Reverse Engineering by Crayon

Game Changing Hypervisor Based Malware Analysis and Visualization

Danny Quist

Lorie Liebrock

New Mexico Tech Computer Science Dept.

Offensive Computing, LLC

Blackhat / Defcon USA 2009

Danny Quist

- Offensive Computing, LLC - Founder
- Ph.D. Candidate at New Mexico Tech
- Reverse Engineer
- Instructor Reverse Engineering

Lorie Liebrock

- Computer Science Department Chair, New Mexico Tech
- Associate Professor
- New Mexico Tech Scholarship for Service Principal Investigator

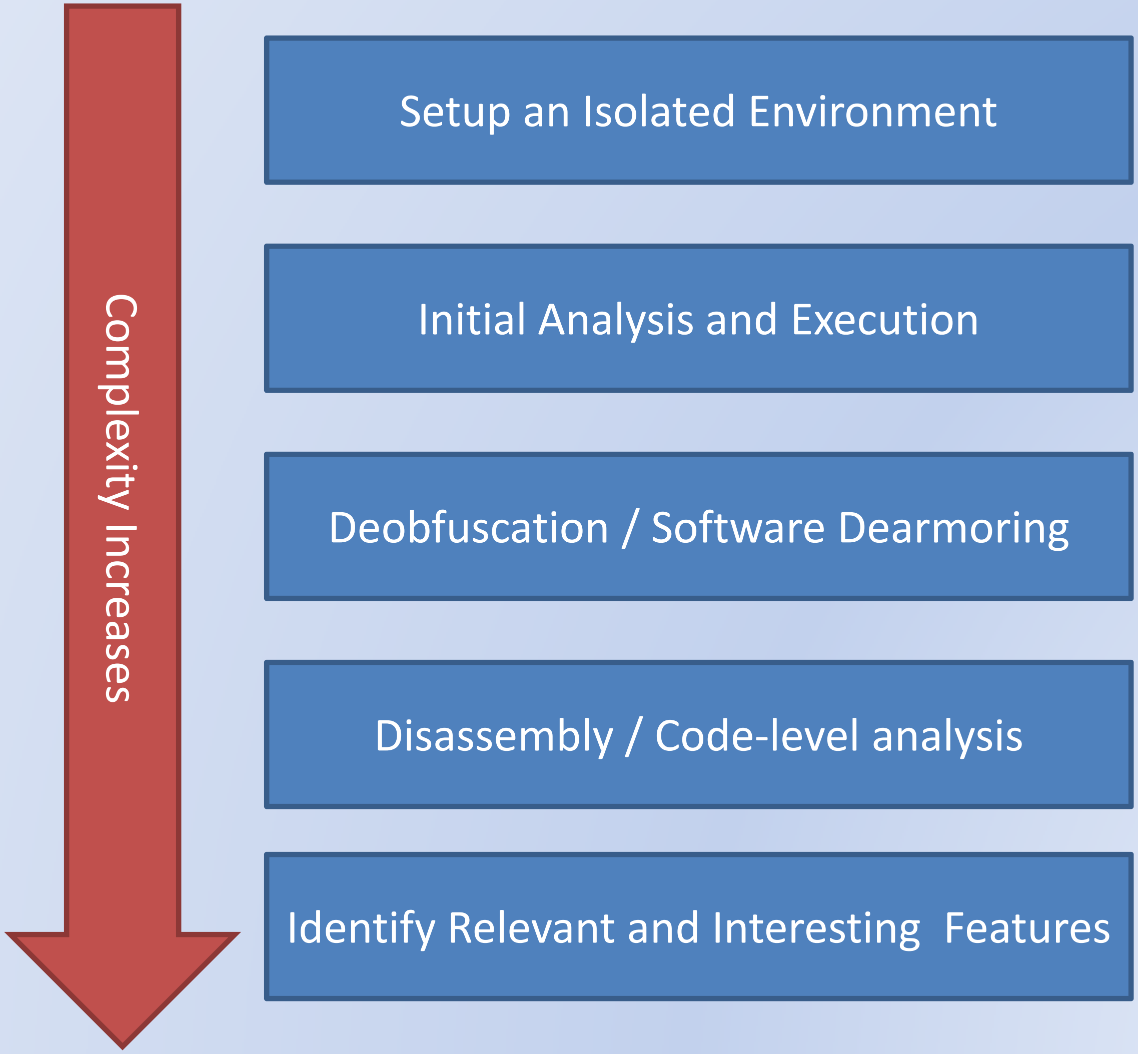
Overview

- Reverse Engineering Process
- Hypervisors and You
- Xen and Ether
- Modifying the Process
- VERA
- Real! Live! Reversing!
- Results

Overview

- **Reverse Engineering Process**
- Hypervisors and You
- Xen and Ether
- Modifying the Process
- VERA
- Real! Live! Reversing!
- Results

Reversing Process



Complexity Increases

Setup an Isolated Environment

- VMWare, Xen, Virtual PC
- Dedicated Hardware

Initial Analysis and Execution

- Sysinternals, CWSandbox
- Look for OS State Changes
  - Files, registry, network

Deobfuscation / Software Dearmoring

- Unpacking
- Debuggers, Saffron, Ether

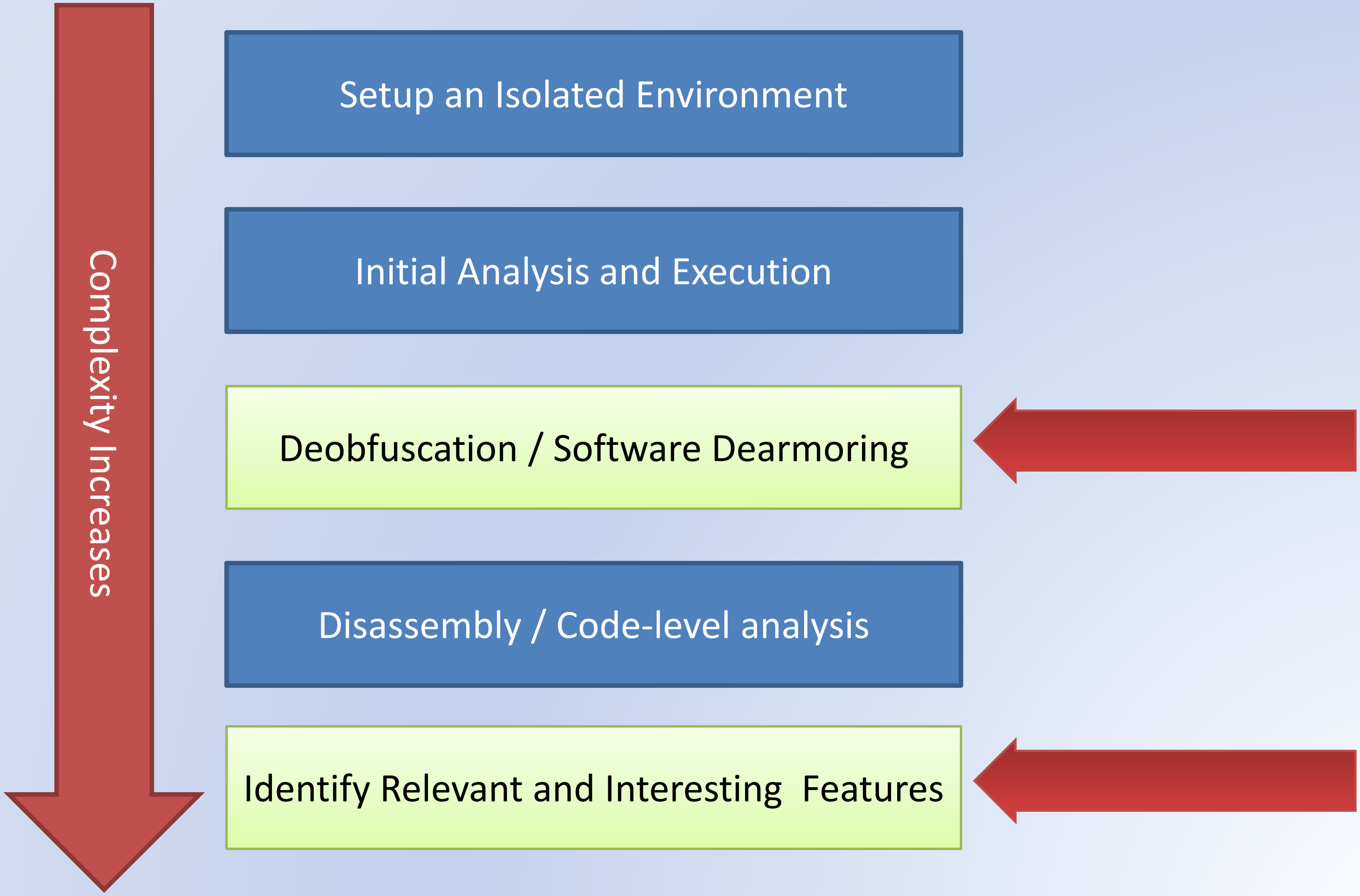
Disassembly / Code-level analysis

- IDA Pro
- OllyDbg

Identify Relevant and Interesting Features

- Experience based
- Newbies have trouble with this

What We Want to Address



Complexity Increases

Setup an Isolated Environment

Initial Analysis and Execution

Deobfuscation / Software Dearmoring

Disassembly / Code-level analysis

Identify Relevant and Interesting Features

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

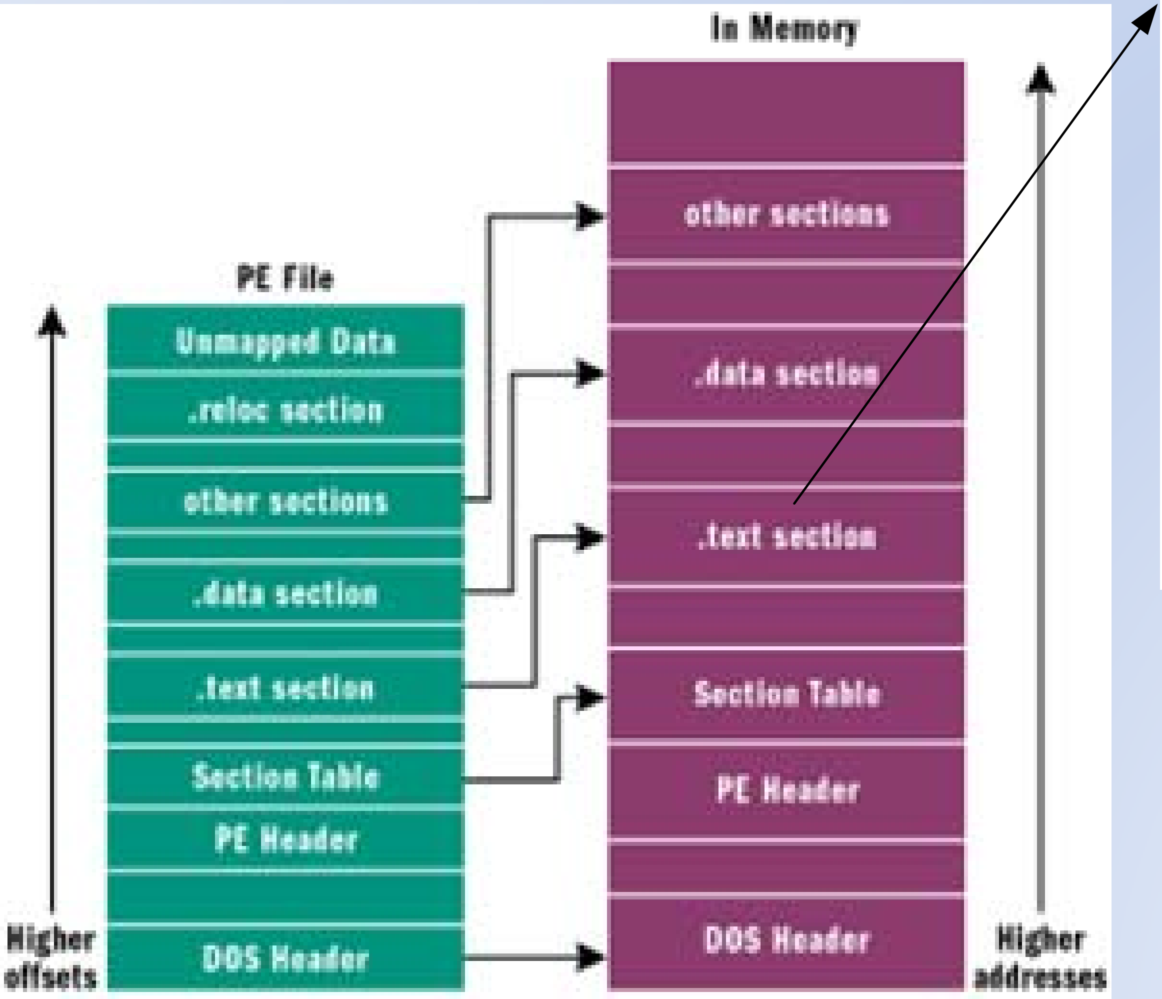
Remove Software Armoring

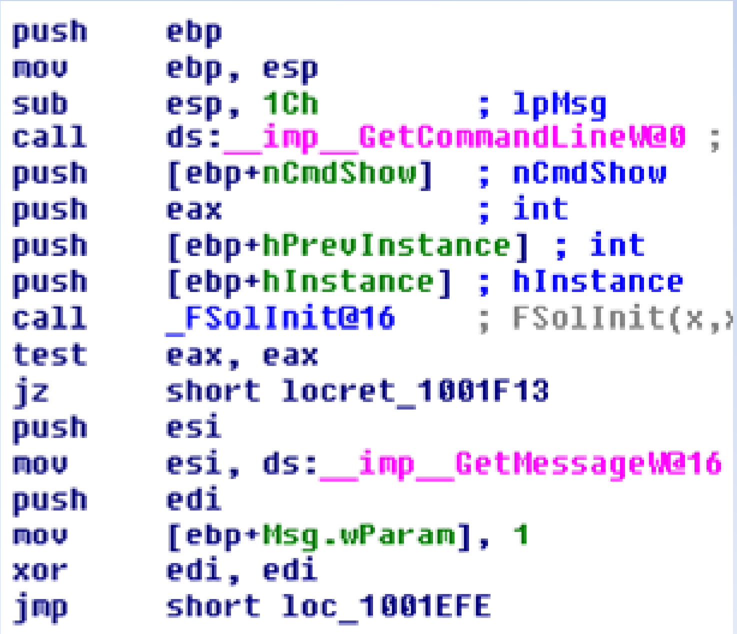
- Program protections to prevent reverse engineering
- Done via packers – Small encoder/decoder
- Self-modifying code
- Lots of research about this
  - OllyBonE, Saffron, Polyunpack, Renovo, Ether, Azure
  - Our research uses Ether

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

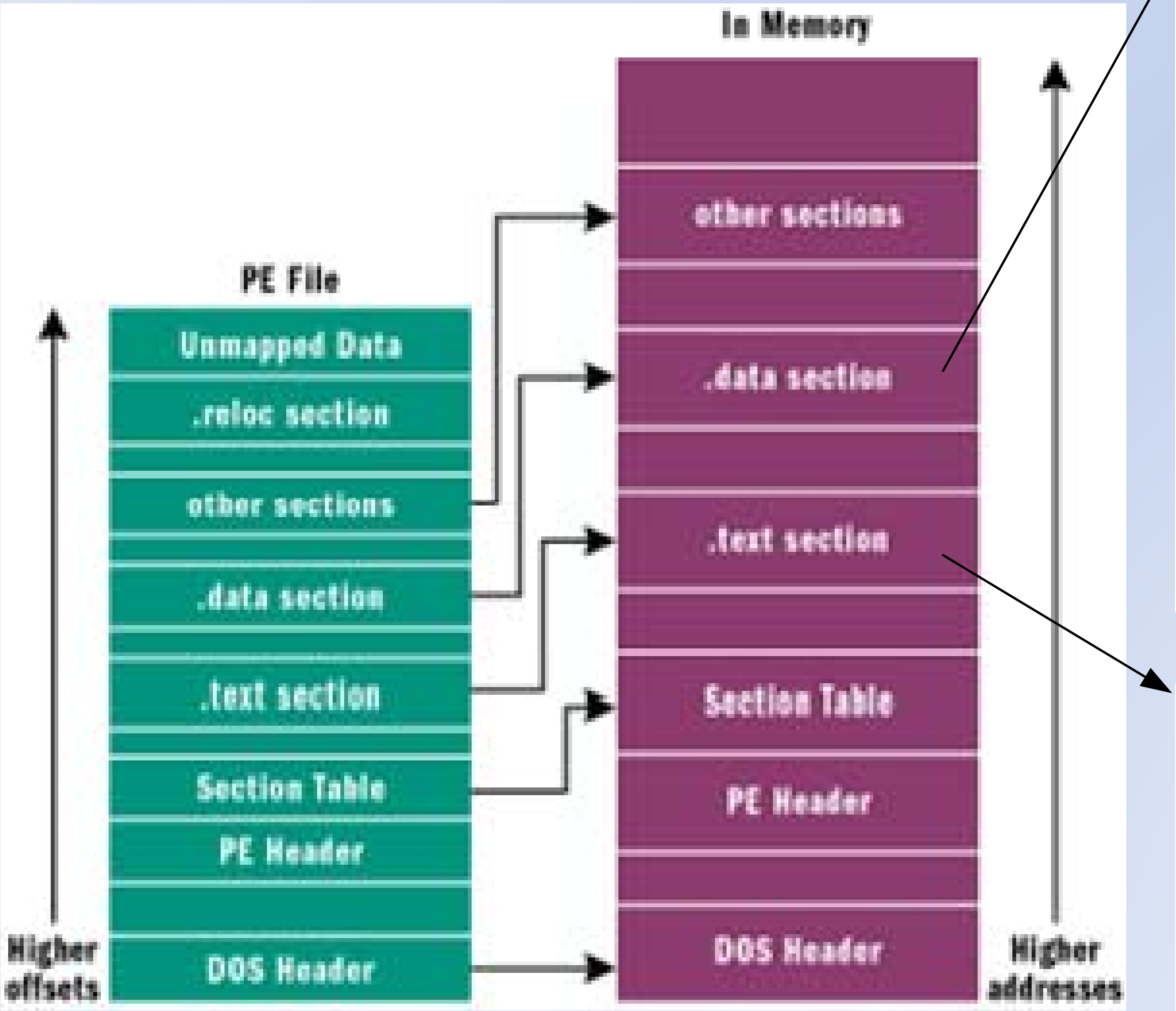
Normal PE File

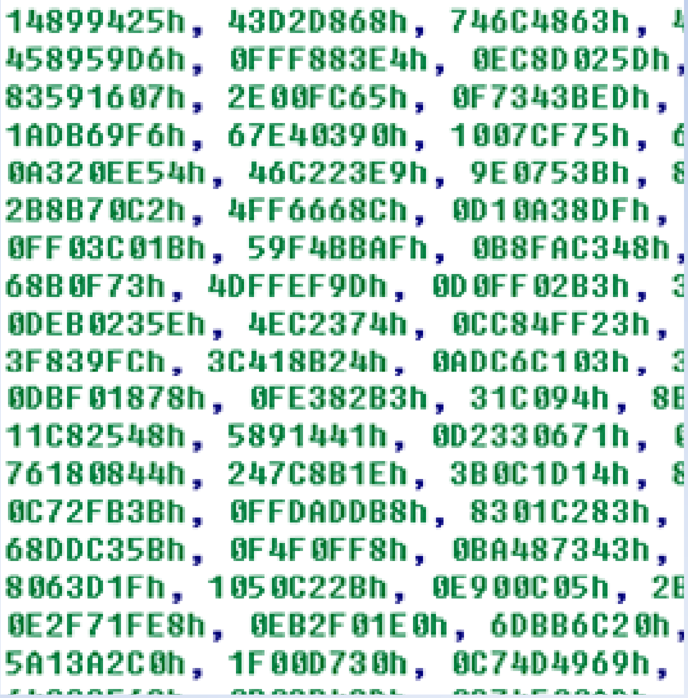




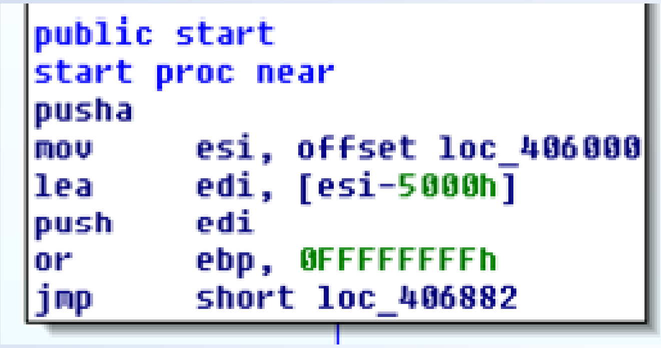
```
push    ebp
mov     ebp, esp
sub     esp, 1Ch          ; lpMsg
call    ds:__imp__GetCommandLineW@0 ;
push    [ebp+nCmdShow]    ; nCmdShow
push    eax               ; int
push    [ebp+hPrevInstance] ; int
push    [ebp+hInstance] ; hInstance
call    _FSolInit@16      ; FSolInit(x,
test    eax, eax
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, 0F7343BEDh,
1ADB69F6h, 67E40390h, 1007CF75h, 6
0A320EE54h, 46C223E9h, 9E0753Bh, 8
2B8B70C2h, 4FF6668Ch, 0D10A38DFh,
0FF03C01Bh, 59F4BBAFh, 0B8FAC348h,
68B0F73h, 4DFFEF9Dh, 0D0FF02B3h, 3
0DEB0235Eh, 4EC2374h, 0CC84FF23h,
3F839FCh, 3C418B24h, 0ADC6C103h, 3
0DBF01878h, 0FE382B3h, 31C094h, 8B
11C82548h, 5891441h, 0D2330671h, 0
76180844h, 247C8B1Eh, 3B0C1D14h, 8
0C72FB3Bh, 0FFDADDB8h, 8301C283h,
68DDC35Bh, 0F4F0FF8h, 0BA487343h,
8063D1Fh, 1050C22Bh, 0E900C05h, 2B
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
  - “Attacks on Virtual Machine Emulators” Peter Ferrie, Symantec
    http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf
- 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
- Problem
  - Requires dedicated hardware

Disassembly and Code Analysis

- Most nebulous portion of the process
- Largely depends on intuition
- Looking at assembly is tedious
- Suffers from “not seeing the forest for 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

Overview

- Reverse Engineering Process
- **Hypervisors and You**
- Xen and Ether
- Modifying the Process
- VERA
- Real! Live! Reversing!
- Results

Hypervisors

- Lots of hype over the past few years
- New hypervisor rootkits lead defensive tools
  Rutkowska, Tereshkin, Ptacek, et. Al.
- Covert methods for analyzing runtime behavior are extremely useful
- Detection of hardware virtualization not widely implemented

Hypervisor Implementations

- VMWare ESX Server
  - Commercial grade solution for VMs
  - Avoids VM detection issues (mostly)
- Linux Kernel Virtual Machines (KVM)
  - Separates analysis OS from target OS (slightly safer?)
  - Uses well-tested Linux algorithms for resource management
- Xen
  - Excellent set of tools for introspection
  - Uses standard QEMU image formats
  - API controlled via Python – Integration into tools is easier

Overview

- Reverse Engineering Process
- Hypervisors and You
- **Xen and Ether**
- Modifying the Process
- VERA
- Real! Live! Reversing!
- Results

What is Ether?

- Patches to the Xen Hypervisor
- Instruments a Windows system
- Base modules available
  - Instruction tracing
  - API tracing
  - Unpacking
- "Ether: Malware Analysis via Hardware Virtualization Extensions"
  Dinaburg, Royal, Sharif, Lee

  ACM CCS 2008

Ether Event Tracing

- Detects events on an instrumented system
  - System call execution
  - Instruction execution
  - Memory writes
  - Context switches
- Covert monitoring
  - No modifications to the system means no detection

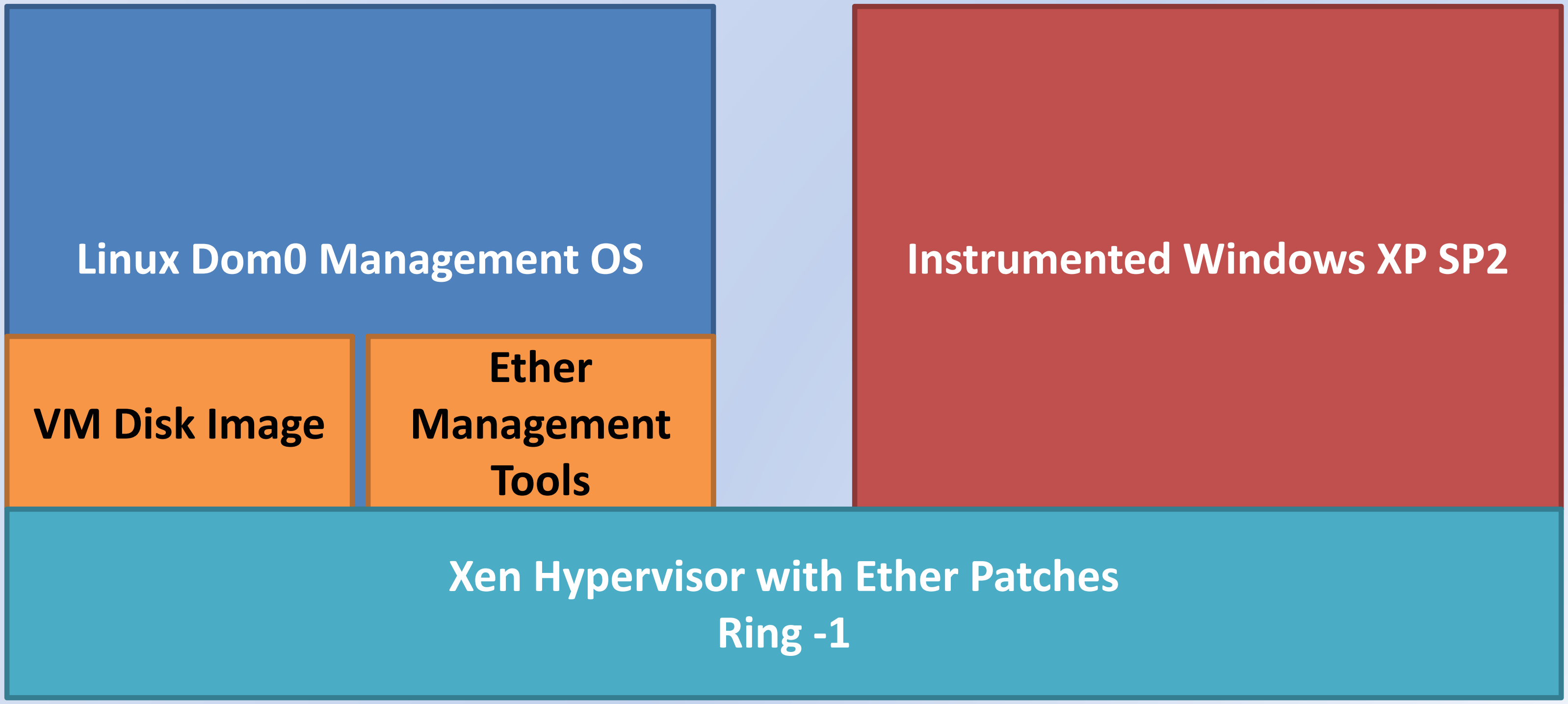
Instruction Tracing

- EFLAGS register modified for single-step (trap flag)
- PUSHF and POPF instructions are intercepted
  - Only direct way for reading and modifying the trap flag
- Modifications to this single-stepping effectively hidden

Memory and System Calls

- Memory Writes
  - Tracked by manipulating the shadow page table
  - Gives access to the written and read memory addresses
- System Calls
  - Modifies the SYSENTER_EIP register to point to non-paged address space
  - Logged, returned to Ether
  - Overrides 0x2e interrupt to catch older syscalls

Ether System Architecture



Linux Dom0 Management OS

VM Disk Image

Ether Management Tools

Instrumented Windows XP SP2

Xen Hypervisor with Ether Patches
Ring -1

Extensions to Ether

- Moved unpacking code from hypervisor into user-space
- Better user mode analysis
- PE repair system – Allows for disassembly of executables
- Added enhanced monitoring system for executables

User mode Unpacking

- Watch for and monitor all memory writes
- Allow program to execute
- When execution occurs in written memory, dump memory
- Each dump is a candidate for the OEP
- Not perfect, but decent
- Scaffolding for future modifications

PE Repair

- Dumped PE files had problems
  - Sections were not file aligned
  - Address of Entry Point invalid
  - Would not load in IDA correctly
- Ported OllyDump code to Ether user mode
  - Fix section offsets to match data on disk
  - Repair resources as much as possible
  - Set AddressOfEntryPoint to be the candidate OEP

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

Ether Unpacking Demo!

Open Problems

- Unpacking process produces lots of candidate dump files
- Better Original Entry Point discovery method
- Import rebuilding is still an issue
- Now that there is a nice tool for tracing programs covertly, we need to do analysis

Overview

- Reverse Engineering Process
- Hypervisors and You
- Xen and Ether
- **Modifying the Process**
- VERA
- Real! Live! Reversing!
- Results

Modifying the Process

- Knowing what to look for is often what most new reversers have trouble with
- Having an idea of the execution flow of a program is extremely useful
  - IDA is focused on the function view
  - Extend to the basic block view
- Software armoring removal made easy

Overview

- Reverse Engineering Process
- Hypervisors and You
- Xen and Ether
- Modifying the Process
- **VERA**
- Real! Live! Reversing!
- Results

Visualization of Ether 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

NEW MEXICO TECH
SCIENCE • ENGINEERING • RESEARCH • UNIVERSITY

offensive computing

VERA

- **V**isualization of **E**xecutables for **R**eversing and **A**nalysis
- Windows MFC Application
- Integrates with IDA Pro
- Fast, small memory footprint

VERA: Graphs

- Each vertex (node) represents an address
- Each edge represents execution
- Thicker edges represent larger execution
- Two display modes:
  - Basic blocks
  - Instructions

Vertices (Nodes)

- Basic blocks
  - Fundamental small grouping of code
  - Reduces data size
  - Useful for large commercial programs
- Instructions
  - Useful for small programs
  - Greater aesthetic value
  - Larger datasets can produce useless graphs

Edges (Lines)

- Transitions between addresses
- Thicker lines represent more executions
  - Easy identification of loops
  - Find heavy concentration of execution
- Multiple edges from a node represent decision point

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

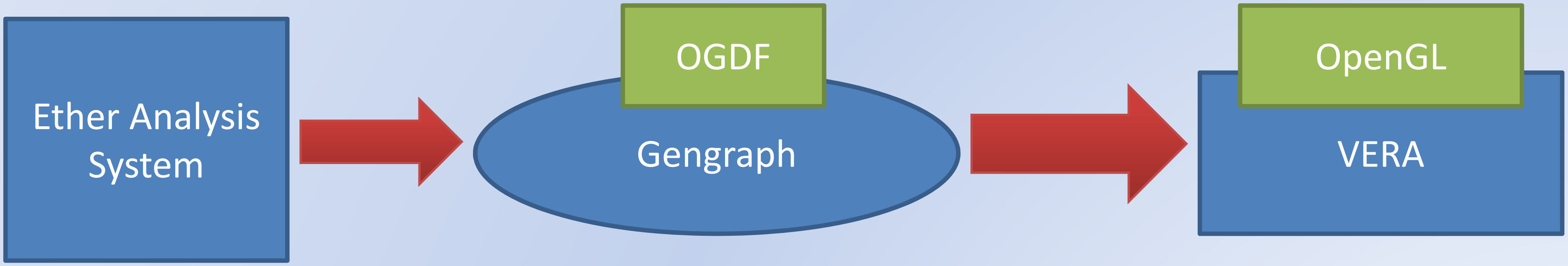
Colors

- 🟨 **Yellow** – Normal uncompressed low-entropy section data
- 🟩 **Dark Green** – Section not present in the packed version
- 🟪 **Light Purple** – SizeOfRawData = 0
- 🟥 **Dark Red** – High Entropy
- 🟥 **Light Red** – Instructions not in the packed exe
- 🟩 **Lime Green** – Operands don’t match

Colors

- Chosen arbitrarily (aesthetically?)
- Alternate set available for red-green color blind users
  - Uncomment in the code if you want this
  - Change it to your own
- Feedback would be appreciated

VERA Architecture



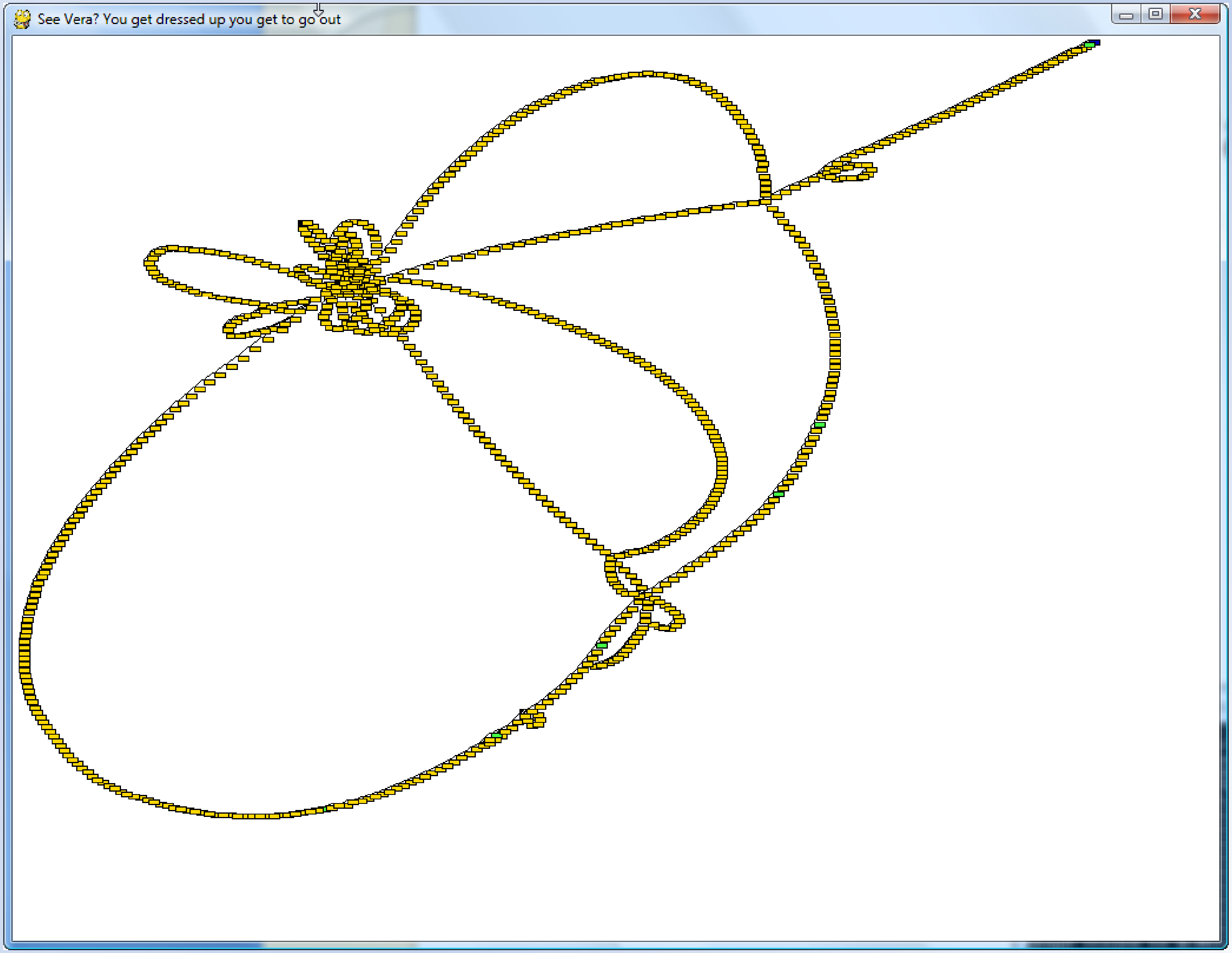
Open Graph Display Framework
- Handles all layout and arrangement of the graphs
- Similar to Graphviz
- Works with large datasets

Using Vera

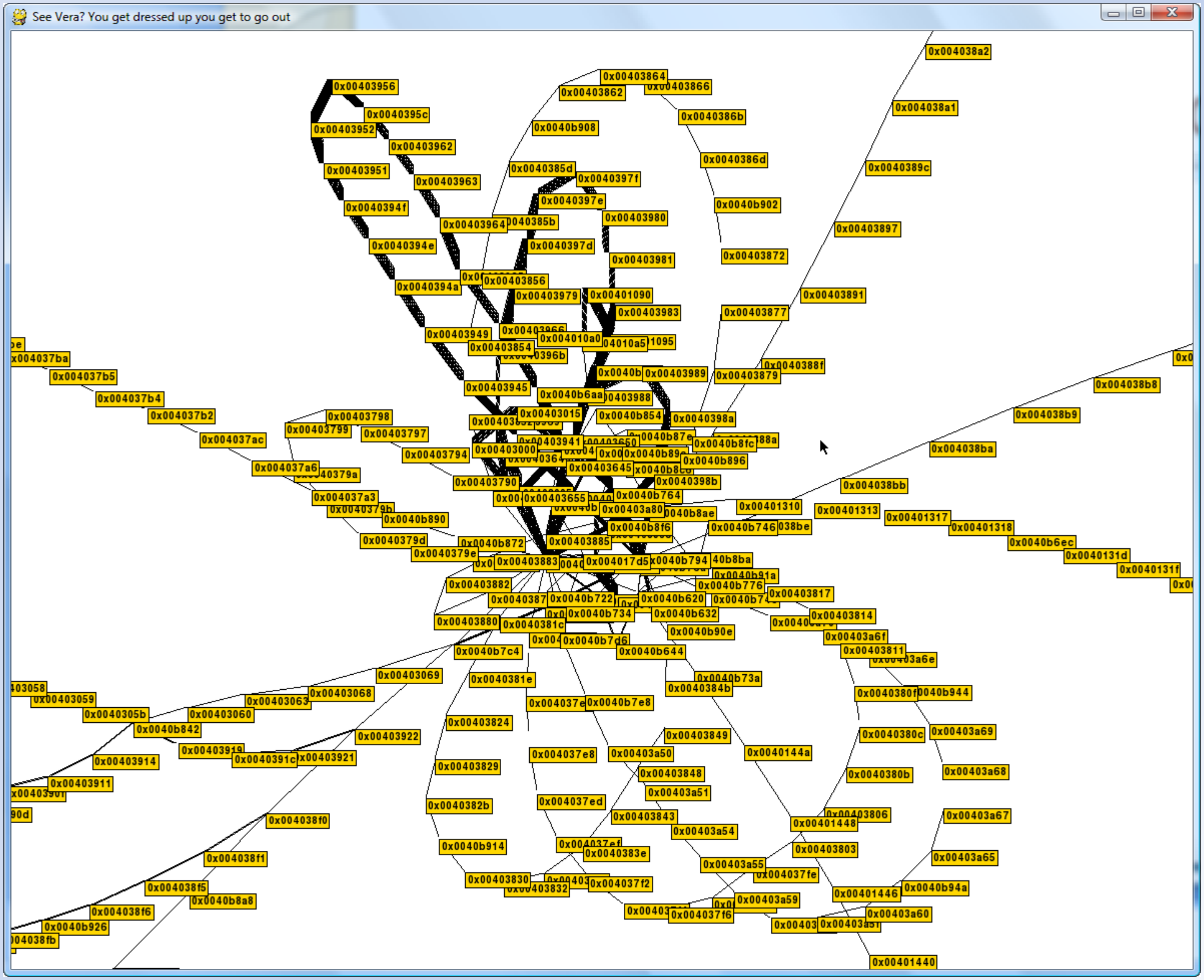
- Run an instruction trace with Ether
- Transfer the trace file to your analysis box
- Run gengraph.exe on the output
- Open the resulting .GML file in Vera
- Correlate data with the graph

Vera Demo!

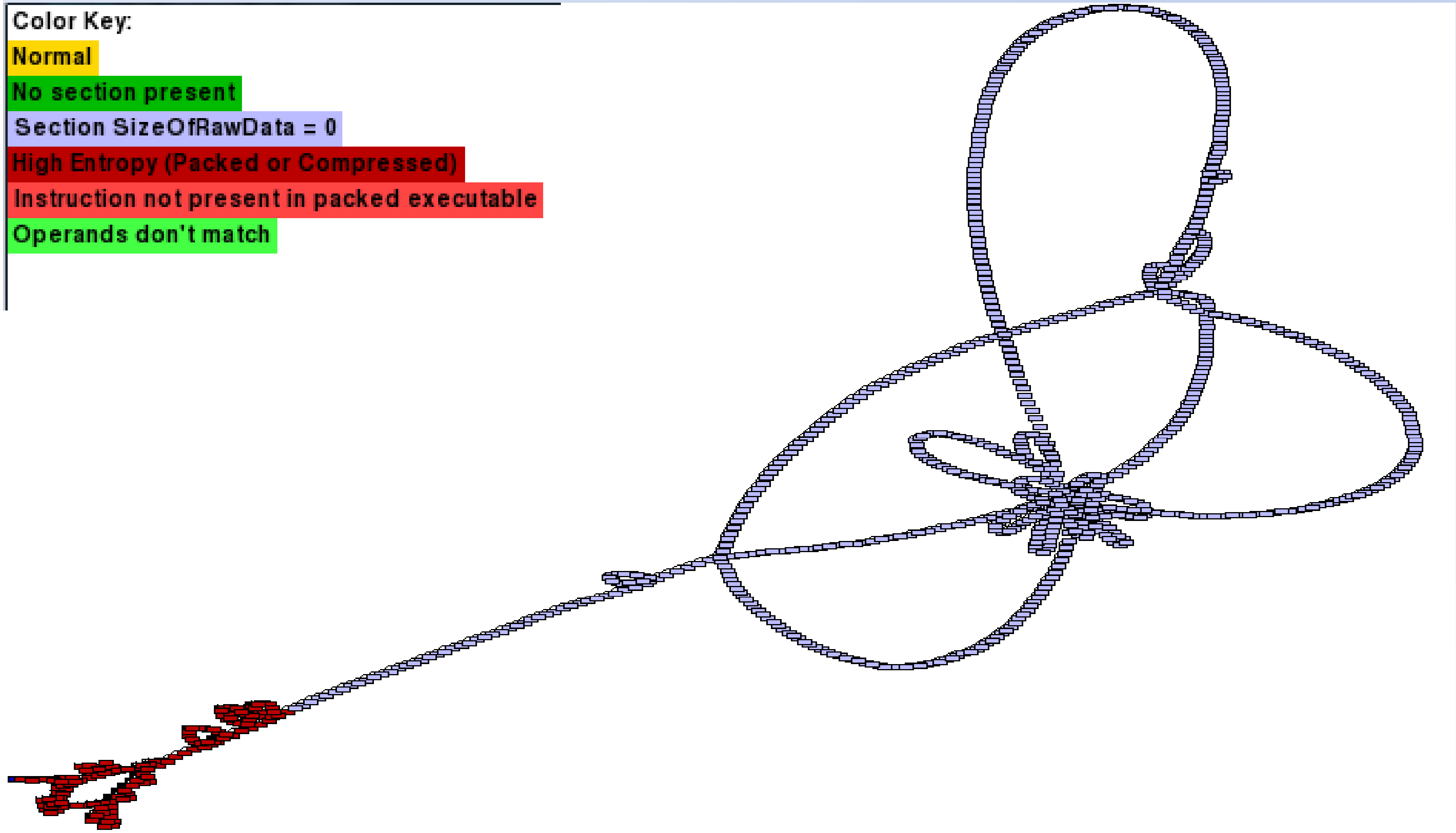
Netbull Virus (Not Packed)



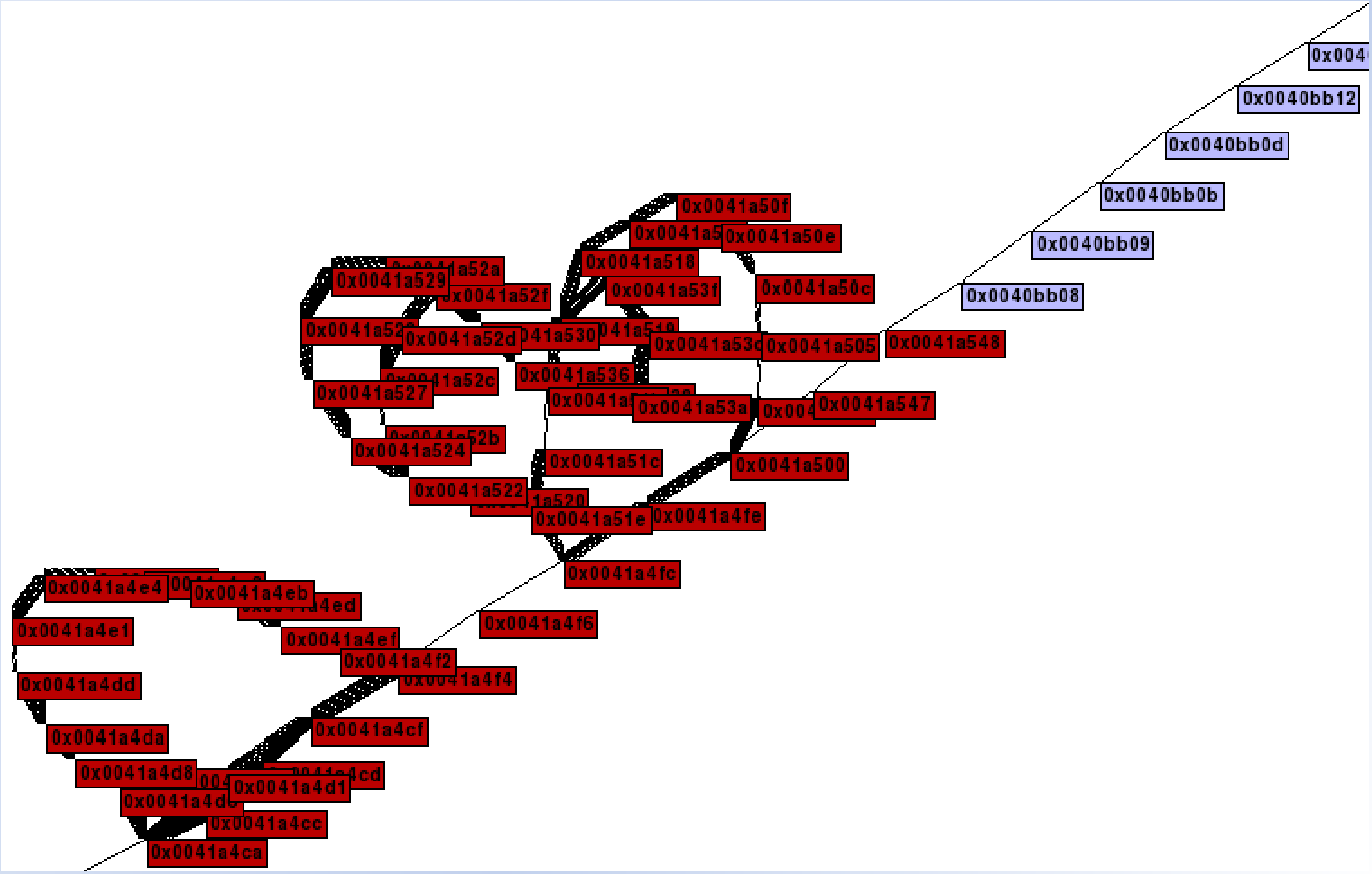
Netbull Zoomed View



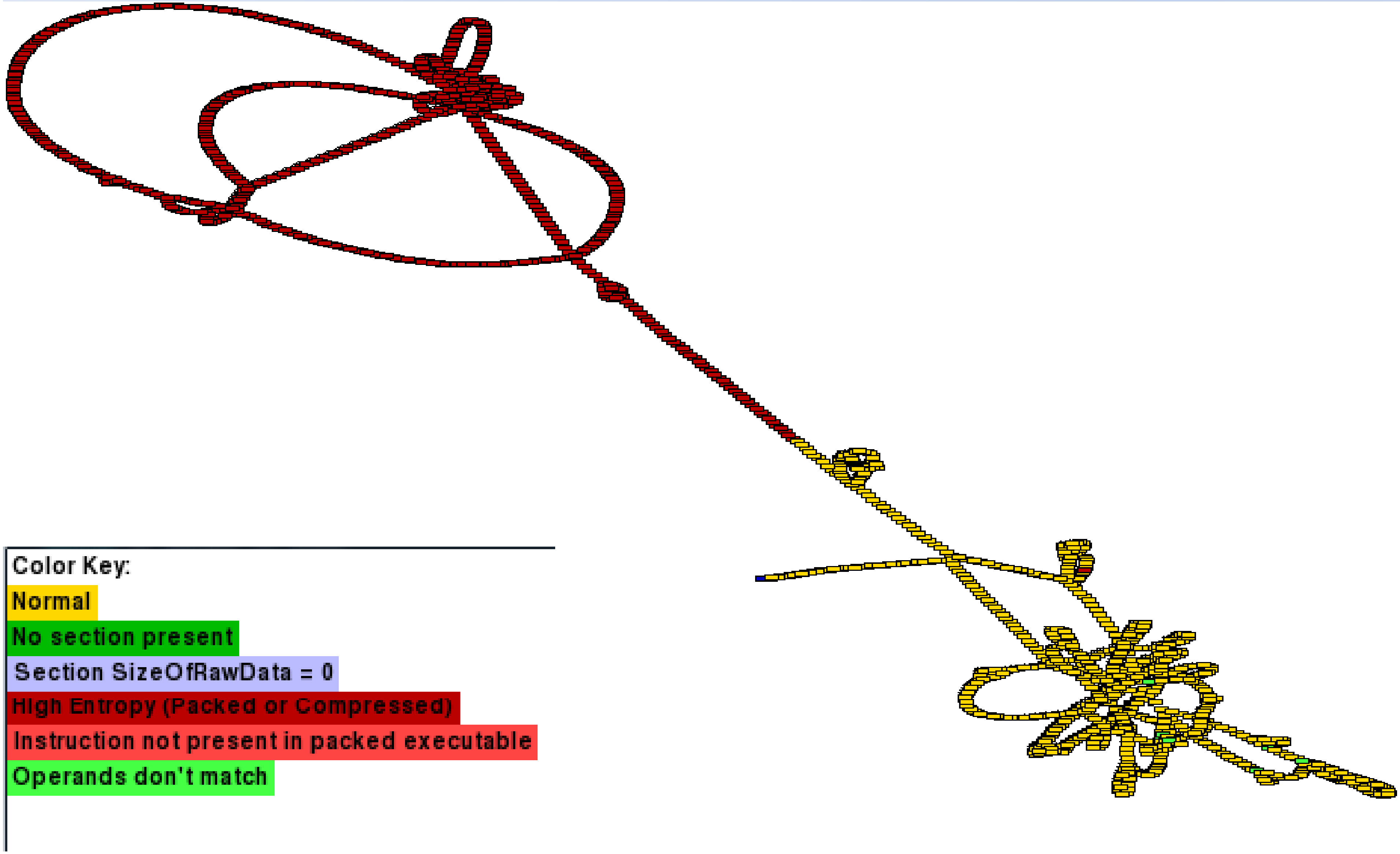
UPX


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

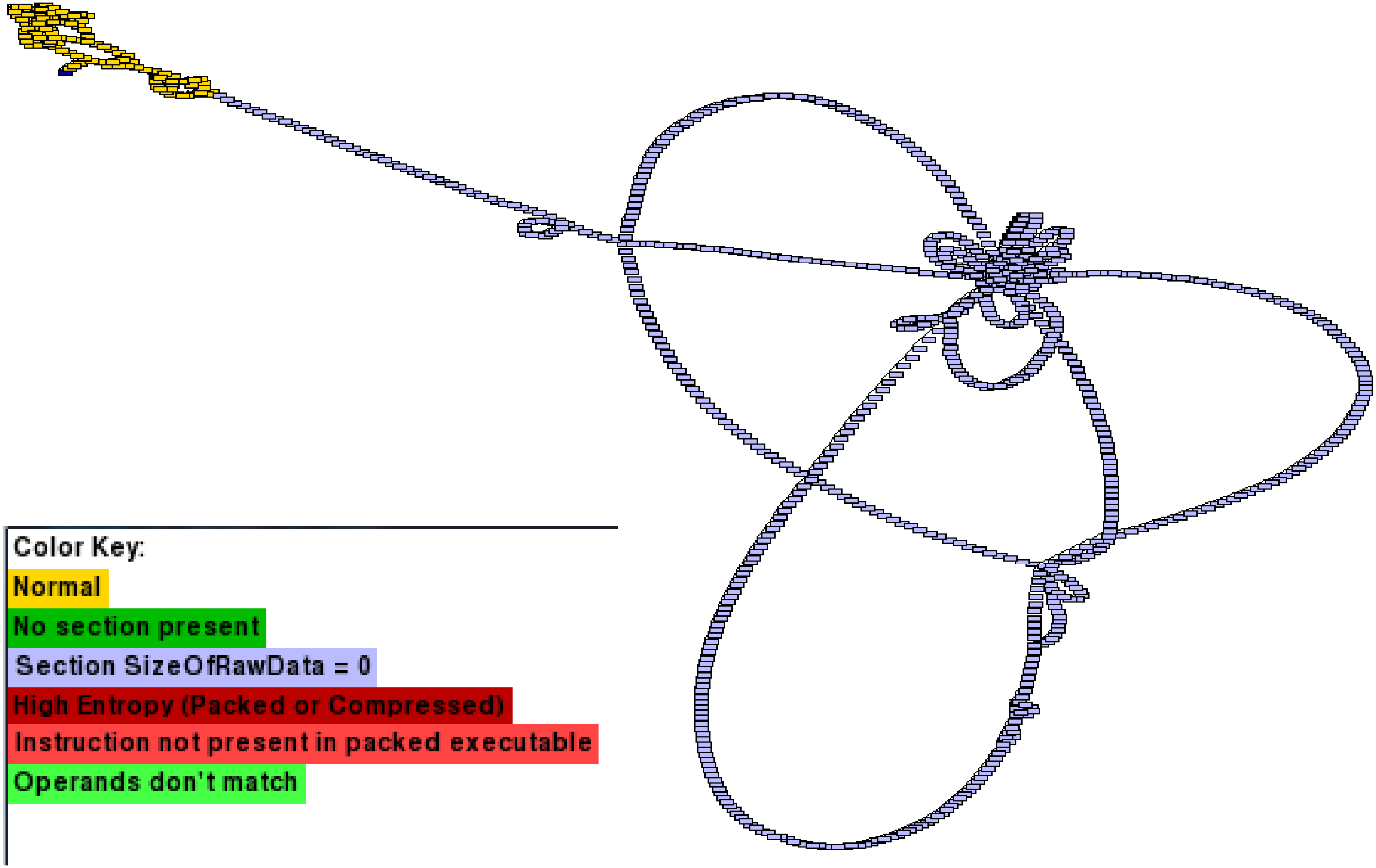
UPX - OEP


0x0040bb12
0x0040bb0d
0x0040bb0b
0x0040bb09
0x0040bb08
0x0041a50f
0x0041a50e
0x0041a518
0x0041a53f
0x0041a50c
0x0041a529
0x0041a52f
0x0041a52d
0x0041a53c
0x0041a505
0x0041a548
0x0041a536
0x0041a52c
0x0041a527
0x0041a53a
0x0041a547
0x0041a52b
0x0041a524
0x0041a51c
0x0041a500
0x0041a522
0x0041a51e
0x0041a4fe
0x0041a4fc
0x0041a4e4
0x0041a4eb
0x0041a4ed
0x0041a4f6
0x0041a4e1
0x0041a4ef
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

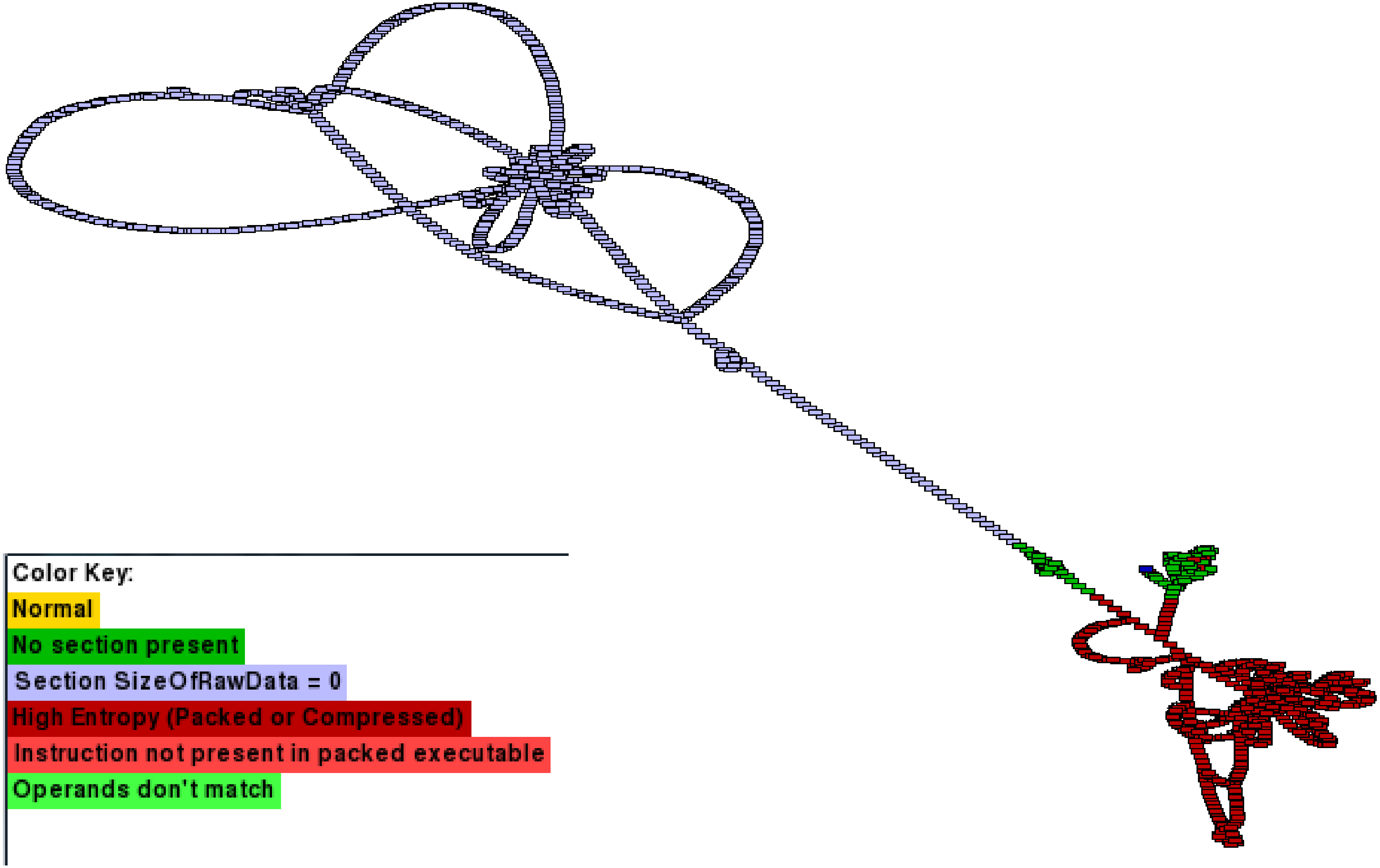
Section SizeOfRawData = 0

High Entropy (Packed or Compressed)

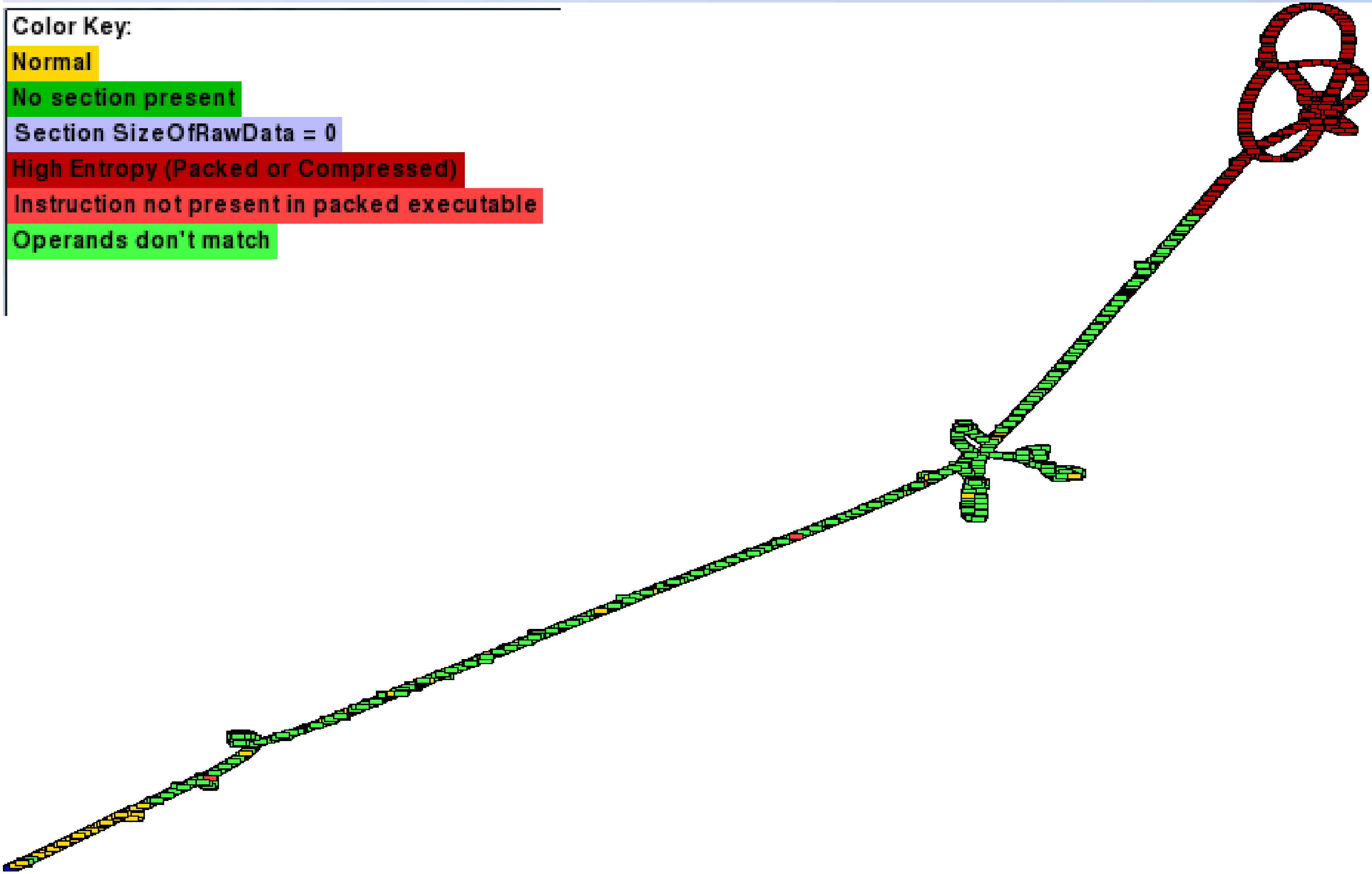
Instruction not present in packed executable

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

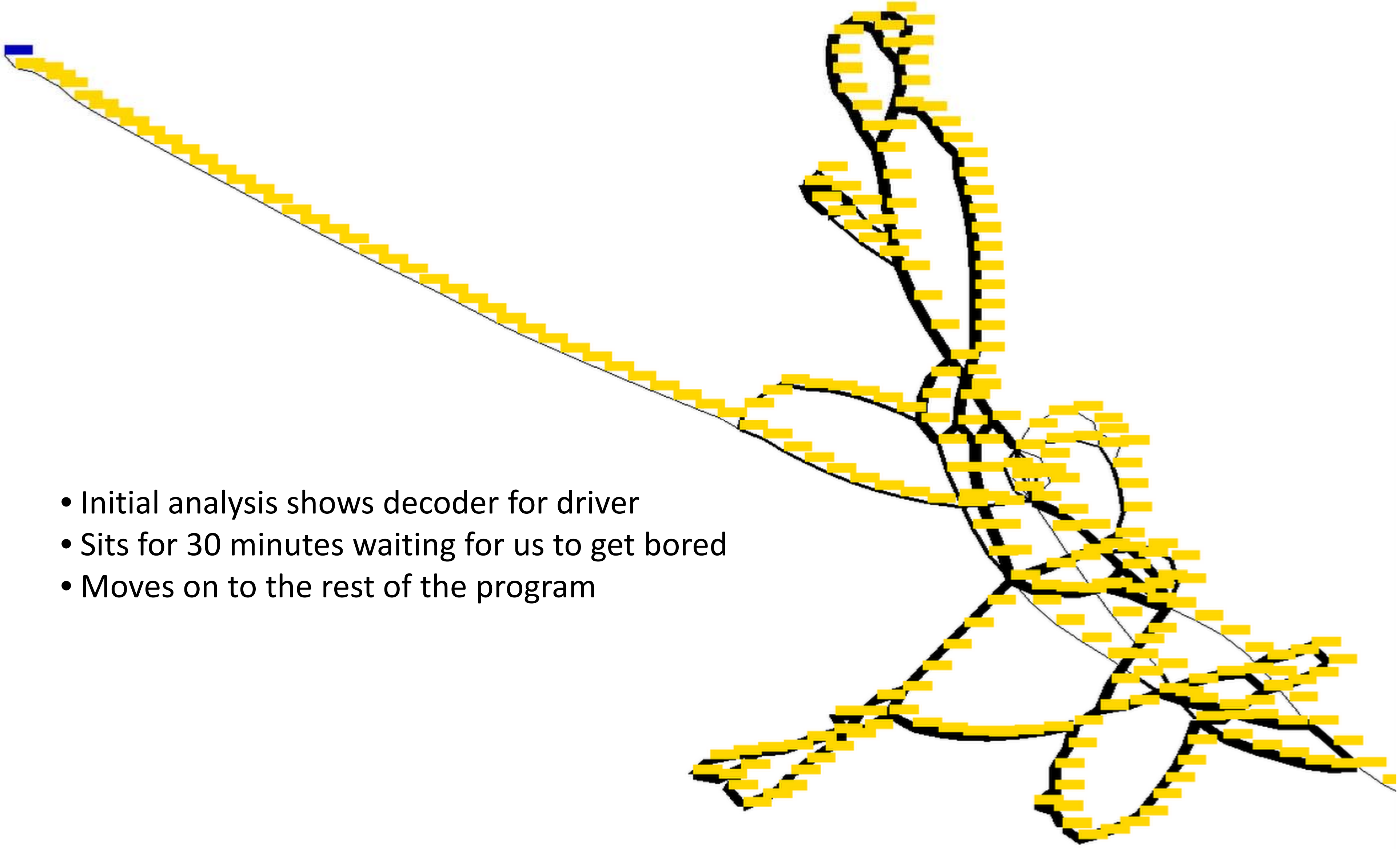
TeLock


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

Real! Live! Reversing!

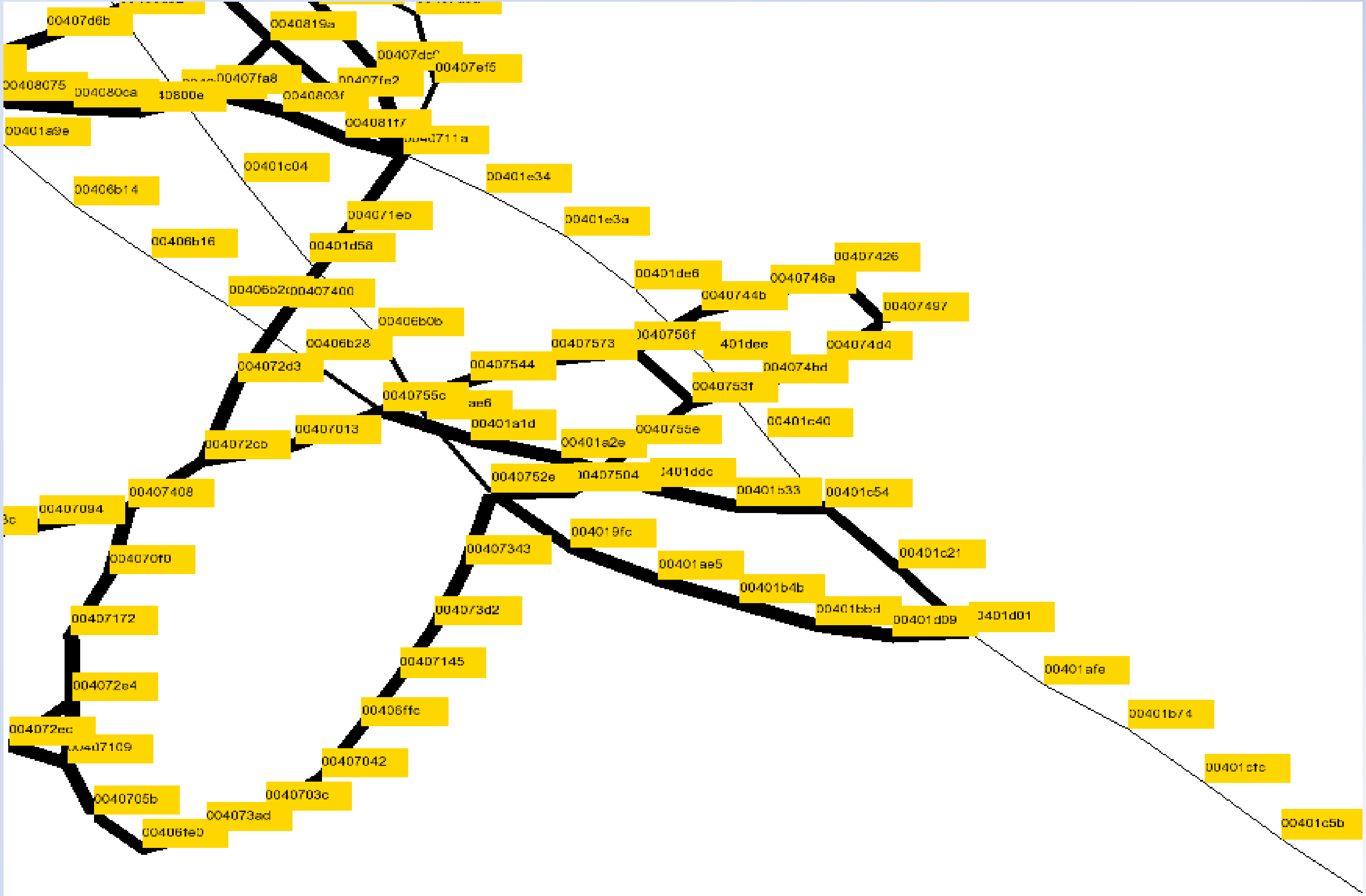
- Took latest Mebroot sample from Offensive Computing collection
- Analyzed inside of VERA
- Seemed to be idling for long periods of time
- Actually executed based on network traffic
- Hybrid user mode / kernel malware

Mebroot – Initial Busy Loop

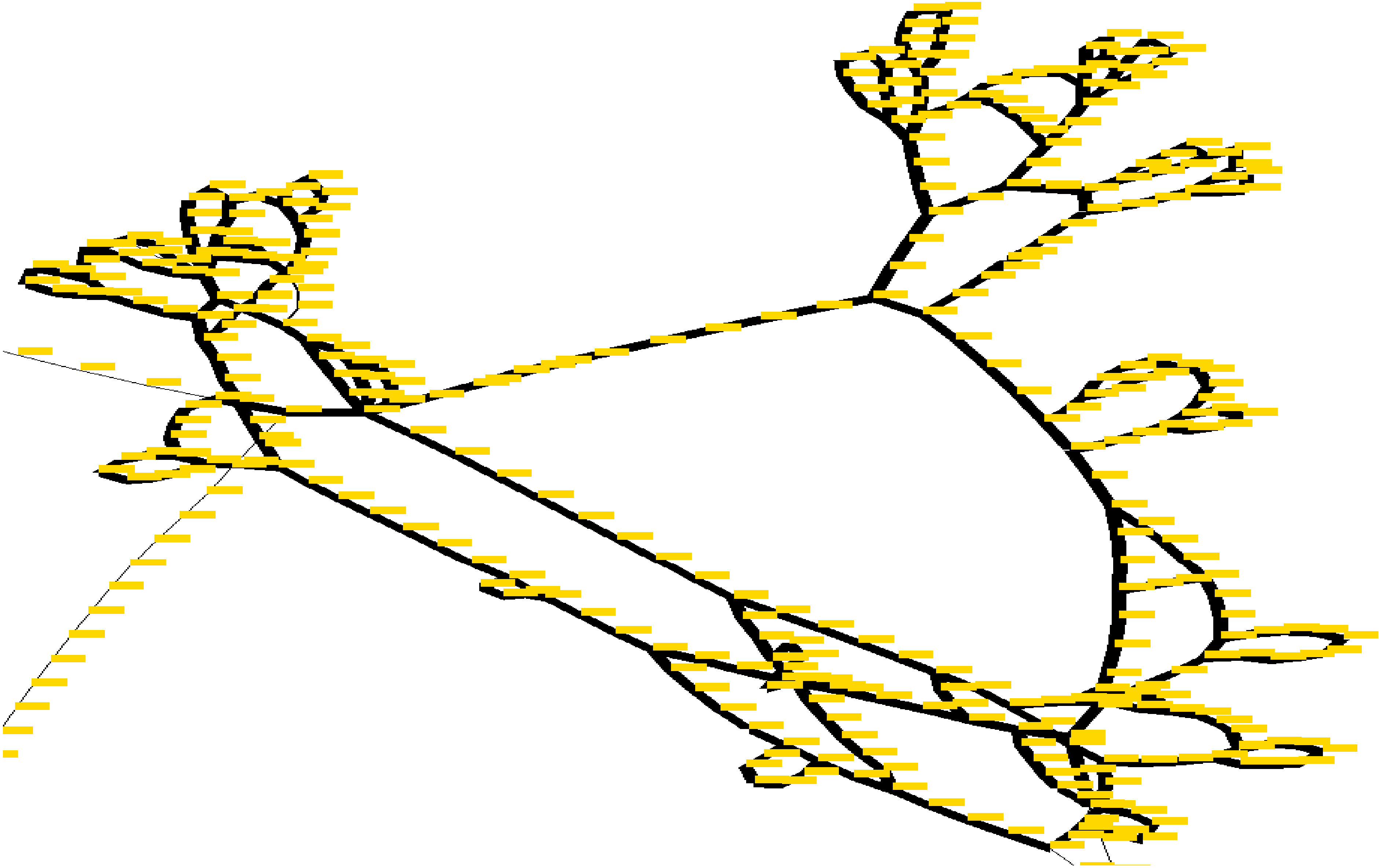


- Initial analysis shows decoder for driver
- Sits for 30 minutes waiting for us to get bored
- Moves on to the rest of the program

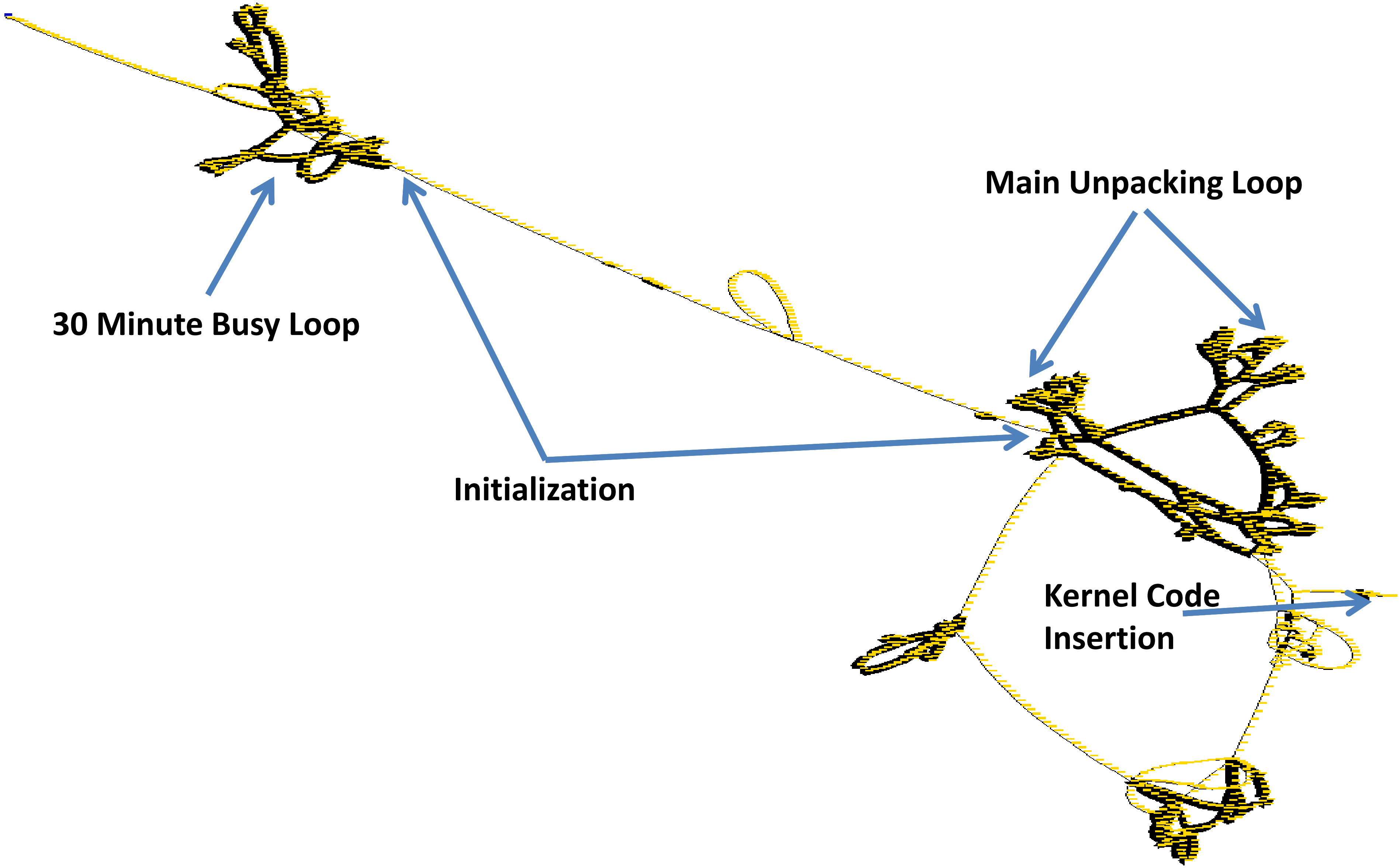
Mebroot – After Busy Loop



Mebroot – Main Unpacking Loop



Mebroot – Entire View



30 Minute Busy Loop

Initialization

Main Unpacking Loop

Kernel Code Insertion

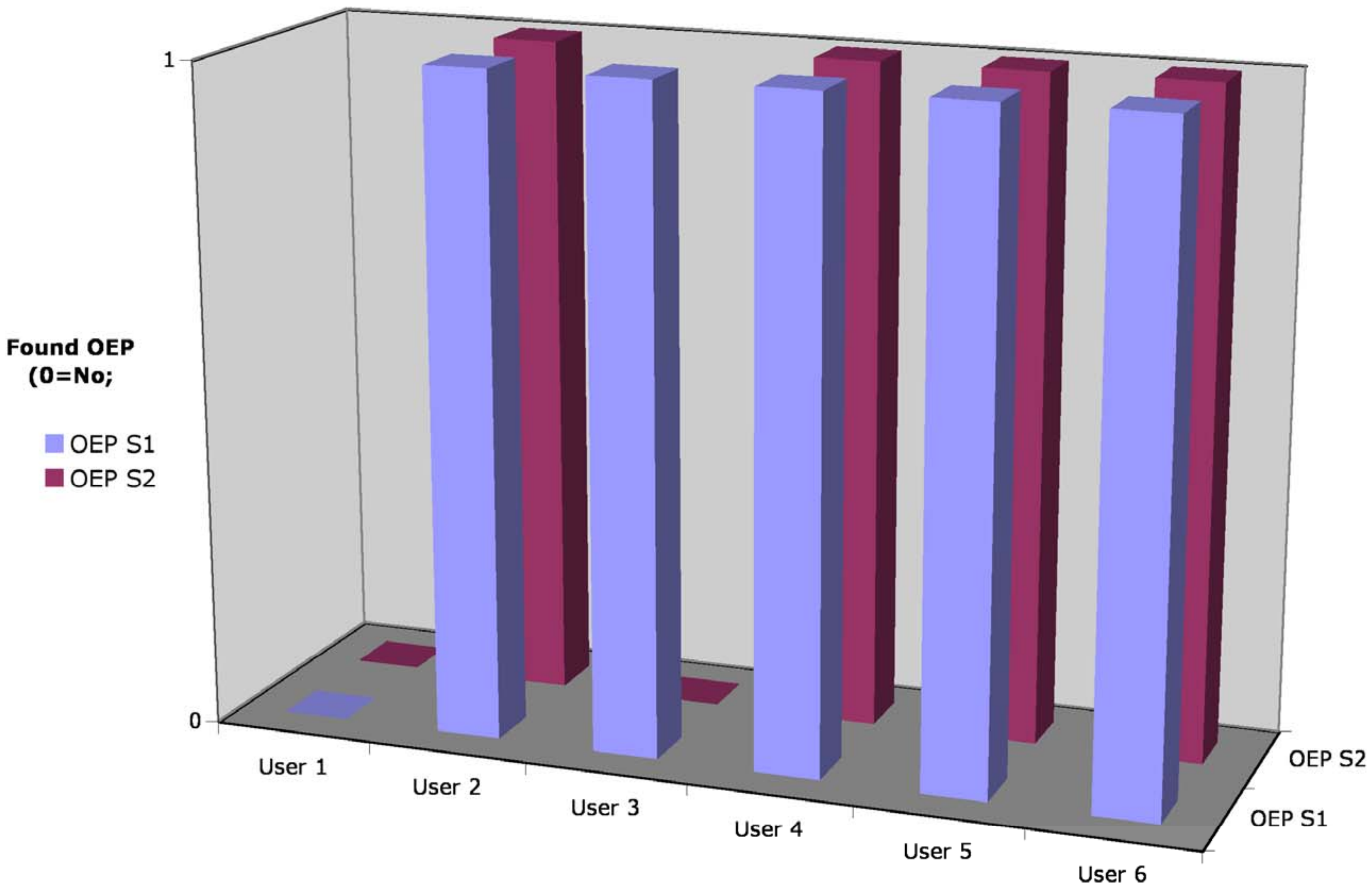
User Study

- Students had just completed week long reverse engineering course
- Analyzed two packed samples of the Netbull Virus with UPX and MEW
- Asked to perform a series of tasks based on the typical reverse engineering process
- Asked about efficacy of visualization tool

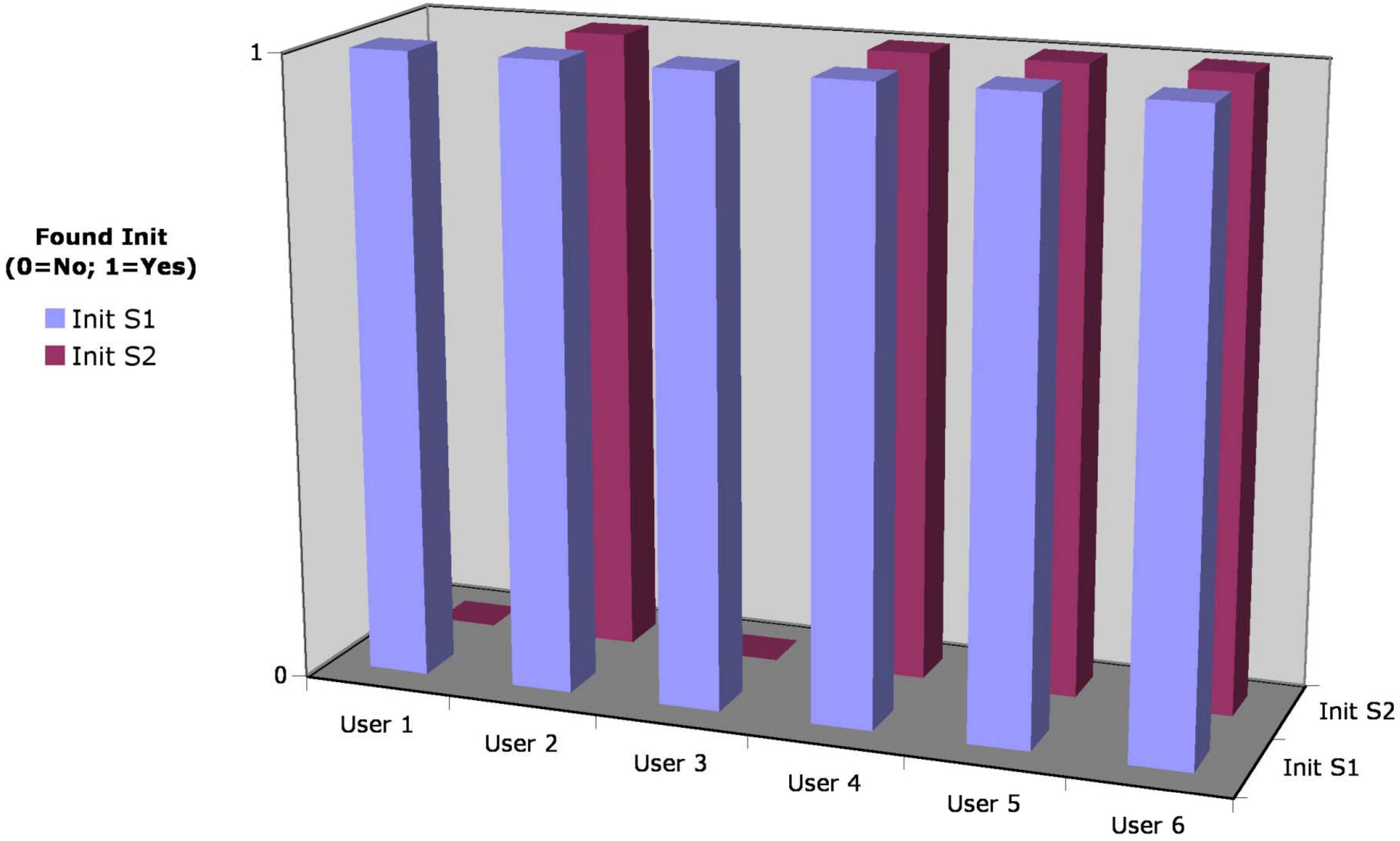
User Study: Tasks Performed

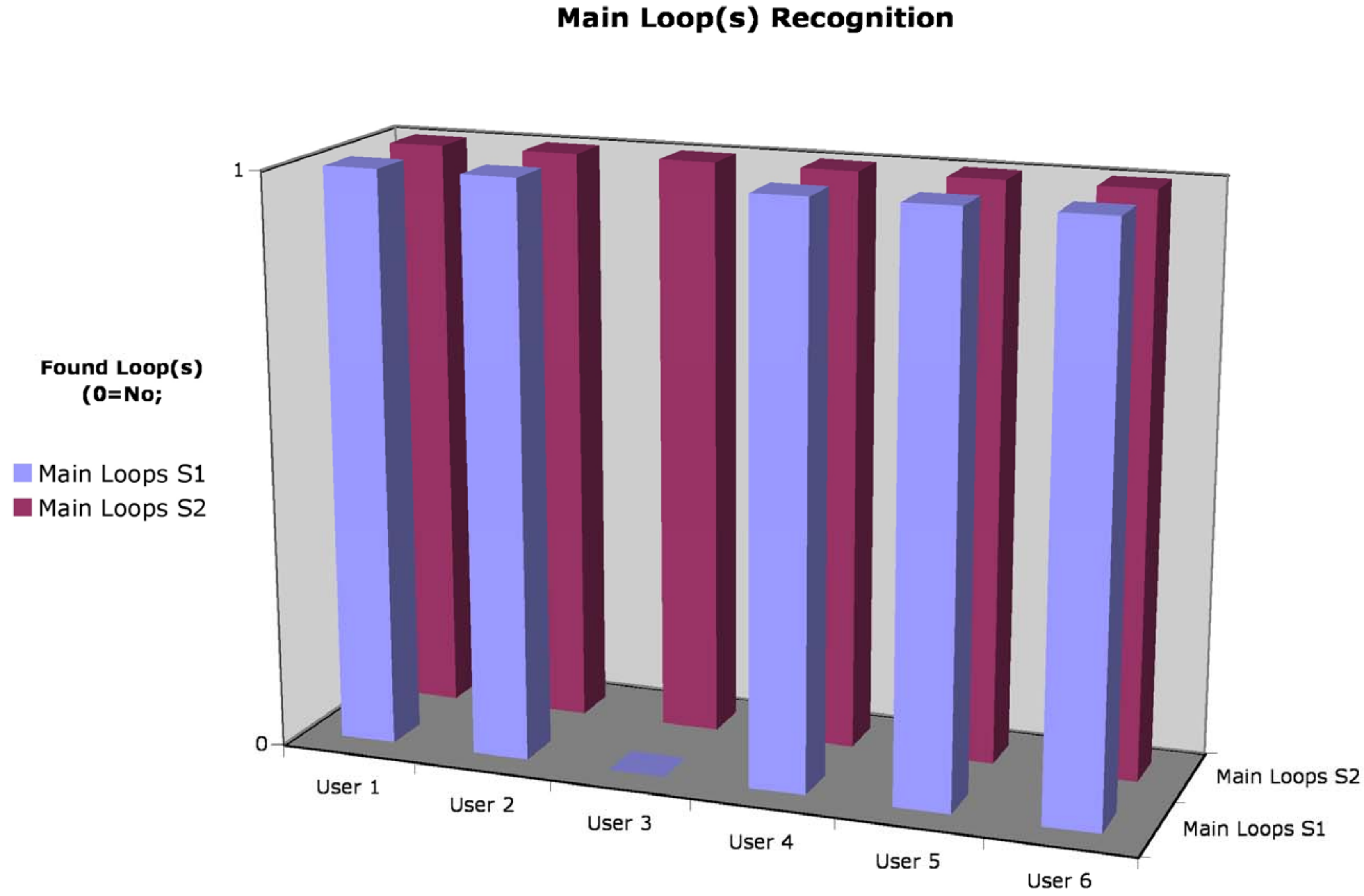
- Find the original entry point (OEP) of the packed samples
- Execute the program to look for any identifying output
- Identify portions of the executable:
  - Packer code
  - Initialization
  - Main loops

Original Entry Point Recognition



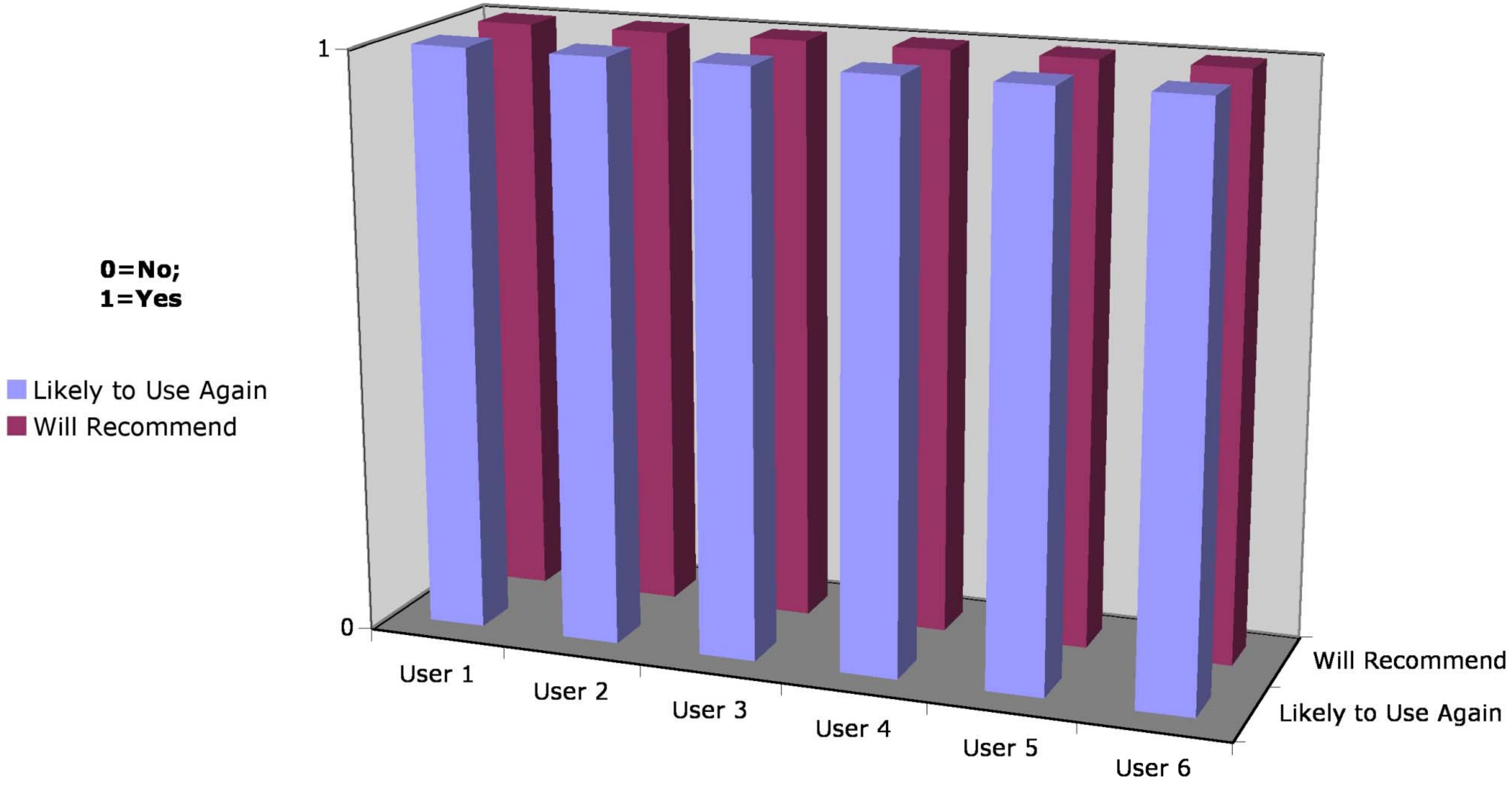
Initialization Recognition





Main Loop(s) Recognition
Found Loop(s)
(0=No;
🟦 Main Loops S1
🟥 Main Loops S2
1
0
User 1
User 2
User 3
User 4
User 5
User 6
Main Loops S2
Main Loops S1

Overall Evaluation



Selected Comments

- “Wonderful way to visualize analysis and to better focus on areas of interest”
- “Fantastic tool. This has the potential to significantly reduce analysis time.”
- “It rocks. Release ASAP.”

Recommendations for improvement

- Need better way to identify beginning and end of loops
- Many loops overlap and become convoluted
- Be able to enter memory address and see basic blocks that match

Future Work

- General GUI / bug fixes
- Stabilization of analysis environment
- 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

Installation Tripping Hazards

- Install 64-bit Debian Sarge
  - Doesn't work on other distributions
- Install Ether using instructions on their page: http://ether.gtisc.gatech.edu/
- Setup a 32-bit Windows XP SP2 Image
  - Disable: DEP, large pages, multiple CPUs
- Kill target program before stopping Ether
  - Pretty serious bug causes reboot

Closing thoughts

- Ether is awesome. Thanks Artem Dinaburg and Paul Royal.
- Source, tools, and latest slides can be found at: http://www.offensivecomputing.net
- If you use the tool, please give feedback
- Look for the paper at Vizsec 2009

Thanks!

- Artem Dinaburg
- Paul Royal
- Cort Dougan
- Moses Schwartz
- Alan Erickson
- Alex Kent
- New Mexico Tech SFS Program NSF / DHS