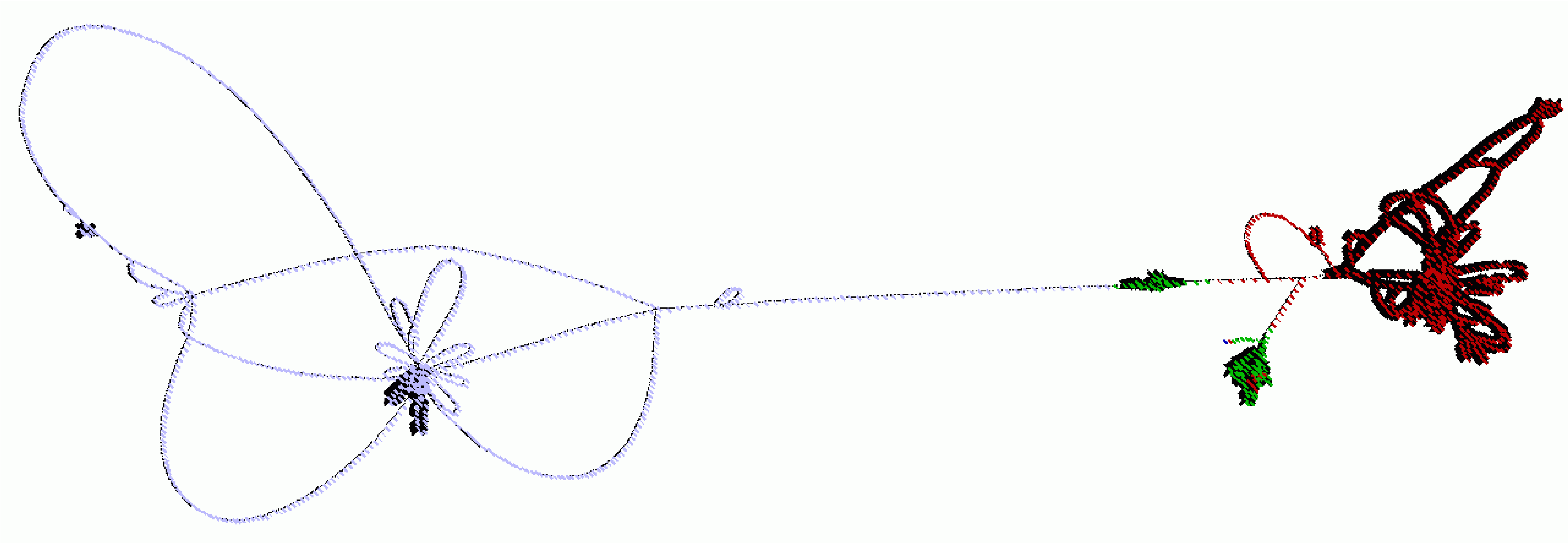
Visualizing Compiled Executables for Malware Analysis



Daniel Quist
Lorie Liebrock
New Mexico Tech
Los Alamos National Laboratory

Overview

Explanation of Problem
Overview of Reverse Engineering Process
Related Work
Visualization for Reverse Engineering
VERA Architecture
Case Study: Mebroot
User Study
Contributions

Explanation of Problem

- Reverse engineering is a difficult and esoteric skill to learn
- Most new reversers struggle with understanding overall structure
- Knowing where to start is the most difficult task

Reverse Engineering 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

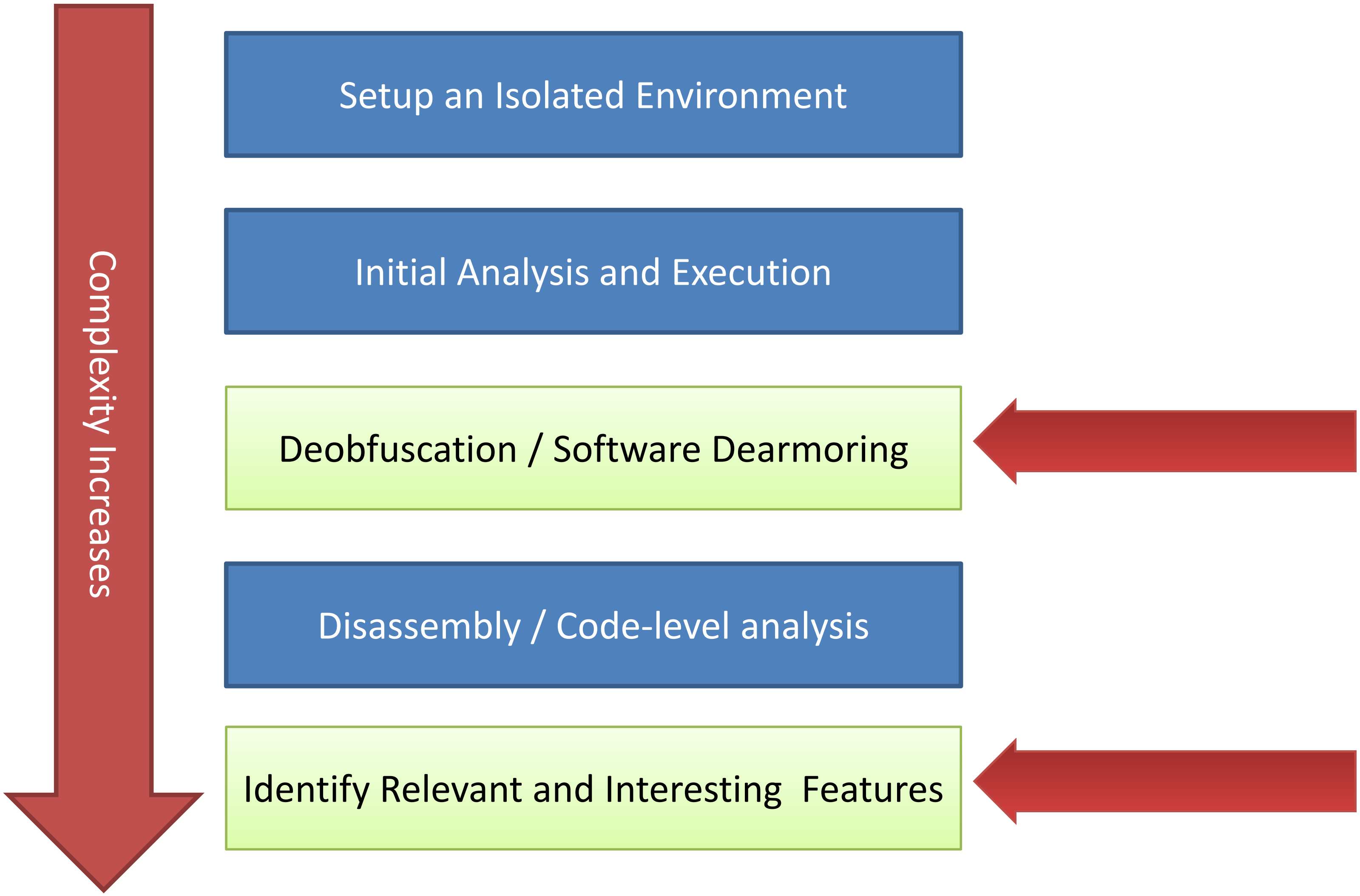
Disassembly / Code-level analysis

- IDA Pro
- OllyDbg

Identify Relevant and Interesting Features

- Experience based
- Newbies have trouble with this

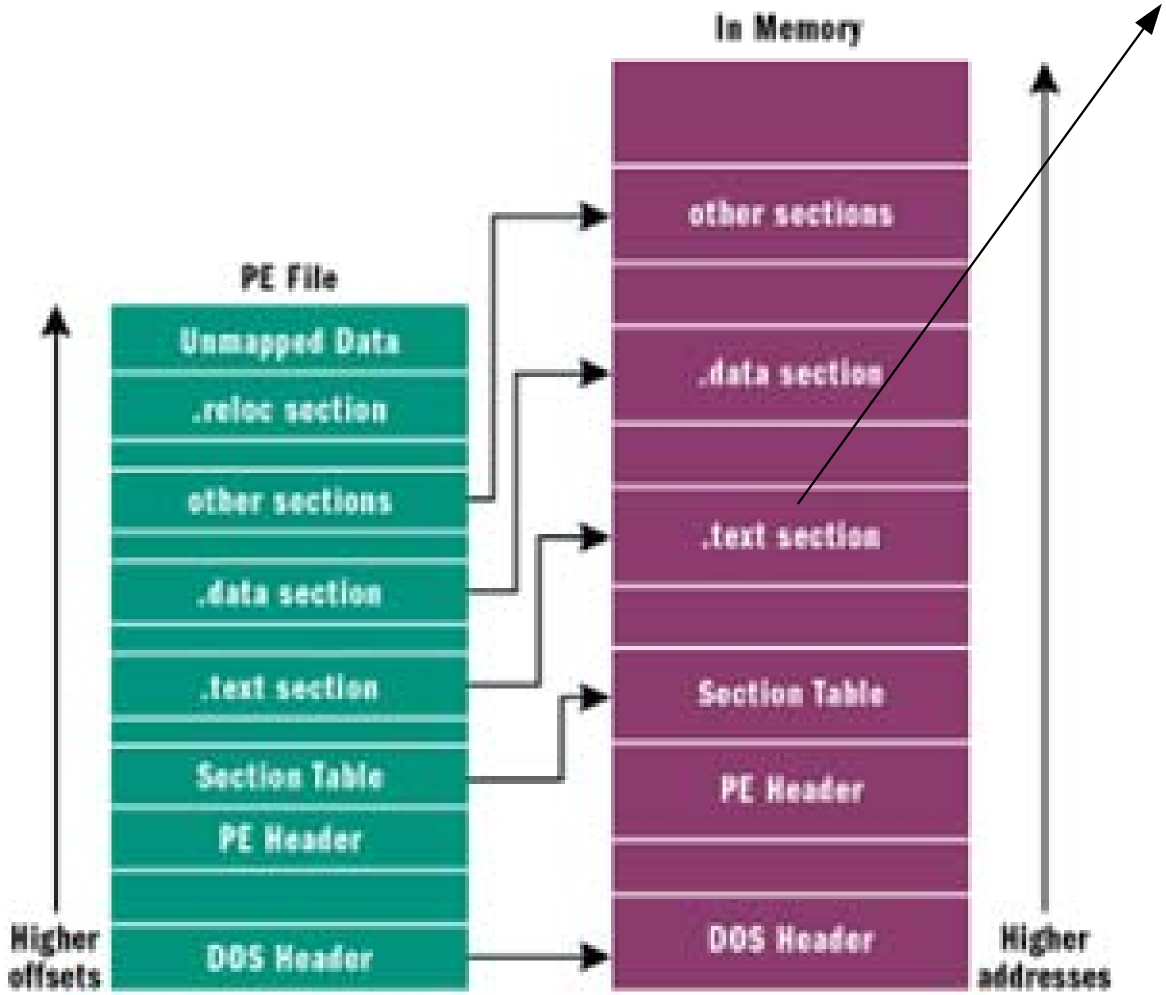
Addressing the Situation



Packing and Encryption

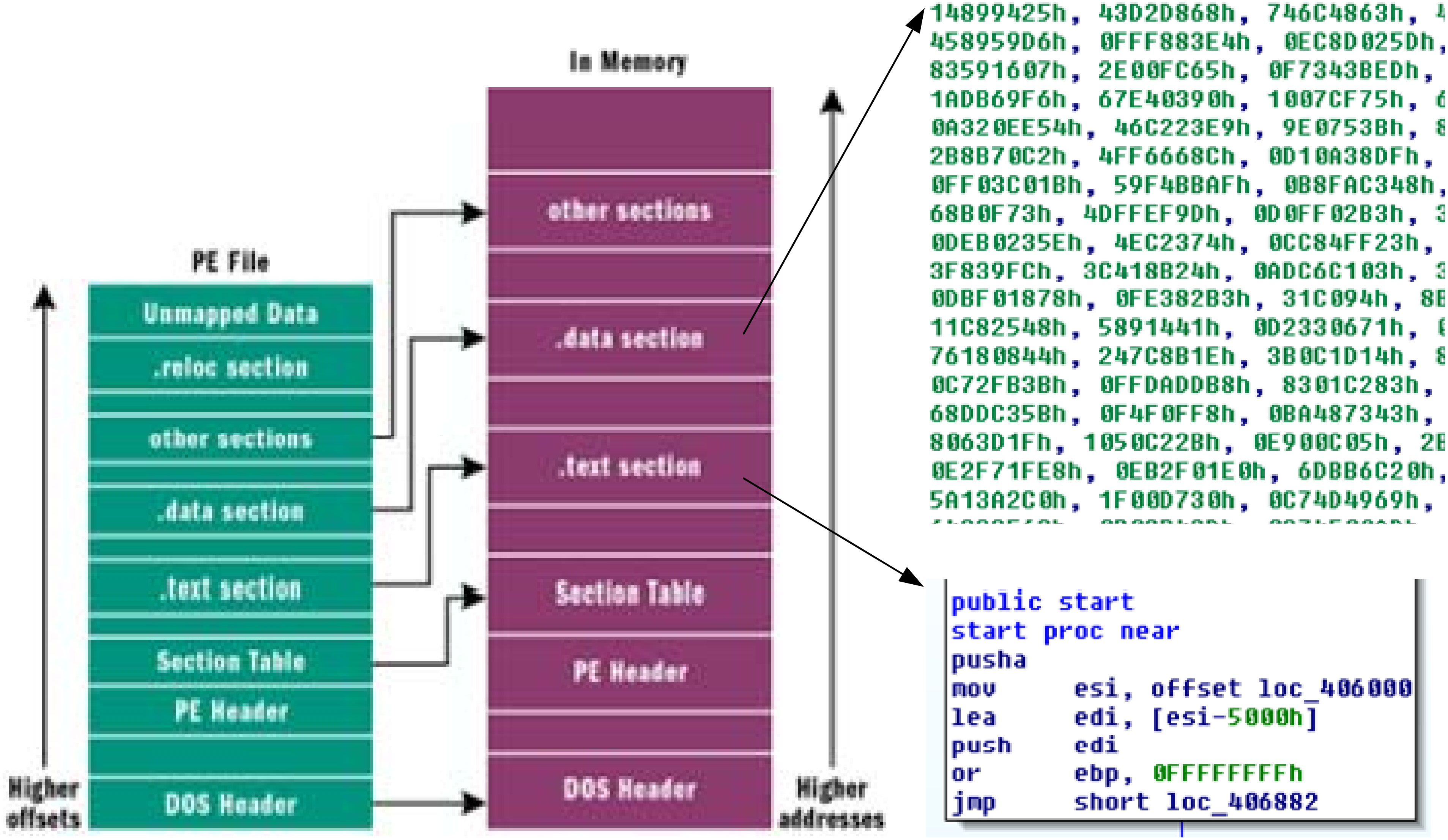
- Self-modifying code
  – Small decoder stub
  – Decompress the main executable
  – Restore imports
- Play “tricks” with the executable
  – OS Loader is inherently lazy (efficient)
  – Hide the imports
  – Obscure relocations
  – Use bogus values for various unimportant fields

Normal PE File



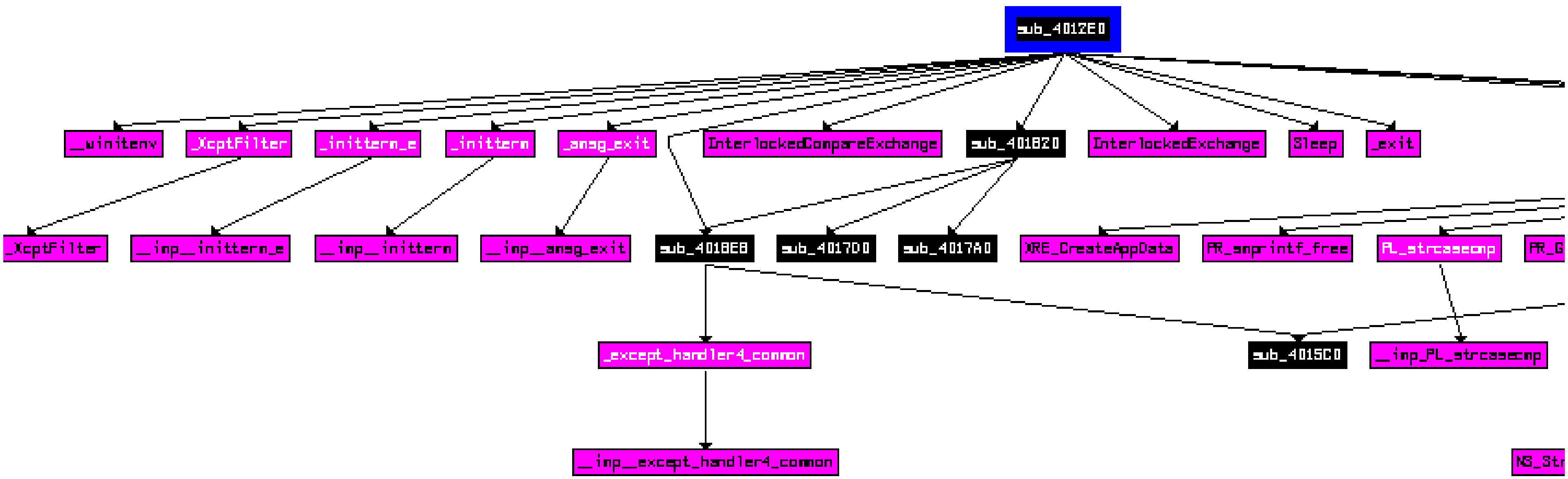

```
push    ebp
mov     ebp, esp
sub     esp, 1Ch           ; lpMsg
call    ds:__imp__GetCommandLineW@0 ;
push    [ebp+nCmdShow]     ; nCmdShow
push    eax                ; int
push    [ebp+hPrevInstance] ; int
push    [ebp+hInstance]  ; hInstance
call    _FSolInit@16       ; FSolInit(x,
test    eax, 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



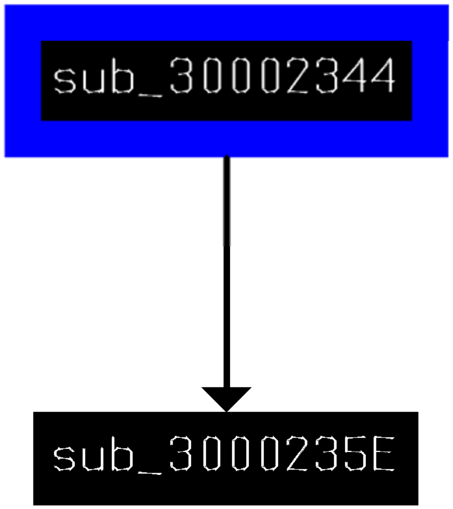
Related Work

IDA Pro - Graphing Crossreferences



- Illustrates Relationship of Function Calls
- Magenta represents imported API calls
- Black represents module subroutines

IDA Pro – Visualization Problems

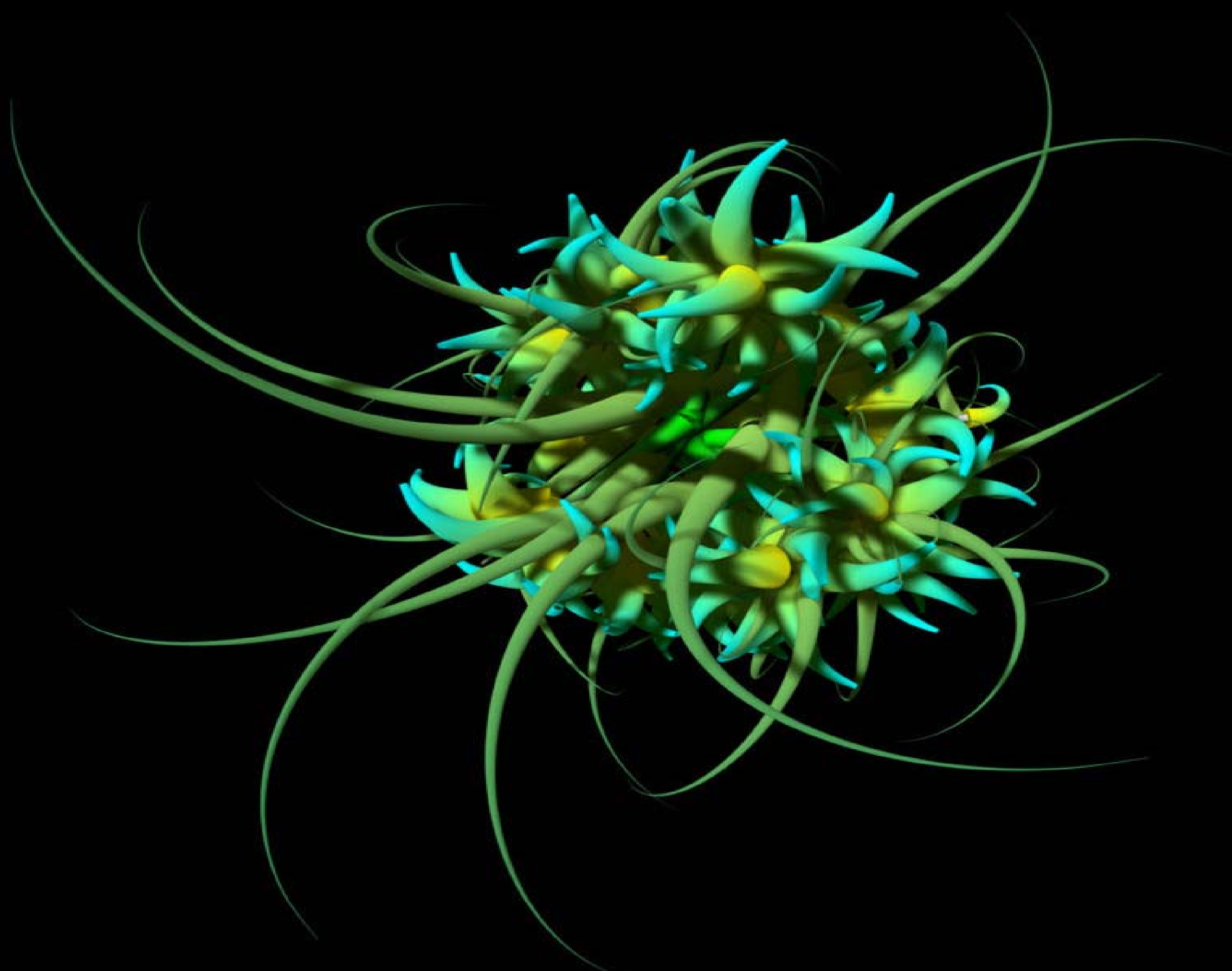


Firefox Initialization

idag.exe (IDA Pro) overview

- Some graphs are useless
- Some graphs are too complex
- No indication of heavily executed portions
- Obfuscated code is gibberish

Alex Dragulescu – MyDoom Visualization



http://www.sq.ro/malwarez.php

Visualization for Reverse Engineering

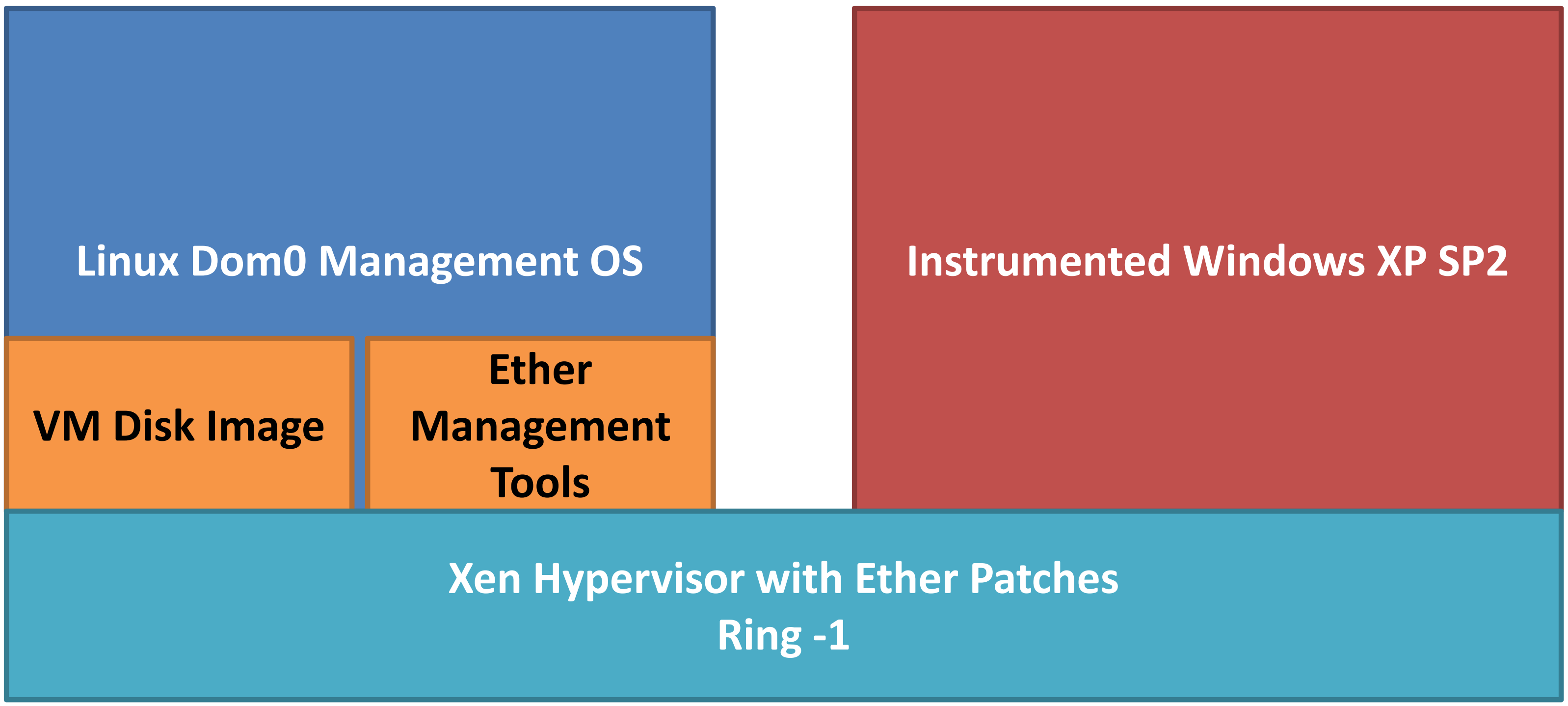
- Identify major program functional areas
  - Initialization
  - Main loops
  - Communications / organizational structure
- Deobfuscation / dearmoring
  - Identify packing loops
  - Find self-modifying code
- Take “intuition” out of the reversing process

Enabling Technology: 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

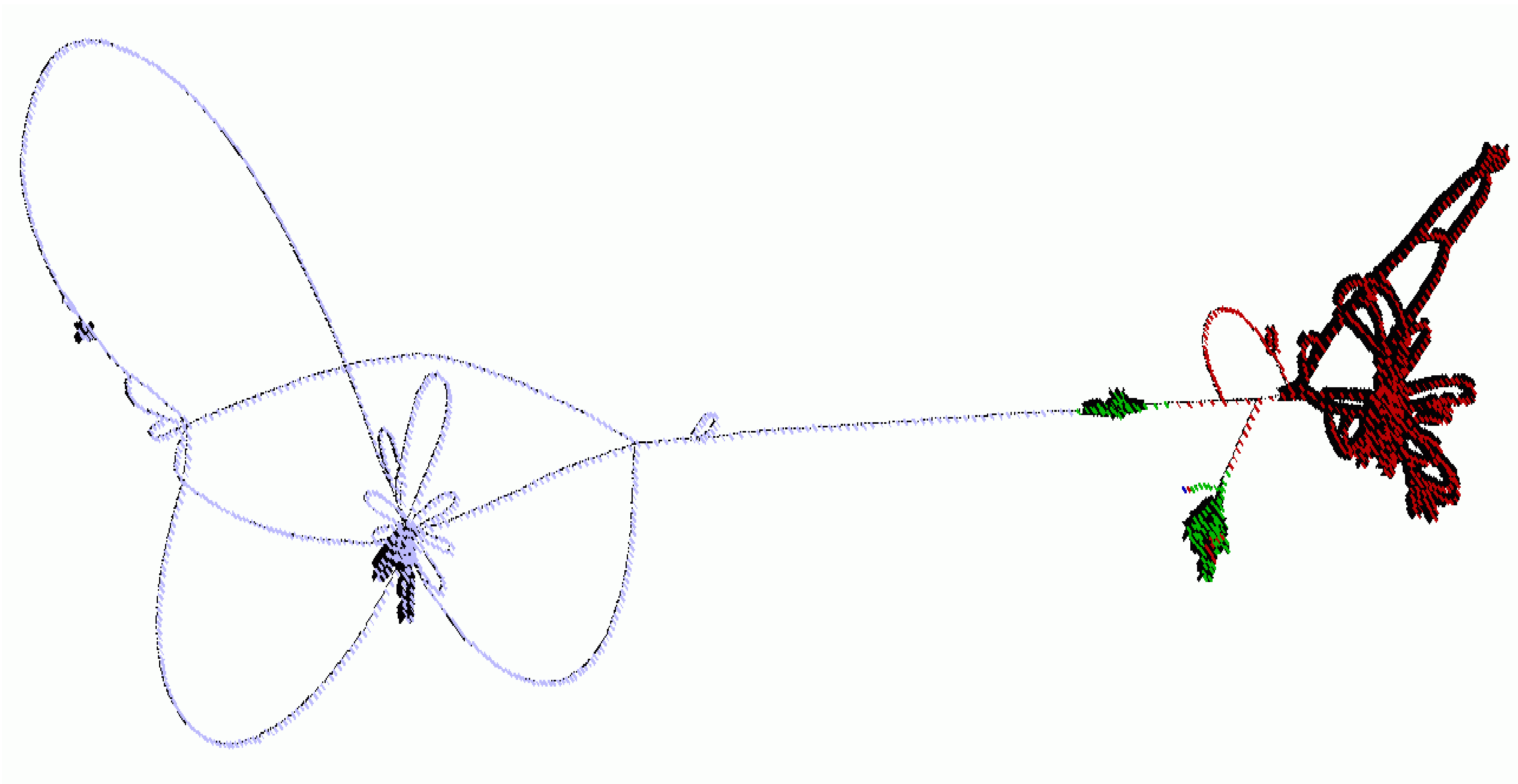
Ether System Architecture



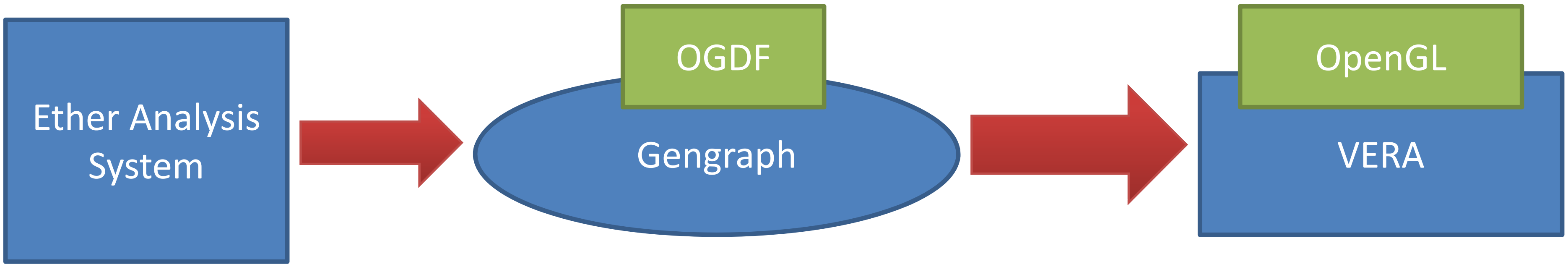
Visualizing Executables for Reversing and Analysis

- OpenGL rendering of dynamic program execution
- Vertices represent addresses
- Edges represent execution from one address to another
- Thicker edges represent multiple executions
- Colors to help identify type of code

Graph Preview



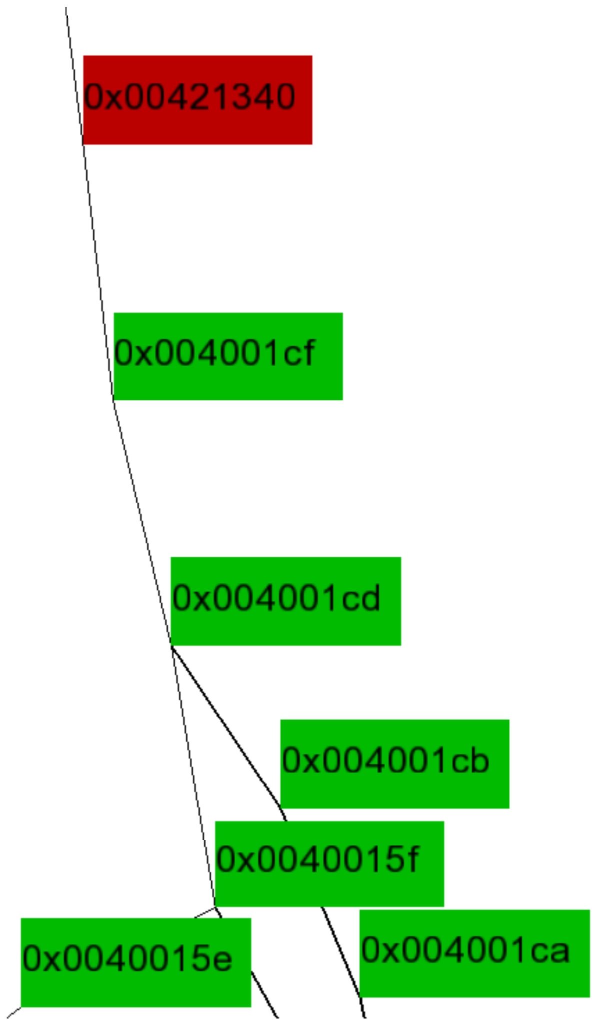
VERA Architecture



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

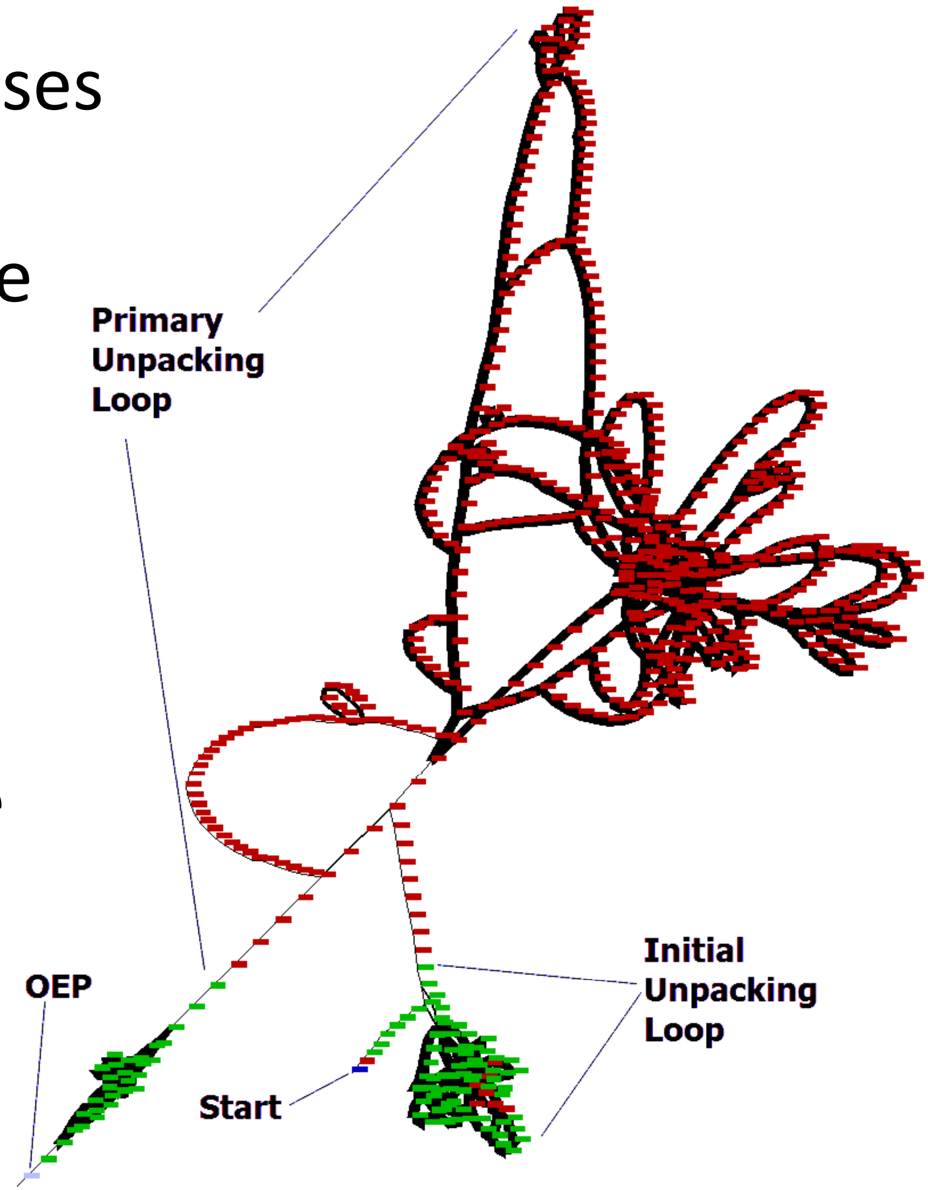
Vertices (Addresses)

- 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 (Transition)

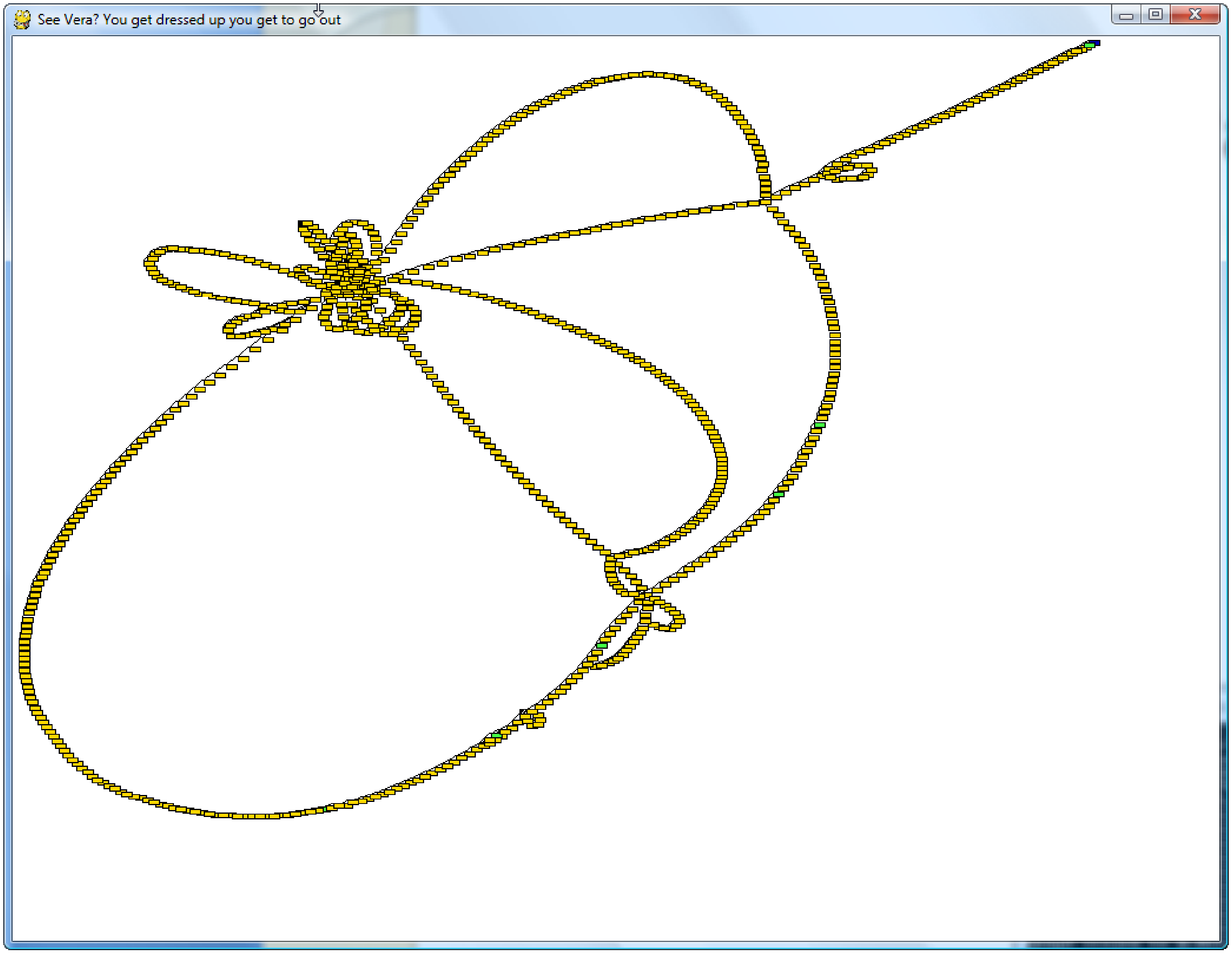
- Transitions between addresses
- Thicker lines represent more executions
  - Easy identification of loops
  - Find heavy concentration of execution
- Multiple edges from a node represent decision point



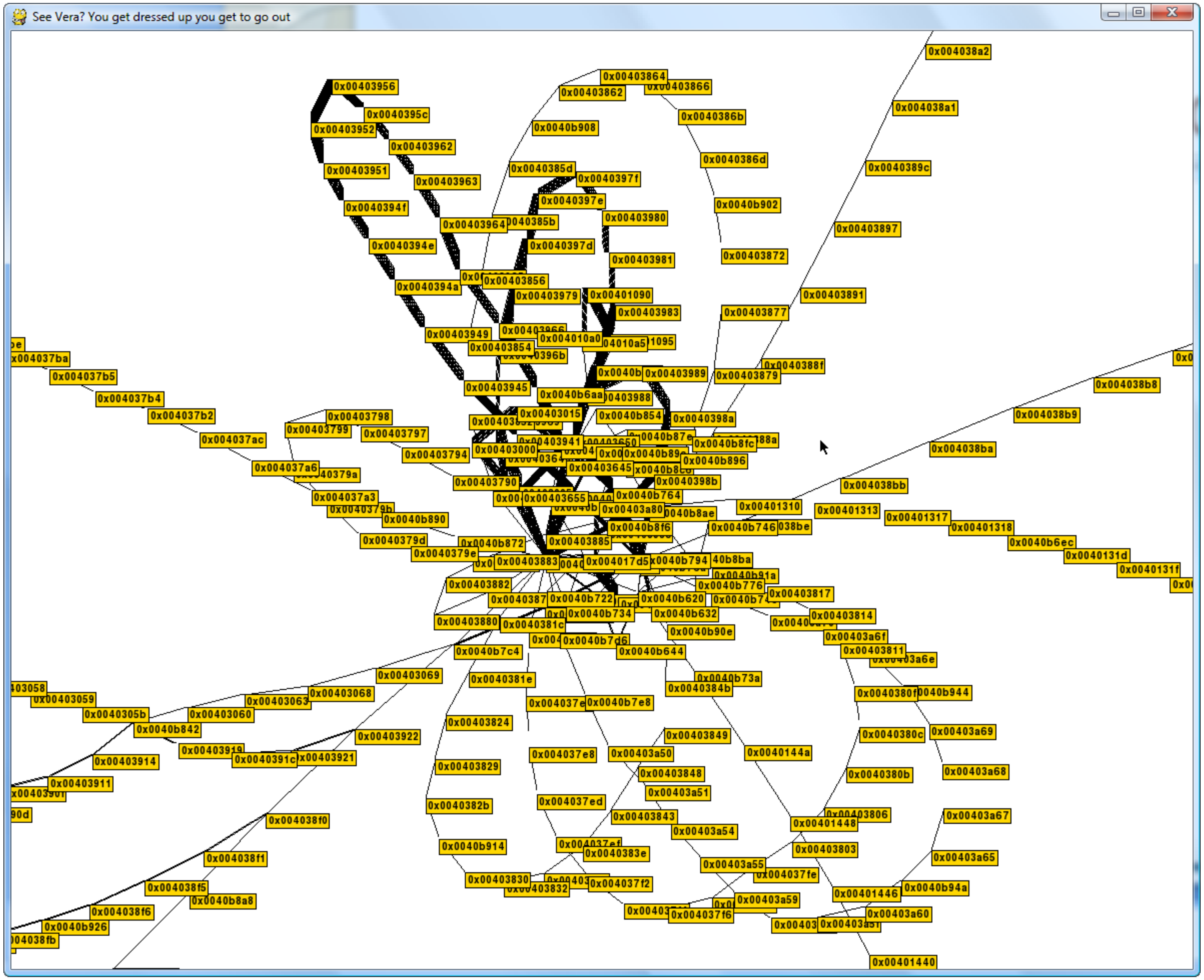
Colors

- **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

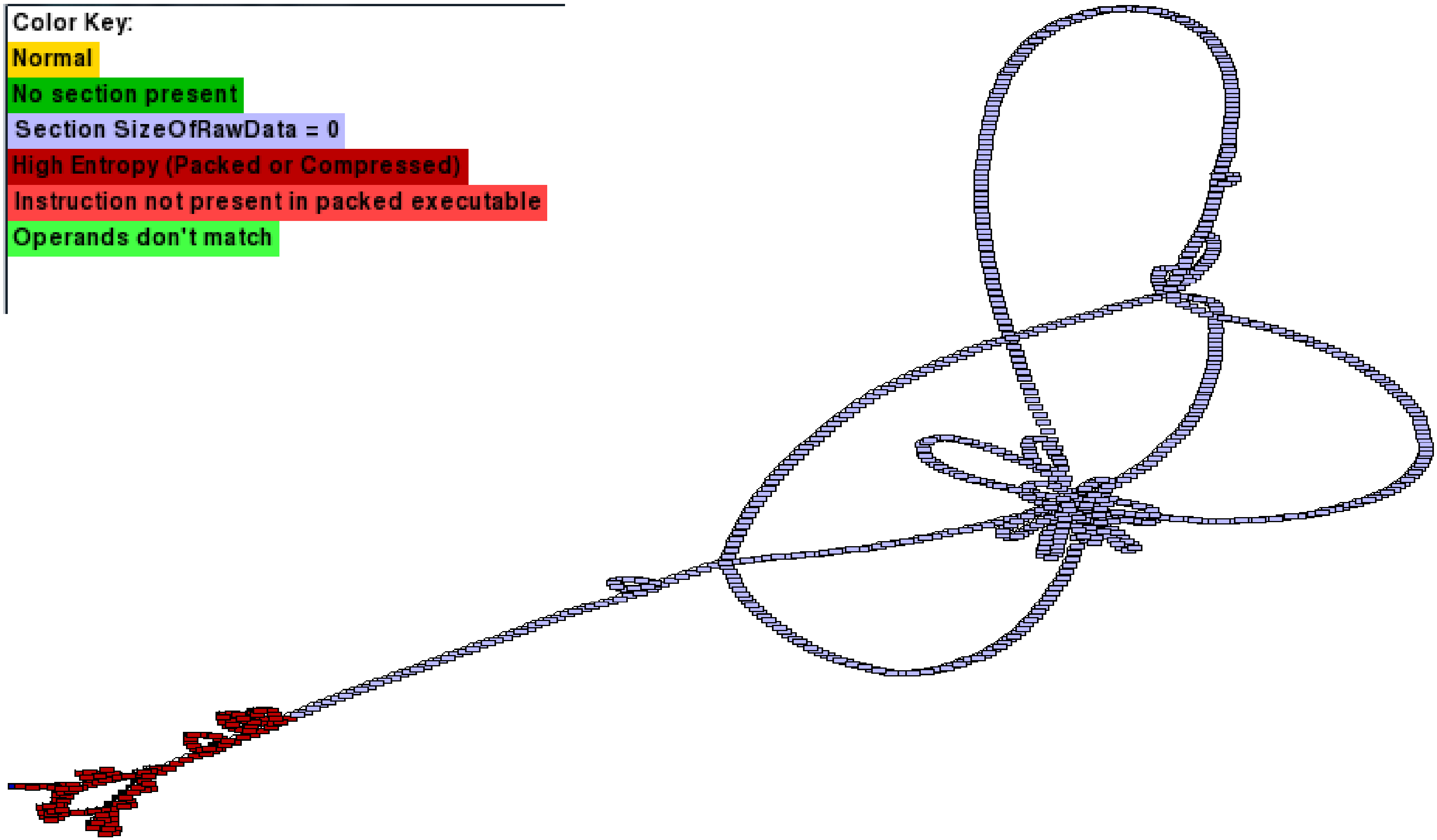
Netbull Virus (Not Packed)



Netbull Zoomed View

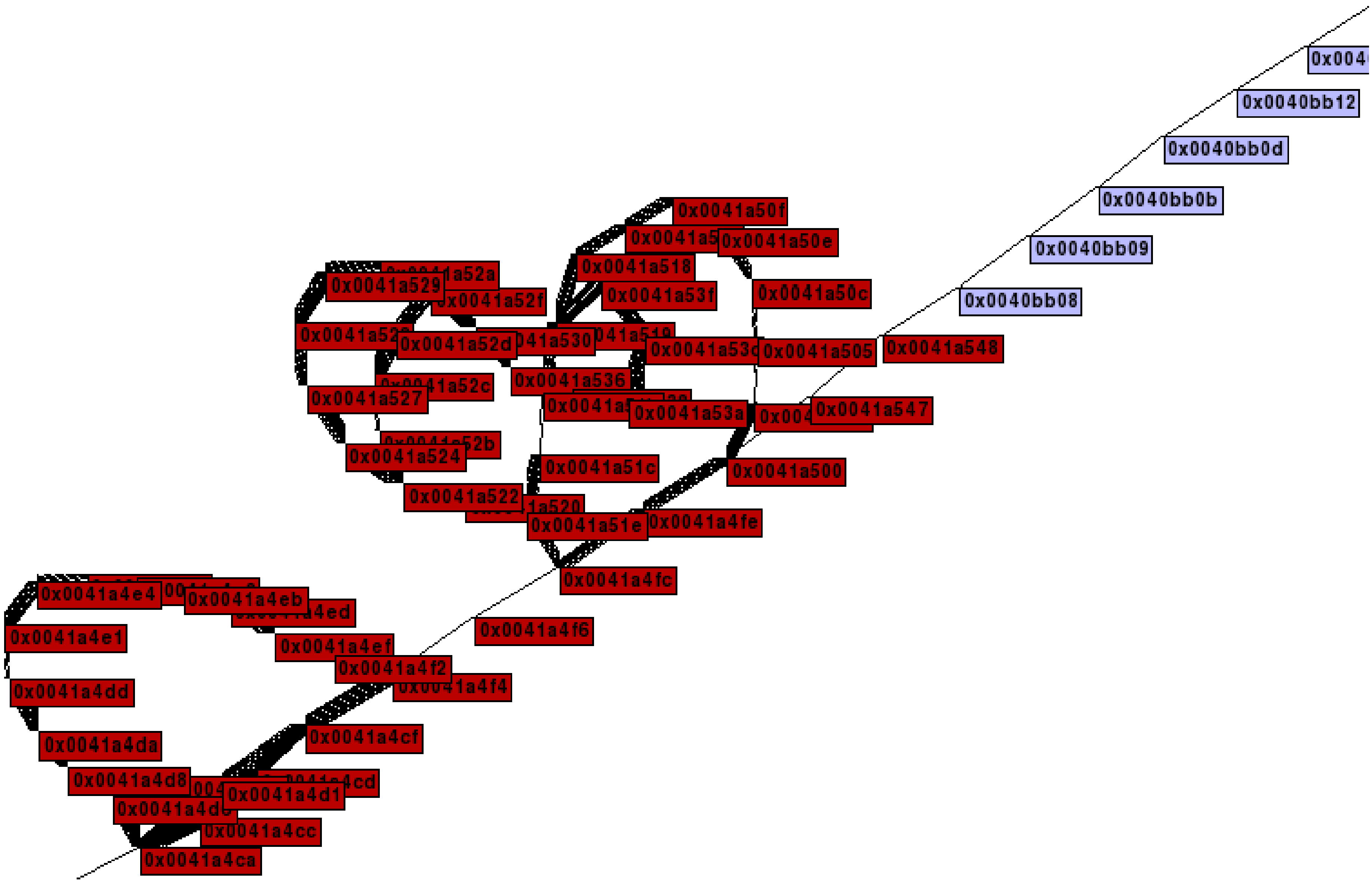


UPX



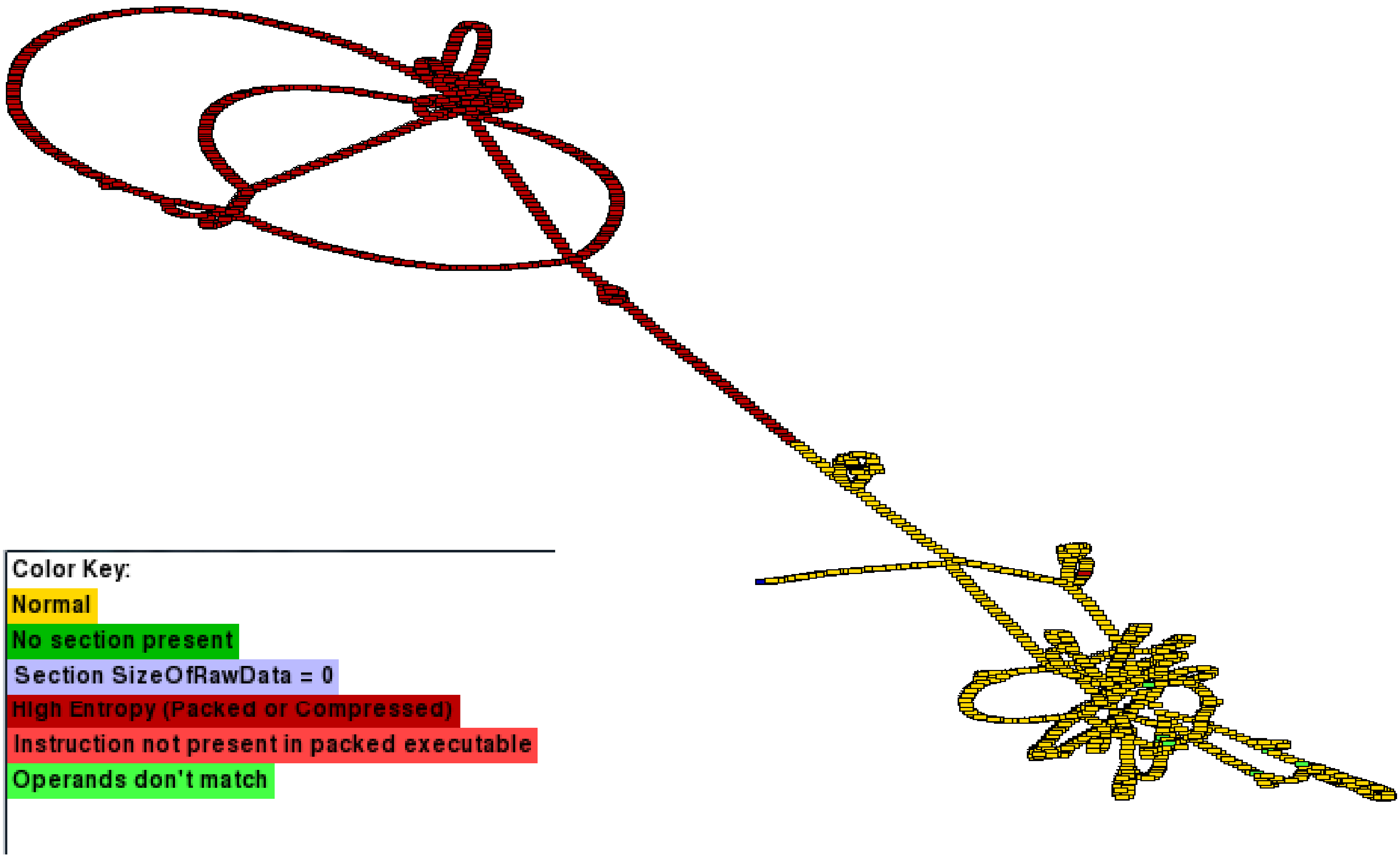
Color Key:
Normal
No section present
Section SizeOfRawData = 0
High Entropy (Packed or Compressed)
Instruction not present in packed executable
Operands don't match

UPX - OEP



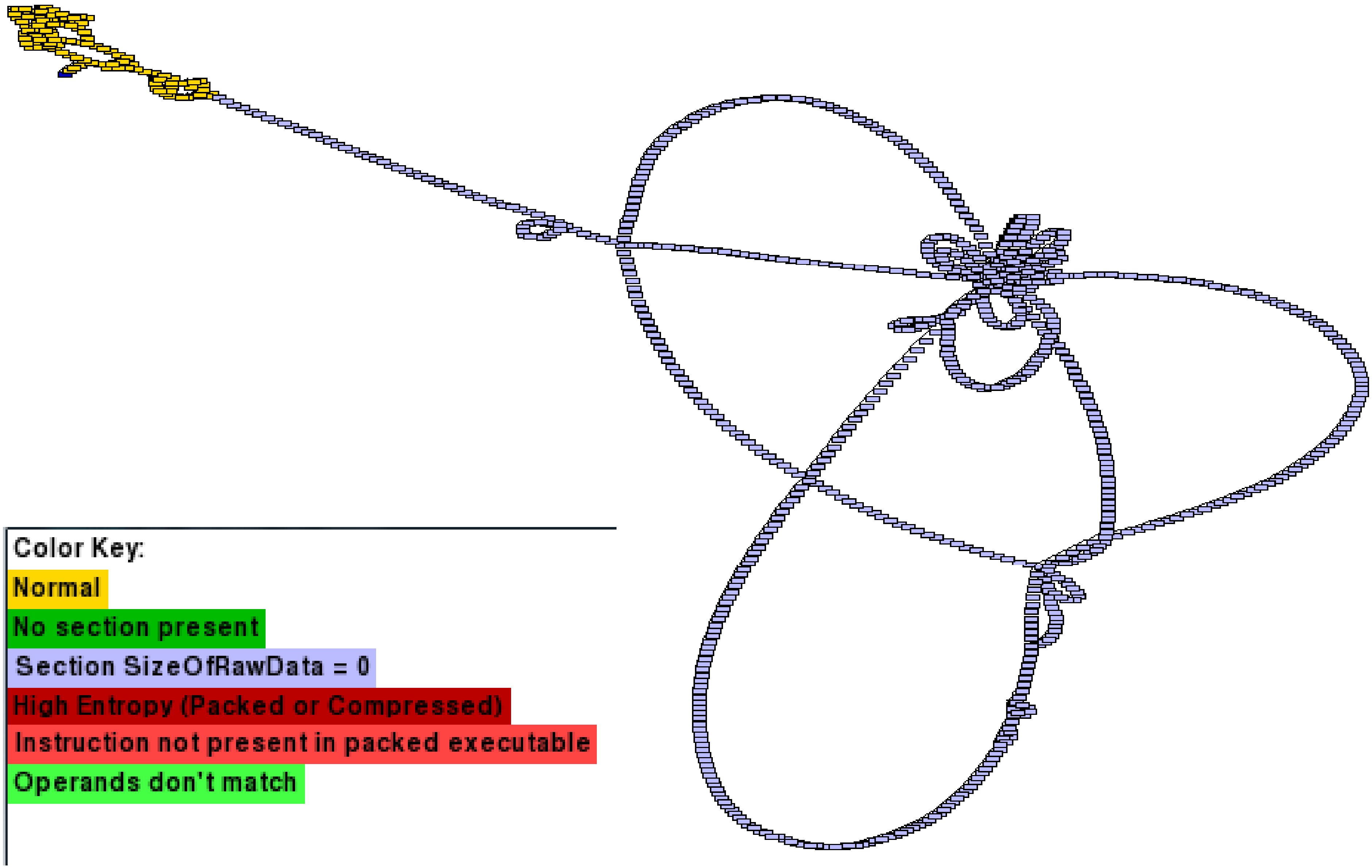
0x0040bb12
0x0040bb0d
0x0040bb0b
0x0040bb09
0x0040bb08
0x0041a50f
0x0041a50e
0x0041a518
0x0041a53f
0x0041a50c
0x0041a529
0x0041a52f
0x0041a52d
0x0041a536
0x0041a505
0x0041a548
0x0041a527
0x0041a53a
0x0041a547
0x0041a524
0x0041a51c
0x0041a500
0x0041a522
0x0041a51e
0x0041a4fe
0x0041a4fc
0x0041a4e4
0x0041a4eb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

ASPack

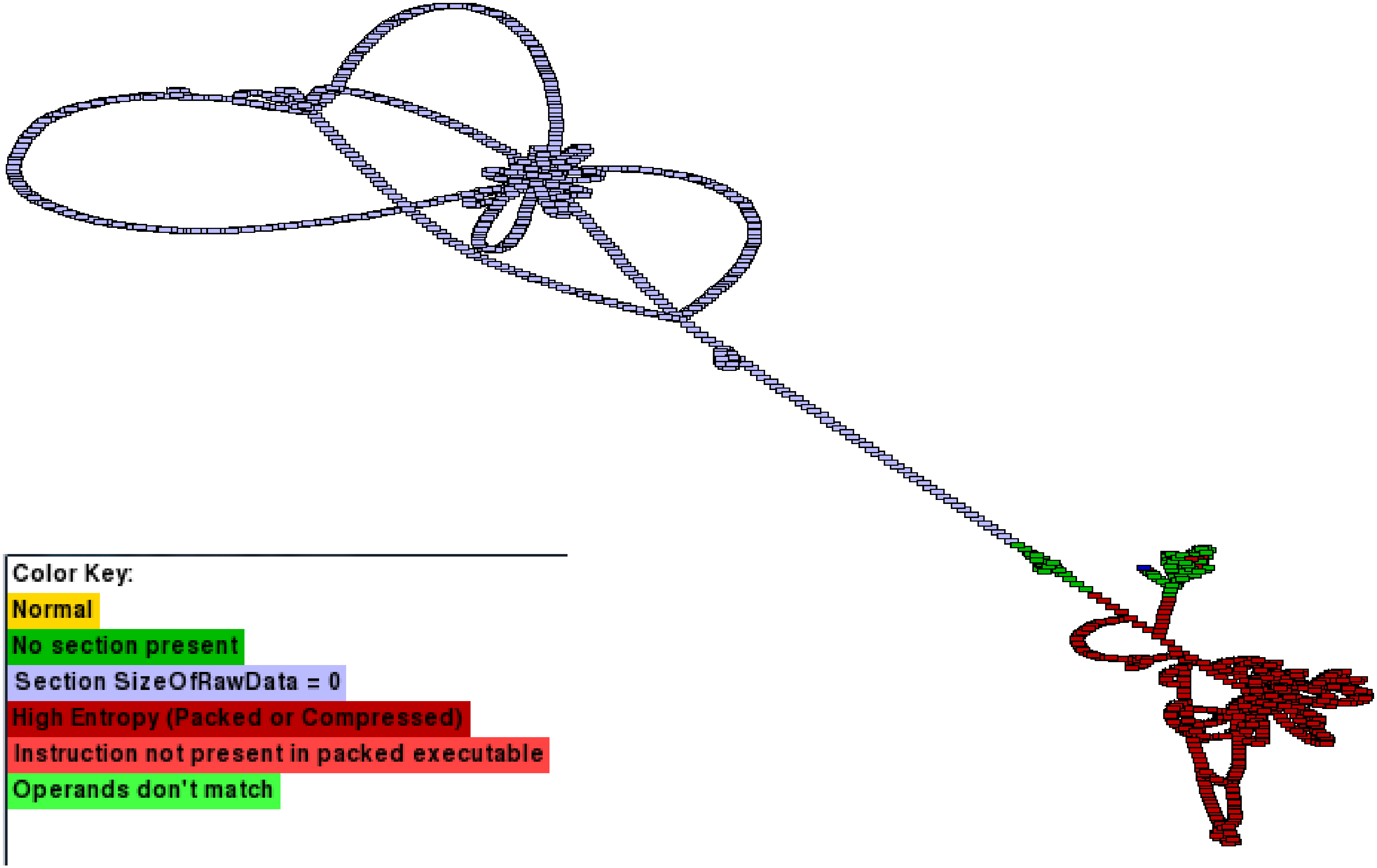


Color Key:
Normal
No section present
Section SizeOfRawData = 0
High Entropy (Packed or Compressed)
Instruction not present in packed executable
Operands don't match

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

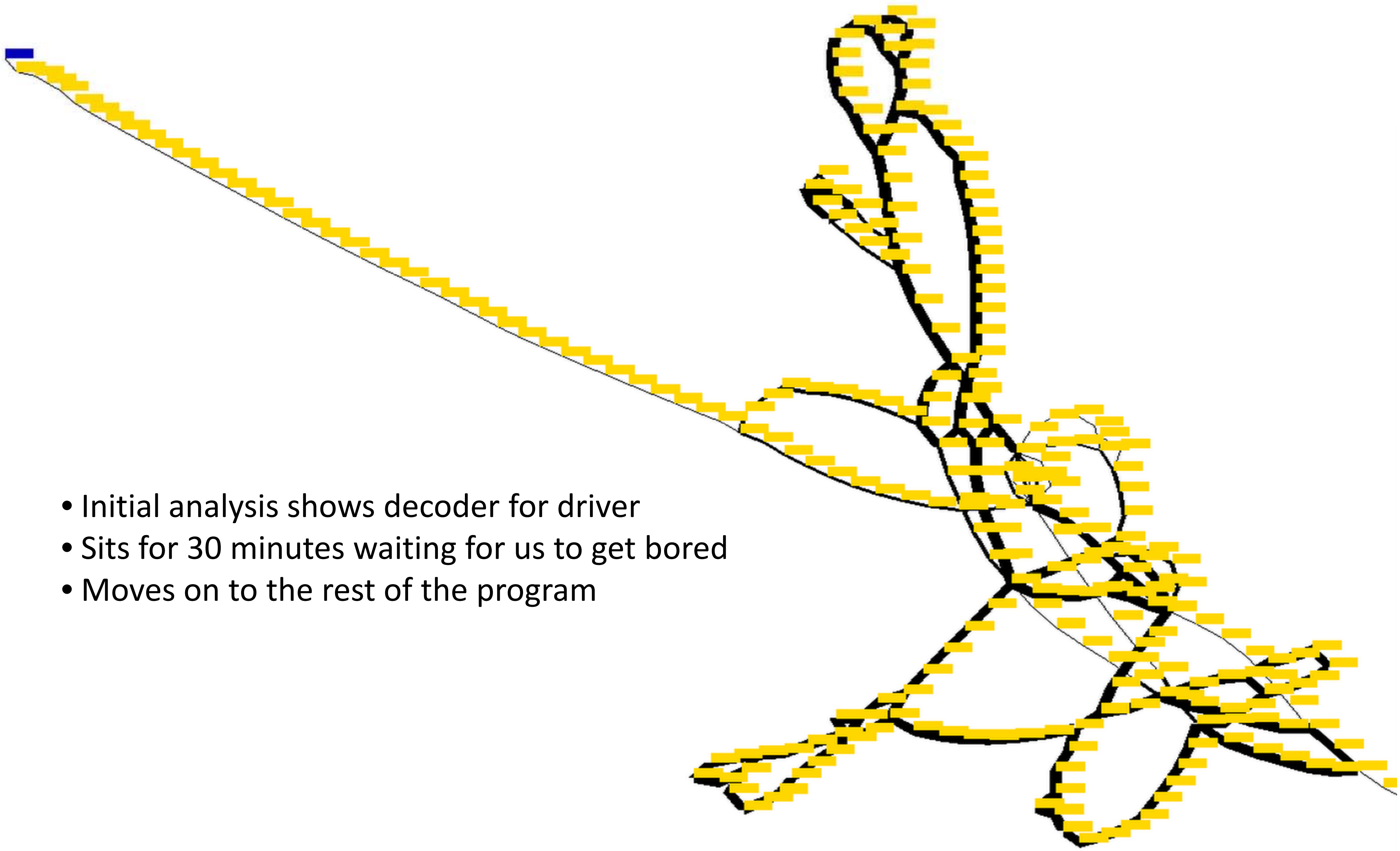
MEW


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

Case Study: Mebroot

- 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

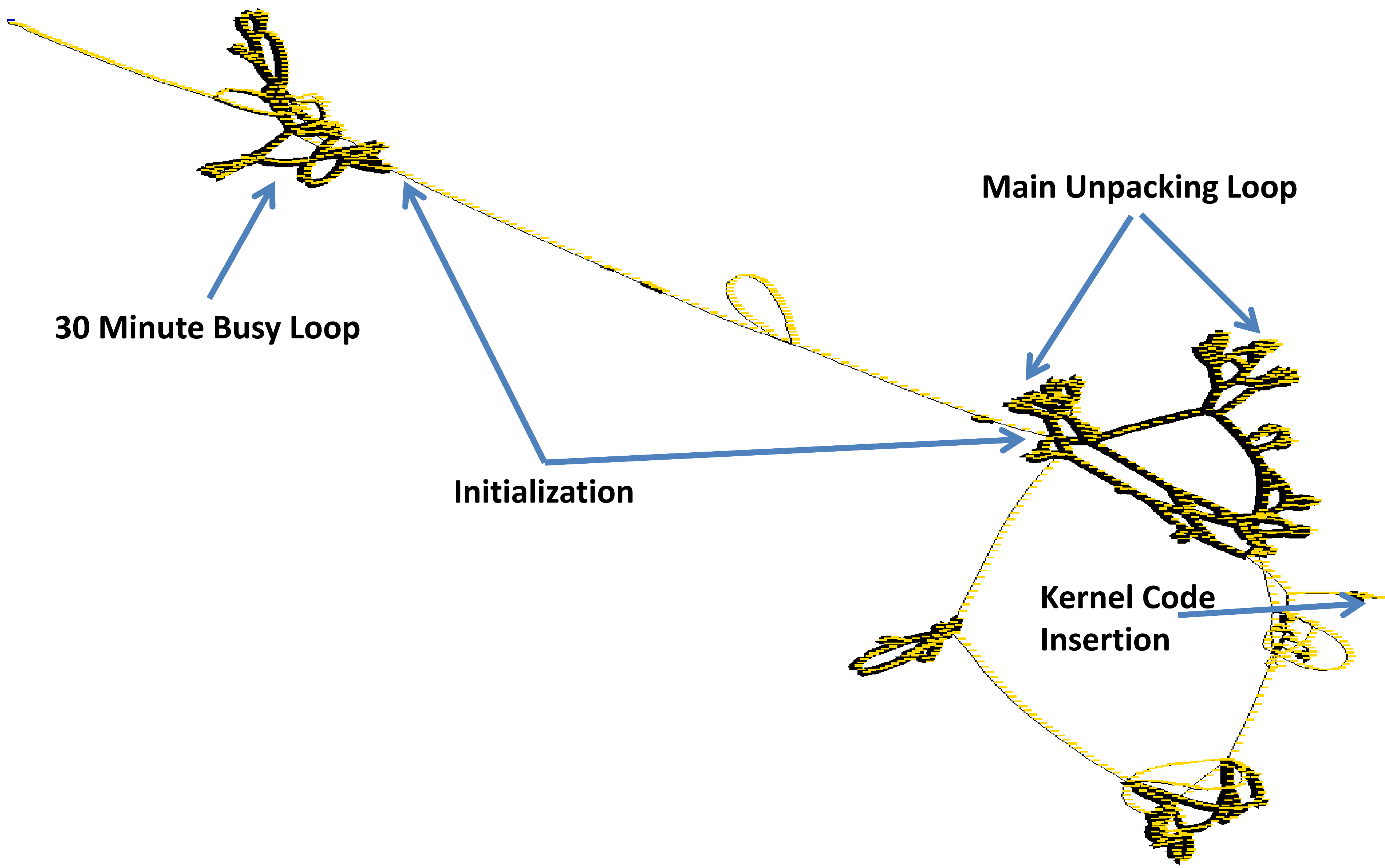
Mebroot – Initial Busy Loop



- 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 – Entire View

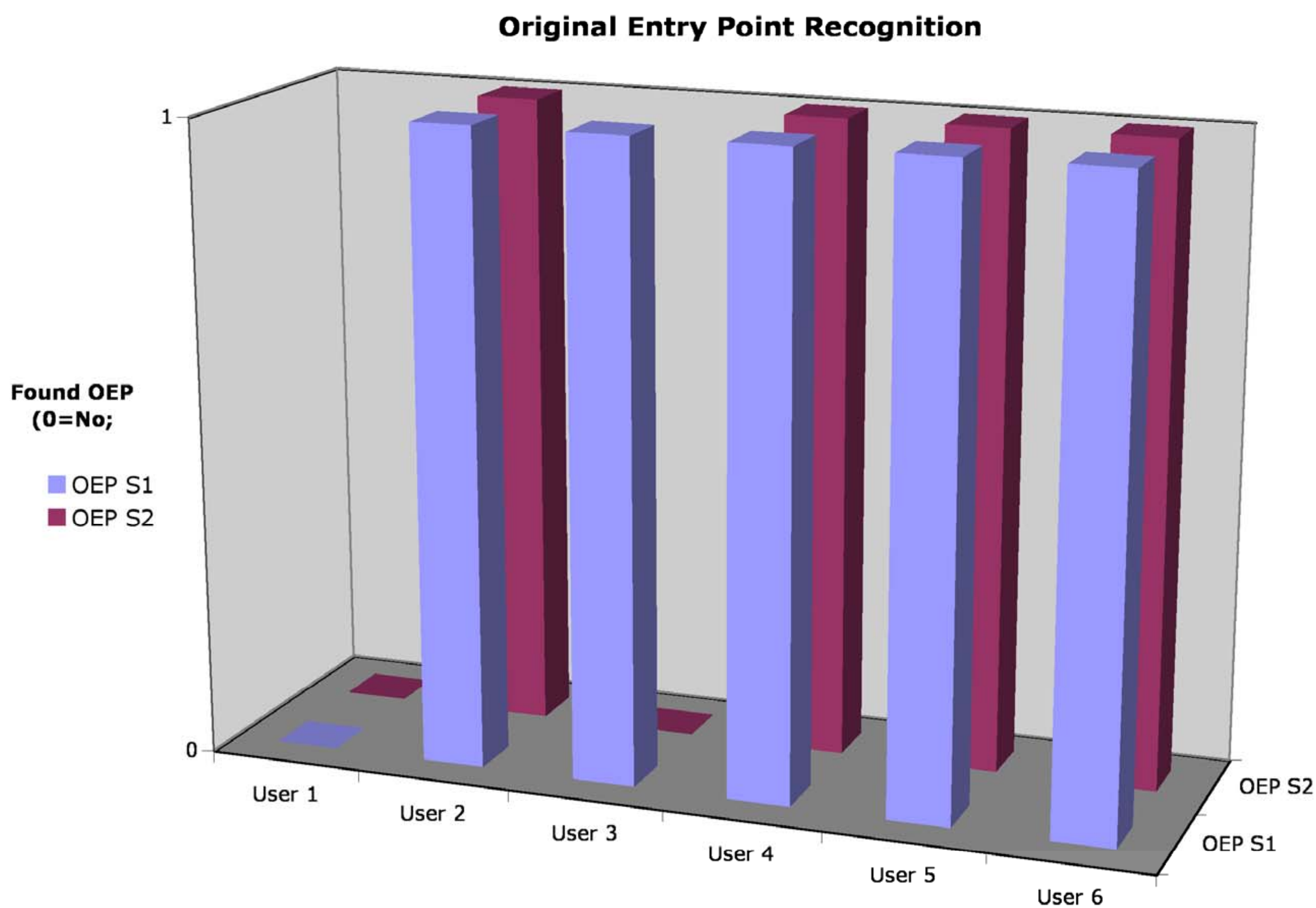


User Study

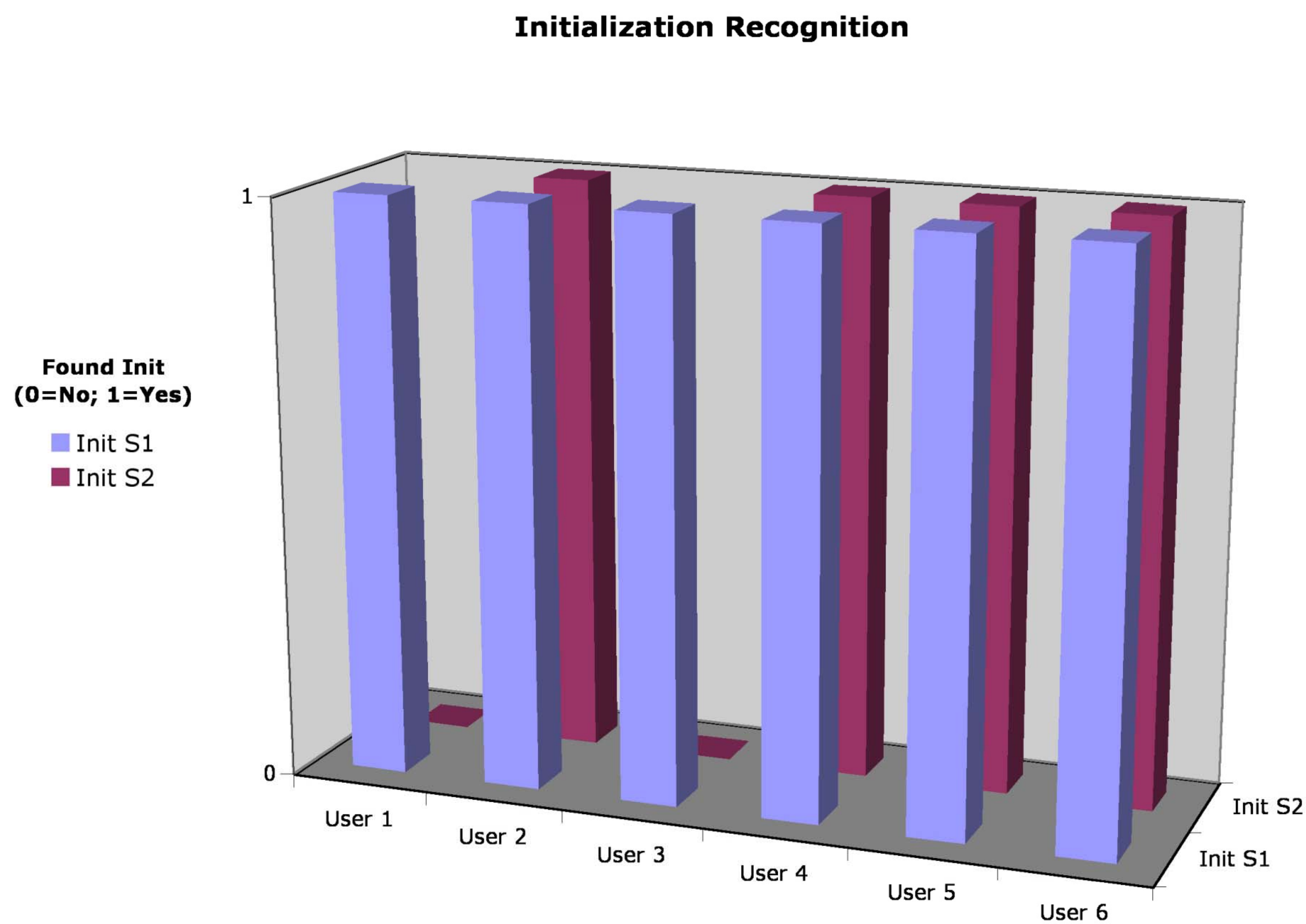
- Students had just completed week long reverse engineering course
- Analyzed two packed samples of the Netbull Virus with UPX and MEW
- Asked to perform a series of tasks based on the typical reverse engineering process
- Asked about efficacy of visualization tool

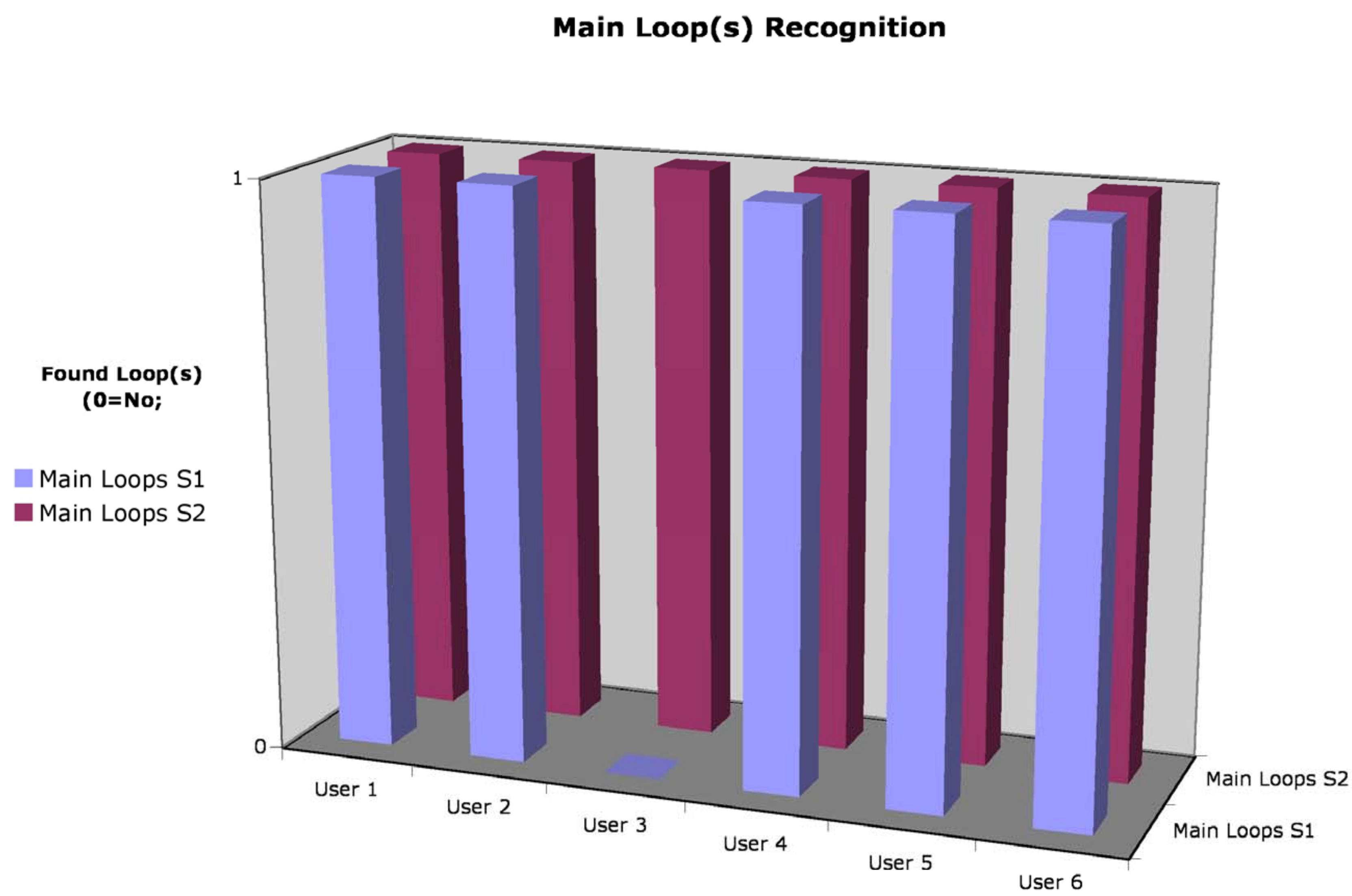
User Study: Tasks Performed

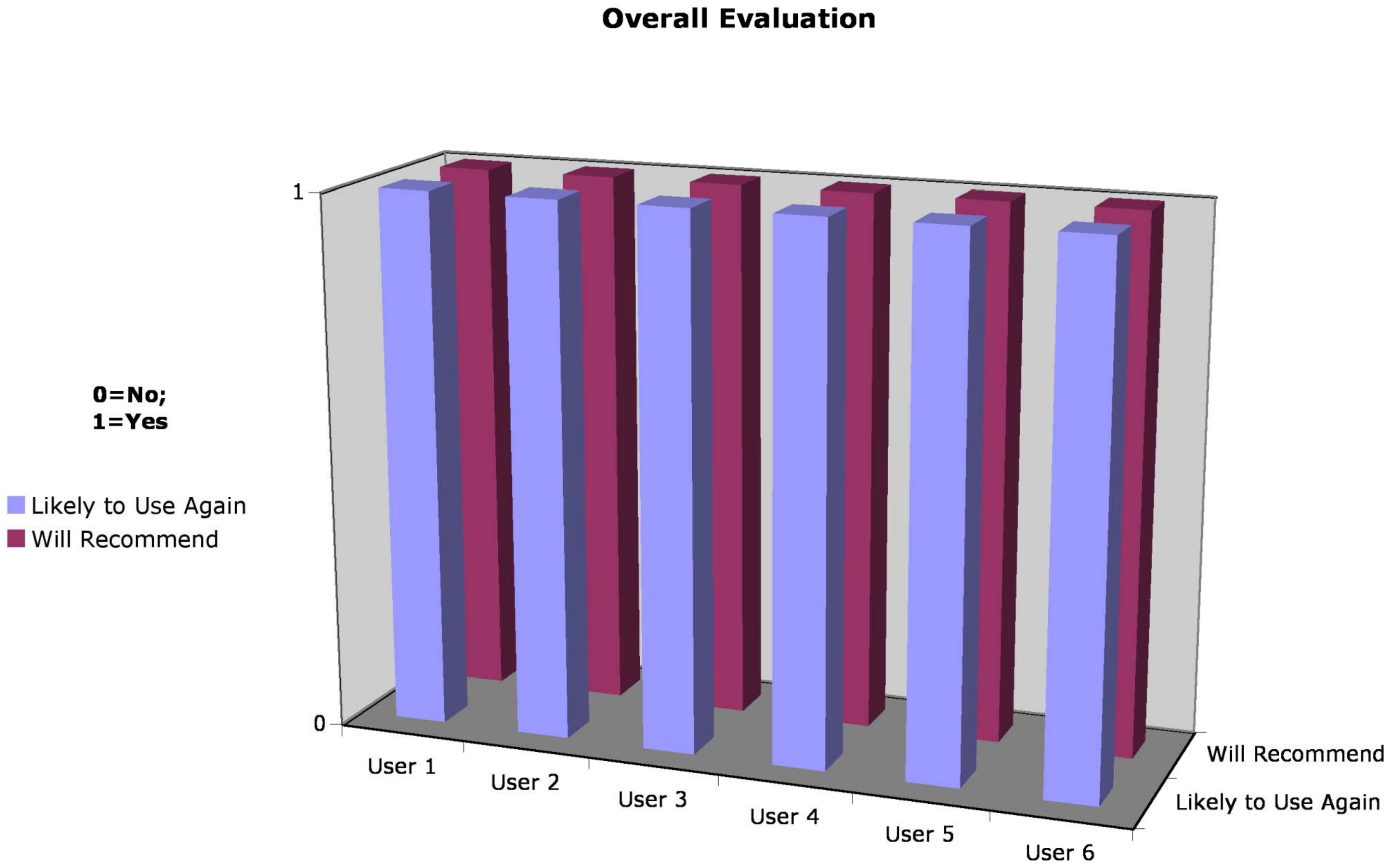
- Find the original entry point (OEP) of the packed samples
- Execute the program to look for any identifying output
- Identify portions of the executable:
  - Packer code
  - Initialization
  - Main loops



Original Entry Point Recognition
1
0
Found OEP
(0=No;
OEP S1
OEP S2
User 1
User 2
User 3
User 4
User 5
User 6
OEP S2
OEP S1

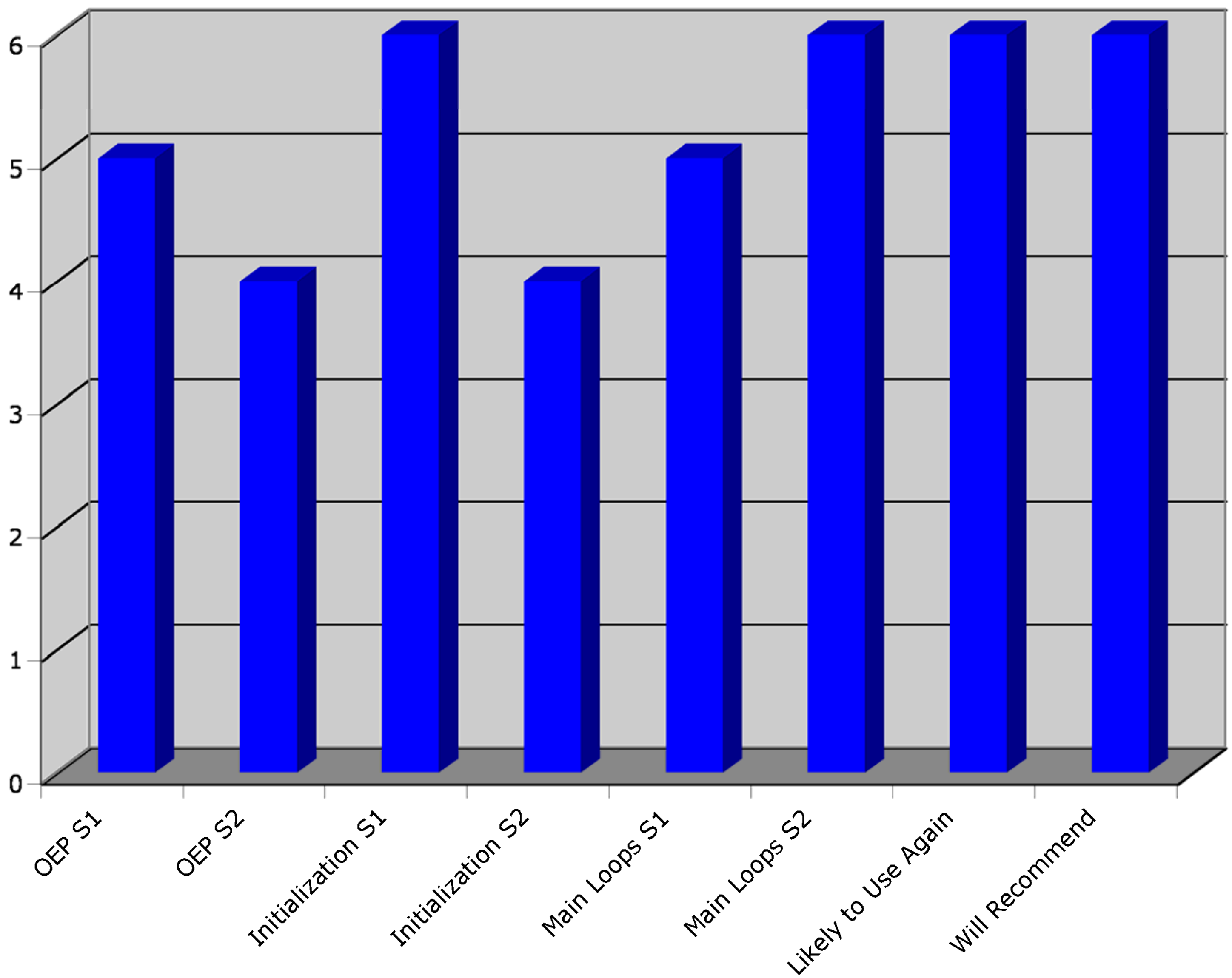

Initialization Recognition
Found Init
(0=No; 1=Yes)
Init S1
Init S2
1
0
User 1
User 2
User 3
User 4
User 5
User 6
Init S2
Init S1


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
0=No;
1=Yes
Likely to Use Again
Will Recommend
1
0
User 1
User 2
User 3
User 4
User 5
User 6
Will Recommend
Likely to Use Again

Results of User Study



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
- Highlight temporal nature of execution
- Memory access visualization
- System call integration
- Function boundaries
- Interactivity with unpacking process

Conclusion

- Overall process for analyzing and reverse engineering malware is shortened
- Program phases readily identified
- Integration with existing tools
- Preliminary user study shows tool holds promise for speeding up reverse engineering

Questions?

- Source, tools, and latest slides can be found at: http://www.offensivecomputing.net
- If you use the tool, please give feedback
- Contact info: dquist@nmt.edu