# Reverse Engineering by Crayon: Game Changing Hypervisor and Visualization Analysis

## Fine-grained covert debugging using hypervisors and analysis via visualization

**Daniel A. Quist**
**Lorie M. Liebrock**
Offensive Computing, LLC
New Mexico Tech

Defcon 17
Las Vegas, NV

# Introduction

- Reverse Engineering is Hard!
- Hypervisor based executable monitoring
- Modifications for improved performance
- Visualization tool for rapid analysis
- Modifying the reverse engineering process

# Difficulties of RE

- Time consuming process
- Difficult set of skills to acquire
- Tools are advanced, but still don’t provide adequate views.
- Focused on static analysis
- Software armoring makes process even more difficult

# Process for Reverse Engineering

- Setup an isolated run-time environment
- Execution and initial analysis
- Deobfuscate compressed or packed code
- Disassembly / Code-level Analysis
- Identify and analyze relevant and interesting portions of the program

# Isolated Analysis Environment

- Setup an Isolated Runtime Environment
  - Virtual machines: VMWare, Xen, KVM, …
  - Need to protect yourself from malicious code
  - Create a known-good baseline environment
  - Quickly allows backtracking if something bad happens

# Execution and Initial Analysis

- **Goal**: Quickly figure out what the program is doing without looking at assembly
- Look for:
  - Changes to the file system
  - Changes to the behavior of the system
    - Network traffic
    - Overall performance
    - Ads or changed browser settings

# Remove Software Armoring

- Program protections to prevent reverse engineering
- Done via packers – Small encoder/decoder
- Self-modifying code
- Lots of research about this
  - OllyBonE, Saffron, Polyunpack, Renovo, Ether, Azure
  - My research uses Ether

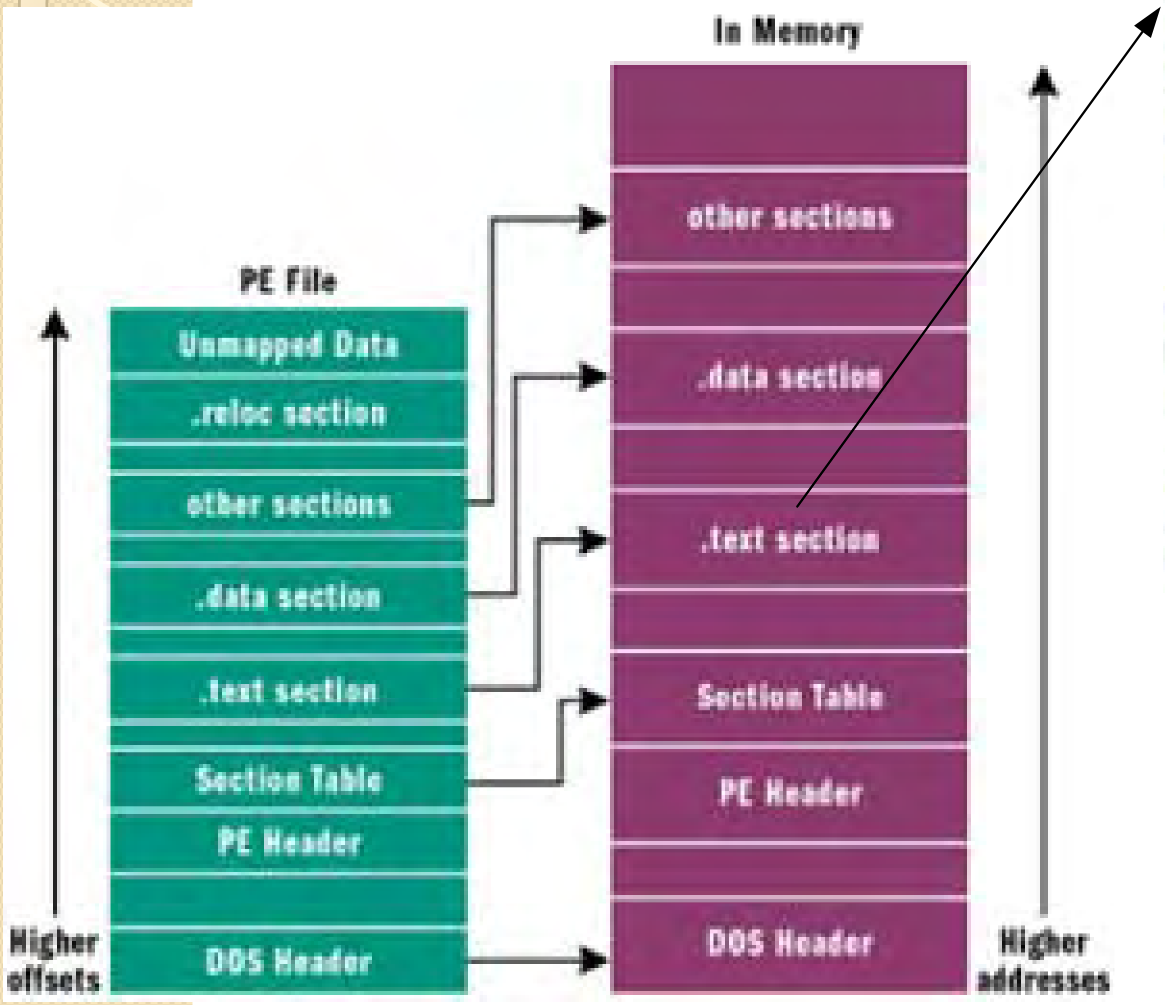
# Packing and Encryption

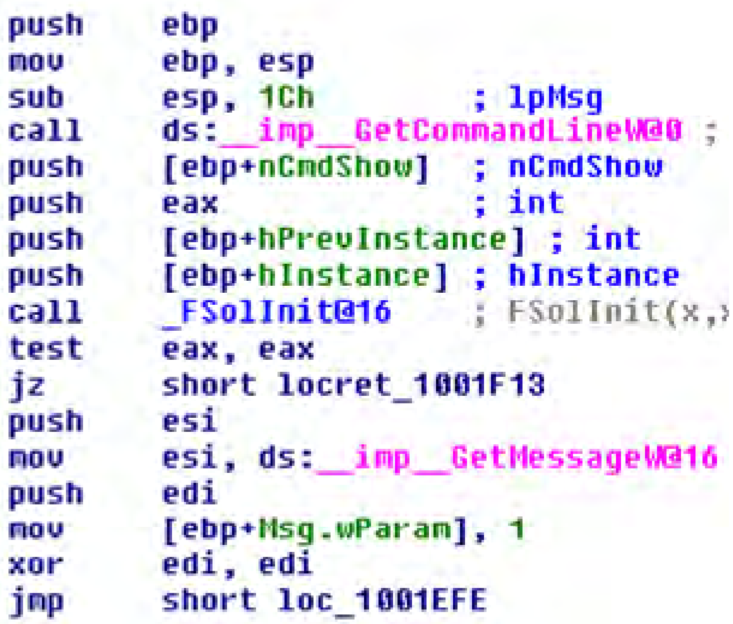
- Self-modifying code
  - Small decoder stub
  - Decompress the main executable
  - Restore imports
- Play “tricks” with the executable
  - OS Loader is inherently lazy (efficient)
  - Hide the imports
  - Obscure relocations
  - Use bogus values for various unimportant fields

# Software Armoring

- Compressed, obfuscated, hidden code
- Virtual machine detection
- Debugger detection
- Shifting decode frames

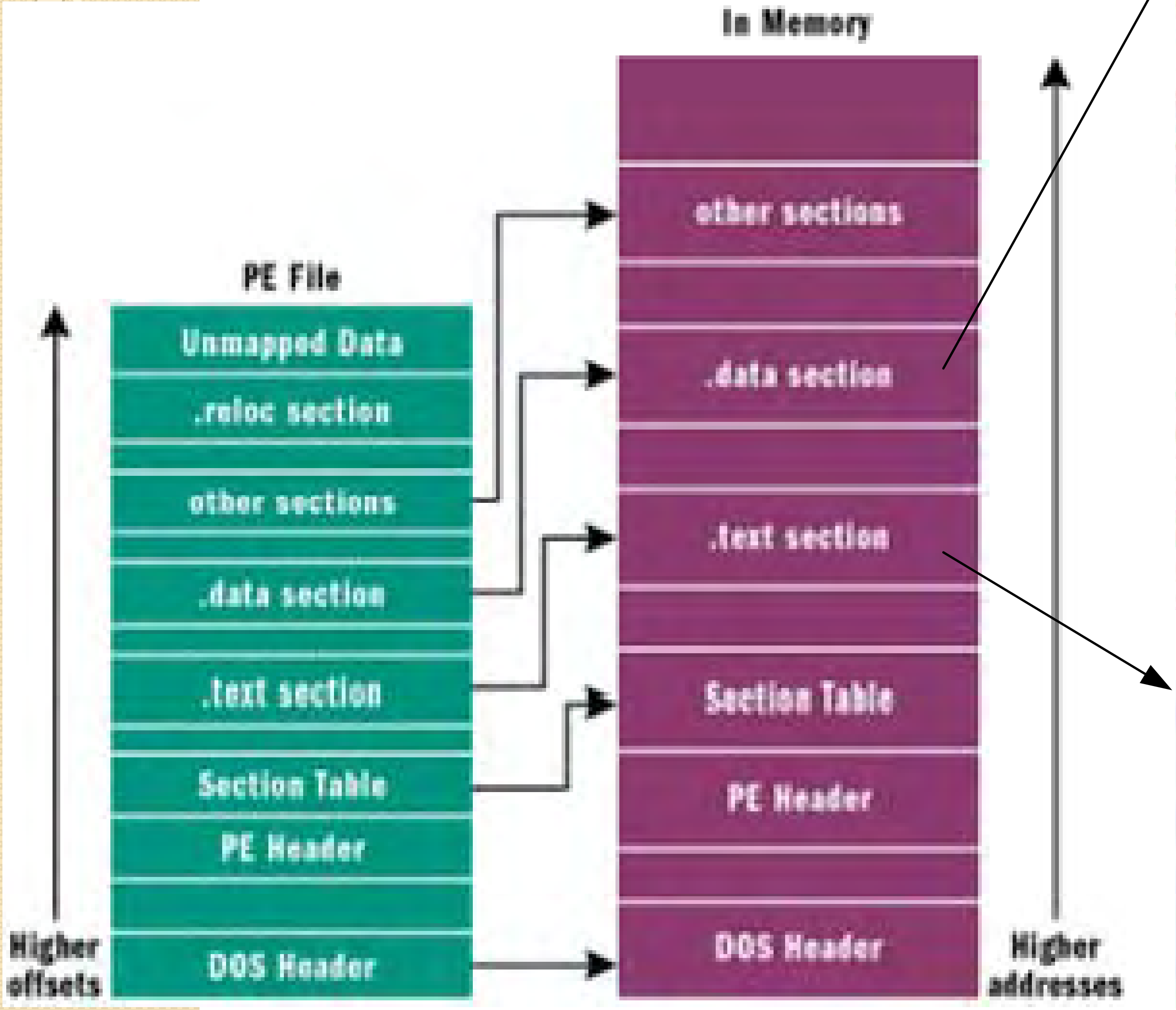
# Normal PE File

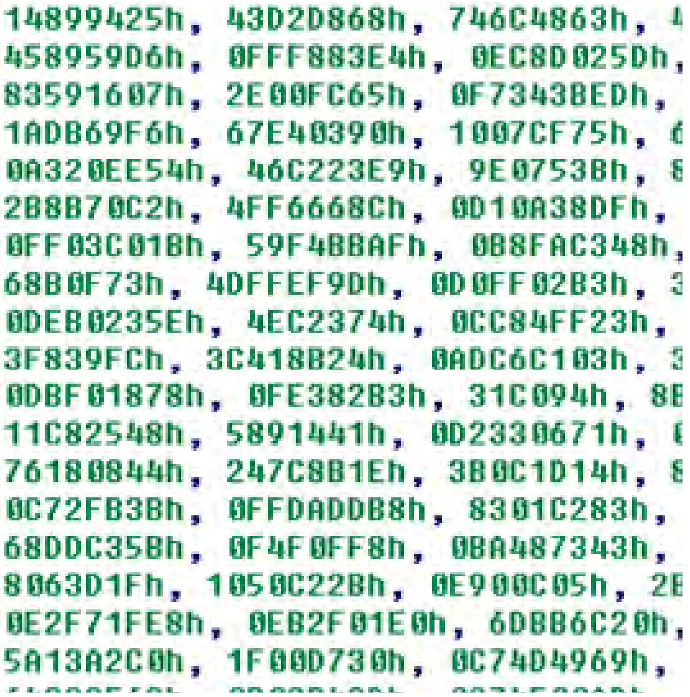




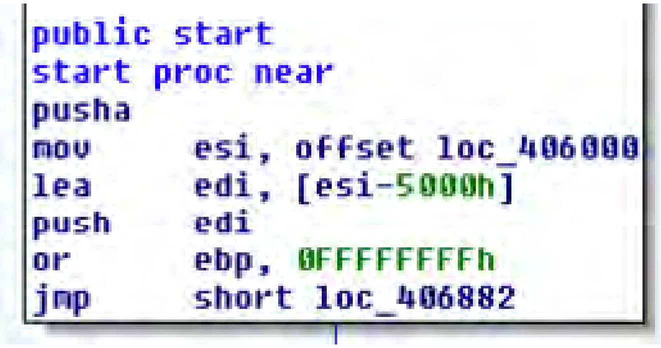
```
push    ebp
mov     ebp, esp
sub     esp, 1Ch          ; lpMsg
call    ds:__imp__GetCommandLineW@0 ;
push    [ebp+nCmdShow]    ; nCmdShow
push    eax               ; int
push    [ebp+hPrevInstance] ; int
push    [ebp+hInstance] ; hInstance
call    _FSolInit@16      ; FSolInit(x,
test    eax, eax
jz      short locret_1001F13
push    esi
mov     esi, ds:__imp__GetMessageW@16
push    edi
mov     [ebp+Msg.wParam], 1
xor     edi, edi
jmp     short loc_1001EFE
```

# Packed PE File





```
14899425h, 43D2D868h, 746C4863h, 4
458959D6h, 0FFF883E4h, 0EC8D025Dh,
83591607h, 2E00FC65h, 0F73438EDh,
1ADB69F6h, 67E40390h, 1007CF75h, 6
0A320EE54h, 46C223E9h, 9E07538h, 8
2B8B70C2h, 4FF6668Ch, 0D10A38DFh,
0FF03C01Bh, 59F4BBAFh, 0B8FAC348h,
68B0F73h, 4DFFEF9Dh, 0D0FF02B3h, 3
0DEB0235Eh, 4EC2374h, 0CC84FF23h,
3F839FCh, 3C418B24h, 0ADC6C103h, 3
0DBF01878h, 0FE382B3h, 31C094h, 8E
11C82548h, 5891441h, 0D2330671h, 6
76180844h, 247C8B1Eh, 3B0C1D14h, 8
0C72FB3Bh, 0FFDADDB8h, 8301C283h,
68DDC35Bh, 0F4F0FF8h, 0BA487343h,
8063D1Fh, 1050C22Bh, 0E900C05h, 2E
0E2F71FE8h, 0EB2F01E0h, 6DBB6C20h,
5A13A2C0h, 1F00D730h, 0C74D4969h,
```



```
public start
start proc near
pusha
mov     esi, offset loc_406000
lea     edi, [esi-5000h]
push    edi
or      ebp, 0FFFFFFFFh
jmp     short loc_406882
```

# Troublesome Protections

- Virtual Machine Detection
  - Redpill, ocvmdetect, Paul Ferrie's paper
- Debugger Detection
  - IsDebuggerPresent()
  - EFLAGS bitmask
- Timing Attacks
  - Analyze value of RDTSC before and after
  - Really effective

# Thwarting Protections

Two methods for circumvention

1. Know about all the protections before hand and disable them

2. Make yourself invisible

# Virtual Machine Monitoring

- Soft VM Based systems
  - Renovo
  - Polyunpack
  - Zynamics Bochs unpacker
- Problems
  - Detection of virtual machines is easy
  - Intel CPU never traditionally designed for virtualization
  - Do not emulate x86 bug-for-bug

# OS Integrated Monitoring

- Saffron, OllyBonE
  - Page-fault handler based debugger
  - Abuses the supervisor bit on memory pages
  - High-level executions per page
- Problems
  - Destabilizes the system
  - Need dedicated hardware
  - Fine-grain monitoring not possible

# Fully Hardware Virtualizations

- Ether: A. Dinaburg, P. Royal
  - Xen based hypervisor system
  - Base functions for monitoring
    - System calls
    - Instruction traces
    - Memory Writes
  - All interactions done by memory page mapping
- Problems
  - Unpacking code primitive
  - Dumps mangled and not possible to dissassemble
  - Old version of Xen hypervisor

# Disassembly and Code Analysis

- Most nebulous portion of the process
- Largely depends on intuition
  - Example: When we reversed the MP3 Cutter and MIRC programs
  - Takes time and experience
- Looking at assembly is tedious
- Suffers from “not seeing the forest from the trees” syndrome
- Analyst fatigue – Level of attention required yields few results

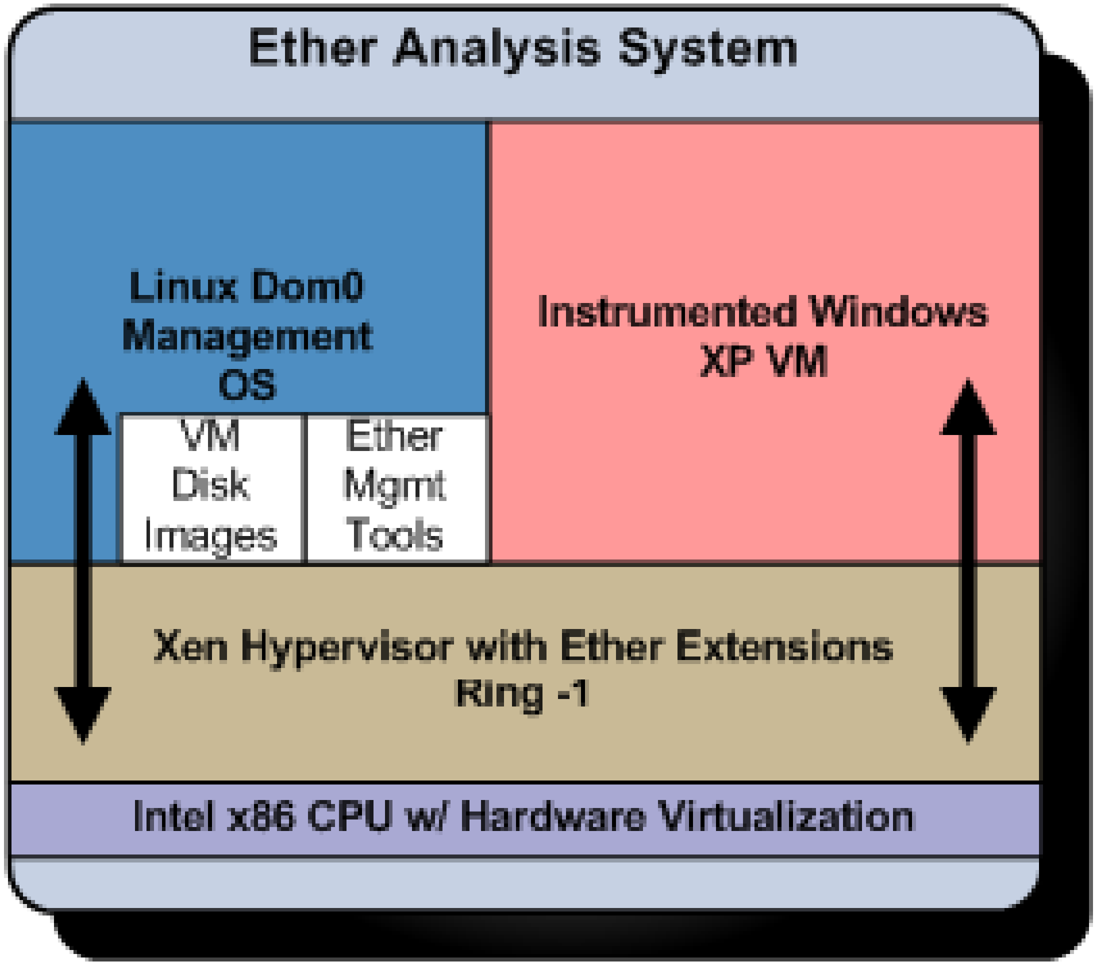
# Find Interesting and Relevant Portions of the Executable

- Like disassembly, this relies on a lot of intuition and experience
- Typical starting points:
  - Look for interesting strings
  - Look for API calls
  - Examine the interaction with the OS
- This portion is fundamentally imprecise, tedious, and often frustrating for beginners and experts

# Contributions

- Modifications to Ether
  - Improve malware unpacking
  - Enable advanced tracing mechanisms
  - Automate much of the tedious portions
- Visualizing Execution for Reversing and Analysis (VERA)
  - Speed up disassembly and finding interesting portions of an executable
  - Faster identification of the Original Entry Point

# Ether System Architecture


Ether Analysis System
Linux Dom0 Management OS
VM Disk Images
Ether Mgmt Tools
Instrumented Windows XP VM
Xen Hypervisor with Ether Extensions Ring -1
Intel x86 CPU w/ Hardware Virtualization

# Extensions to Ether

- Removed unpacking code from hypervisor into userspace
- Better user mode analysis
- PE Repair system – Allows for disassembly of executables
- Added enhanced monitoring system for executables

# Results

- Close to a truly covert analysis system
  - Ether is nearly invisible
  - Still subject to bluepill detections
- Fine-grain resolution of program execution
- Application memory monitoring and full analysis capabilities
- Dumps from Ether can now be loaded in IDA Pro without modification

# Open Problems

- Unpacking process produces lots of candidate dump files
- Need to figure out what the OEP is
- Import rebuilding is still an issue
- Now that there is a nice tool for tracing programs covertly, we need to do analysis

# Visualization of Trace Data

- Goals:
  - Quickly visually subvert software armoring
  - Identify modules of the program
    - Initialization
    - Main loops
    - End of unpacking code
  - Figure out where the self-modifying code ends (OEP detection)
  - Discover dynamic runtime program behavior
  - Integrate with existing tools

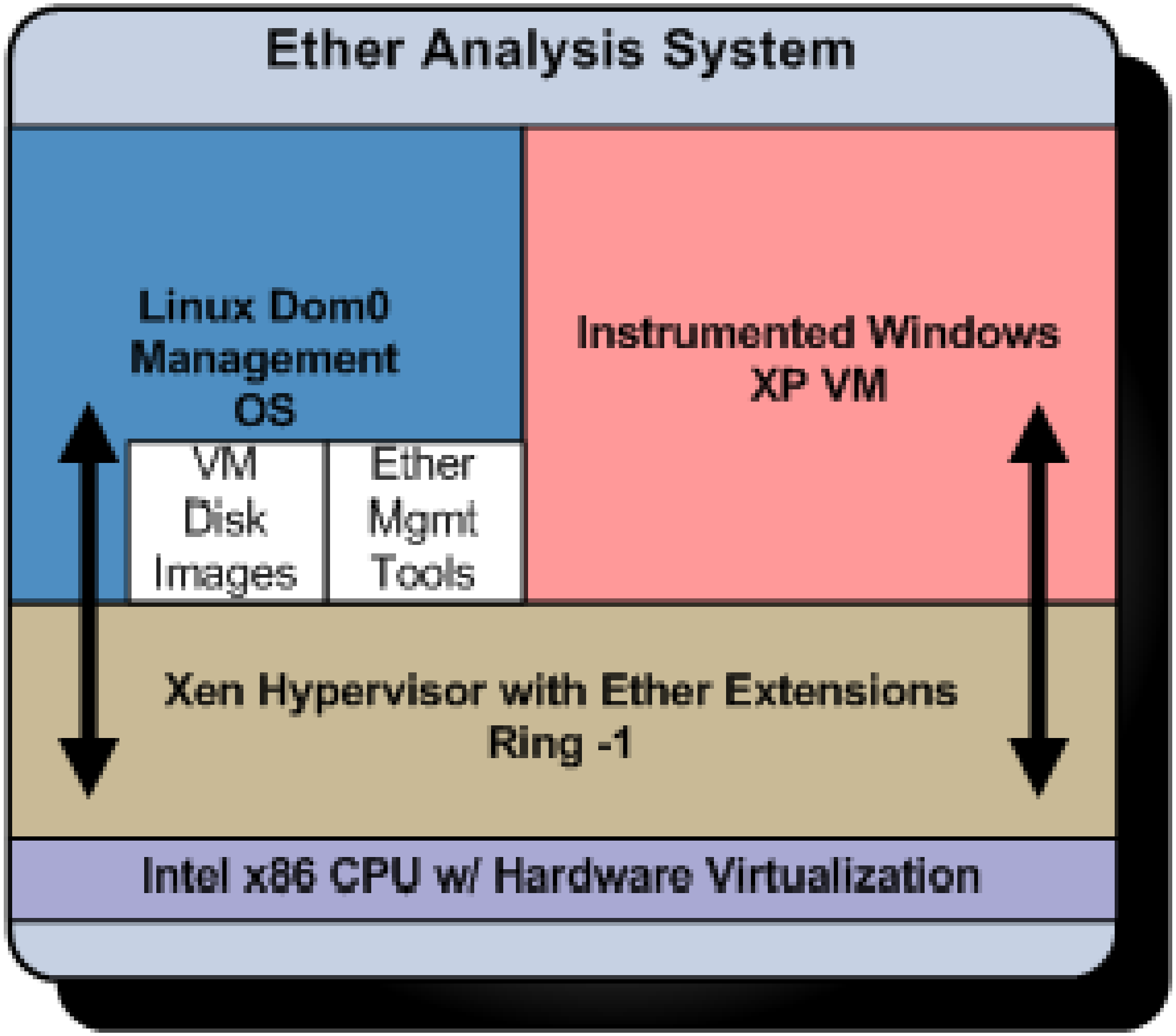
# Visualizing the OEP Problem

- Each block (vertex) represents a basic block executed in the user mode code
- Each line represents a transition
- The thicker the line, the more it was executed
- Colors represent areas of memory execution

# VERA

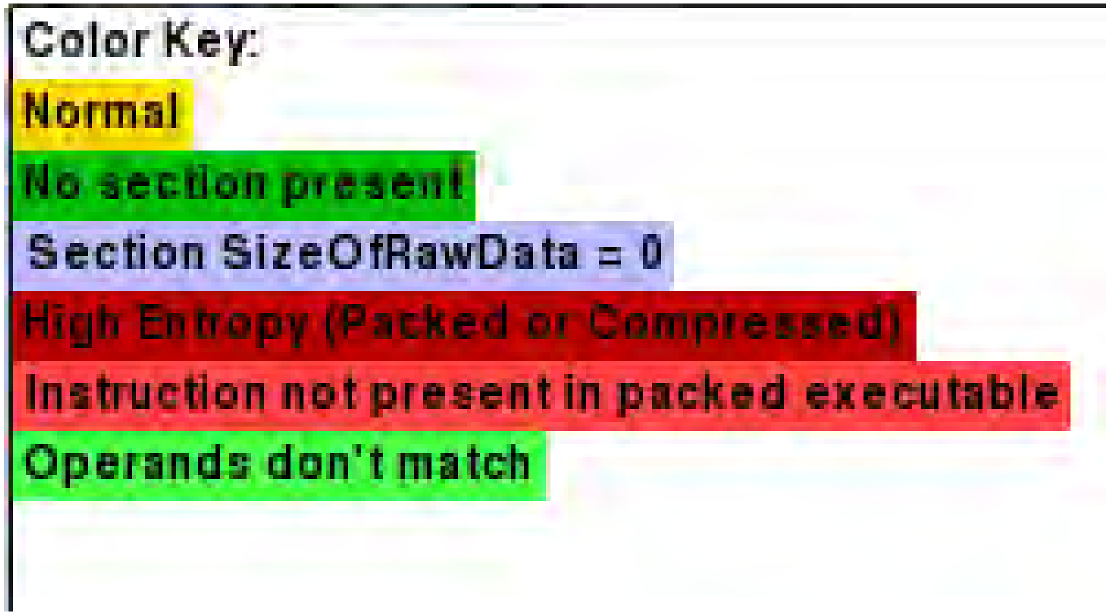
- Visualization of Executables for Reversing and Analysis
- Windows MFC Application
- Integrates with IDA Pro
- Fast, small memory footprint

# VERA Architecture


Ether Analysis System
Linux Dom0 Management OS
VM Disk Images
Ether Mgmt Tools
Instrumented Windows XP VM
Xen Hypervisor with Ether Extensions Ring -1
Intel x86 CPU w/ Hardware Virtualization

# Visualizing Packers

- Memory regions marked for PE heuristics



Color Key:

Normal

No section present

Section SizeOfRawData = 0

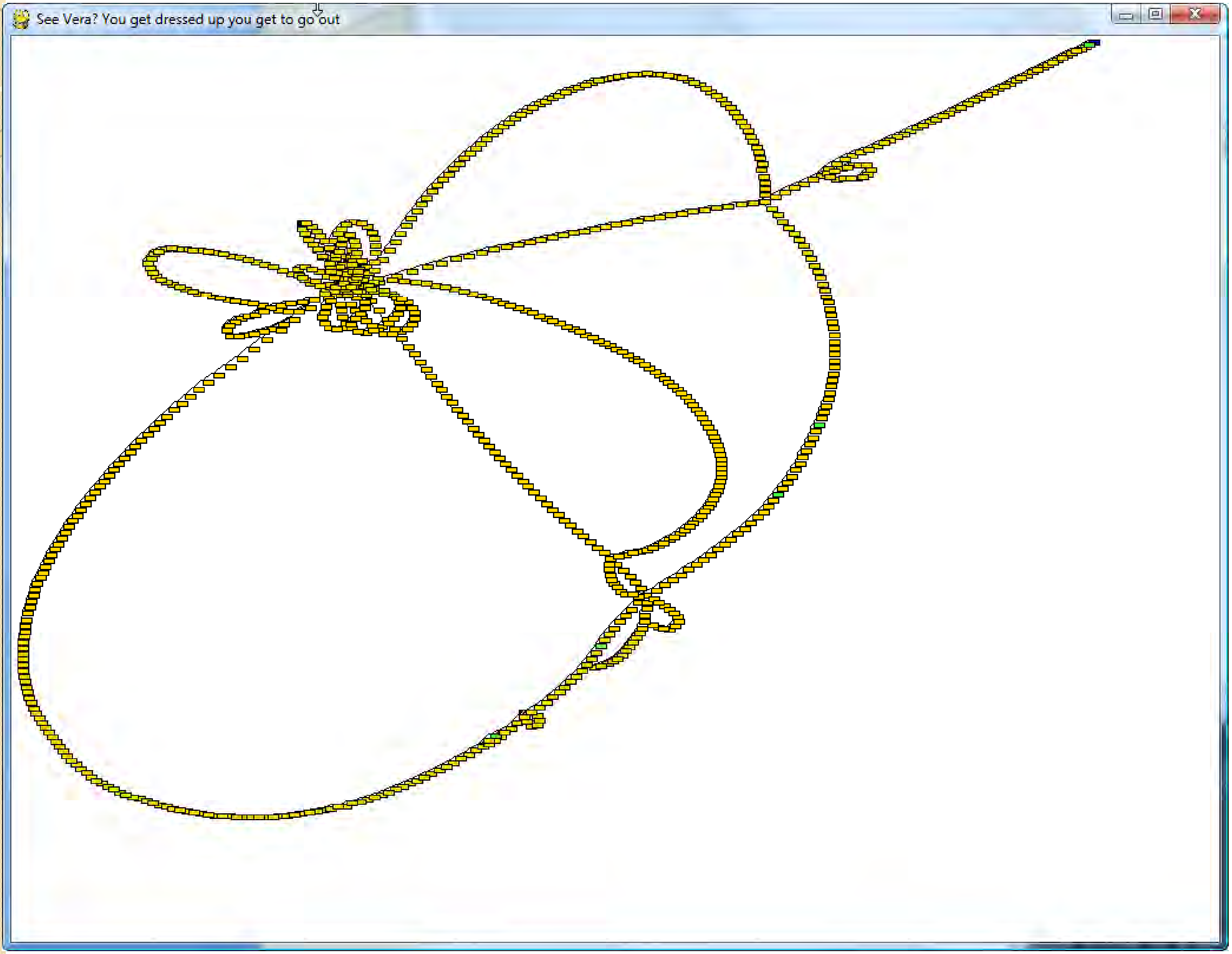
High Entropy (Packed or Compressed)

Instruction not present in packed executable

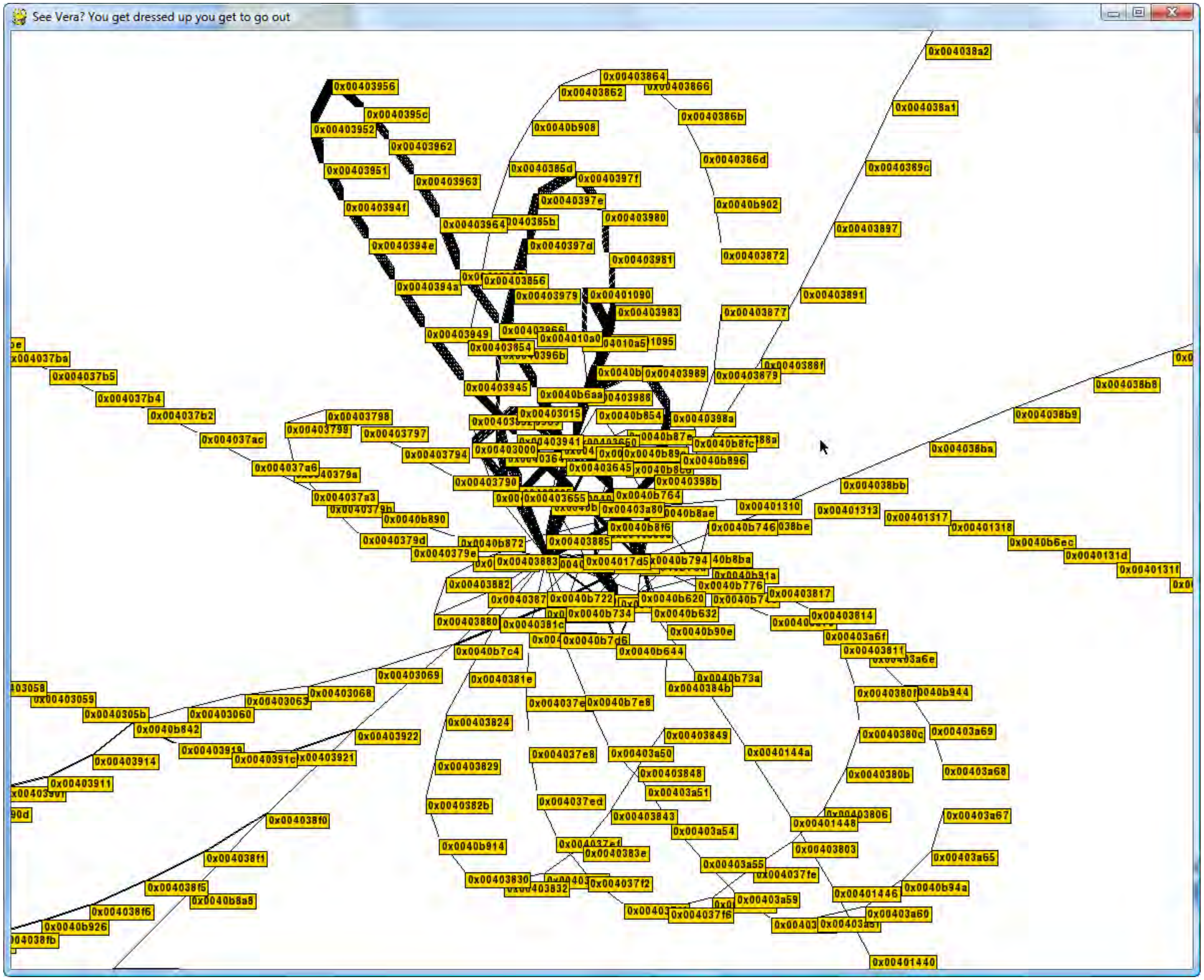
Operands don't match

# Demo!

# Netbull Virus (Not Packed)

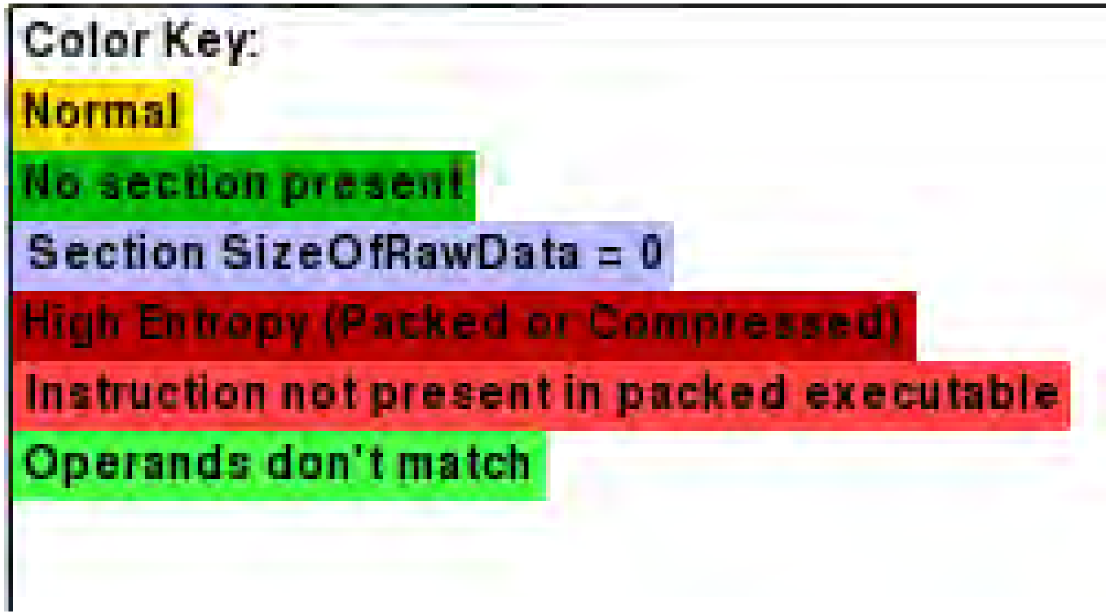


# Netbull Zoomed View



# Visualizing Packers

- Memory regions marked for PE heuristics



Color Key:

Normal

No section present

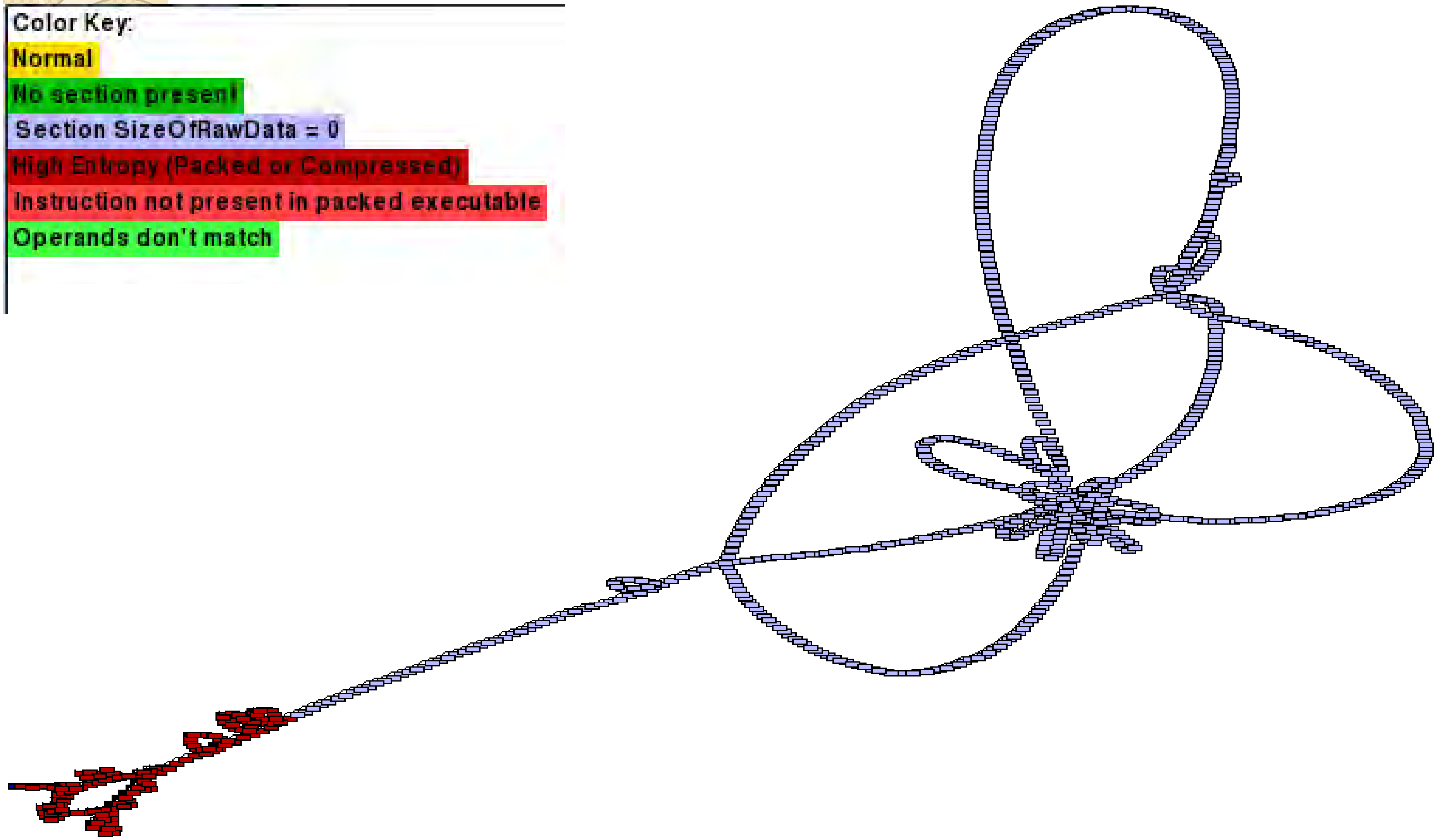
Section SizeOfRawData = 0

High Entropy (Packed or Compressed)

Instruction not present in packed executable

Operands don't match

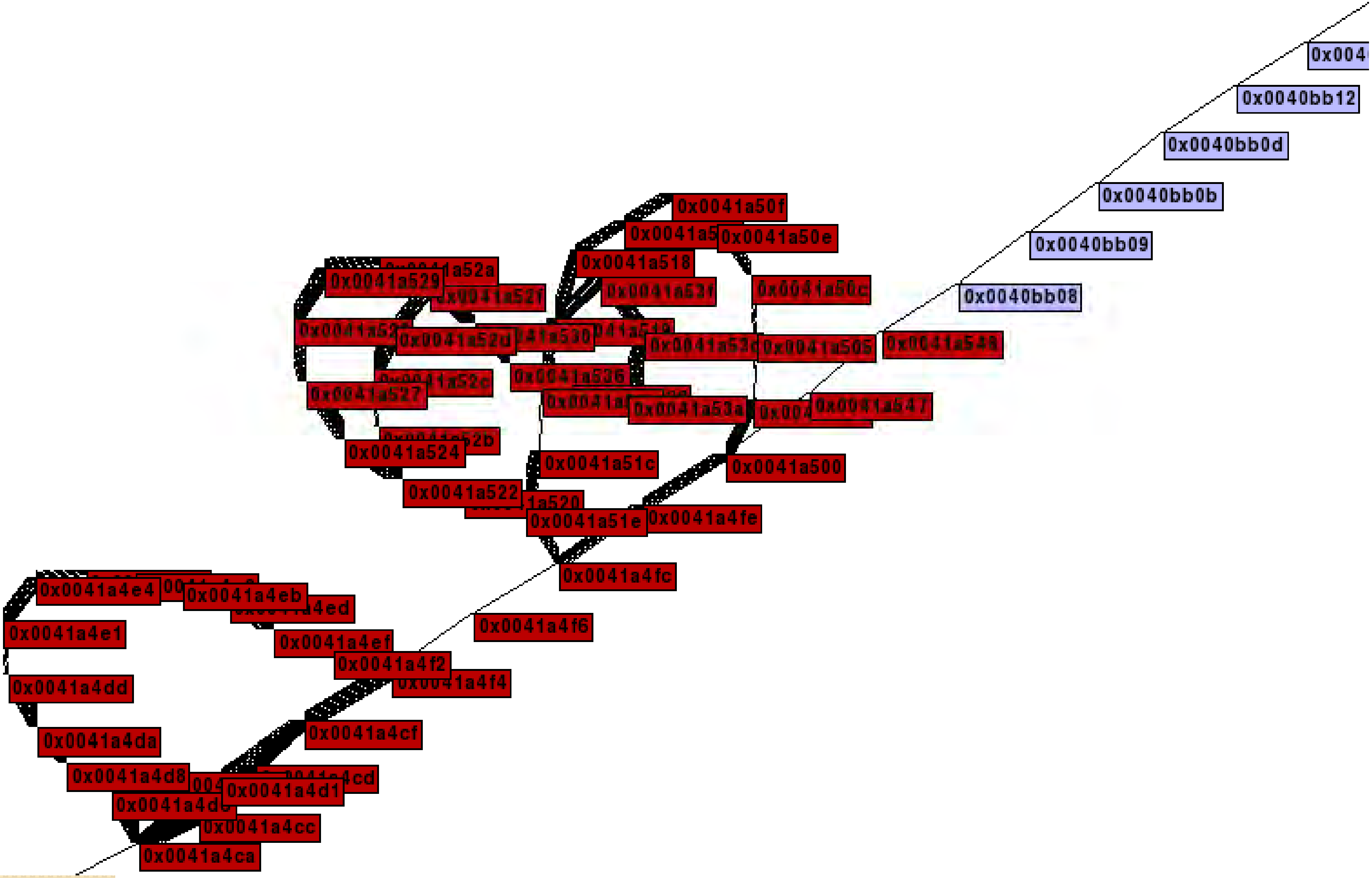
# UPX



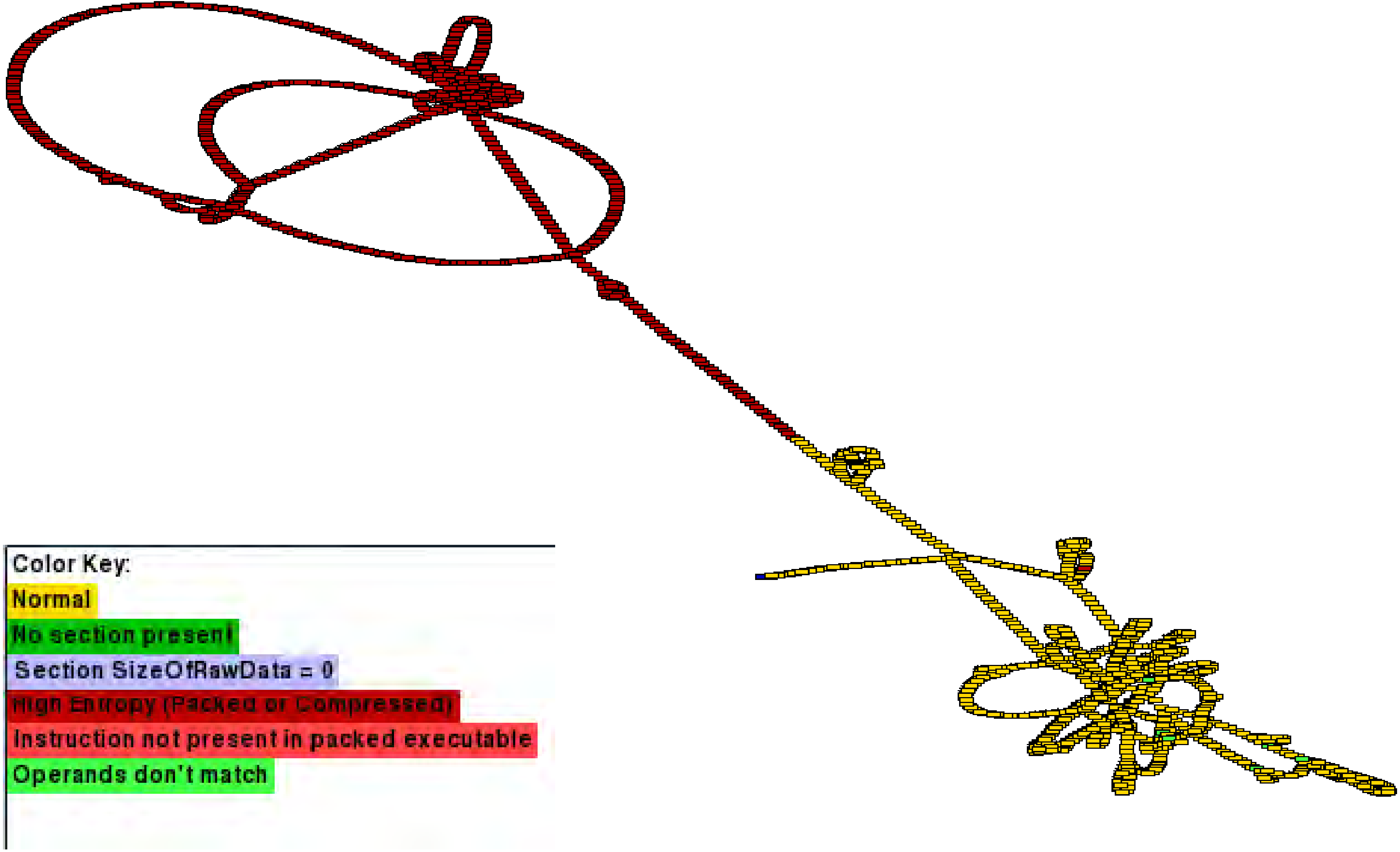
Color Key:

- Normal
- No section present
- Section SizeOfRawData = 0
- High Entropy (Packed or Compressed)
- Instruction not present in packed executable
- Operands don't match

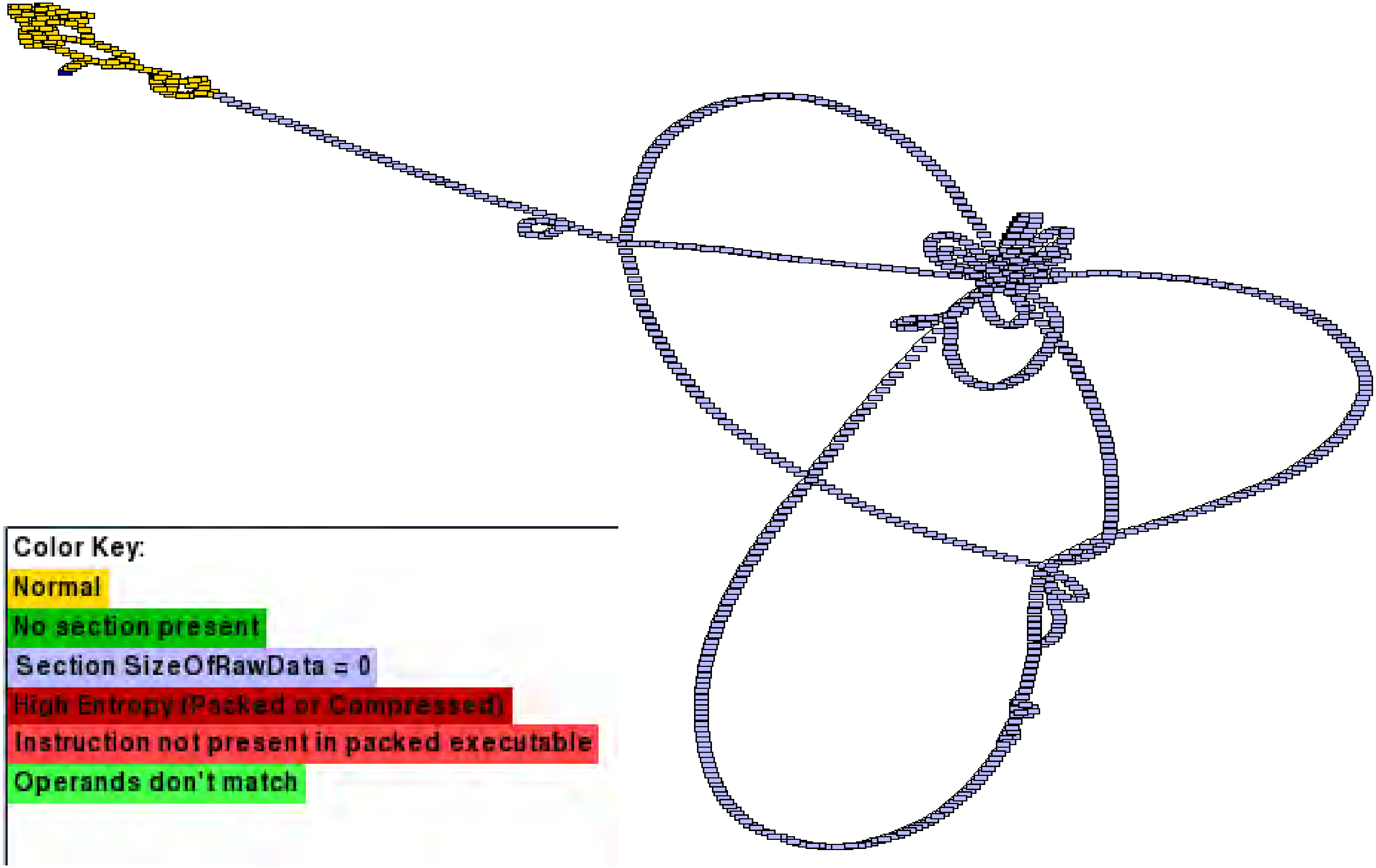
# UPX - OEP


0x004
0x0040bb12
0x0040bb0d
0x0040bb0b
0x0040bb09
0x0040bb08
0x0041a50f
0x0041a50e
0x0041a518
0x0041a52a
0x0041a529
0x0041a52f
0x0041a53f
0x0041a50c
0x0041a52d
0x0041a530
0x0041a505
0x0041a548
0x0041a536
0x0041a52c
0x0041a527
0x0041a53a
0x0041a547
0x0041a52b
0x0041a524
0x0041a51c
0x0041a500
0x0041a522
0x0041a51e
0x0041a4fe
0x0041a4fc
0x0041a4e4
0x0041a4eb
0x0041a4ed
0x0041a4e1
0x0041a4ef
0x0041a4f6
0x0041a4f2
0x0041a4f4
0x0041a4dd
0x0041a4cf
0x0041a4da
0x0041a4d8
0x0041a4cd
0x0041a4d1
0x0041a4cc
0x0041a4ca

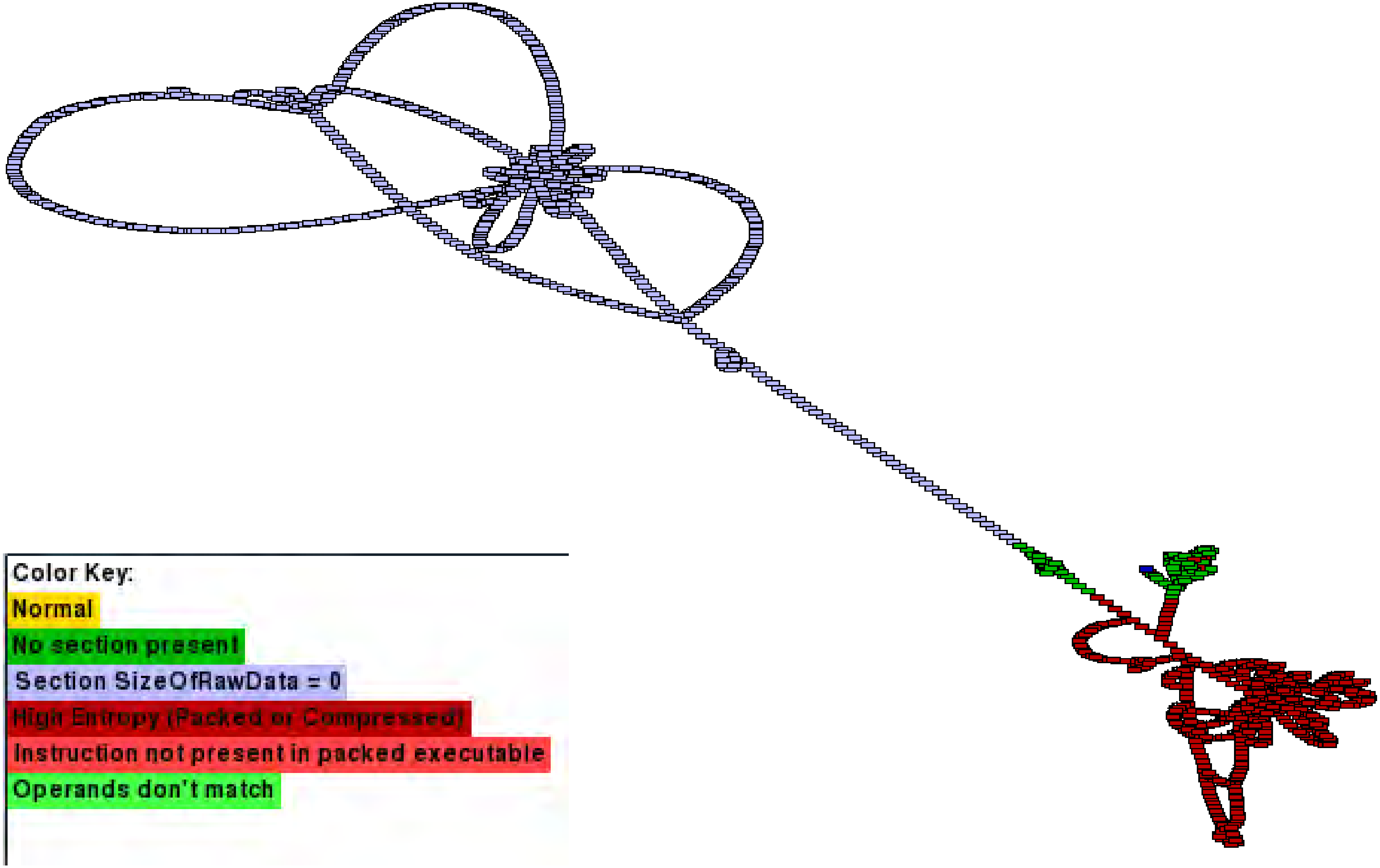
# ASPack


Color Key:
🟨 Normal
🟩 No section present
▨ Section SizeOfRawData = 0
🟥 High Entropy (Packed or Compressed)
🟥 Instruction not present in packed executable
🟩 Operands don't match

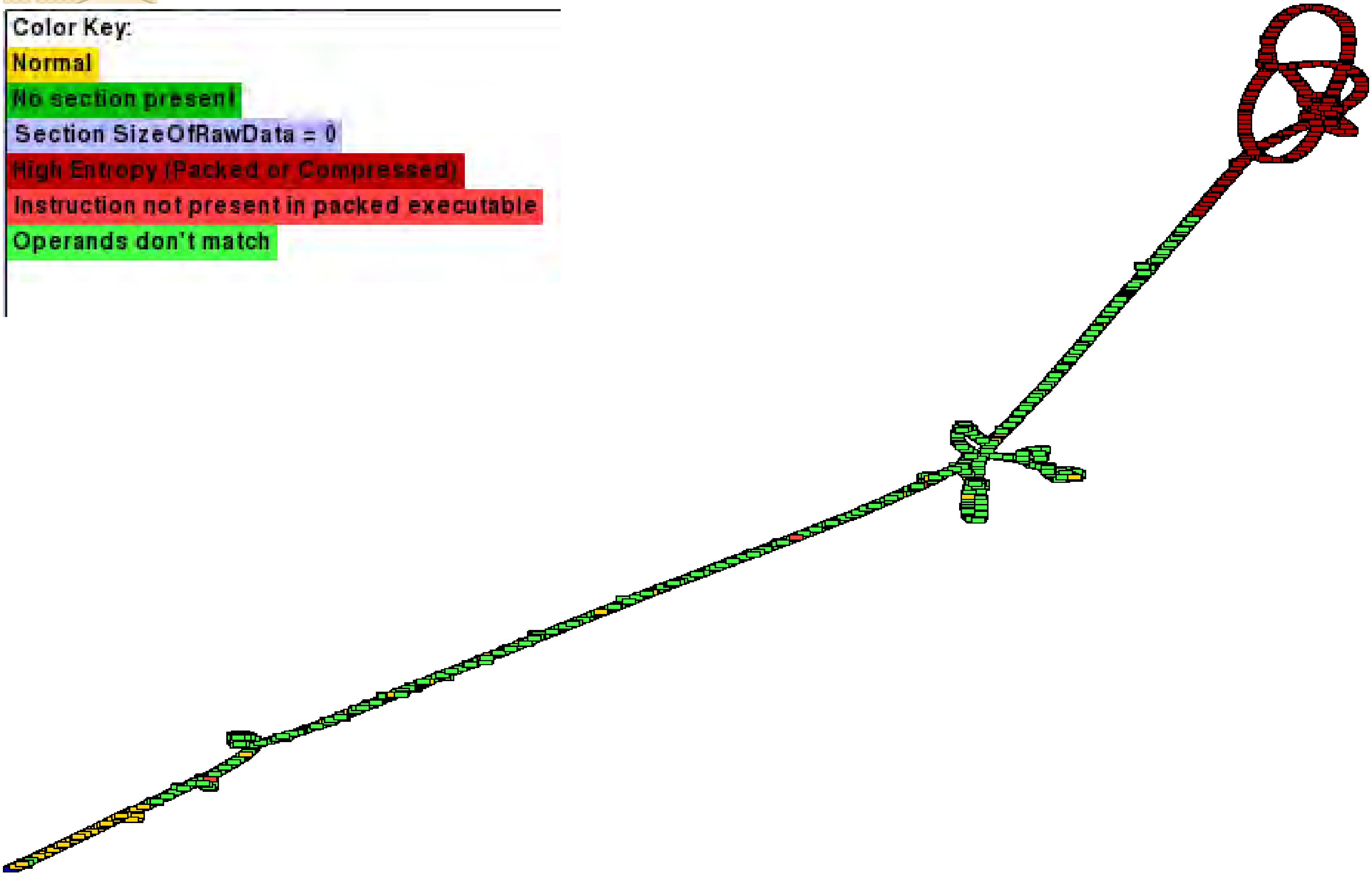
# FSG


Color Key:
Normal
No section present
Section SizeOfRawData = 0
High Entropy (Packed or Compressed)
Instruction not present in packed executable
Operands don't match

# MEW


Color Key:
Normal
No section present
Section SizeOfRawData = 0
High Entropy (Packed or Compressed)
Instruction not present in packed executable
Operands don't match

# TeLock


Color Key:
Normal
No section present
Section SizeOfRawData = 0
High Entropy (Packed or Compressed)
Instruction not present in packed executable
Operands don't match

# Future Work

- General GUI / bug fixes
- Integration with IDA Pro
- Memory access visualization
- System call integration
- Function boundaries
- Interactivity with unpacking process
- Modify hypervisor to work with WinDBG, OllyDbg, IDA Debugger

# Conclusions

- Visualizations make it easy to identify the OEP
- No statistical analysis of data needed
- Program phases readily identified
- Graphs are relatively simple
- Preliminary user study shows tool holds promise for speeding up reverse engineering

# Questions?

These slides are out of date! Find the latest ones at:

http://www.offensivecomputing.net/