

Visual Malware Reversing

How to Stop Reading Assembly and Love the Code

Danny Quist

ShmooCon 111

January 30, 2011

Danny Quist

- Founder of OffensiveComputing.net
 - Free malware!
 - RE Training!
 - 2.3 million samples!
- Ph.D. Computer Science, New Mexico Tech
- Twitter: @ocomputing
- Research scientist
Los Alamos National Laboratory

Goals

- Identify structure of malware quickly
- Remove difficulty of unpacking
- Remove dependence on tools like IDA
- Play nicely with others
 - IDA
 - OllyDbg
 - WinDbg
 - GDB

VERA Overview







- Functionality
- New features
 - Ether Import Rebuilding
 - Better OEP detection
 - IDA Pro Integration
 - VERAttrace – No Xen / Ether hardware required!
 - Imports in the VERA GUI

What is VERA?

- Visualizing Executables for Reversing and Analysis
- High-level overview of entire program
- Low-level drill-down of details
- Free!
- Development is funded!

Demo1 - VERA

What the Colors Mean

-  • **Yellow** – Normal uncompressed low-entropy section data
-  • **Dark Green** – DLL / API / Section not present
-  • **Light Purple** – SizeOfRawData = 0
-  • **Dark Red** – High Entropy
-  • **Light Red** – Instructions not in the packed exe
-  • **Lime Green** – Operands don't match

Generating Traces

- Ether
 - Set of patches to the Xen hypervisor
 - Allows for covert tracing of executables
- Veratrace – NEW!
 - Intel PIN system suitable for use in VMWare
 - Commercial code analysis
- Output from debuggers (GDB/WinDbg/...)

Ether Improvements

- Import reconstruction using kernel data structures
- OEP detection from stack back-tracking technique
- Antivirus scanning performance improved

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]

```
loc_1001832:  
xor     eax, eax  
cmp     edi, 7  
setz    al  
push   eax  
call    dword_1001118  
jmp     short loc_100183E
```

UPX0:01001114	dword_1001114	dd	7E42E020h
• UPX0:01001118	dword_1001118	dd	7E42FA6Eh
UPX0:0100111C	dword_100111C	dd	7E429A5Ah
• UPX0:01001120	dword_1001120	dd	7E418C42h
• UPX0:01001124	dword_1001124	dd	7E42908Eh
UPX0:01001124			
• UPX0:01001128	dword_1001128	dd	7E4297FFh
• UPX0:0100112C	dword_100112C	dd	7E42EE76h

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?

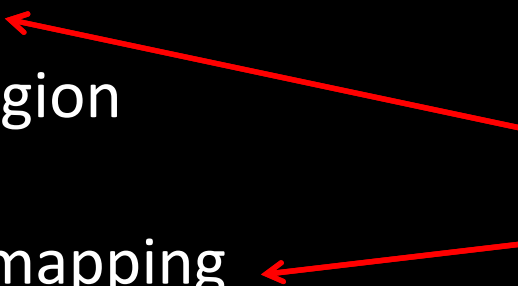
Determining DLL Address Space

- 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

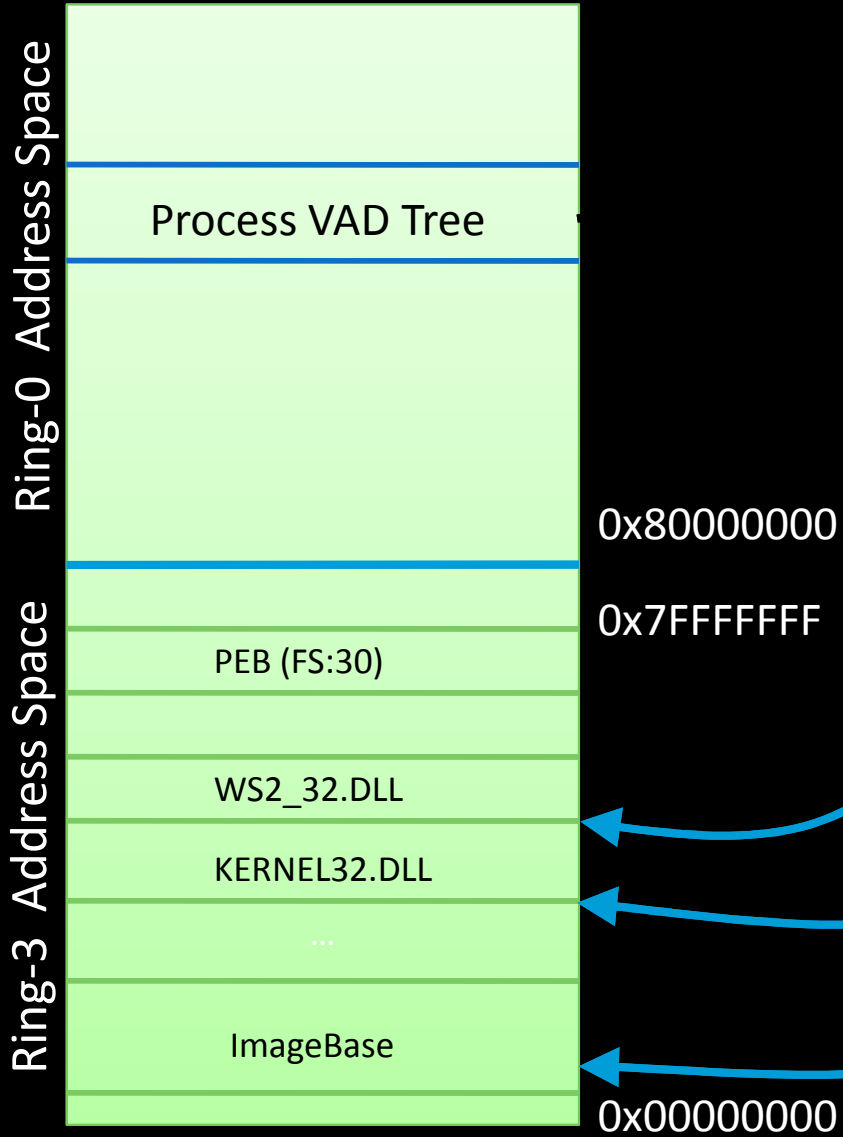
Import Repair Process

- 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
- Data Structure - Balanced Binary Tree
 - Address space
 - Size of memory region
 - Execution flags
 - Module memory mapping

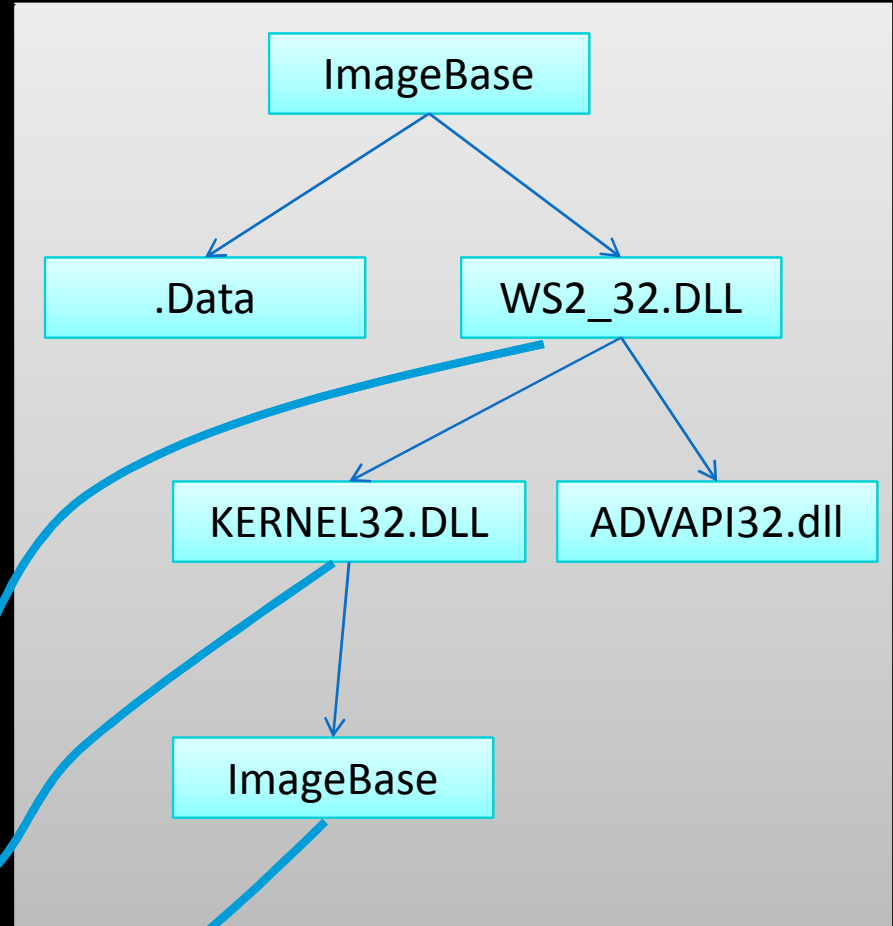
This is all the information
needed to rebuild imports



Executable Memory Space



Process Virtual Address Descriptor Tree



0x80000000

0x7FFFFFFF

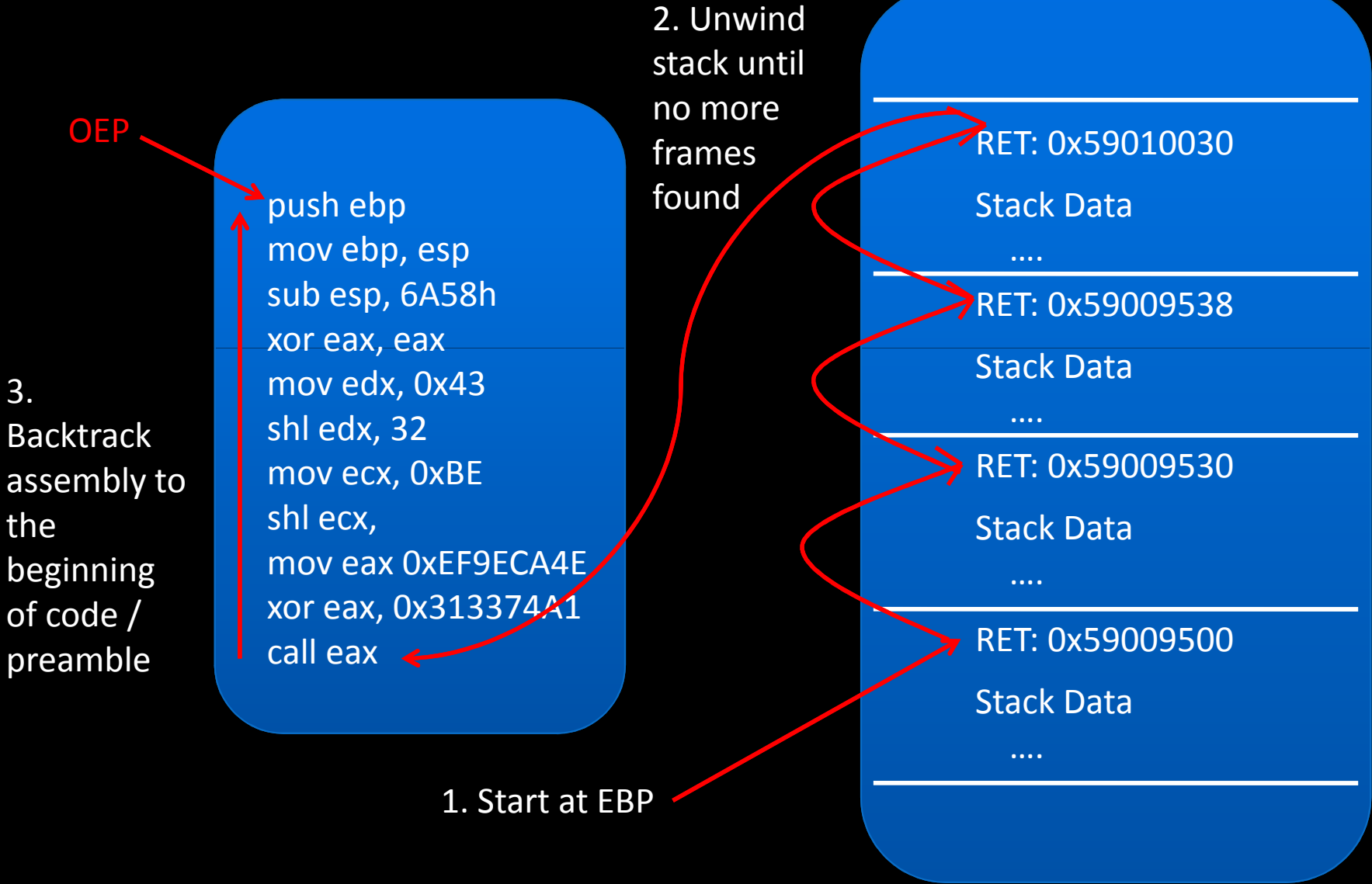
0x00000000

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



Problems with Ether

- Heavy-weight analysis system
- Not portable to common VMs like Vmware
- Problems Installing Ether
 - Old version of Xen (3.1.x)
 - Debian
 - Other bugs
- Provide alternate way to collect information

VERATrace

- Intel PIN based instruction tracing program
- Usable on VMWare / VirtualBox / VirtualPC
- Useful for analyzing non-obfuscated programs
- Extensions planned to hide from malware
- Unpacking (See Saffron-DI)
- Adds import data to VERA

Veratrace Demonstration

Demo 2

Veratrace Malware

Demo 3

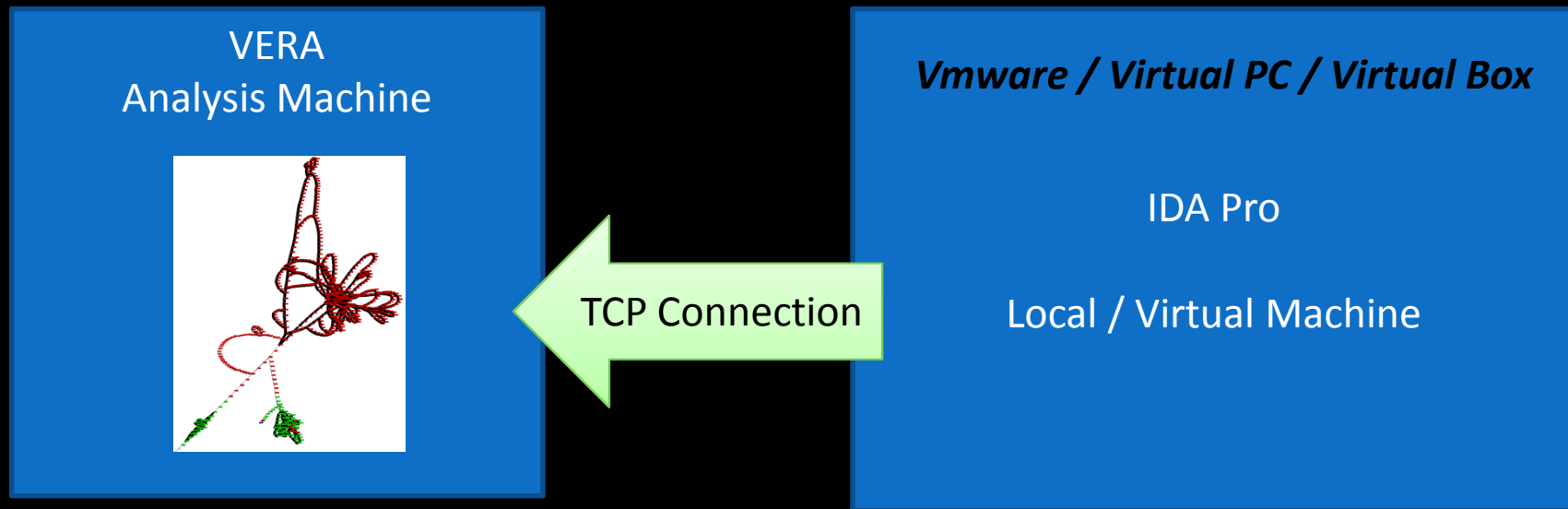
Caveat Emptor

- Intel PIN is very detectable
 - Malware doesn't always run well
 - Ether was made for malware analysis
 - Sometimes useful results are found

VERA IDA Plugin

- Used to correlate VERA graphs with IDA
- TCP Connection from IDA Pro system to VERA
- Synchronization with IDA representation
- Identify core constructs

VERA – IDA Plugin



IDA Integration

Demo 4

Future Work

- Memory usage analysis
- Integrating string analysis in the graph
- Explore 3D visualizations
- Better integration with Debuggers

Conclusion

- New Features
 - Better import recovery in Ether
 - New OEP Algorithm
 - New tracing tool VERATrace
 - IDA Pro Plugin
 - Imports in the visualization

<http://www.offensivecomputing.net/vera>

Twitter: @ocomputing

Thanks ShmooCon!