Improving Antivirus Accuracy with Hypervisor Assisted Analysis

Danny Quist
Offensive Computing, LLC

dquist@offensivecomputing.net
Twitter: ocomputing

Danny Quist

- Reverse engineer
  - Automated reverse engineering
  - Unpacker of strange malware
  - RE Training Course
- Founder Offensive Computing
  - Largest open collection of malware
  - Blog with research (when able)
- Ph.D. New Mexico Tech, 2010

Overview

- Complexities of reverse engineering
- Discussion of malware detection problem
- The commercial antivirus industry
- Hypervisors and Reverse Engineering
- Improving AV scanning results

Complexities of Reverse Engineering

- Most malware is compiled Intel x86 Assembly code

```
char pw[] = "\xdc\xc5\xdc\xca\xb0\xa3\xb0\xe9"
            "\xf5\xf1\xe2\xe3\xb0\xff\xf6\xb0"
            "\xfd\xe9\xb0\xfc\xf9\xf6\xf5\xb0"
            "\xff\xfe\xb0\xe4\xf8\xf5\xb0\xfc"
            "\xf9\xfe\xf5\xbe";

int main(int argc, char* argv[])
{
    char in[256] = {0};
    size_t i = 0;
    size_t inlen = 0;
    bool isgood = 1;

    printf("Enter your password: ");
    fflush(stdout);

    fgets(in, sizeof(in)-1, stdin);

    inlen = strlen(in);

    for (i = 0 ; i < inlen - 1; i++)
    {
        if (pw[i] != (char) (in[i] ^ 0x90))
        {
            isgood = 0;
            break;
        }
    }

    if (isgood)
        printf("Good password\n");
    else
        printf("Bad password\n");

    getchar();

    return 0;
}
```

C Code – 45 lines

Compiler

- Machine code is more complex
- Optimizations make analysis more difficult
- Total code size is 1,200 instructions
- 118 Relevant assembly instructions
- Much of machine code is compiler boiler plate

Reverse Engineering

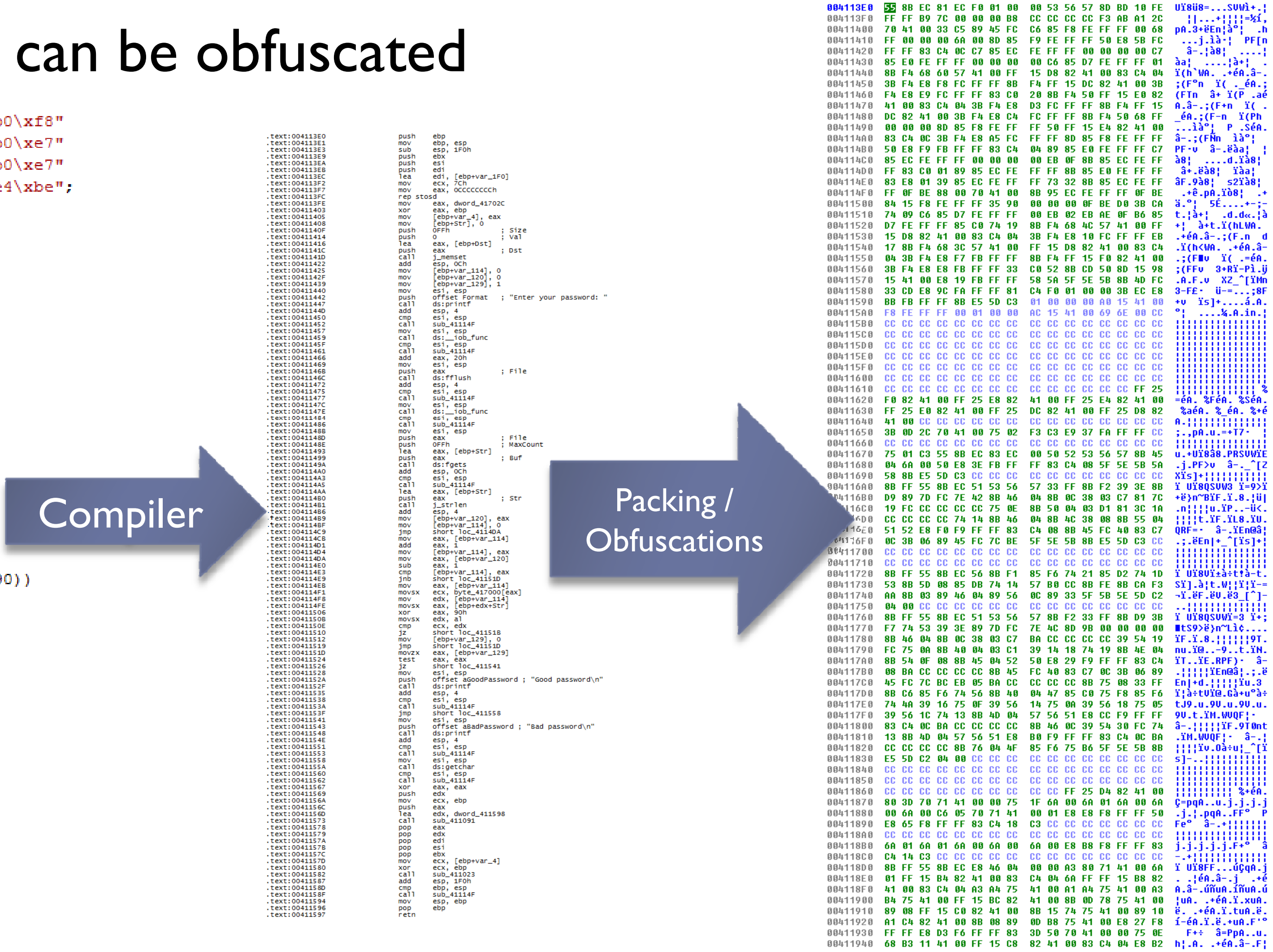
```
.text:004113E0            push    ebp
.text:004113E1            mov     ebp, esp
.text:004113E3            sub     esp, 1F0h
.text:004113E9            push    ebx
.text:004113EA            push    esi
.text:004113EB            push    edi
.text:004113EC            lea     edi, [ebp+var_1F0]
.text:004113F2            mov     ecx, 7Ch
.text:004113F7            mov     eax, 0CCCCCCCCh
.text:004113FC            rep stosd
.text:004113FE            mov     eax, dword_41702C
.text:00411403            xor     eax, ebp
.text:00411405            mov     [ebp+var_4], eax
.text:00411408            mov     [ebp+Str], 0
.text:0041140F            push    0FFh            ; Size
.text:00411414            push    0               ; Val
.text:00411416            lea     eax, [ebp+Dst]
.text:0041141C            push    eax             ; Dst
.text:0041141D            call    j_memset
.text:00411422            add     esp, 0Ch
.text:00411425            mov     [ebp+var_114], 0
.text:0041142F            mov     [ebp+var_120], 0
.text:00411439            mov     [ebp+var_129], 1
.text:00411440            mov     esi, esp
.text:00411442            push    offset Format   ; "Enter your password: "
.text:00411447            call    ds:printf
.text:0041144D            add     esp, 4
.text:00411450            cmp     esi, esp
.text:00411452            call    sub_41114F
.text:00411457            mov     esi, esp
.text:00411459            call    ds:__iob_func
.text:0041145F            cmp     esi, esp
.text:00411461            call    sub_41114F
.text:00411466            add     esp, 20h
.text:00411469            mov     esi, esp
.text:0041146B            push    eax             ; File
.text:0041146C            call    ds:fflush
.text:00411472            add     esp, 4
.text:00411475            cmp     esi, esp
.text:00411477            call    sub_41114F
.text:0041147C            mov     esi, esp
.text:0041147E            call    ds:__iob_func
.text:00411484            cmp     esi, esp
.text:00411486            call    sub_41114F
.text:00411488            mov     esi, esp
.text:0041148D            push    eax             ; File
.text:0041148F            push    0FFh            ; MaxCount
.text:00411493            lea     eax, [ebp+Str]
.text:00411499            push    eax             ; Buf
.text:0041149A            call    ds:fgets
.text:004114A0            add     esp, 0Ch
.text:004114A3            cmp     esi, esp
.text:004114A5            call    sub_41114F
.text:004114AA            lea     eax, [ebp+Str]
.text:004114B0            push    eax             ; Str
.text:004114B1            call    j_strlen
.text:004114B6            add     esp, 4
.text:004114B9            mov     [ebp+var_120], eax
.text:004114BF            mov     [ebp+var_114], 0
.text:004114C9            jmp     short loc_4114DA
.text:004114CB            mov     eax, [ebp+var_114]
.text:004114D1            add     eax, 1
.text:004114D4            mov     [ebp+var_114], eax
.text:004114DA            mov     eax, [ebp+var_120]
.text:004114E0            sub     eax, 1
.text:004114E3            cmp     [ebp+var_114], eax
.text:004114E9            jnb     short loc_41151D
.text:004114EB            mov     eax, [ebp+var_114]
.text:004114F1            movsx   ecx, byte_417000[eax]
.text:004114F8            mov     edx, [ebp+var_114]
.text:004114FE            movsx   eax, [ebp+edx+Str]
.text:00411506            xor     eax, 90h
.text:0041150B            movsx   edx, al
.text:0041150E            cmp     ecx, edx
.text:00411510            jz      short loc_411518
.text:00411512            mov     [ebp+var_129], 0
.text:00411519            jmp     short loc_41151D
.text:0041151D            movzx   eax, [ebp+var_129]
.text:00411524            test    eax, eax
.text:00411526            jz      short loc_411541
.text:00411528            mov     esi, esp
.text:0041152A            push    offset aGoodPassword ; "Good password\n"
.text:0041152F            call    ds:printf
.text:00411535            add     esp, 4
.text:00411538            cmp     esi, esp
.text:0041153A            call    sub_41114F
.text:0041153F            jmp     short loc_411558
.text:00411541            mov     esi, esp
.text:00411543            push    offset aBadPassword ; "Bad password\n"
.text:00411548            call    ds:printf
.text:0041154E            add     esp, 4
.text:00411551            cmp     esi, esp
.text:00411553            call    sub_41114F
.text:00411558            mov     esi, esp
.text:0041155A            call    ds:getchar
.text:00411560            cmp     esi, esp
.text:00411562            call    sub_41114F
.text:00411567            xor     eax, eax
.text:00411569            push    edx
.text:0041156A            mov     ecx, ebp
.text:0041156C            push    eax
.text:0041156D            lea     edx, dword_411598
.text:00411573            call    sub_411091
.text:00411578            pop     eax
.text:00411579            pop     edx
.text:0041157A            pop     edi
.text:0041157B            pop     esi
.text:0041157C            pop     ebx
.text:0041157D            mov     ecx, [ebp+var_4]
.text:00411580            xor     ecx, ebp
.text:00411582            call    sub_41103
.text:00411587            add     esp, 1F0h
.text:0041158D            cmp     ebp, esp
.text:0041158F            call    sub_41114F
.text:00411594            mov     esp, ebp
.text:00411596            pop     ebp
.text:00411597            retn
```

Relevant Assembly Code

Complexities of Reverse Engineering

- Executables can be obfuscated

```c
char pw[] = "\xc7\xf5\xfc\xfc\xb0\xd9\xb0\xf8"
            "\xff\xe0\xf5\xb0\xf9\xe4\xb0\xe7"
            "\xf1\xe3\xb0\xf1\xfc\xfc\xb0\xe7"
            "\xff\xe2\xe4\xf8\xb0\xf9\xe4\xbe";

int main(int argc, char* argv[])
{
    char in[256] = {0};
    size_t i = 0;
    size_t inlen = 0;
    bool isgood = 1;

    printf("Enter your password: ");
    fflush(stdout);

    fgets(in, sizeof(in)-1, stdin);

    inlen = strlen(in);

    for (i = 0 ; i < inlen - 1; i++)
    {
        if (pw[i] != (char) (in[i] ^ 0x90))
        {
            isgood = 0;
            break;
        }
    }

    if (isgood)
        printf("Good password\n");
    else
        printf("Bad password\n");

    getchar();

    return 0;
}
```



Information Loss - (Comments, Variable Names, Original Structure of Code

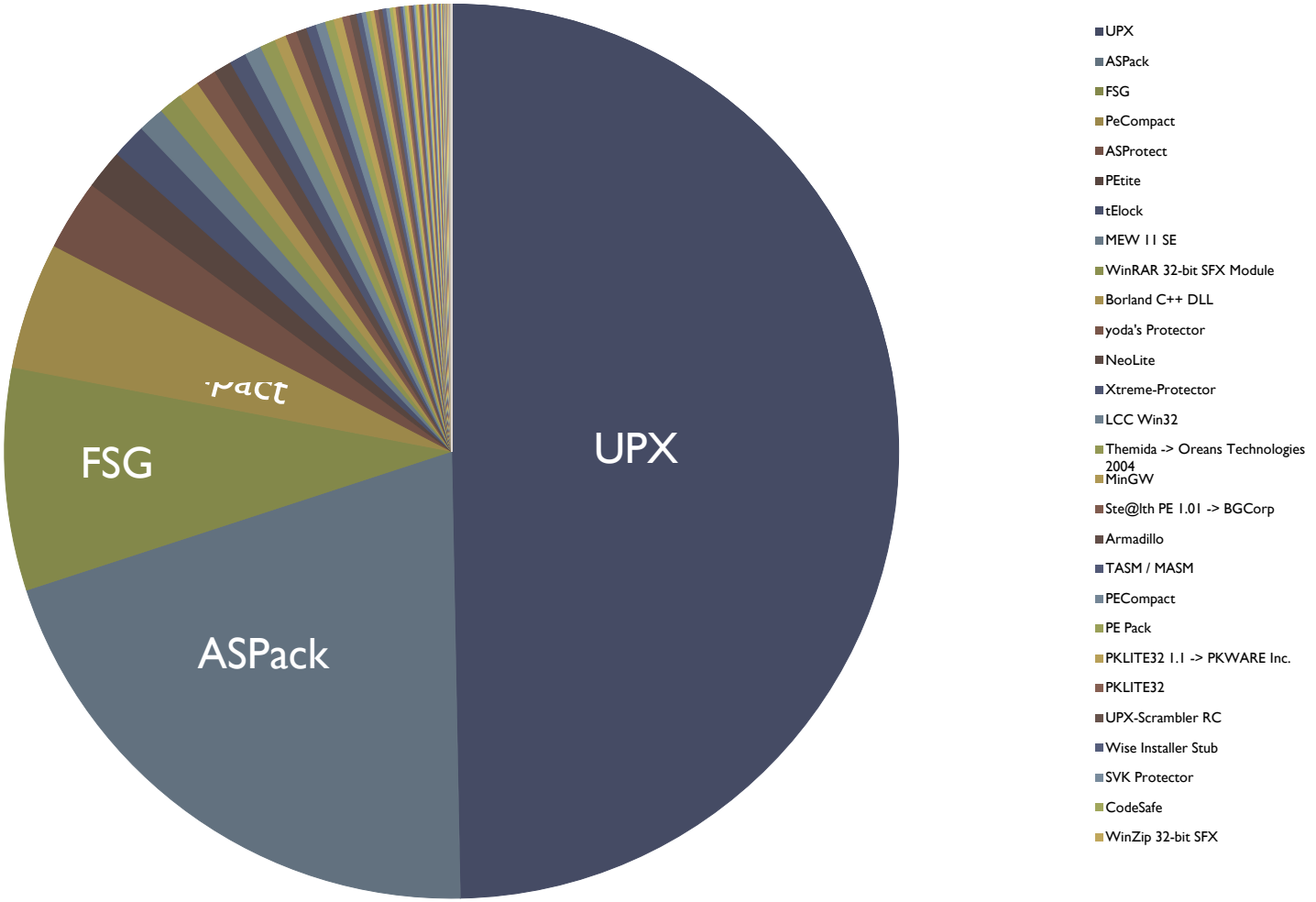
Commercial Antivirus

- Limited by time and resources
  - Customers get annoyed if results take too long
  - If AV is too invasive, software is uninstalled
  - Example: Symantec Endpoint Protection 11 has 14 kernel mode modules that are loaded
- Signatures heavily favored by Vendors
  - Fast and easy to implement
  - Decoders, as long as they are fast, used for known obfuscations
  - Time is AV's achilles heel.
- Detection of new, unknown threats is only 45%

Malware Authors Have an Easy Life

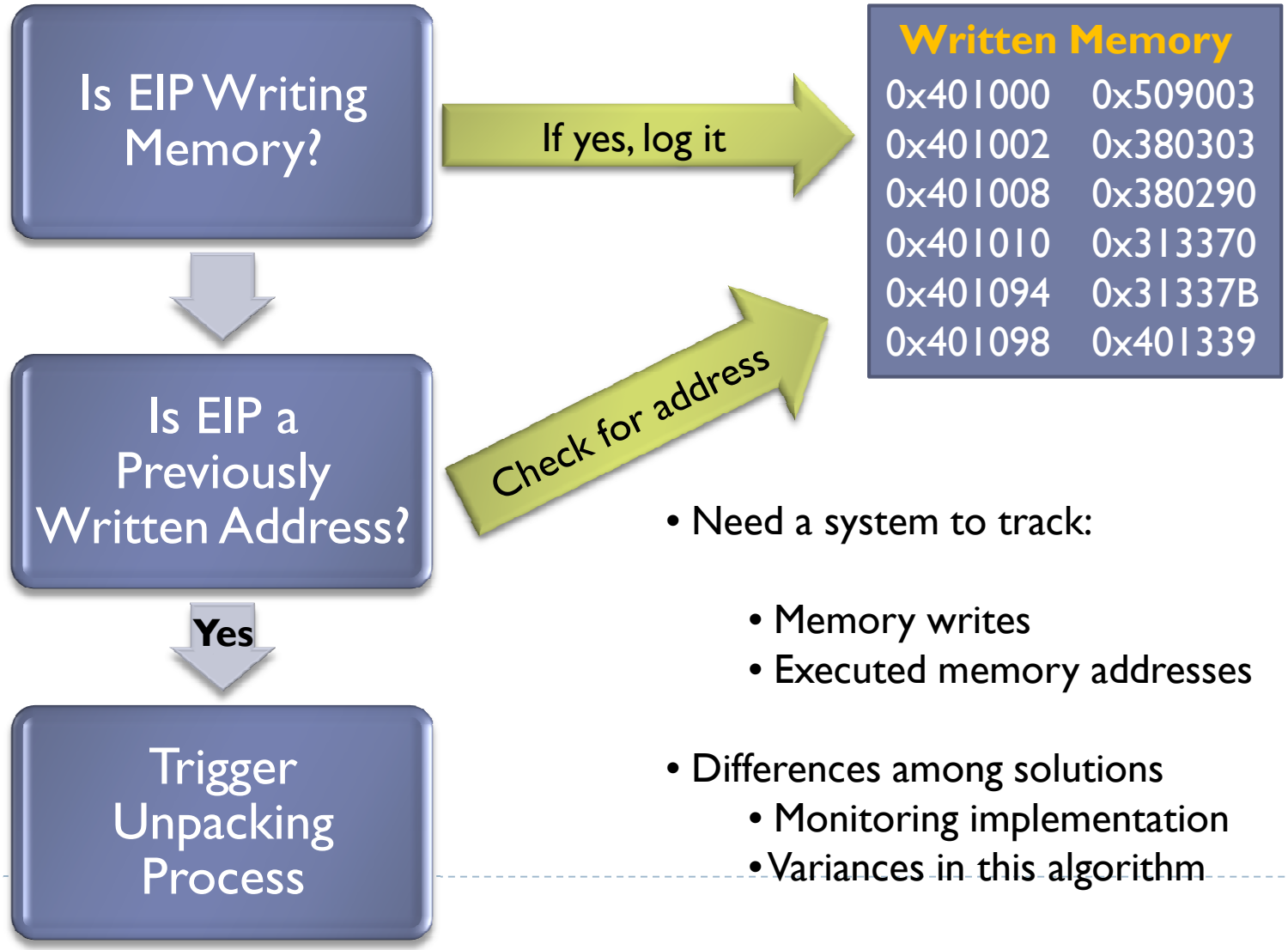
- Slight modifications yield zero detection
  - Modify the icons inside the PE files
  - Remove imports
  - Slight modification of code
- Most common exploit kits sold for N iterations of AV
  - Guaranteed not detectable
  - Provides a funding source on detection
- Generic deobfuscation is not possible for AV vendors

Types of Packers



PEiD scanning results from 1.6 million samples from Offensive Computing

Unpacking: The Generic Algorithm



- Need a system to track:
  - Memory writes
  - Executed memory addresses
- Differences among solutions
  - Monitoring implementation
  - Variances in this algorithm

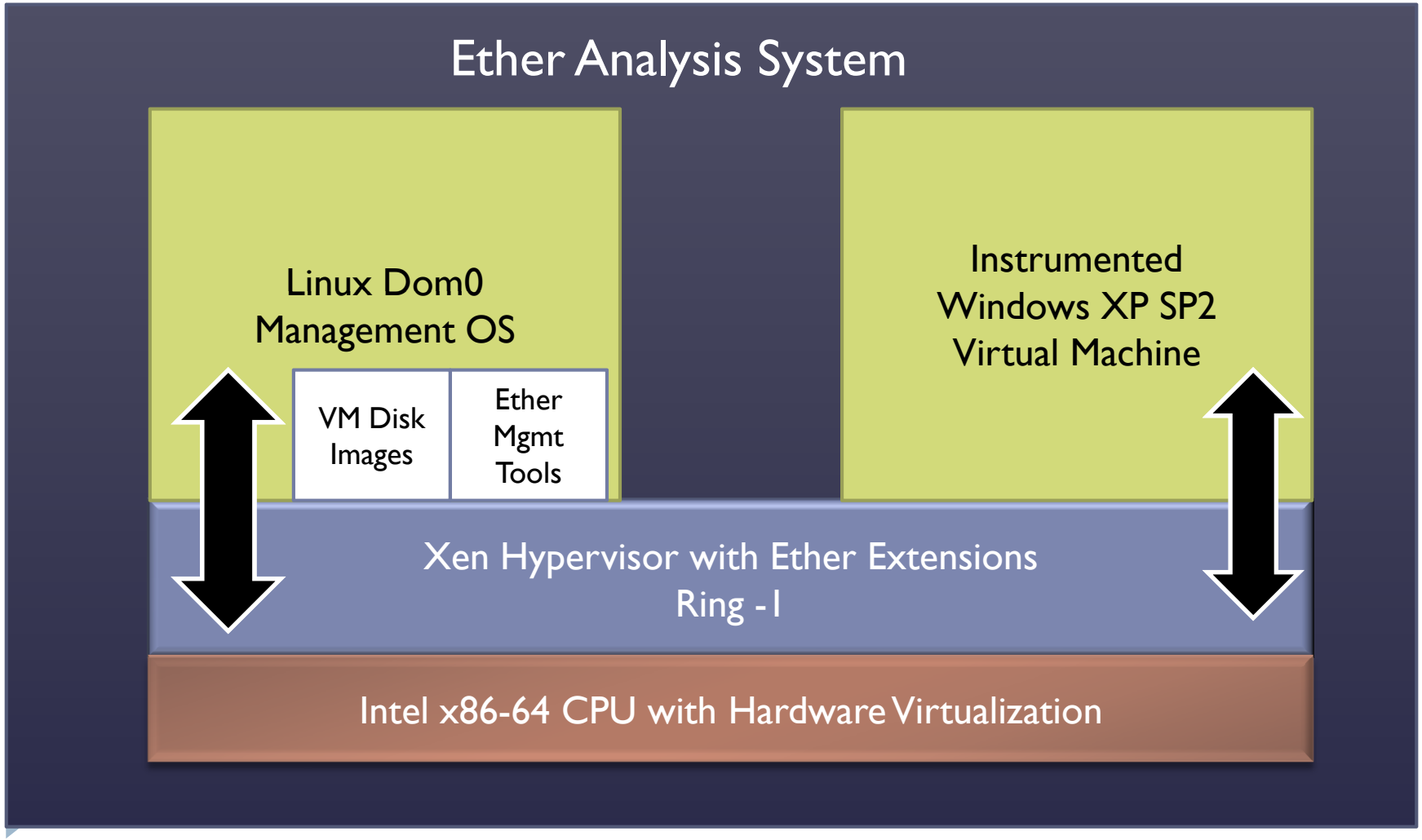
Related Work – Improving Antivirus Accuracy

- Automated unpacking system performance can be measured based on antivirus detection performance
  - Polyunpack, Renovo, Ether
    - Automated unpacking systems
    - Monitor memory writes, flag on execution of written data
  - Josse
    - QEMU virtual machine used for analysis (detectable)
    - Instruction level resolution executable monitoring
    - Emulation makes analysis slow
- Repair mechanisms of each of system primitive or non-existent

Improving Antivirus Accuracy with Hypervisor Assisted Analysis

- Contributions
  - Improved unpacking technique leveraging Ether hypervisor system
  - Better import rebuilding using kernel data structures
  - Better OEP detection from stack back-tracking technique
  - Antivirus scanning performance improved

Ether System Architecture



Importance of Repairs

- Viruses can be packed and avoid detection
- Removing imported APIs takes data away from analysis engines
- Original Entry Point (OEP) Detection hasn't progressed in years
  - Watch for all written memory, log into a hash table
  - If there is an execution in written memory guessed to be OEP
  - Dump contents of memory
  - Problems
    - Multiple obfuscations
    - Staged unpacking
    - Lots of candidate OEPs
- Restoring this information improves existing AV tools accuracy

Imported API Recovery

- Removing Imported APIs is first obfuscation step
- Reverse engineering is difficult without APIs
  - Provide no context for code
  - Order of magnitude increase in complexity
  - Restoring them is extremely valuable

Which is easier to read?

No Imports

```
loc_1001906:
push    esi
mov     esi, dword_100110C
push    3E9h
push    edi
call    esi ; dword_100110C
mov     eax, dword_1007170
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax
push    3E8h
push    edi
call    esi ; dword_100110C
mov     eax, dword_1007170
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax
push    3EAh
push    edi
call    esi ; dword_100110C
mov     eax, dword_1007170
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax
push    7D0h
push    edi
call    esi ; dword_100110C
mov     edi, [ebp+arg_4]
jmp     loc_10018AE
```

Which is easier to read?

No Imports

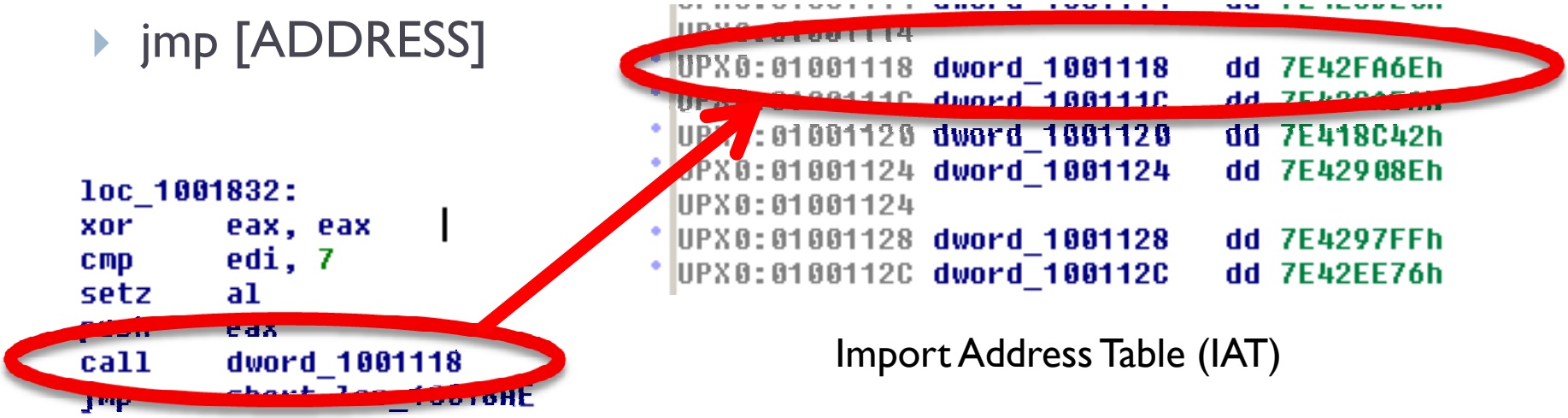
```
loc_1001906:
push    esi
mov     esi, dword_100110C
push    3E9h
push    edi
call    esi ; dword_100110C
mov     eax, dword_1007170
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax
push    3E8h
push    edi
call    esi ; dword_100110C
mov     eax, dword_1007170
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax
push    3EAh
push    edi
call    esi ; dword_100110C
mov     eax, dword_1007170
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax
push    7D0h
push    edi
call    esi ; dword_100110C
mov     edi, [ebp+arg_4]
jmp     loc_10018AE
```

Imports Rebuilt

```
loc_1001906:            ; uEnable
push    esi
mov     esi, ds:__imp__EnableMenuItem@12 ; EnableMenuItem(x,x,
push    3E9h            ; uIDEnableItem
push    edi             ; hMenu
call    esi ; EnableMenuItem(x,x,x) ; EnableMenuItem(x,x,x)
mov     eax, _pgmCur
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax             ; uEnable
push    3E8h            ; uIDEnableItem
push    edi             ; hMenu
call    esi ; EnableMenuItem(x,x,x) ; EnableMenuItem(x,x,x)
mov     eax, _pgmCur
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax             ; uEnable
push    3EAh            ; uIDEnableItem
push    edi             ; hMenu
call    esi ; EnableMenuItem(x,x,x) ; EnableMenuItem(x,x,x)
mov     eax, _pgmCur
mov     eax, [eax+58h]
inc     eax
neg     eax
sbb     eax, eax
and     eax, 3
push    eax             ; uEnable
push    7D0h            ; uIDEnableItem
push    edi             ; hMenu
call    esi ; EnableMenuItem(x,x,x) ; EnableMenuItem(x,x,x)
mov     edi, [ebp+Msg]
jmp     loc_10018AE
```

Import Repair Process

- Find the original entry point
  - Unpack code until this address is found
  - Use OEP method discussed later
- Find references to imported DLLs
  - call [ADDRESS]
  - jmp [ADDRESS]



Import Address Table (IAT)

Import Repair Process

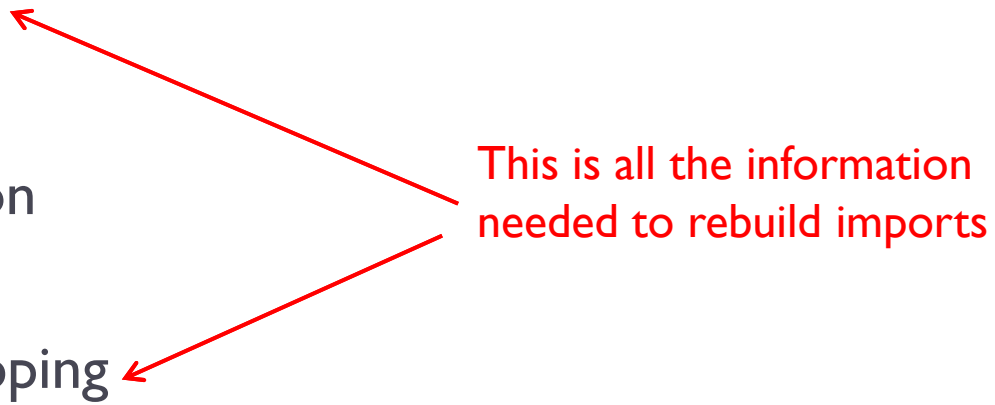
- Each imported DLL has an IAT corresponding to the APIs brought into the application
- The first DLL is found by backtracking the IAT memory until a NULL is found.
- The DWORD after the NULL is the beginning of that DLL's API
- How do we determine which DLL belongs to which memory address?

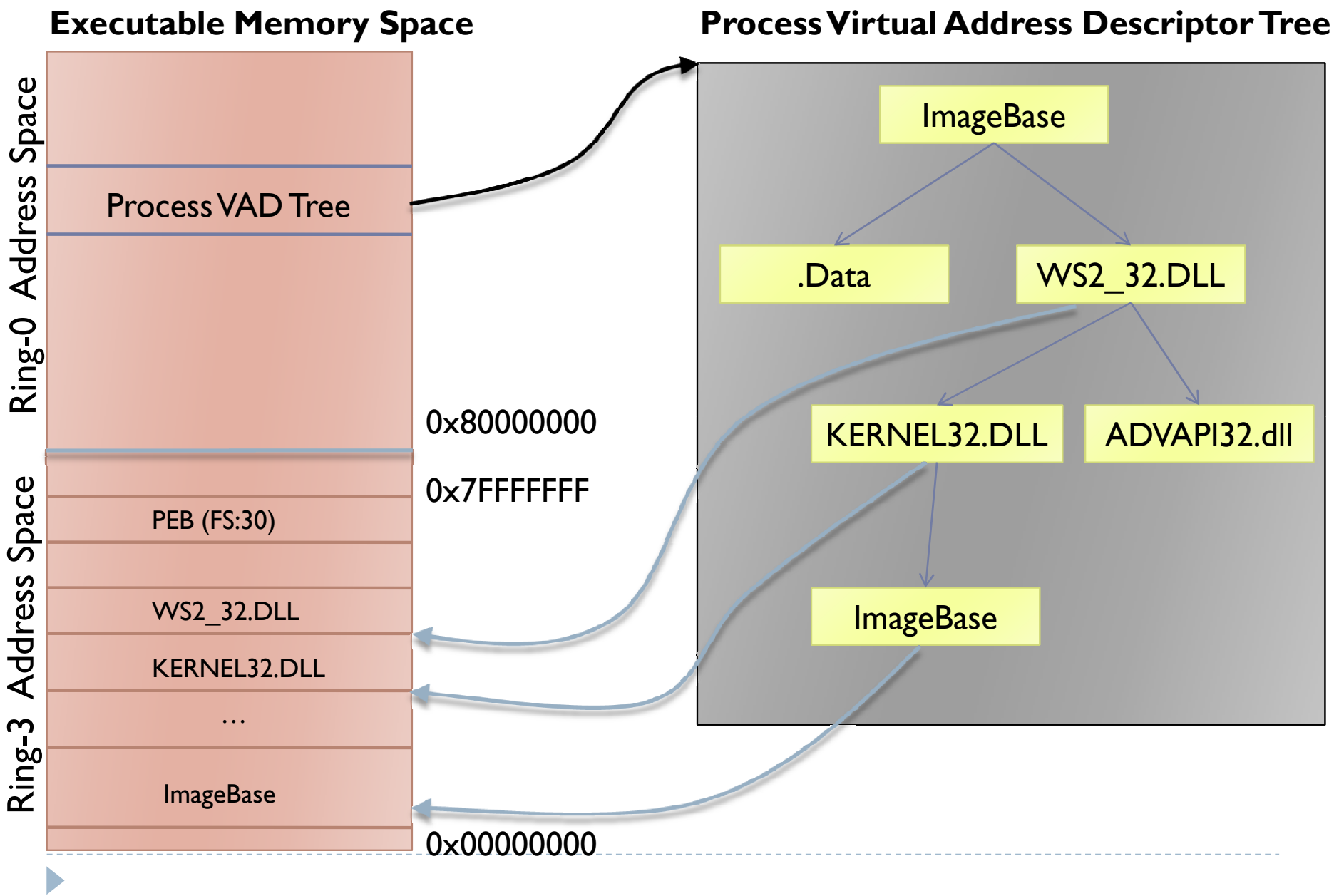
Determining DLL Address Space

- Old Method
  - Attach to process via debugger interface
  - Call windows APIs to query address module
  - Resolve addresses from the DLL listings
- Problems
  - Hypervisor has no access to internal Windows APIs
  - Access to APIs would violate sterility of guest environment (DETECTION)
  - No real way to extract data we need

Import Repair Process

- New Method – Use kernel memory management data structure
- Virtual Address Descriptor – VAD
  - Each process has a VAD to describe memory usage
  - OS uses VADs to interact with CPU MMU
  - Very accurate use of process space
- Balanced Binary Tree
  - Address space
  - Size of memory region
  - Execution flags
  - Module memory mapping



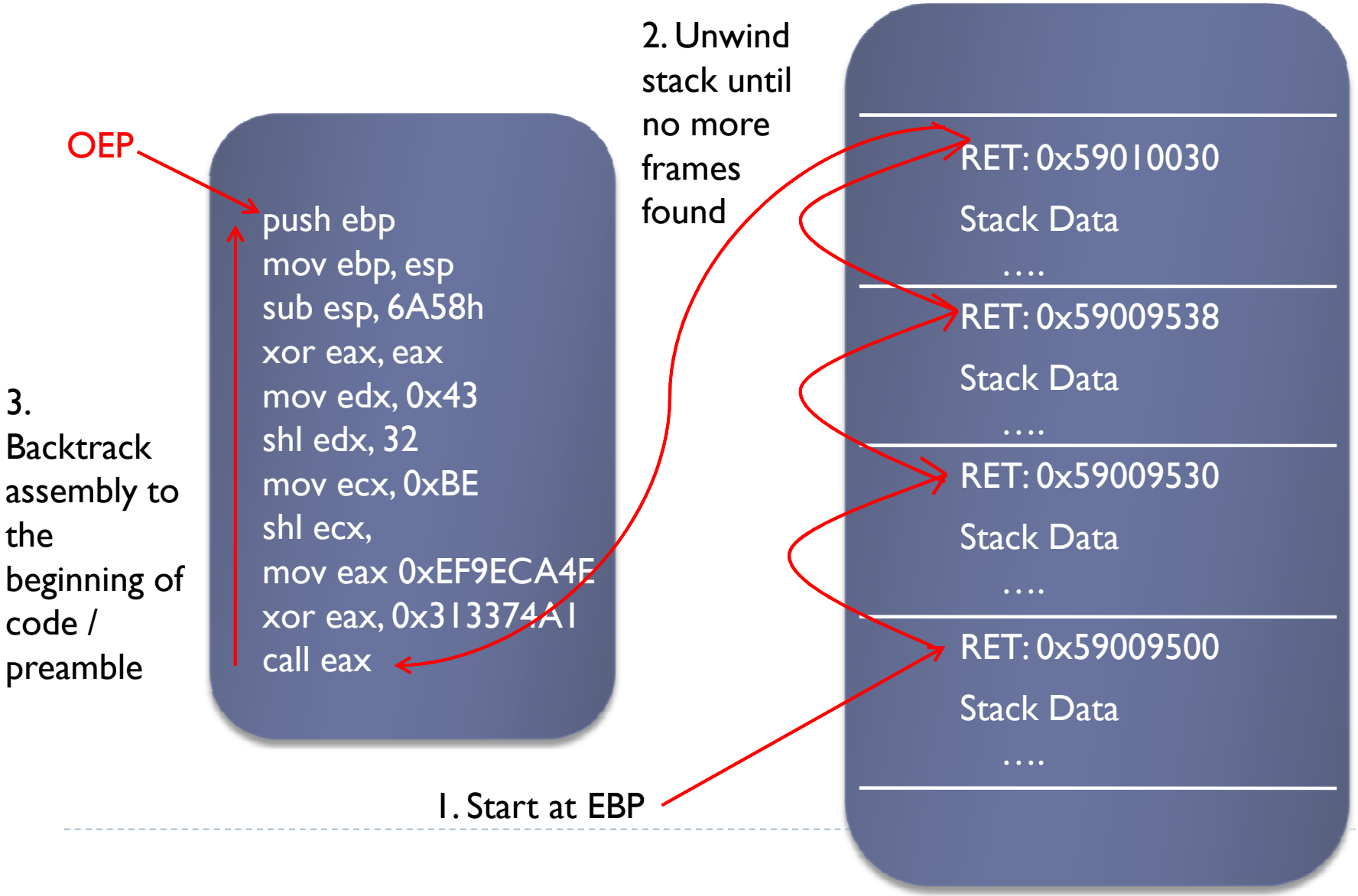

Executable Memory Space
Process Virtual Address Descriptor Tree
Ring-0 Address Space
Process VAD Tree
0x80000000
0x7FFFFFFF
Ring-3 Address Space
PEB (FS:30)
WS2_32.DLL
KERNEL32.DLL
…
ImageBase
0x00000000
ImageBase
.Data
WS2_32.DLL
KERNEL32.DLL
ADVAPI32.dll
ImageBase

Original Entry Point Detection

- Standard OEP discovery produces many file
- Most common packers produce few samples
- Complex packers increase complexity of unpacking
- Requires manual analysis of each candidate dump

Packer	Detected OEPs
Armadillo	1
Petite	1
UPX	1
UPX Scrambler	1
Aspack	2
FSG	2
PECompact	2
VMProtect	12
PEPack	12
AsProtect	15
Themida	33
Yoda	43
PEX	133
MEW	1018

OEP Algorithm – EBP based stack frames



Testing and Analysis

- Verification of malicious file
  - Execution – show that it runs without crashing
  - OS state change – Look for modifications to
    - Registry
    - File system
    - Startup systems
  - Verification of maliciousness
    - Detection by at least 1 AV scanner
    - Good way to scan large sample sets of malware

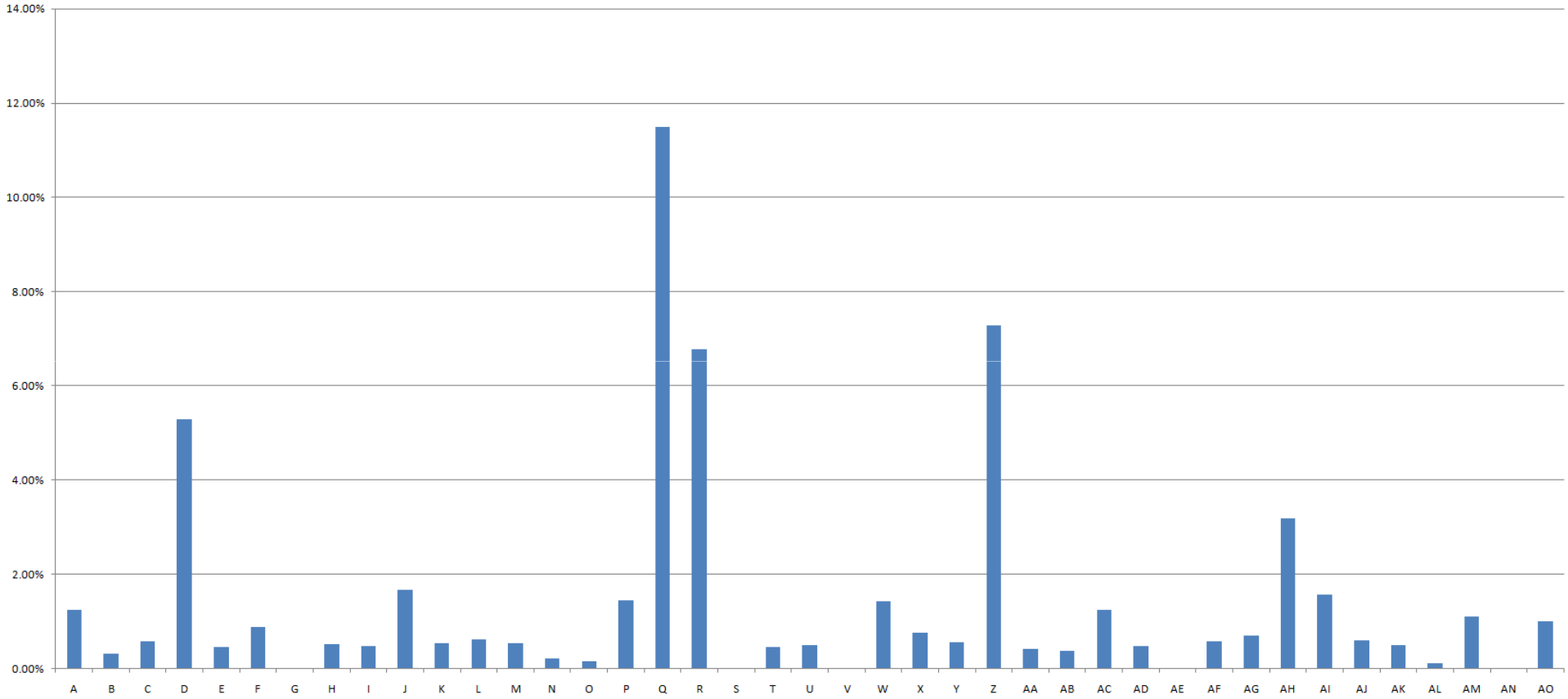
Test 1: Linux Virus Scanners

- Analyze 500,000 samples for samples that are detected by one AV vendor
- Randomly choose 1,000 samples
- Apply verification method, 697 left over
- Results
  - Highest 45.23%
  - Average 19.86%
  - Lowest 0.68%

Test 2: Virus Total

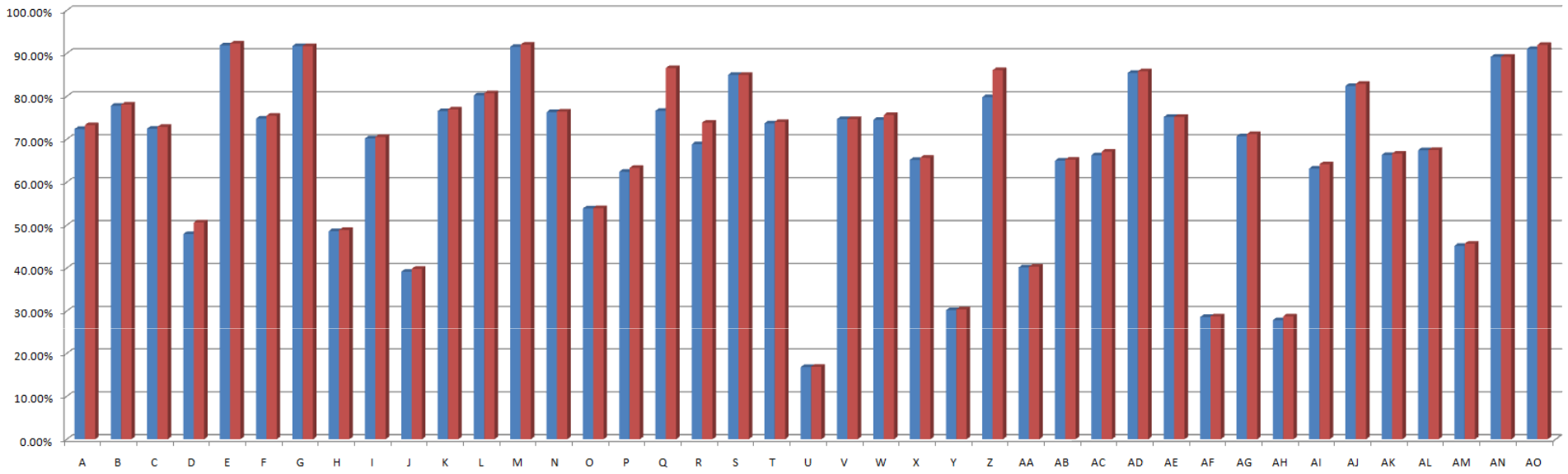
- Virus Total (VT) – Website by Hispasec that aggregates 40 AV scanners' testing results.
- Two weeks passed to allow for improved AV signature development
- Apply verification method, 1,195 left over
- Results
  - Highest 11.54%
  - Average 7.37%
  - Lowest 1.70%

Test 2: Improvement of Scanners



Best:	11.54%
Average	7.37%
Lowest	1.70%

Test 2: Total Detection Percentages



- High value improvements to most AV vendors
- Low improvement means either deobfuscation is poor, or detection is poor
- Blue represents packed, red represents unpacked state

Improving AV Conclusions

- Unpacking and deobfuscation are high value changes
  - In development to incorporate into line-speed e-mail scanner
  - Improved detection of slightly modified malware
- Rebuilding of imports
  - Improves reverse engineering
  - Full recovery of import data
  - VAD is fundamental part of OS (hard to deceive)
- Improved OEP Detection
  - Reduces multiple OEP candidates
  - Reduced analysis time
- Improvement in AV scanning results

Improving AV Future Work

- Unpacking process takes too long
  - Current method is to unpack for 5 minutes
  - Better algorithms can be found to determine if unpacking works
- Integration with existing tools
  - IDA Pro
  - OllyDbg
  - WinDbg
- Build full-fledged debugger
  - PDB / Paimei integration
  - Visual control of unpacking

Questions?

- Contact Information
  - Danny Quist

Email: dquist@offensivecomputing.net
Twitter: Ocomputing