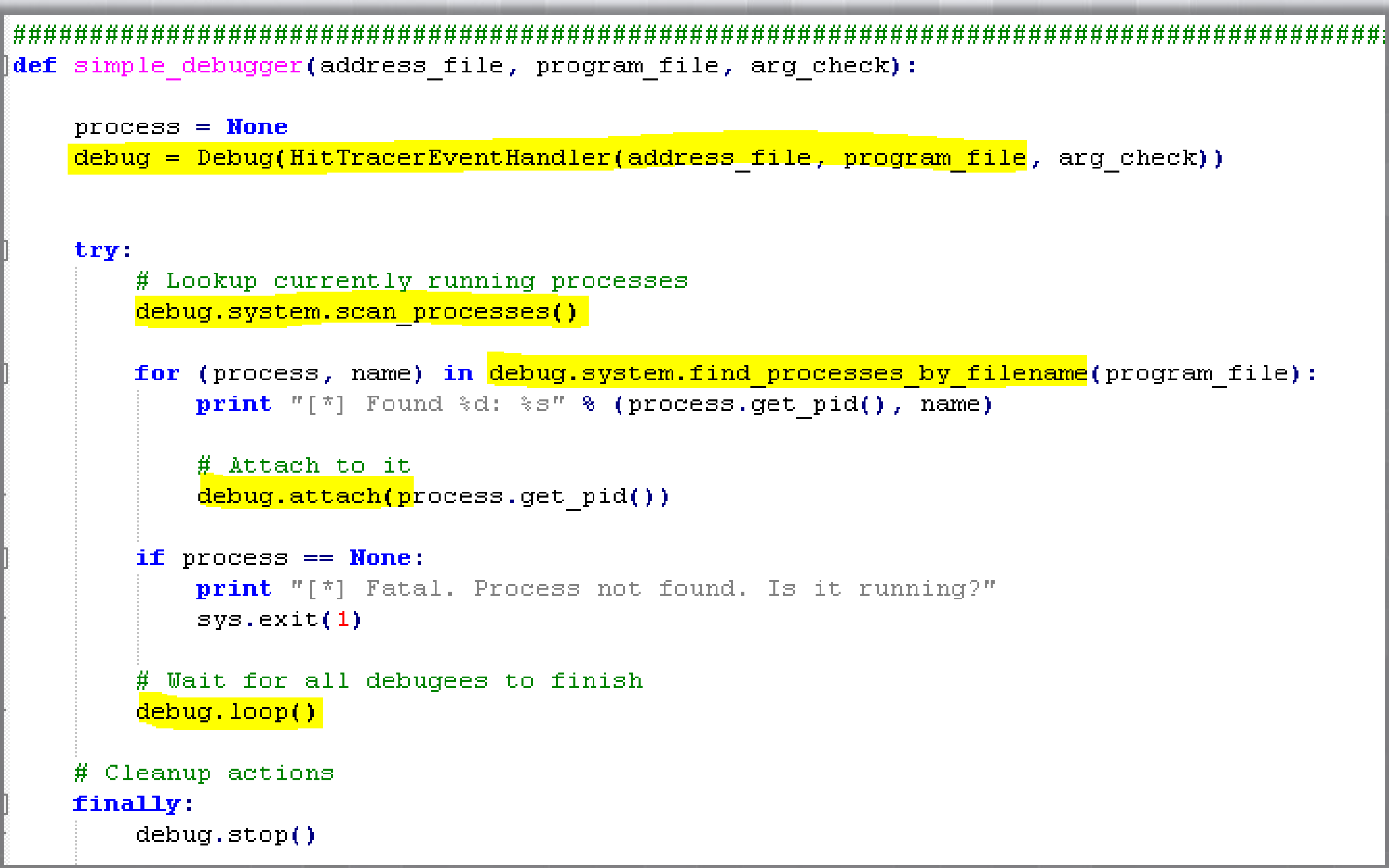
CASE STUDIES

- TRACER.PY
  - FUNCTIONS HIT?
- WTFDLL.PY
  - DLLS LOADED AT RUNTIME?
- TRACER_DOT.PY
  - YAY, GRAPHS!

WinAppDbg

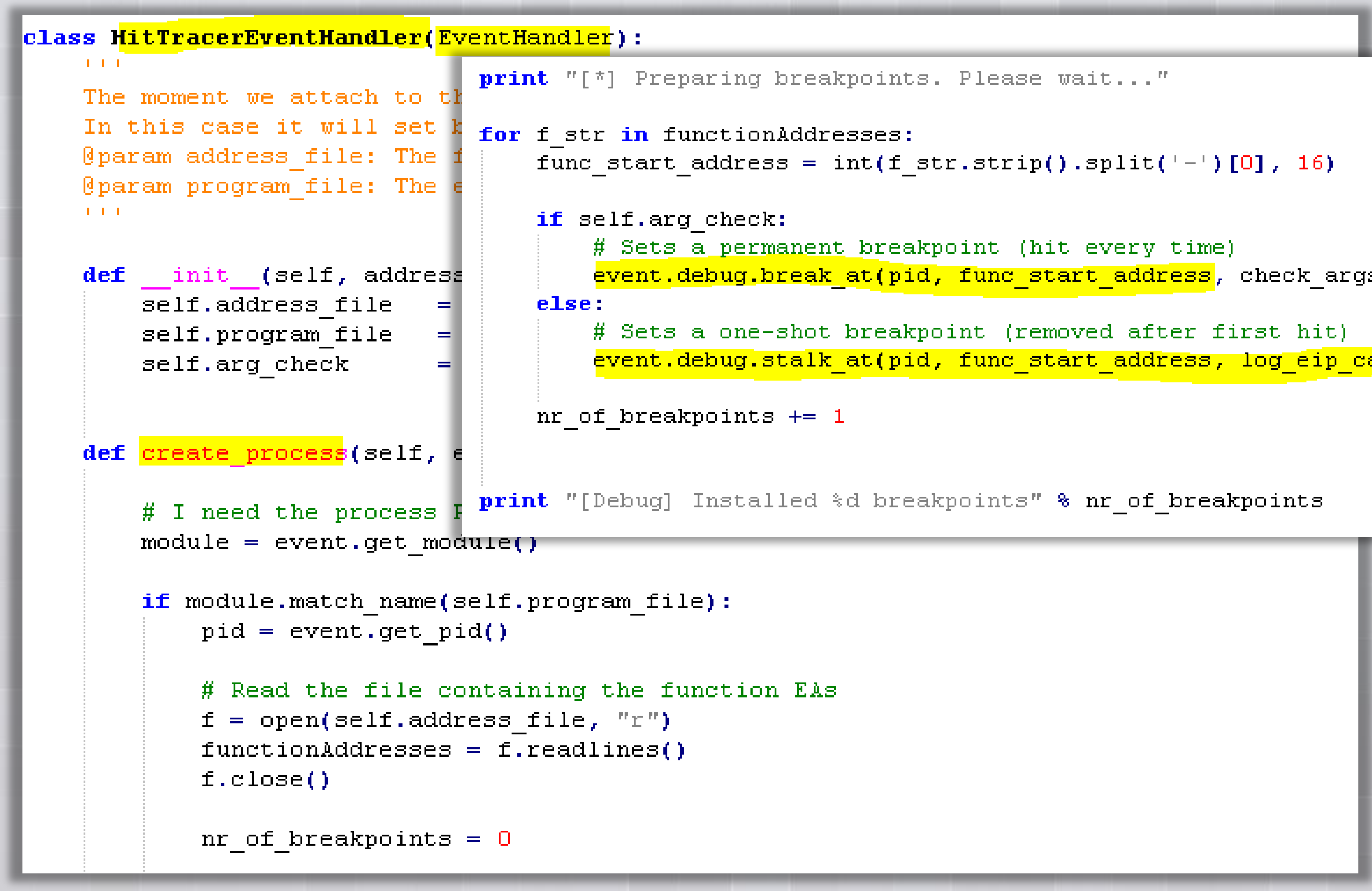
- Tracer.py
  - Performance problems (-1)
  - Slow (-1)
  - Need function list (IDA) (-1)
  - It is Python (+500)
  - PyDot FTW (+500)

TRACER & DERIVATIVES

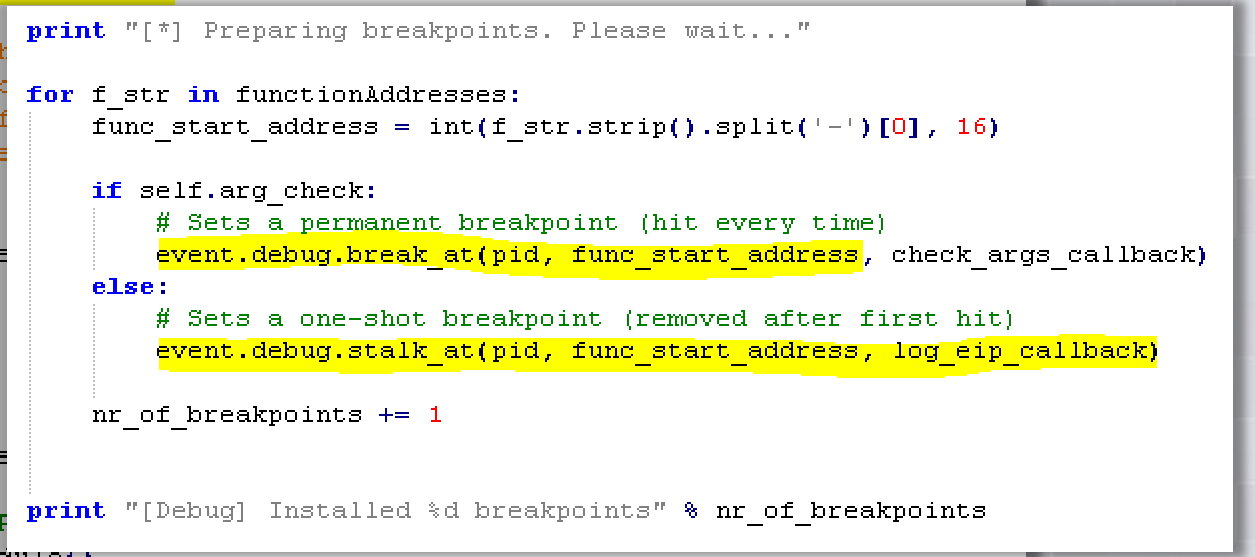


```
#########################################################################################
def simple_debugger(address_file, program_file, arg_check):

    process = None
    debug = Debug(HitTracerEventHandler(address_file, program_file, arg_check))


    try:
        # Lookup currently running processes
        debug.system.scan_processes()

        for (process, name) in debug.system.find_processes_by_filename(program_file):
            print "[*] Found %d: %s" % (process.get_pid(), name)

            # Attach to it
            debug.attach(process.get_pid())

        if process == None:
            print "[*] Fatal. Process not found. Is it running?"
            sys.exit(1)

        # Wait for all debugees to finish
        debug.loop()

    # Cleanup actions
    finally:
        debug.stop()
```

TRACER & DERIVATIVES



```
class HitTracerEventHandler(EventHandler):
    '''
    The moment we attach to th
    In this case it will set b
    @param address_file: The f
    @param program_file: The e
    '''

    def __init__(self, address
        self.address_file    =
        self.program_file    =
        self.arg_check       =


    def create_process(self, e

        # I need the process P
        module = event.get_module()

        if module.match_name(self.program_file):
            pid = event.get_pid()

            # Read the file containing the function EAs
            f = open(self.address_file, "r")
            functionAddresses = f.readlines()
            f.close()

            nr_of_breakpoints = 0
```



```
print "[*] Preparing breakpoints. Please wait..."

for f_str in functionAddresses:
    func_start_address = int(f_str.strip().split('-')[0], 16)

    if self.arg_check:
        # Sets a permanent breakpoint (hit every time)
        event.debug.break_at(pid, func_start_address, check_args_callback)
    else:
        # Sets a one-shot breakpoint (removed after first hit)
        event.debug.stalk_at(pid, func_start_address, log_eip_callback)

    nr_of_breakpoints += 1


print "[Debug] Installed %d breakpoints" % nr_of_breakpoints
```

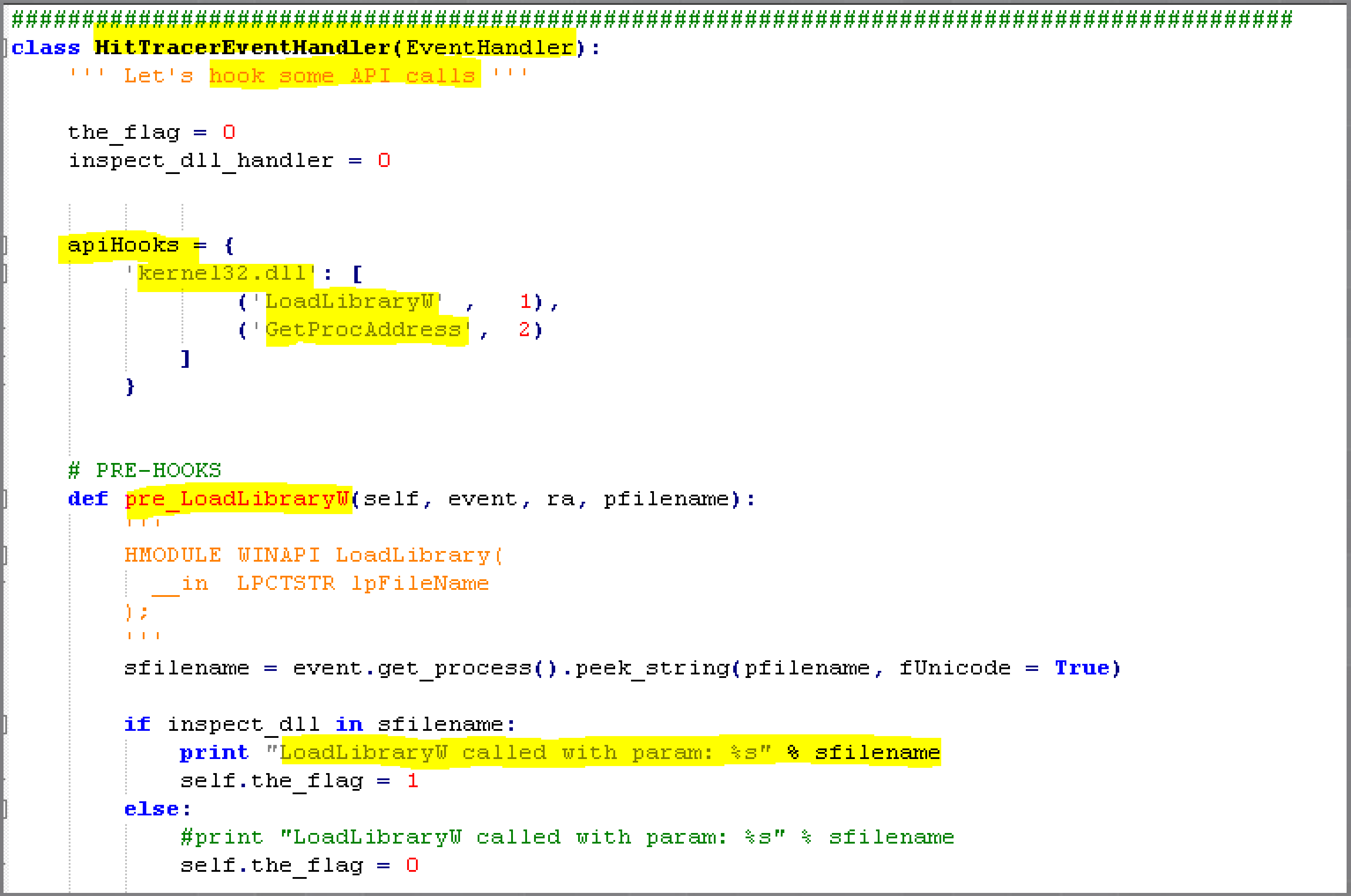
LET'S DO IT!!!

Question

Can you imagine **why in hell** could this be **useful** to anybody?

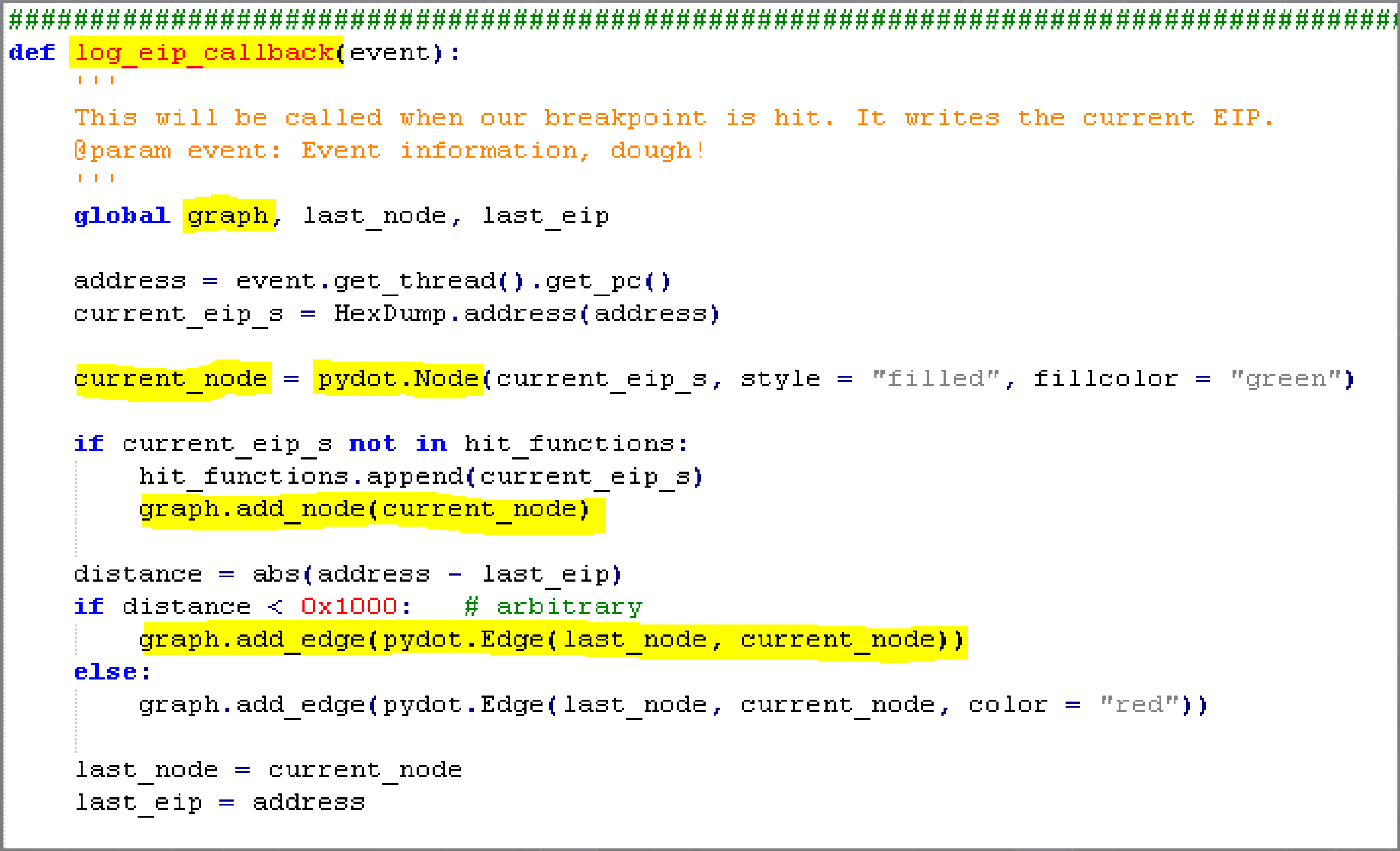
- Facebook CTF
- Fuzzing!?!?!
- Impress chicks
- A lot more

WATCHING DLLS LOAD



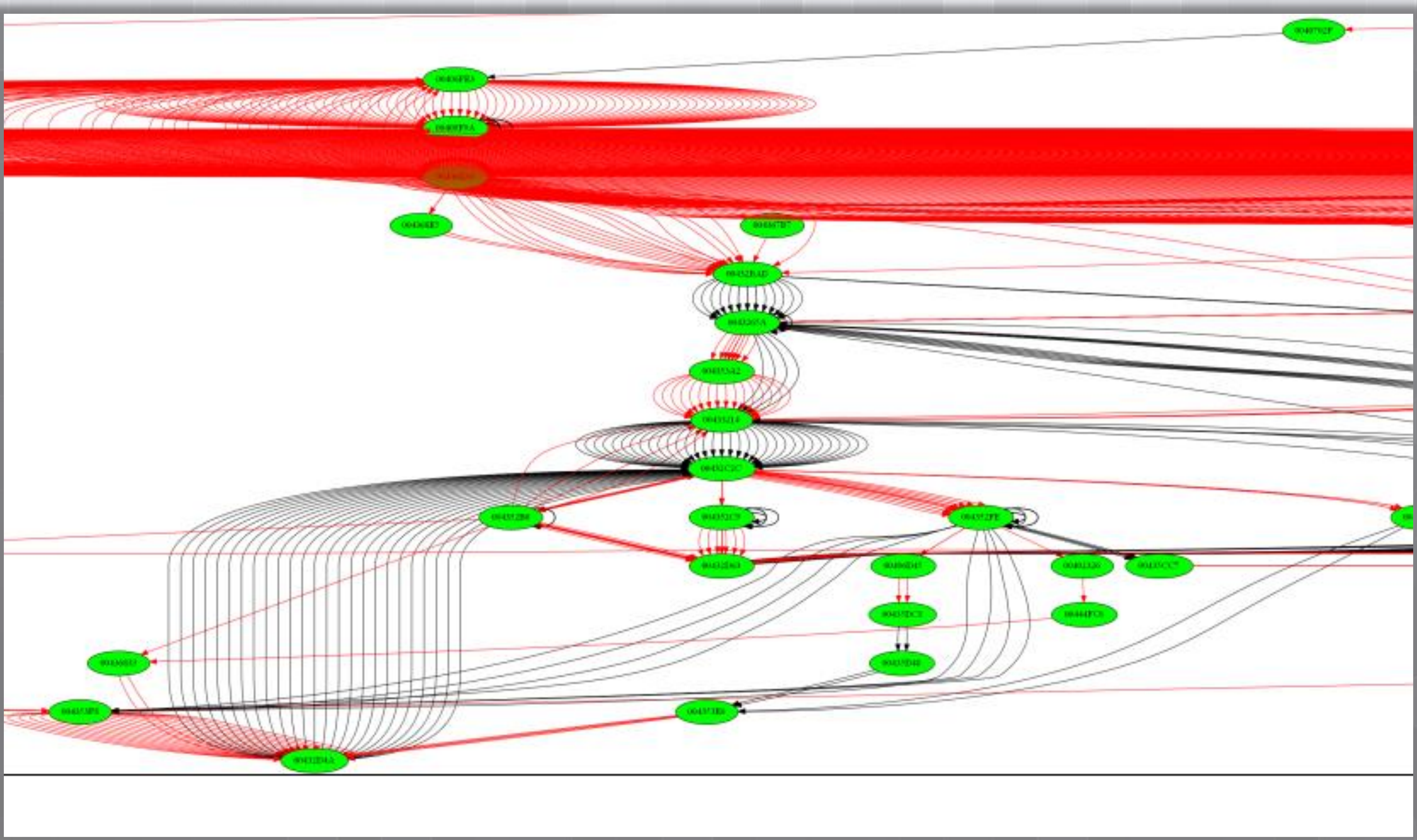
```
############################################################################################
class HitTracerEventHandler(EventHandler):
    ''' Let's hook some API calls '''

    the_flag = 0
    inspect_dll_handler = 0


    apiHooks = {
        'kernel32.dll' : [
                ('LoadLibraryW'  ,   1),
                ('GetProcAddress',   2)
            ]
        }


    # PRE-HOOKS
    def pre_LoadLibraryW(self, event, ra, pfilename):
        '''
        HMODULE WINAPI LoadLibrary(
          __in  LPCTSTR lpFileName
        );
        '''
        sfilename = event.get_process().peek_string(pfilename, fUnicode = True)

        if inspect_dll in sfilename:
            print "LoadLibraryW called with param: %s" % sfilename
            self.the_flag = 1
        else:
            #print "LoadLibraryW called with param: %s" % sfilename
            self.the_flag = 0
```

TRACER & PYDOT



```python
#####################################################################################
def log_eip_callback(event):
    '''
    This will be called when our breakpoint is hit. It writes the current EIP.
    @param event: Event information, dough!
    '''
    global graph, last_node, last_eip

    address = event.get_thread().get_pc()
    current_eip_s = HexDump.address(address)

    current_node = pydot.Node(current_eip_s, style = "filled", fillcolor = "green")

    if current_eip_s not in hit_functions:
        hit_functions.append(current_eip_s)
        graph.add_node(current_node)

    distance = abs(address - last_eip)
    if distance < 0x1000:    # arbitrary
        graph.add_edge(pydot.Edge(last_node, current_node))
    else:
        graph.add_edge(pydot.Edge(last_node, current_node, color = "red"))

    last_node = current_node
    last_eip = address
```

Tracer & PyDot



LET'S DO IT!!!

INTEL PIN

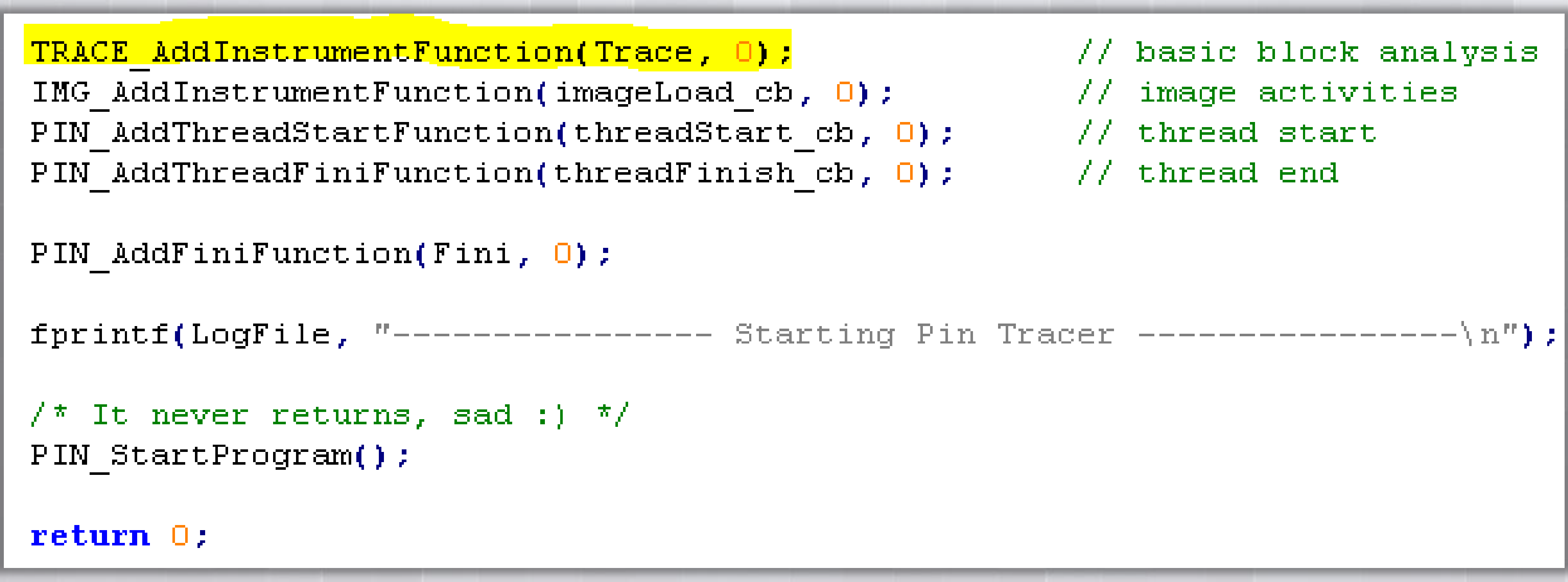
Intel PIN

Case Studies

- A more Efficient tracer
- Detect buffer overflows
  - EIP outside text section

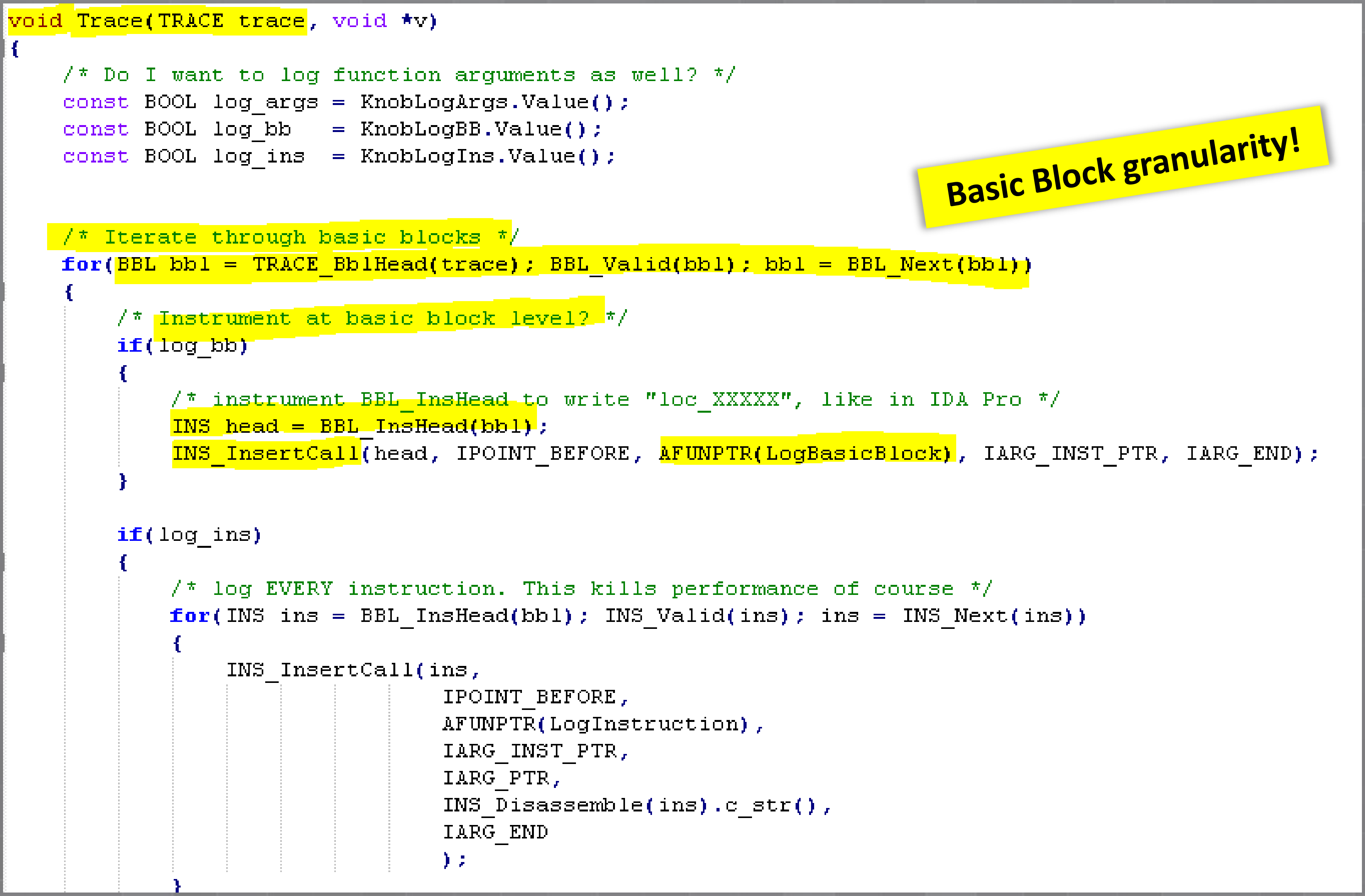
INTEL PIN

- A MORE EFFICIENT TRACER



```
TRACE_AddInstrumentFunction(Trace, 0);                // basic block analysis
IMG_AddInstrumentFunction(imageLoad_cb, 0);           // image activities
PIN_AddThreadStartFunction(threadStart_cb, 0);        // thread start
PIN_AddThreadFiniFunction(threadFinish_cb, 0);        // thread end

PIN_AddFiniFunction(Fini, 0);

fprintf(LogFile, "---------------- Starting Pin Tracer ----------------\n");

/* It never returns, sad :) */
PIN_StartProgram();

return 0;
```

INTEL PIN



```
void Trace(TRACE trace, void *v)
{
    /* Do I want to log function arguments as well? */
    const BOOL log_args = KnobLogArgs.Value();
    const BOOL log_bb   = KnobLogBB.Value();
    const BOOL log_ins  = KnobLogIns.Value();


    /* Iterate through basic blocks */
    for(BBL bbl = TRACE_BblHead(trace); BBL_Valid(bbl); bbl = BBL_Next(bbl))
    {
        /* Instrument at basic block level? */
        if(log_bb)
        {
            /* instrument BBL_InsHead to write "loc_XXXXX", like in IDA Pro */
            INS head = BBL_InsHead(bbl);
            INS_InsertCall(head, IPOINT_BEFORE, AFUNPTR(LogBasicBlock), IARG_INST_PTR, IARG_END);
        }

        if(log_ins)
        {
            /* log EVERY instruction. This kills performance of course */
            for(INS ins = BBL_InsHead(bbl); INS_Valid(ins); ins = INS_Next(ins))
            {
                INS_InsertCall(ins,
                               IPOINT_BEFORE,
                               AFUNPTR(LogInstruction),
                               IARG_INST_PTR,
                               IARG_PTR,
                               INS_Disassemble(ins).c_str(),
                               IARG_END
                               );
            }
```

Basic Block granularity!

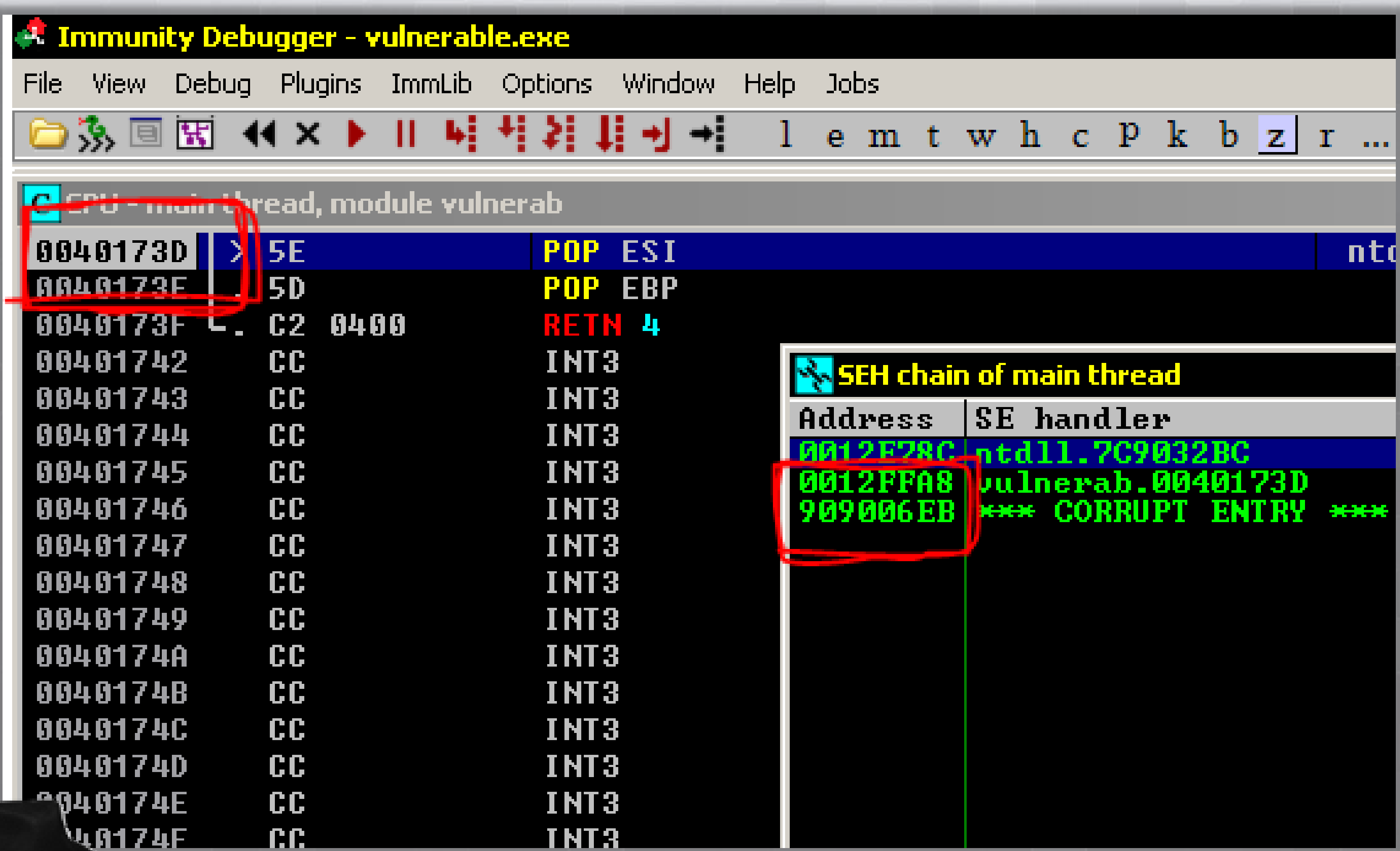
INTEL PIN

LET'S HUNT A BUFFER OVERFLOW!

DOES EVERYBODY KNOW THE SEH OVERWRITE EXPLOITING TECHNIQUE?

(I READ THE IDEA ORIGINALLY AT HTTP://SCRAMMED.BLOGSPOT.COM)

INTEL PIN



Immunity Debugger - vulnerable.exe

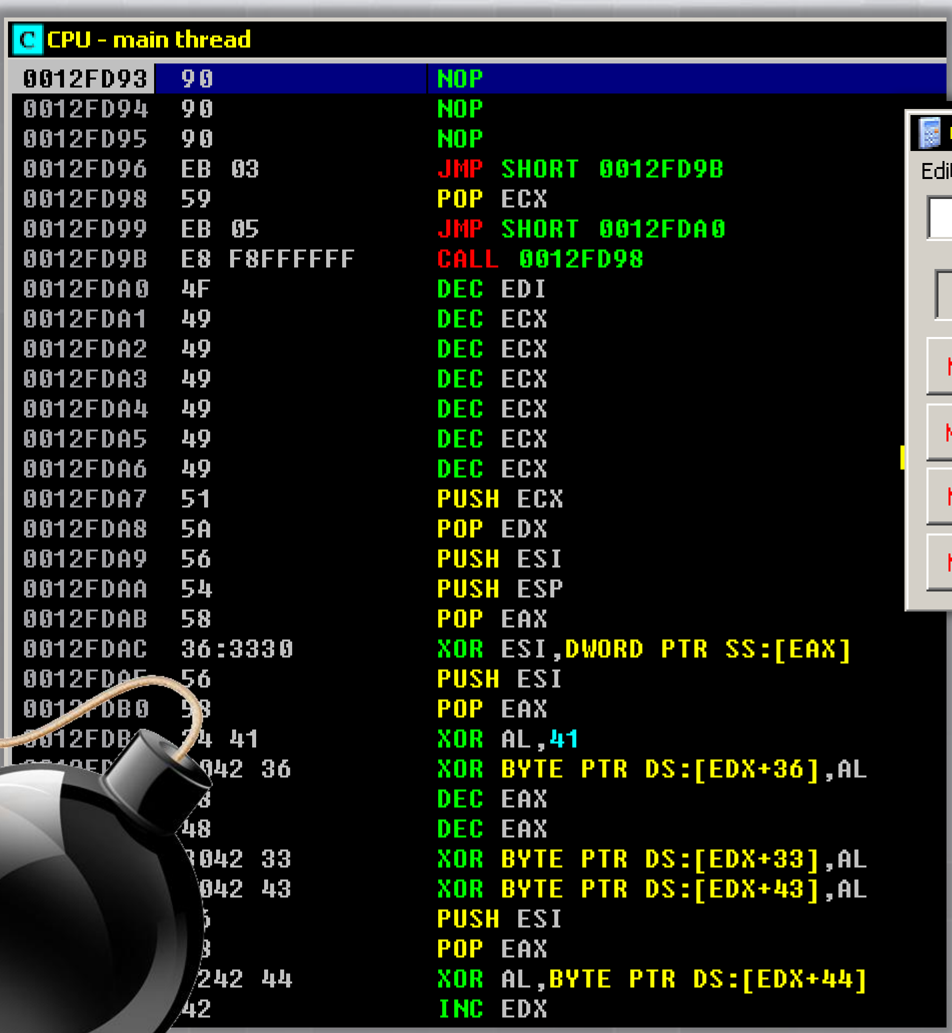
File
View
Debug
Plugins
ImmLib
Options
Window
Help
Jobs
l e m t w h c P k b z r ...
CPU - main thread, module vulnerab

```
0040173D  > 5E        POP ESI
0040173E  . 5D        POP EBP
0040173F  L. C2 0400  RETN 4
00401742    CC        INT3
00401743    CC        INT3
00401744    CC        INT3
00401745    CC        INT3
00401746    CC        INT3
00401747    CC        INT3
00401748    CC        INT3
00401749    CC        INT3
0040174A    CC        INT3
0040174B    CC        INT3
0040174C    CC        INT3
0040174D    CC        INT3
0040174E    CC        INT3
0040174F    CC        INT3
```

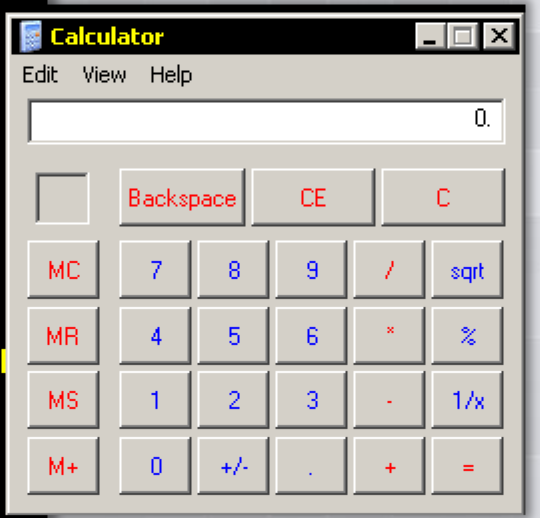
SEH chain of main thread

```
Address   SE handler
0012F78C  ntdll.7C9032BC
0012FFA8  vulnerab.0040173D
909006EB  *** CORRUPT ENTRY ***
```

INTEL PIN

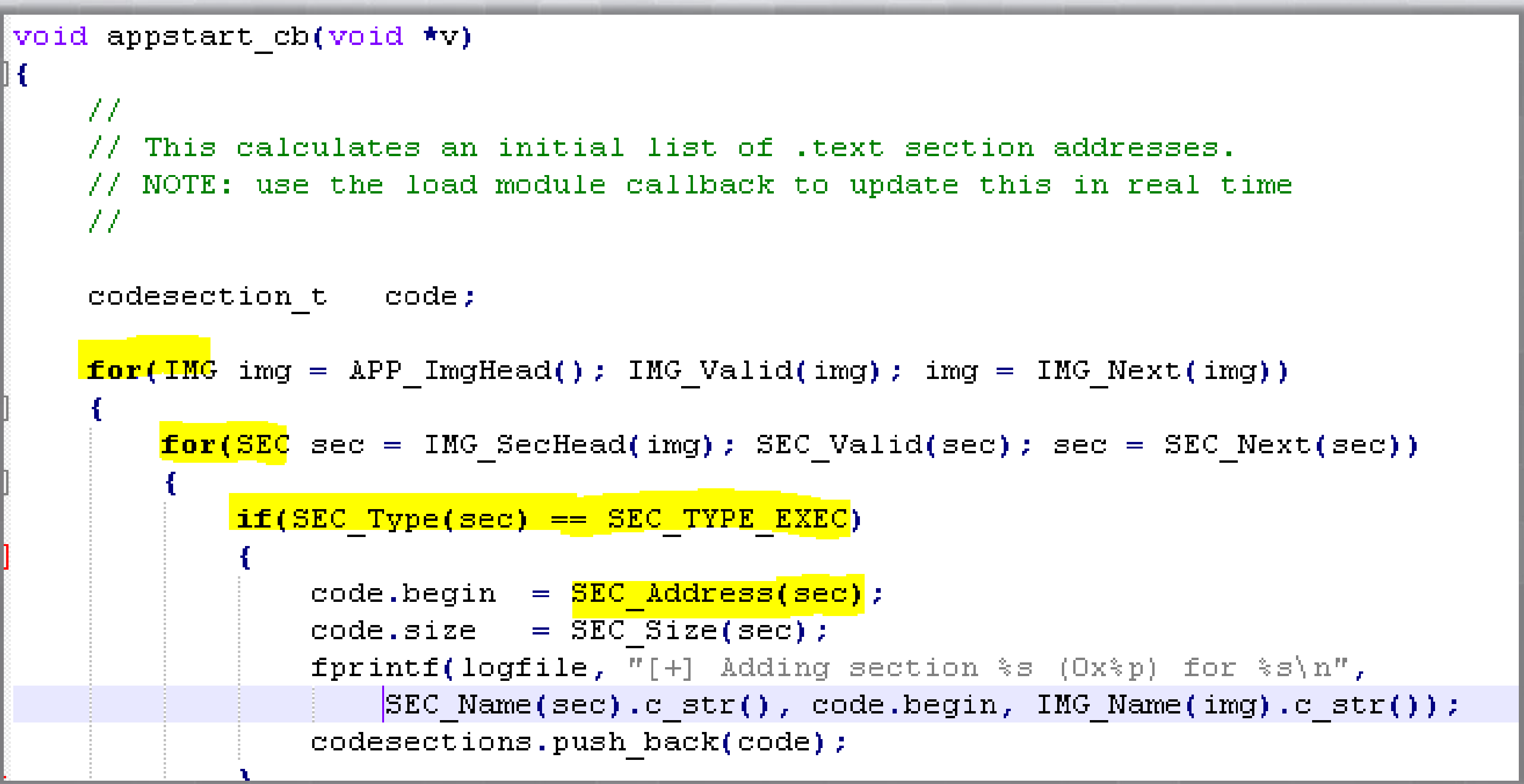


```
0012FD93  90          NOP
0012FD94  90          NOP
0012FD95  90          NOP
0012FD96  EB 03       JMP SHORT 0012FD9B
0012FD98  59          POP ECX
0012FD99  EB 05       JMP SHORT 0012FDA0
0012FD9B  E8 F8FFFFFF CALL 0012FD98
0012FDA0  4F          DEC EDI
0012FDA1  49          DEC ECX
0012FDA2  49          DEC ECX
0012FDA3  49          DEC ECX
0012FDA4  49          DEC ECX
0012FDA5  49          DEC ECX
0012FDA6  49          DEC ECX
0012FDA7  51          PUSH ECX
0012FDA8  5A          POP EDX
0012FDA9  56          PUSH ESI
0012FDAA  54          PUSH ESP
0012FDAB  58          POP EAX
0012FDAC  36:3330     XOR ESI,DWORD PTR SS:[EAX]
0012FDAE  56          PUSH ESI
0012FDB0  58          POP EAX
0012FDB   4 41        XOR AL,41
           042 36     XOR BYTE PTR DS:[EDX+36],AL
           8          DEC EAX
           48         DEC EAX
           3042 33    XOR BYTE PTR DS:[EDX+33],AL
           042 43     XOR BYTE PTR DS:[EDX+43],AL
           5          PUSH ESI
           8          POP EAX
           242 44     XOR AL,BYTE PTR DS:[EDX+44]
           42         INC EDX
```



INTEL PIN

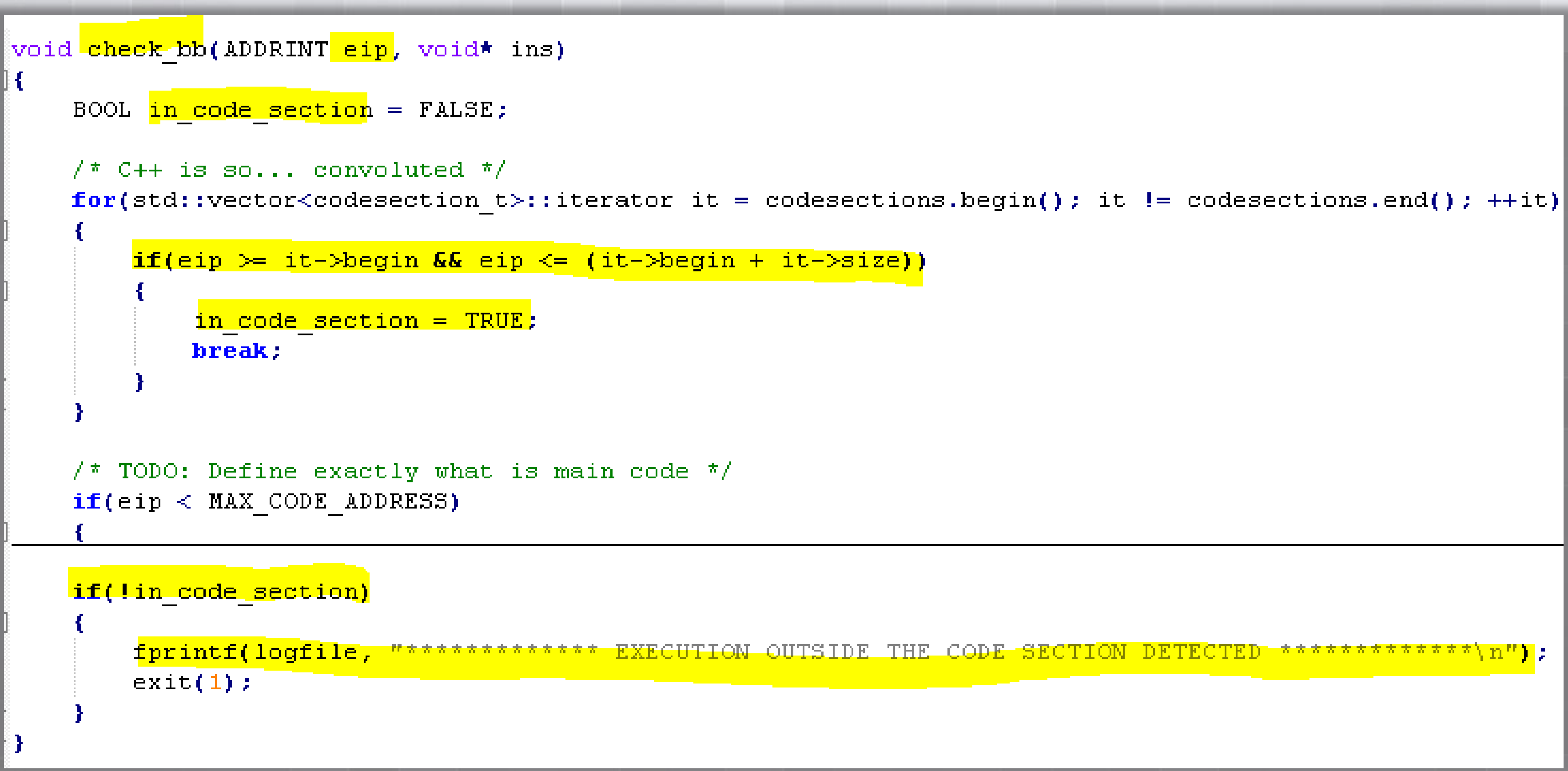
DETECT EIP OUTSIDE TEXT SECTION



```
void appstart_cb(void *v)
{
    //
    // This calculates an initial list of .text section addresses.
    // NOTE: use the load module callback to update this in real time
    //

    codesection_t   code;

    for(IMG img = APP_ImgHead(); IMG_Valid(img); img = IMG_Next(img))
    {
        for(SEC sec = IMG_SecHead(img); SEC_Valid(sec); sec = SEC_Next(sec))
        {
            if(SEC_Type(sec) == SEC_TYPE_EXEC)
            {
                code.begin  = SEC_Address(sec);
                code.size   = SEC_Size(sec);
                fprintf(logfile, "[+] Adding section %s (0x%p) for %s\n",
                        SEC_Name(sec).c_str(), code.begin, IMG_Name(img).c_str());
                codesections.push_back(code);
```

INTEL PIN

DETECT EIP OUTSIDE TEXT SECTION



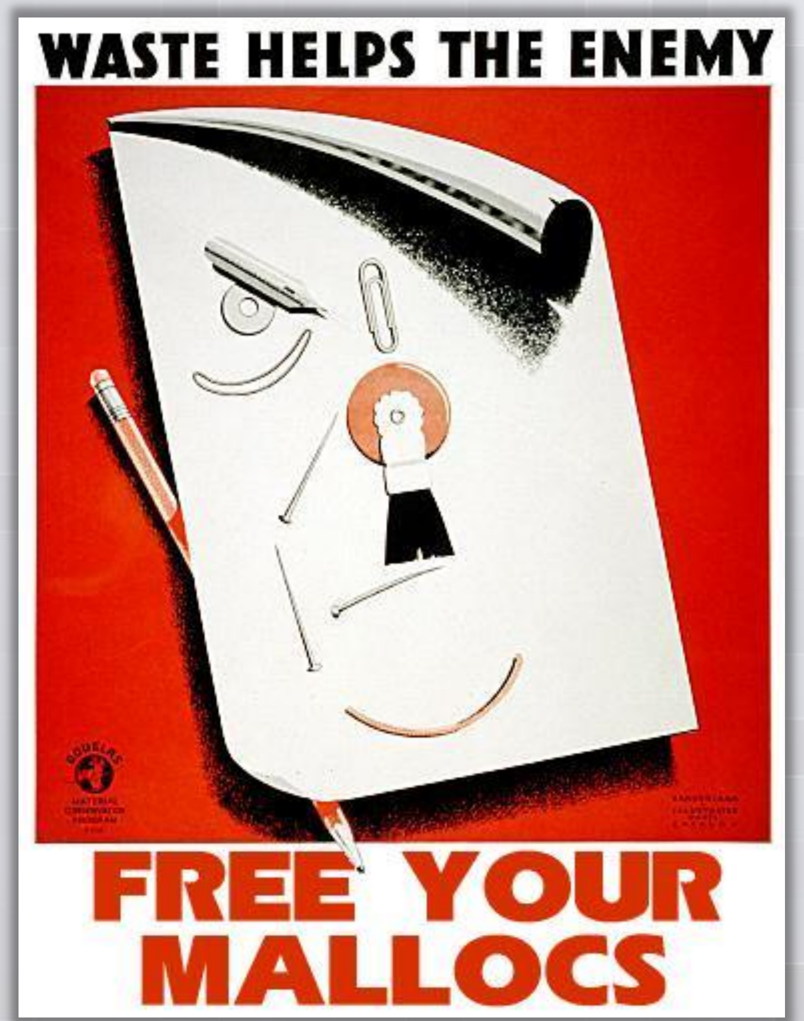
```
void check_bb(ADDRINT eip, void* ins)
{
    BOOL in_code_section = FALSE;

    /* C++ is so... convoluted */
    for(std::vector<codesection_t>::iterator it = codesections.begin(); it != codesections.end(); ++it)
    {
        if(eip >= it->begin && eip <= (it->begin + it->size))
        {
            in_code_section = TRUE;
            break;
        }
    }

    /* TODO: Define exactly what is main code */
    if(eip < MAX_CODE_ADDRESS)
    {

    if(!in_code_section)
    {
        fprintf(logfile, "************** EXECUTION OUTSIDE THE CODE SECTION DETECTED **************\n");
        exit(1);
    }
}
```

EXPERIMENTATION TIME

INTEL PIN

- VALGRIND-LIKE FOR WINDOWS
  - CHECK MEMORY ALLOCATIONS
  - DOUBLE FREE(S)

INTEL PIN


WASTE HELPS THE ENEMY
FREE YOUR MALLOCS

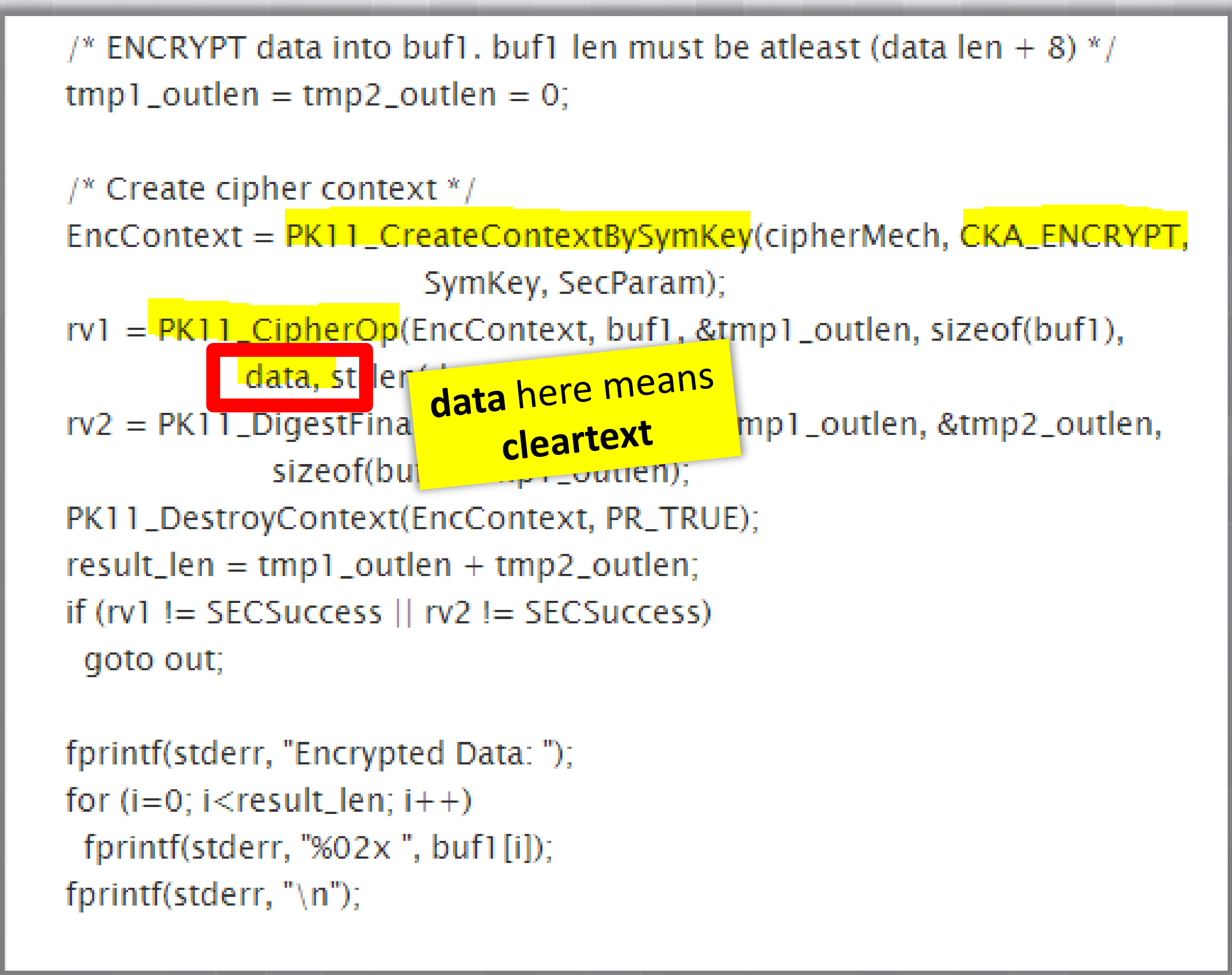
Y U NO FREE
AFTER MALLOC?

VDB / VTRACE

VDB / VTRACE

- YEP, IT IS PYTHON
- IT DOES PRETTY MUCH WHAT OTHERS DO
  - AND LINUX… AND ARM… AND PPC… ETC.
- LOOK, I'M SO COOL!
  - READABLE CODE
  - NO DOCUMENTATION

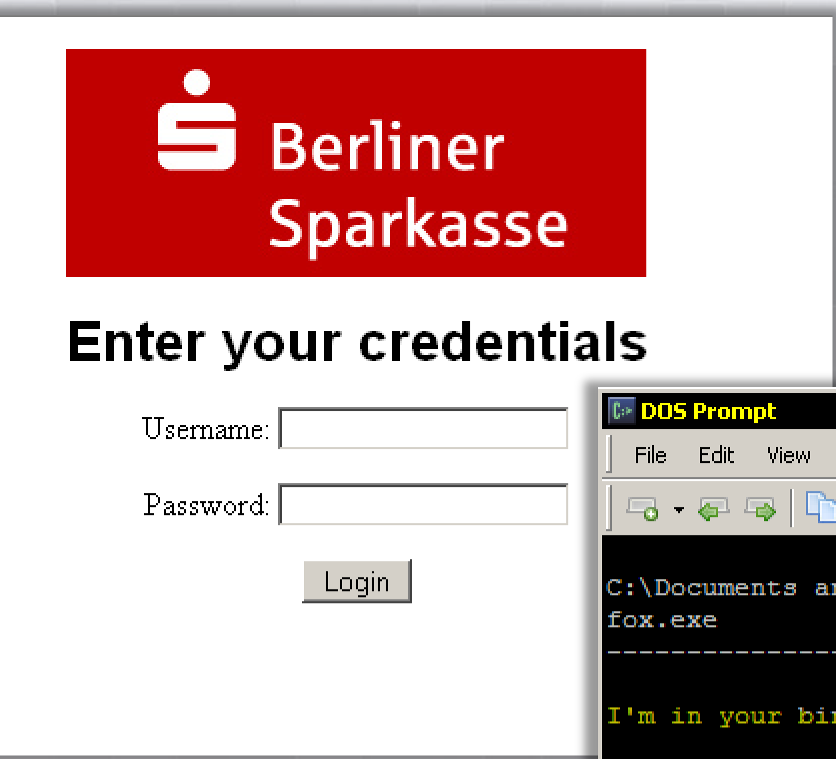
VDB / VTRACE



```
/* ENCRYPT data into buf1. buf1 len must be atleast (data len + 8) */
tmp1_outlen = tmp2_outlen = 0;

/* Create cipher context */
EncContext = PK11_CreateContextBySymKey(cipherMech, CKA_ENCRYPT,
                                        SymKey, SecParam);
rv1 = PK11_CipherOp(EncContext, buf1, &tmp1_outlen, sizeof(buf1),
          data, st len(
rv2 = PK11_DigestFina                        mp1_outlen, &tmp2_outlen,
                sizeof(bu                  p1_outlen);
PK11_DestroyContext(EncContext, PR_TRUE);
result_len = tmp1_outlen + tmp2_outlen;
if (rv1 != SECSuccess || rv2 != SECSuccess)
  goto out;

fprintf(stderr, "Encrypted Data: ");
for (i=0; i<result_len; i++)
  fprintf(stderr, "%02x ", buf1[i]);
fprintf(stderr, "\n");
```

data here means **cleartext**

VDB / VTRACE



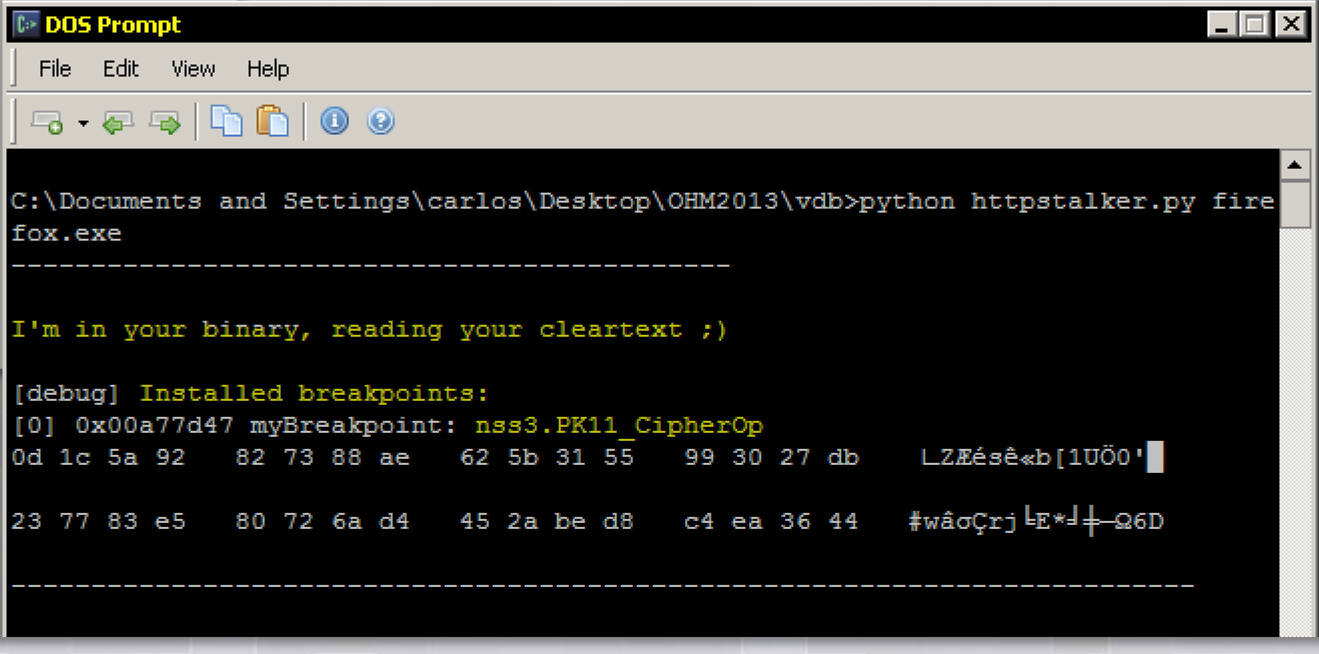
Berliner Sparkasse

Enter your credentials

Username:

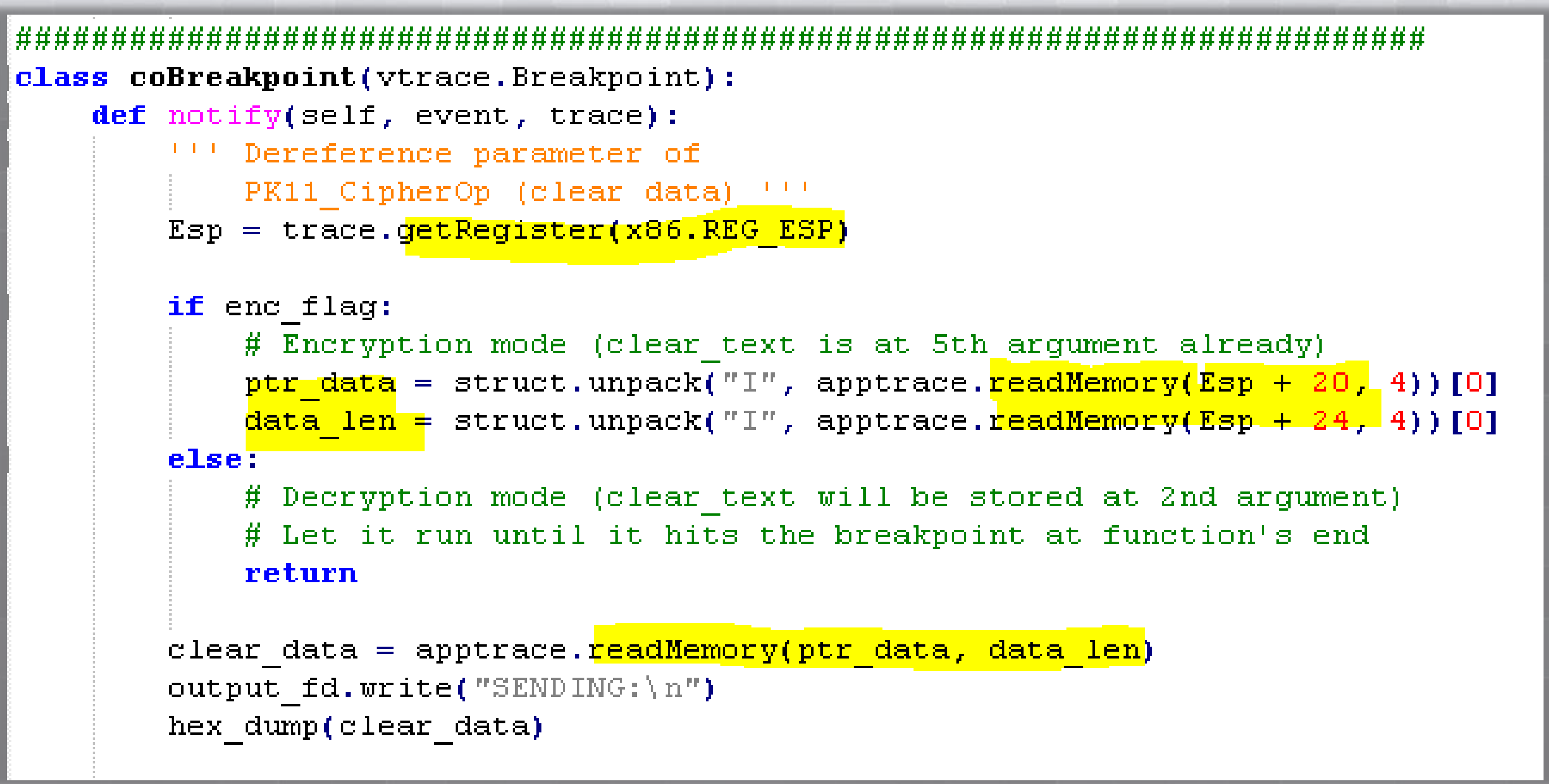
Password:

Login



```
C:\Documents and Settings\carlos\Desktop\OHM2013\vdb>python httpstalker.py fire
fox.exe
---------------------------------------------

I'm in your binary, reading your cleartext ;)

[debug] Installed breakpoints:
[0] 0x00a77d47 myBreakpoint: nss3.PK11_CipherOp
0d 1c 5a 92    82 73 88 ae    62 5b 31 55    99 30 27 db    └ZÆéSê«b[1UÖ0'█

23 77 83 e5    80 72 6a d4    45 2a be d8    c4 ea 36 44   #wâσÇrj╘E*╛╪─Ω6D

---------------------------------------------------------------------------
```

VDB / VTRACE



```
###########################################################################
class coBreakpoint(vtrace.Breakpoint):
    def notify(self, event, trace):
        ''' Dereference parameter of
            PK11_CipherOp (clear data) '''
        Esp = trace.getRegister(x86.REG_ESP)

        if enc_flag:
            # Encryption mode (clear_text is at 5th argument already)
            ptr_data = struct.unpack("I", apptrace.readMemory(Esp + 20, 4))[0]
            data_len = struct.unpack("I", apptrace.readMemory(Esp + 24, 4))[0]
        else:
            # Decryption mode (clear_text will be stored at 2nd argument)
            # Let it run until it hits the breakpoint at function's end
            return

        clear_data = apptrace.readMemory(ptr_data, data_len)
        output_fd.write("SENDING:\n")
        hex_dump(clear_data)
```

EXPERIMENTATION TIME

QUICK RECAP

- PYTHON BASED:
  - FAST PROTOTYPING BUT…
  - DAMN SLOW
- INTEL PIN
  - FAST AND INTELLIGENT BUT…
  - CONVOLUTED

Thanks for coming!

@m0n0sapiens

carlos.garcia@nruns.com