MALWARE VISUALIZATION IN 3D

Shmoocon 2012

Danny Quist, Ph.D.

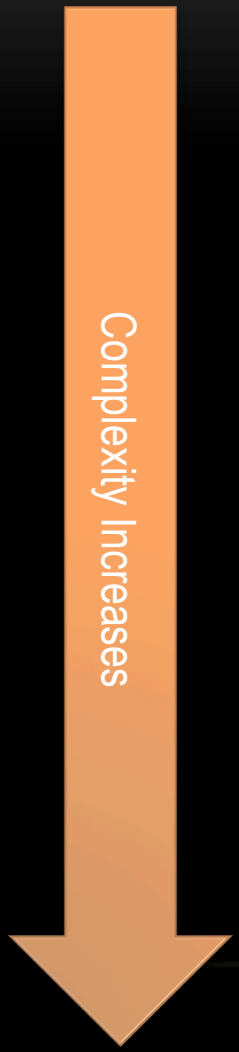
Los Alamos National Laboratory

Advanced Computing Solutions

OVERVIEW

- Complexities of Reverse Engineering
- Malware analysis
- Goals of Visualization
- VERA
- New VERA Features
- DEMOS!

REVERSE ENGINEERING PROCESS



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

COMPLEXITIES OF REVERSE ENGINEERING

- Most malware is compiled Intel x86 Assembly

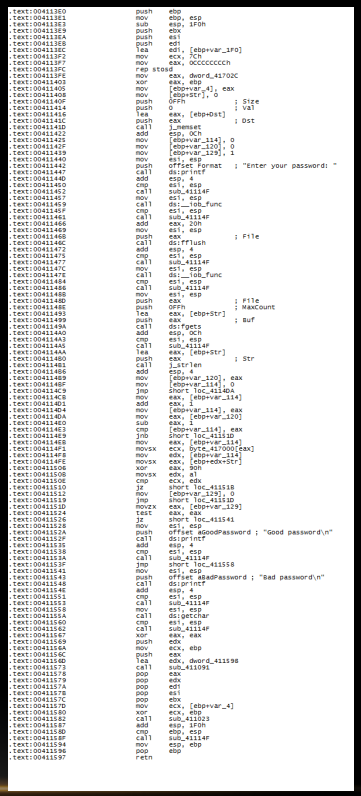
```
char pw[] = "\xdc\xc5\xdc\xca\xb0\xa3\xb0\xe9"
            "\xf5\xf1\xe2\xe3\xb0\xff\xf6\xb0"
            "\xfd\xe9\xb0\xfc\xf9\xf6\xf5\xb0"
            "\xff\xfe\xb0\xe4\xf8\xf5\xb0\xfc"
            "\xf9\xfe\xf5\xbe";

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

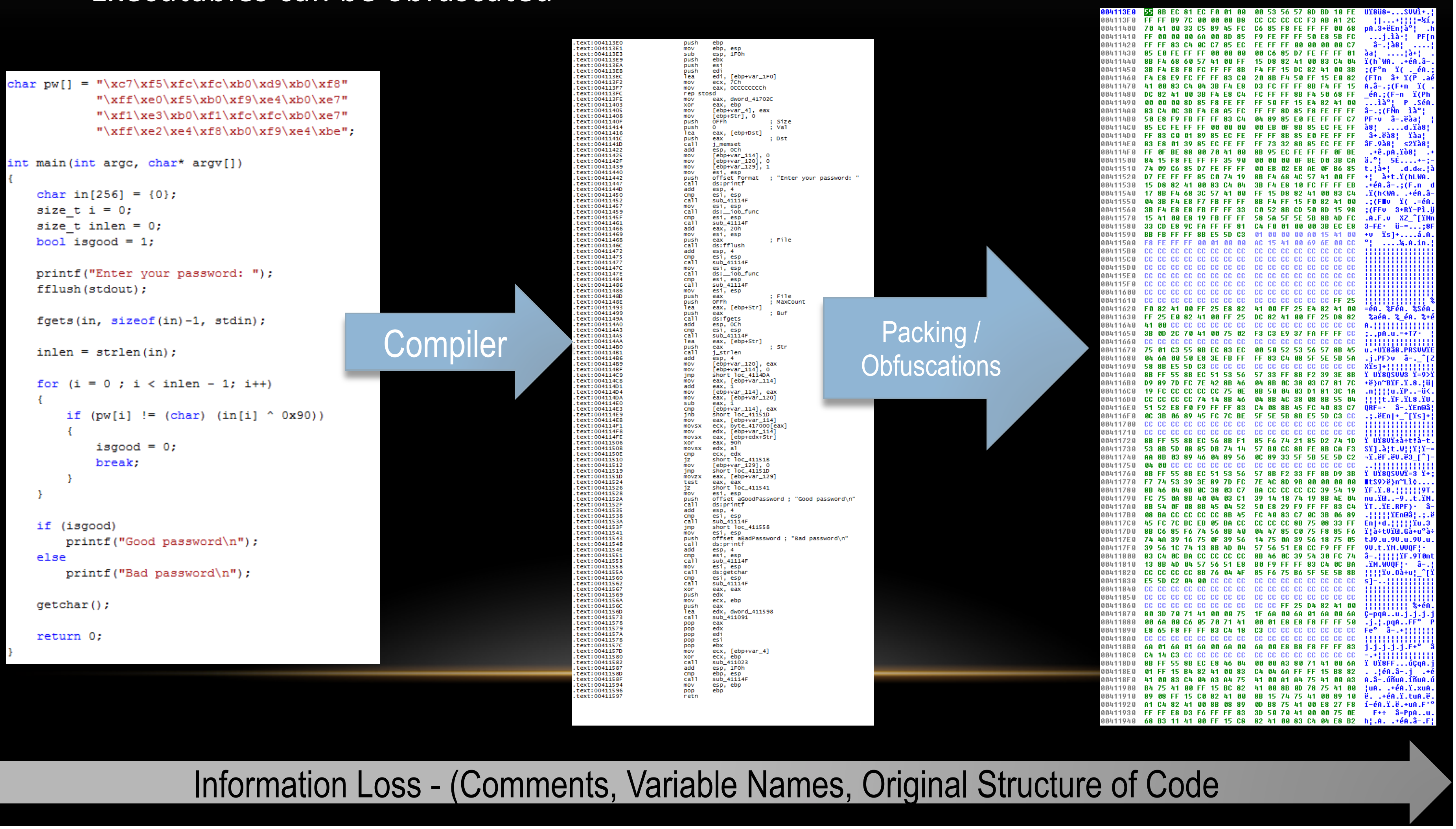
Reverse Engineering



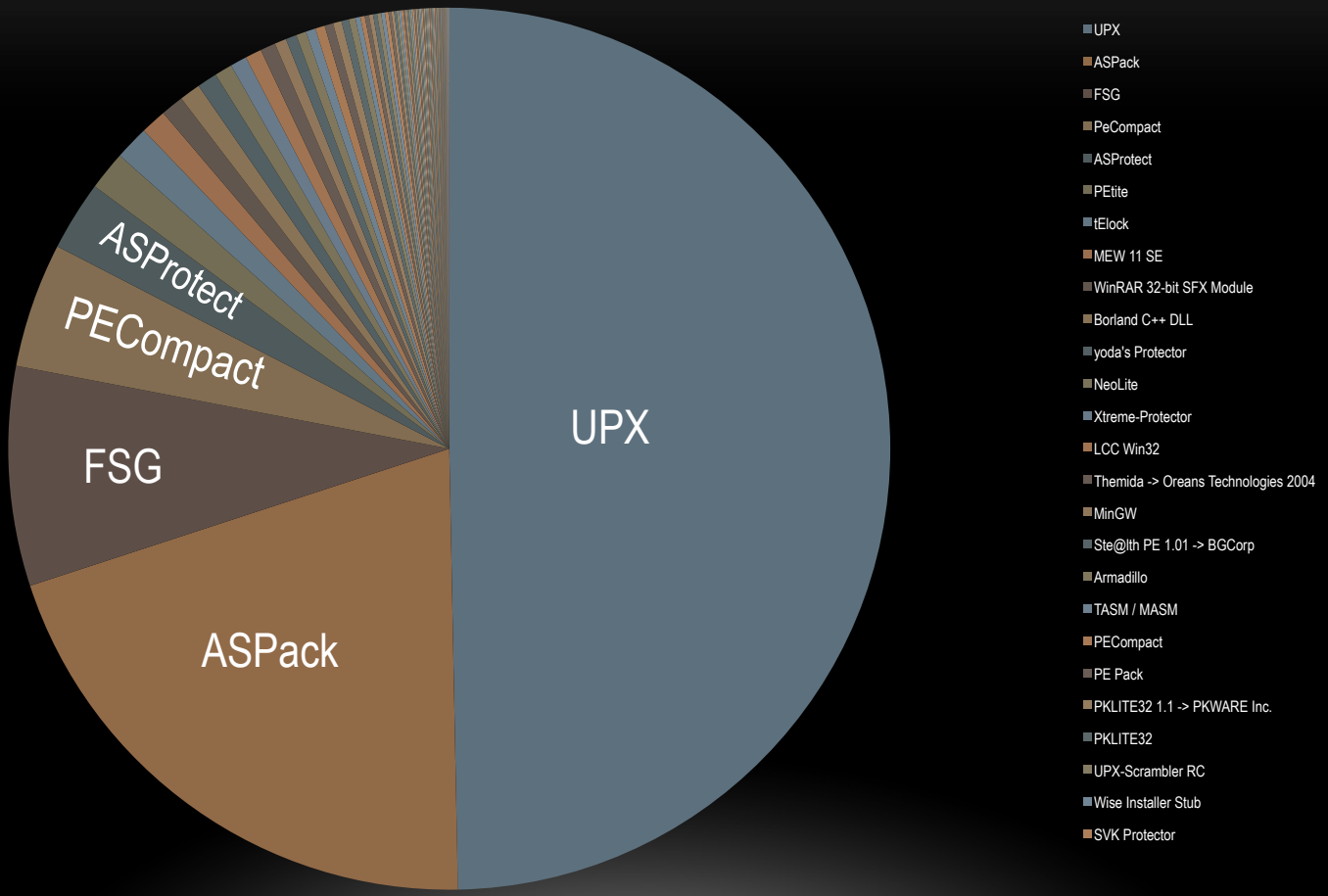
Relevant Assembly Code

COMPLEXITIES OF REVERSE ENGINEERING

- Executables can be obfuscated

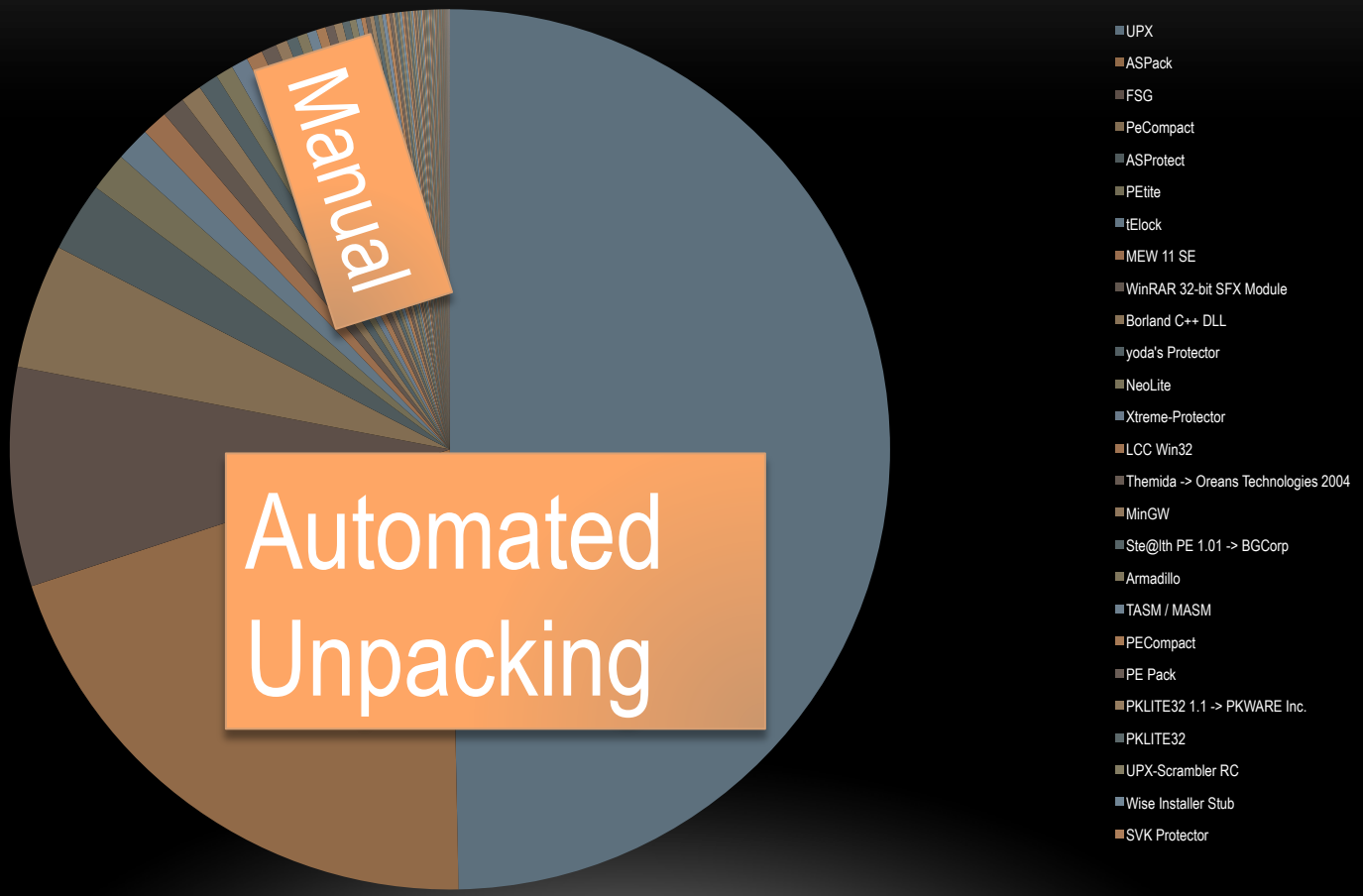


TYPES OF PACKERS



PEiD scanning results from 3.6 million samples from Offensive Computing

UNPACKING TECHNIQUES

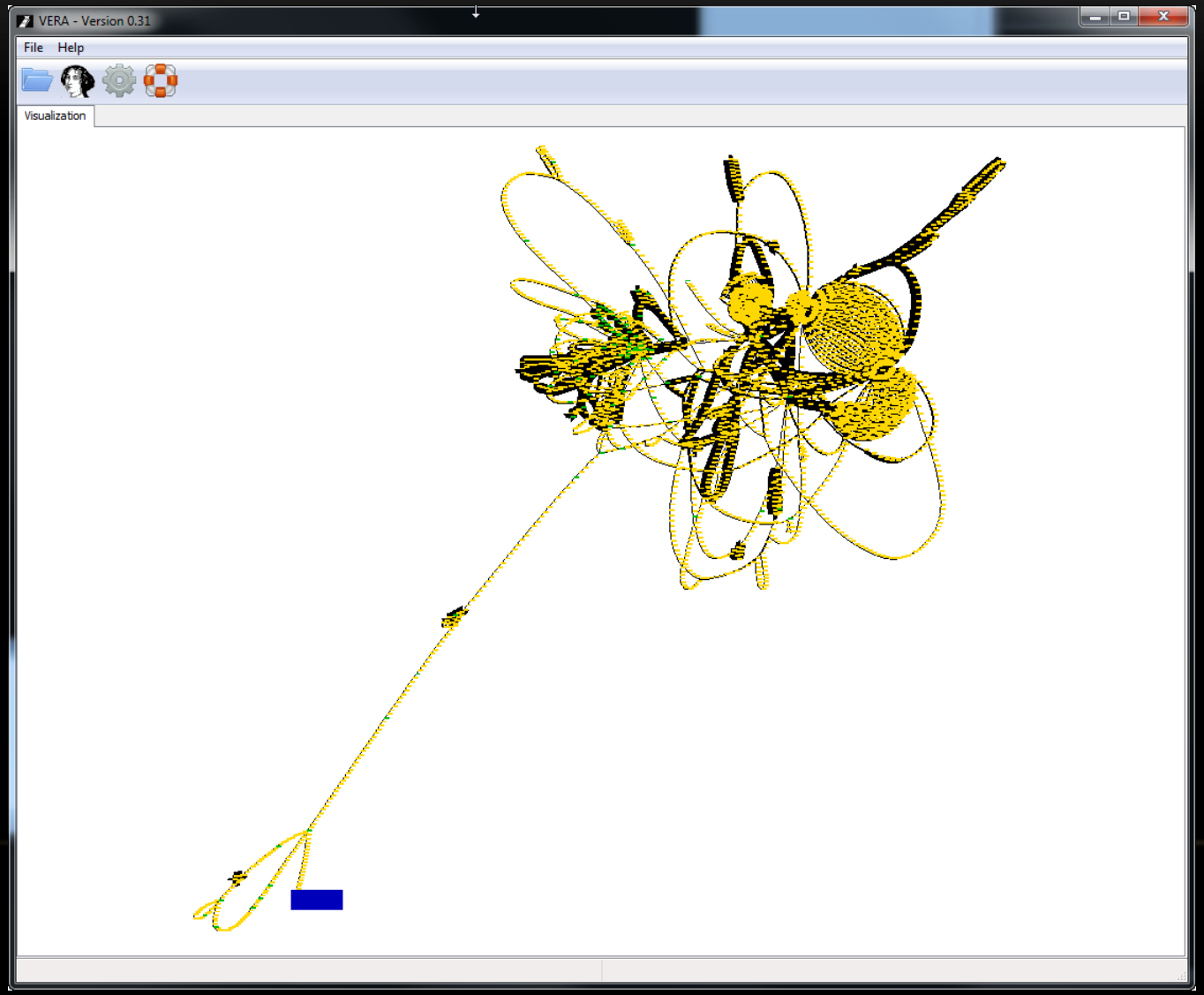


PEiD scanning results from 3.6 million samples from Offensive Computing

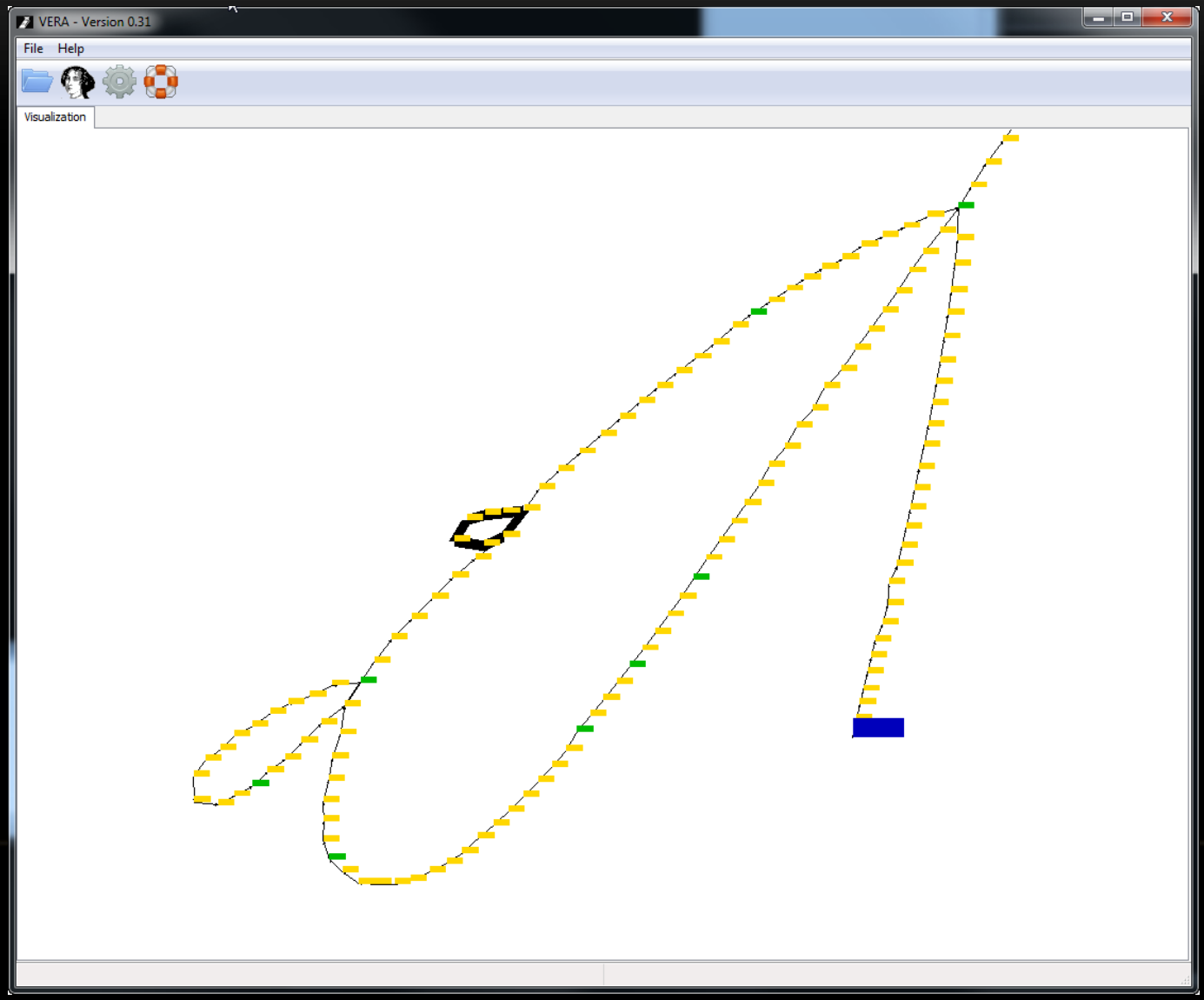
VERA

- Visualization for reverse engineering
- Force directed graph of execution traces
- Helps with determining where to start the reverse engineering process
- Cuts down on RE time
- Makes unpacking easier

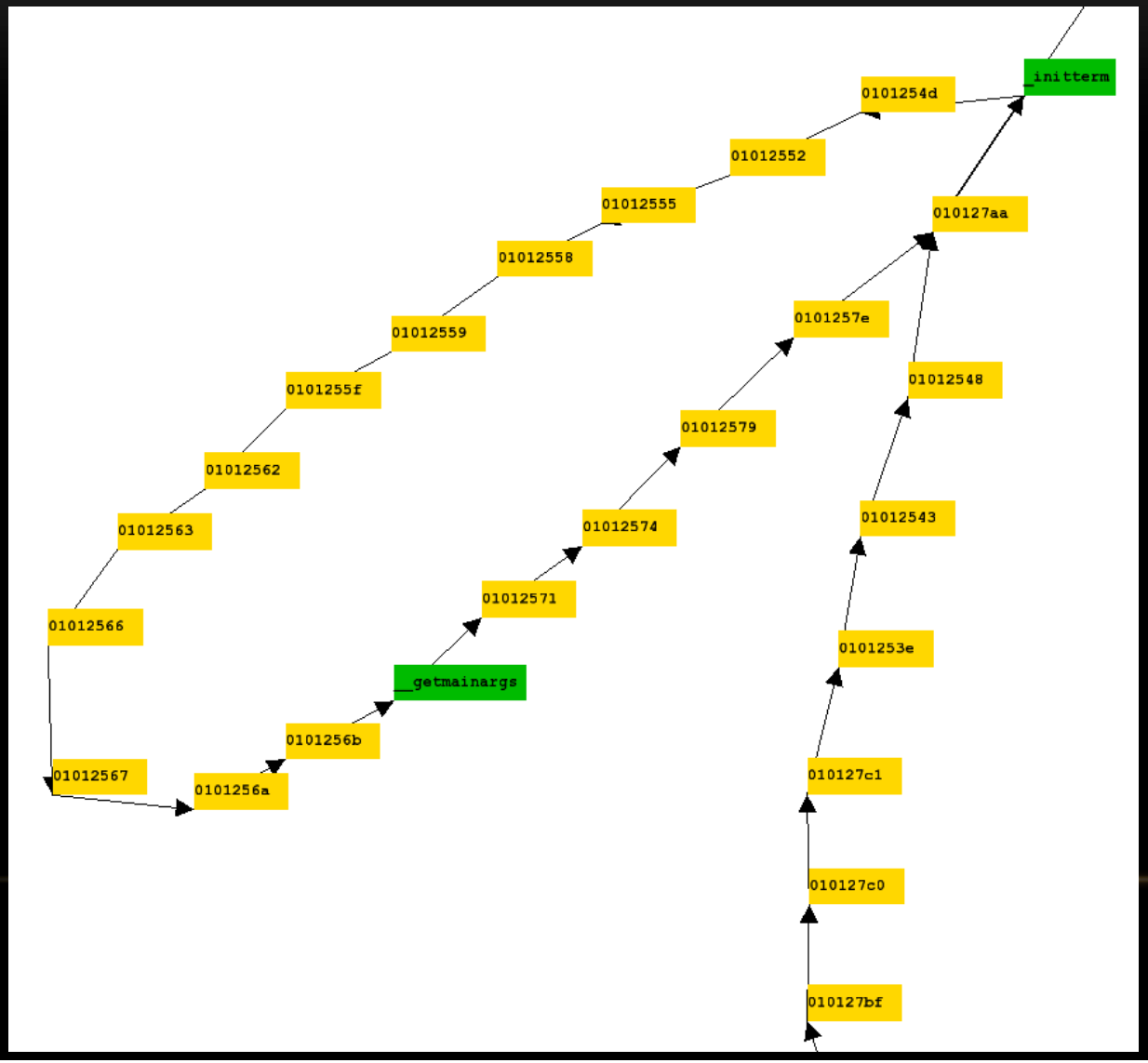
VERA - SCREENSHOTS



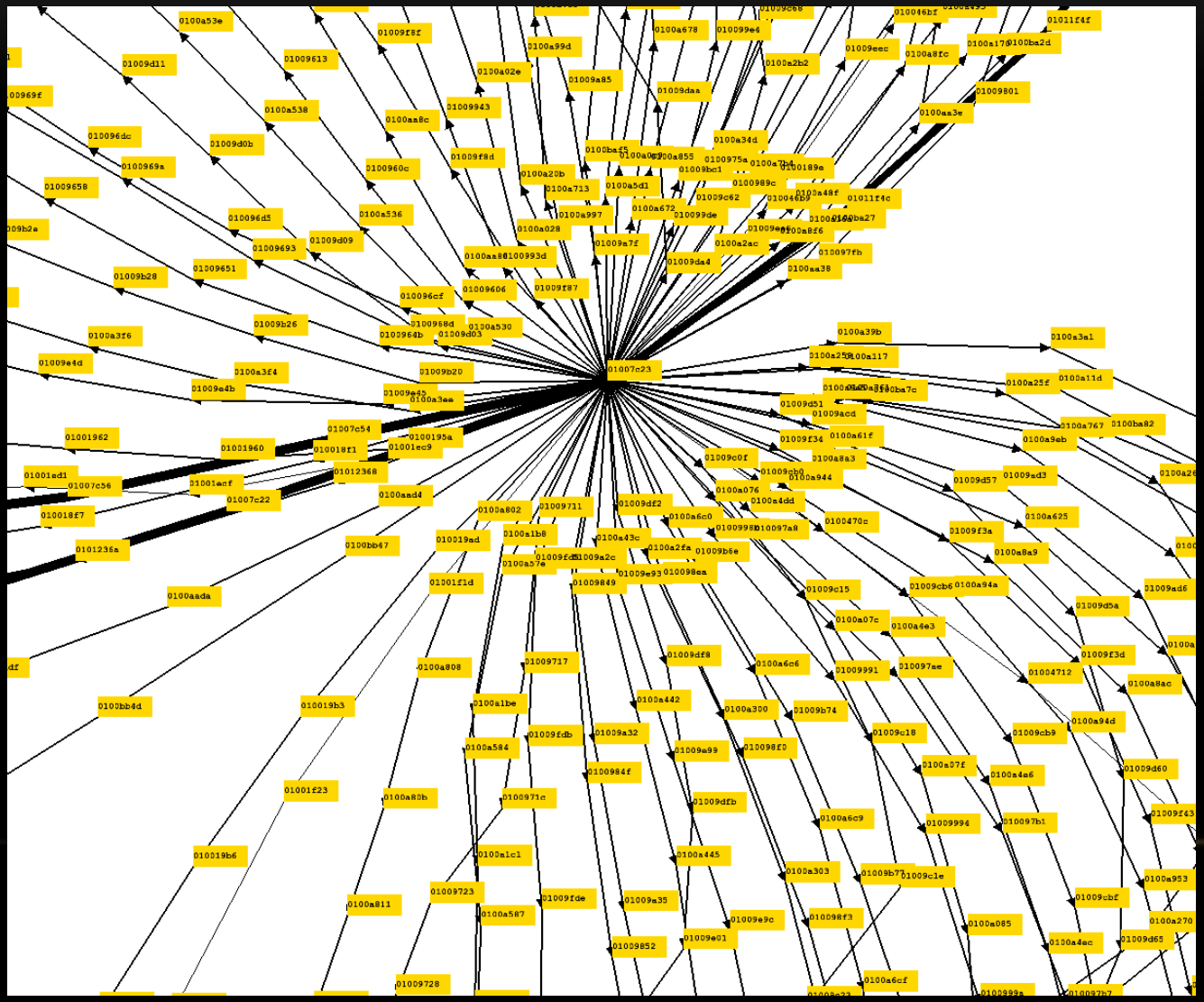
VERA - SCREENSHOTS



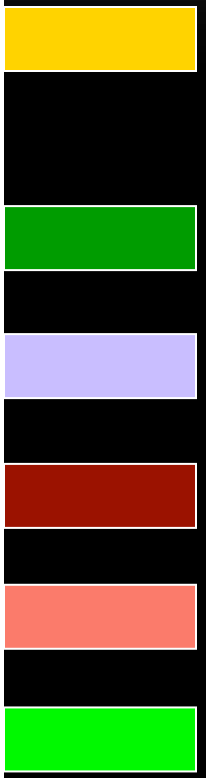
VERA - SCREENSHOTS



VERA - SCREENSHOTS

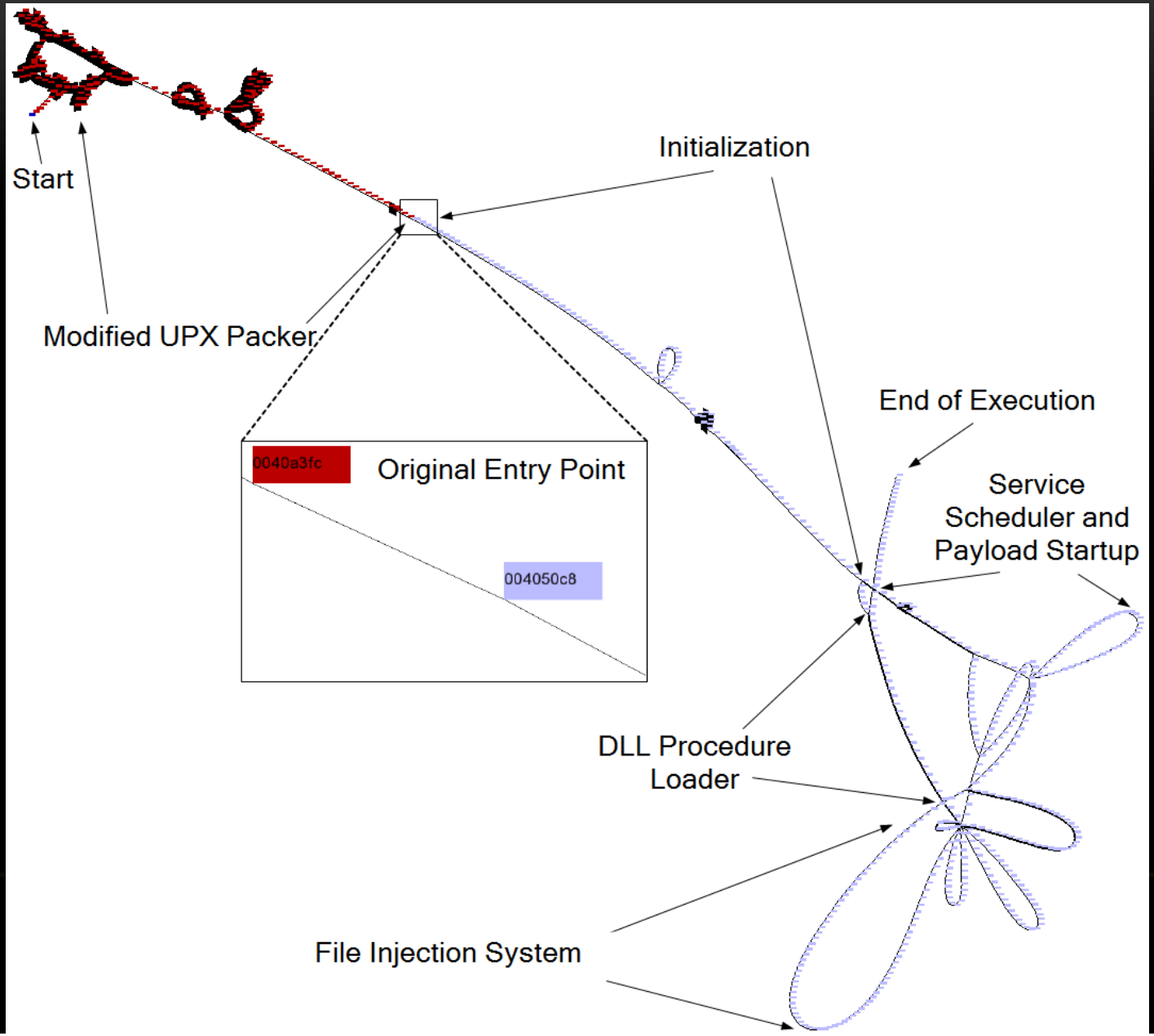


WHAT THE COLORS MEAN

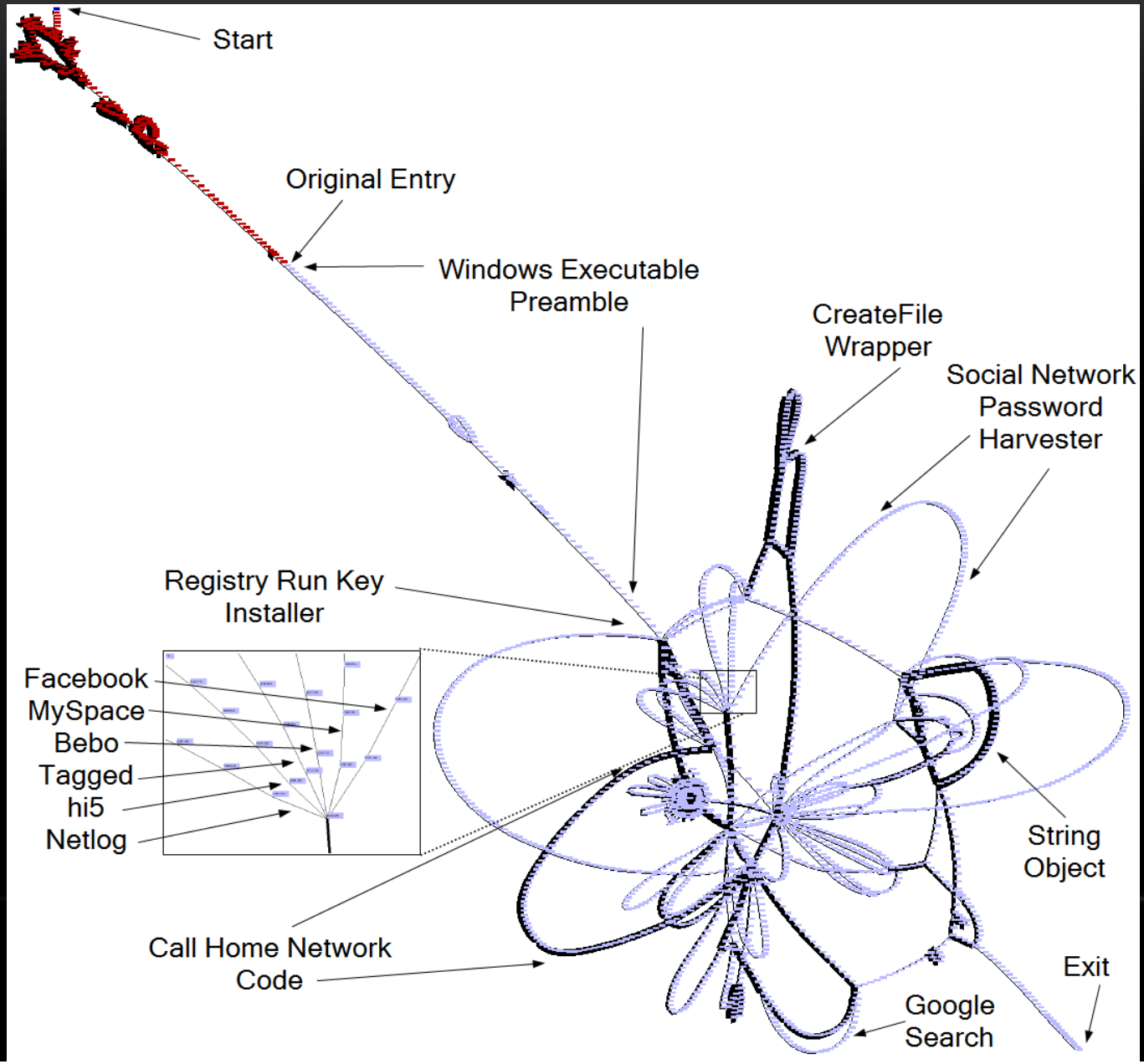


- **Yellow** – Normal uncompressed low-entropy section data
- **Dark Green** – DLL / API / Section not present
- **Light Purple** – SizeOfRawData = 0
- **Dark Red** – High Entropy
- **Light Red** – Instructions not in the packed exe
- **Lime Green** – Operands don’t match

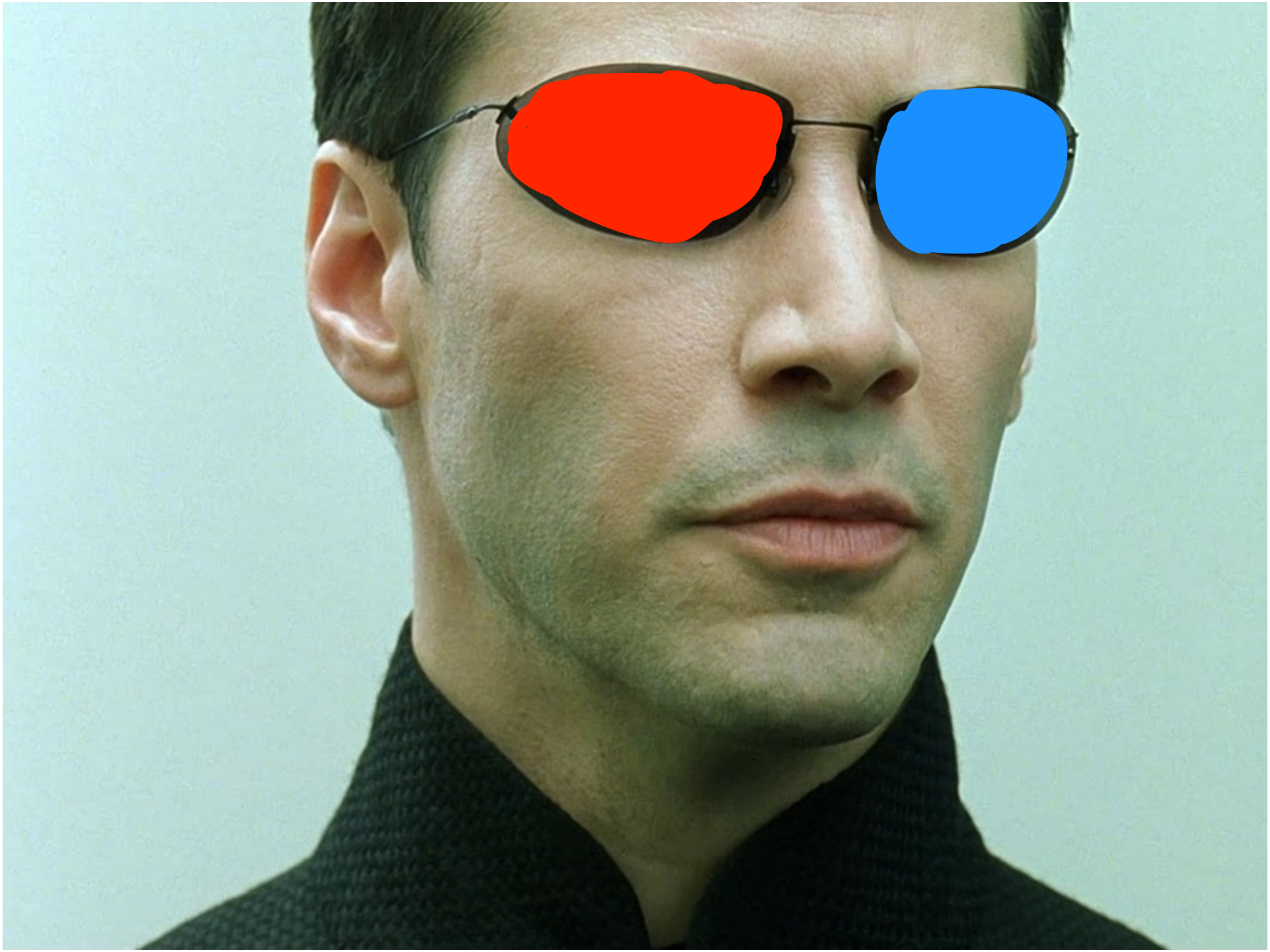
KOOBFACE INITIAL INSTALLATION



KOOBFACE SERVICE



NEW VERA FEATURES



NOTES ON THE 3D VISUALIZATION

- Uses 3D layout from Ubigraph tool
- Force directed, O(|V|log|V| + |E|)
- Heavily threaded, high performance
- Only for the Mac, Linux
- Integrated into main VERA GUI for export

UBIGRAPH

- Dynamic Multilevel Graph Visualization CoRR 2007
- Todd Veldhuizen
  - Where are you?
  - People would like to pay you
- http://ubietylab.net

Dynamic Multilevel Graph Visualization

Todd L. Veldhuizen*

December 11, 2007

Abstract

We adapt multilevel, force-directed graph layout techniques to visualizing dynamic graphs in which vertices and edges are added and removed in an online fashion (i.e., unpredictably). We maintain multiple levels of coarseness using a dynamic, randomized coarsening algorithm. To ensure the vertices follow smooth trajectories, we employ dynamics simulation techniques, treating the vertices as point particles. We simulate fine and coarse levels of the graph simultaneously, coupling the dynamics of adjacent levels. Projection from coarser to finer levels is adaptive, with the projection determined by an affine transformation that evolves alongside the graph layouts. The result is a dynamic graph visualizer that quickly and smoothly adapts to changes in a graph.

1 Introduction

Our work is motivated by a need to visualize dynamic graphs, that is, graphs from which vertices and edges are being added and removed. Applications include visualizing complex algorithms (our initial motivation), ad hoc wireless networks, databases, monitoring distributed systems, realtime performance profiling, and so forth. Our design concerns are:

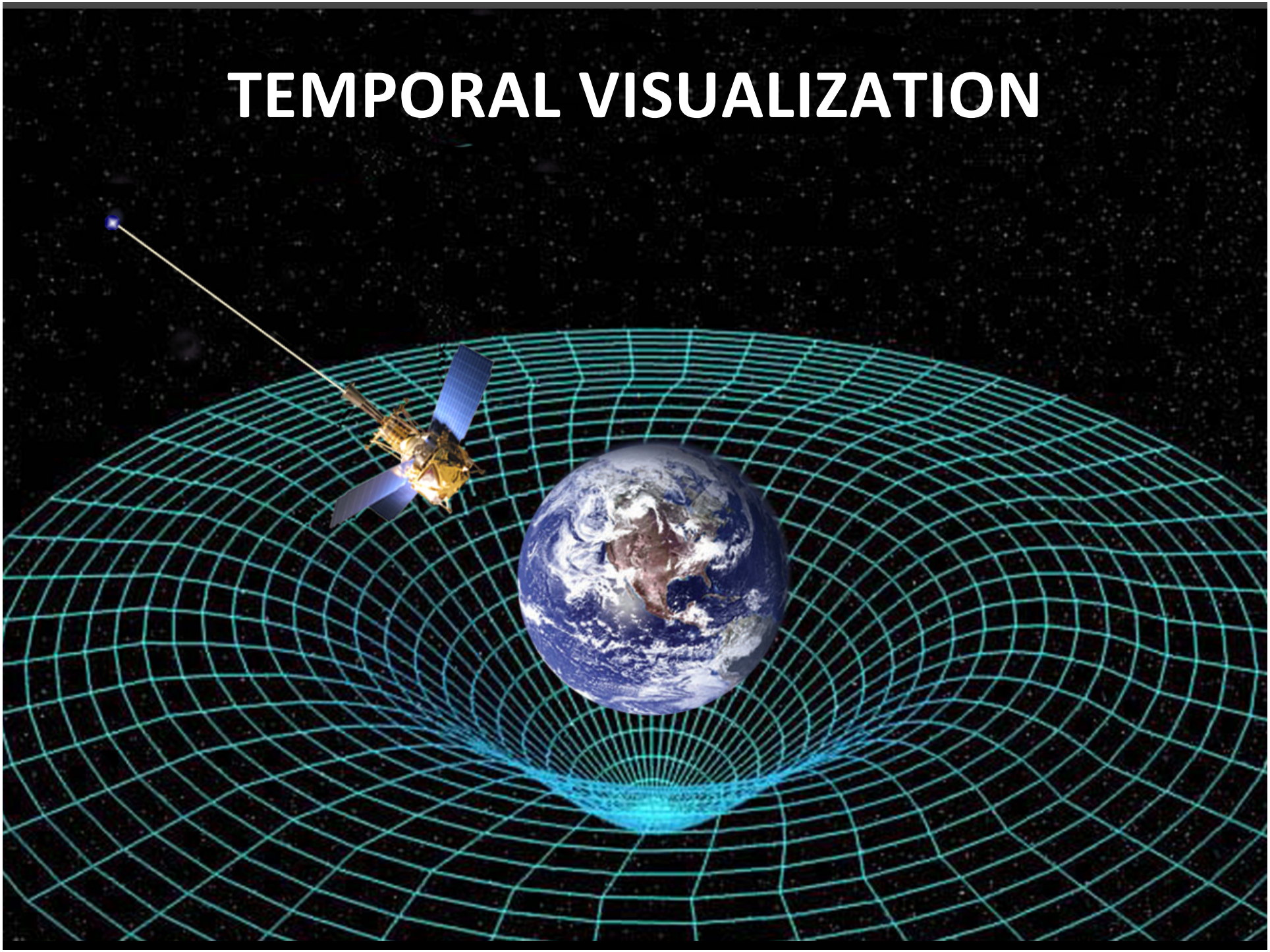
D1. The system should support online revision of the graph, that is, changes to the graph that are not known in advance. Changes made to the graph may radically alter its structure.

D2. The animations should appear smooth. It should be possible to visually track vertices as they move, avoiding abrupt changes.

*Dept. of Electrical and Computer Engineering, University of Waterloo, Canada. Email: tveldhui@acm.org

1

TEMPORAL VISUALIZATION



WHY TIME IS IMPORTANT TO RE

- Understanding the flow of events helps to reconstruct what it does
- Example: Mebroot
  - Initial 30 minute busy loop
  - Is the malware broken?
  - Afterwards functional and persistent
- Complicated samples can get obfuscated quickly

TEMPORAL DEMOS

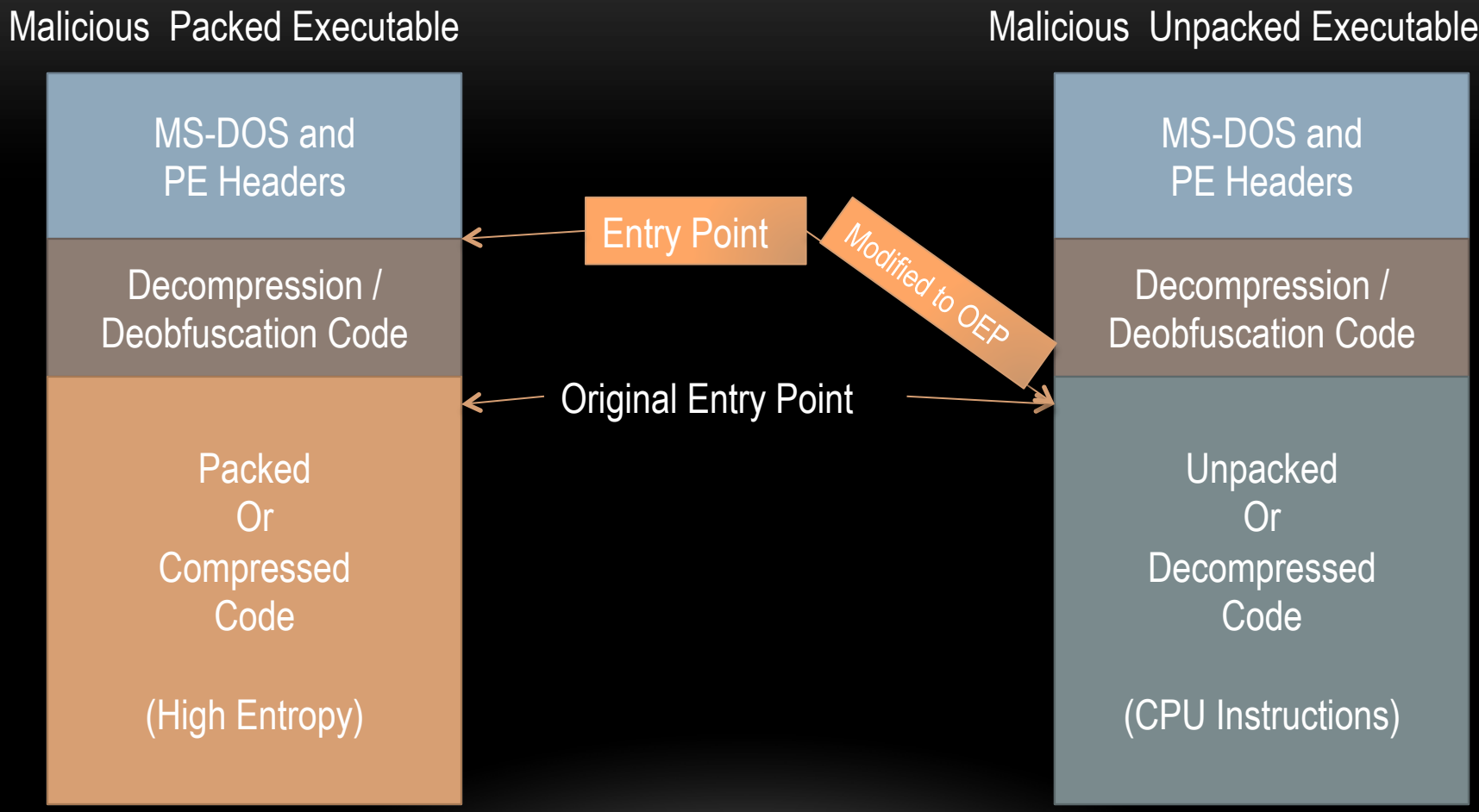
SEARCH FEATURE

- Commonly requested feature to be able to search for addresses in visualization
- Allows for synchronization between IDA/OllyDBG with the main visualization
- Reduces hunting and searching for APIs
- Faster code discovery

VISUAL UNPACKING

- For known packers it is trivially easy (once you know how)
- For unknown packers it's a matter of determining functionality
- Automated methods are fairly robust
  - Ether
  - Polyunpack
  - Etc.

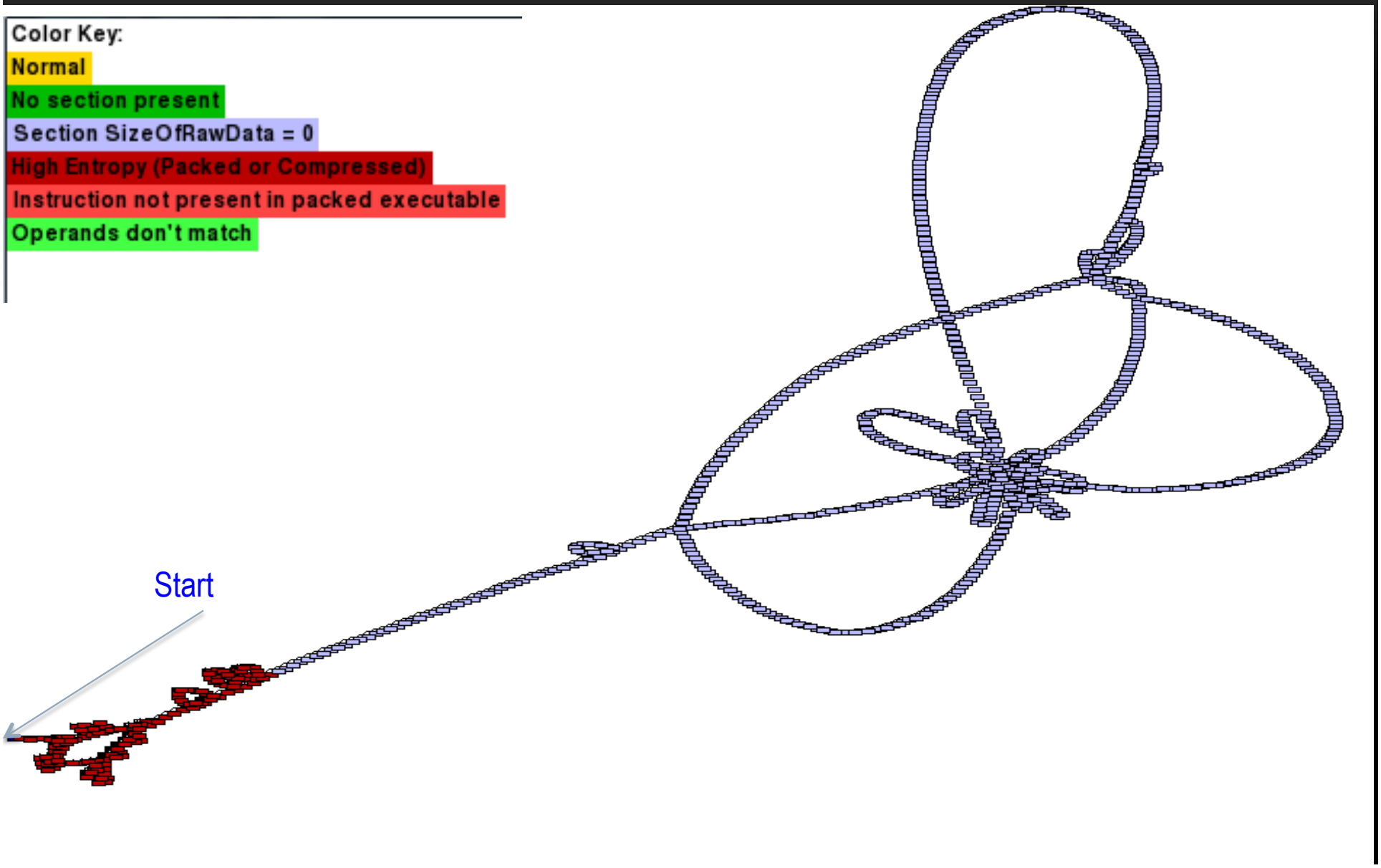
ARCHITECTURE OF A PACKER



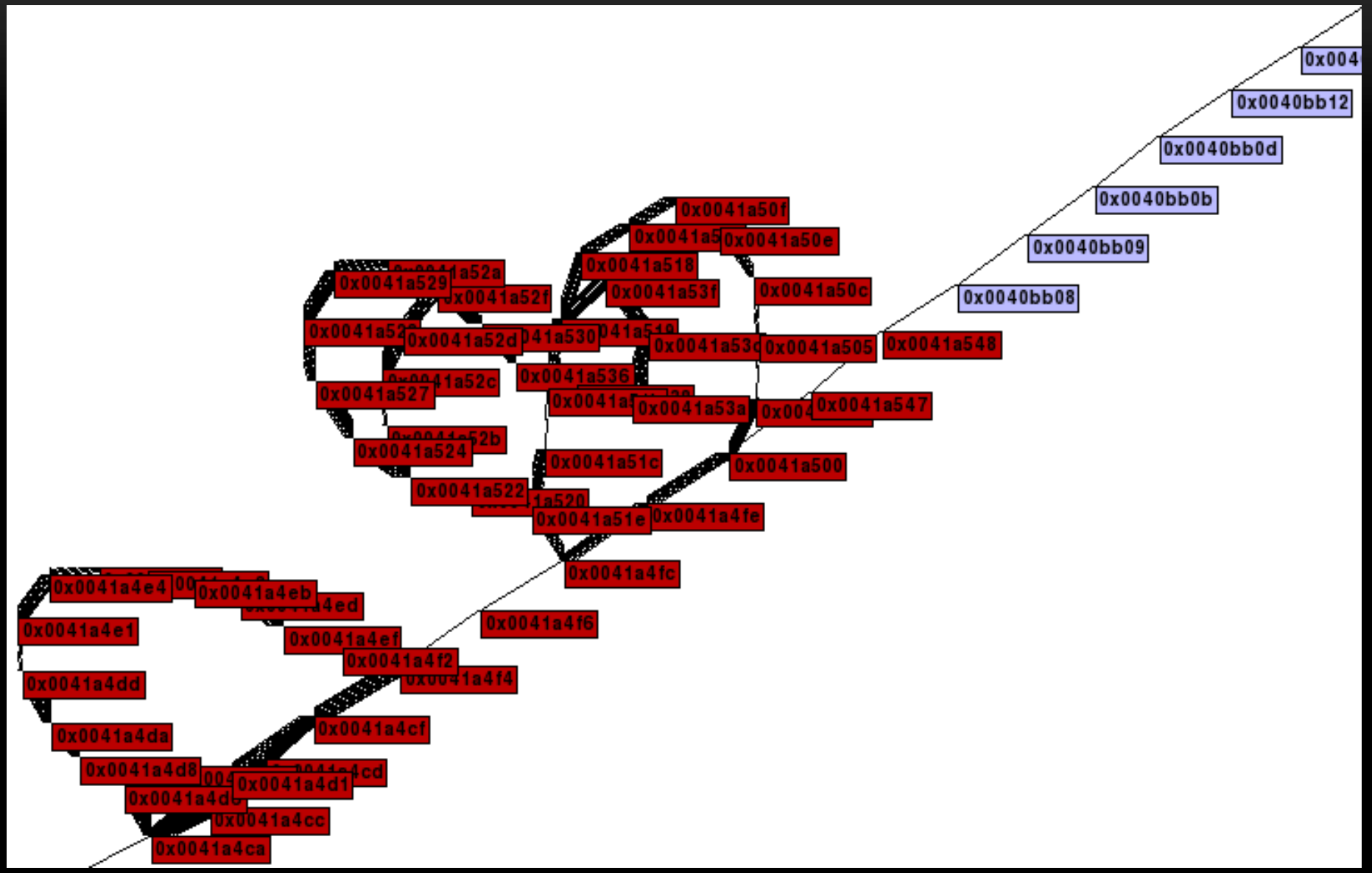
ARCHITECTURE OF A PACKER

- Very rare that malware is written in pure assembly
- Most malware uses traditional software development tools (Compilers, etc.)
- Modern malware is a complex, commercial piece of software
- Obfuscations added afterwards before deployment

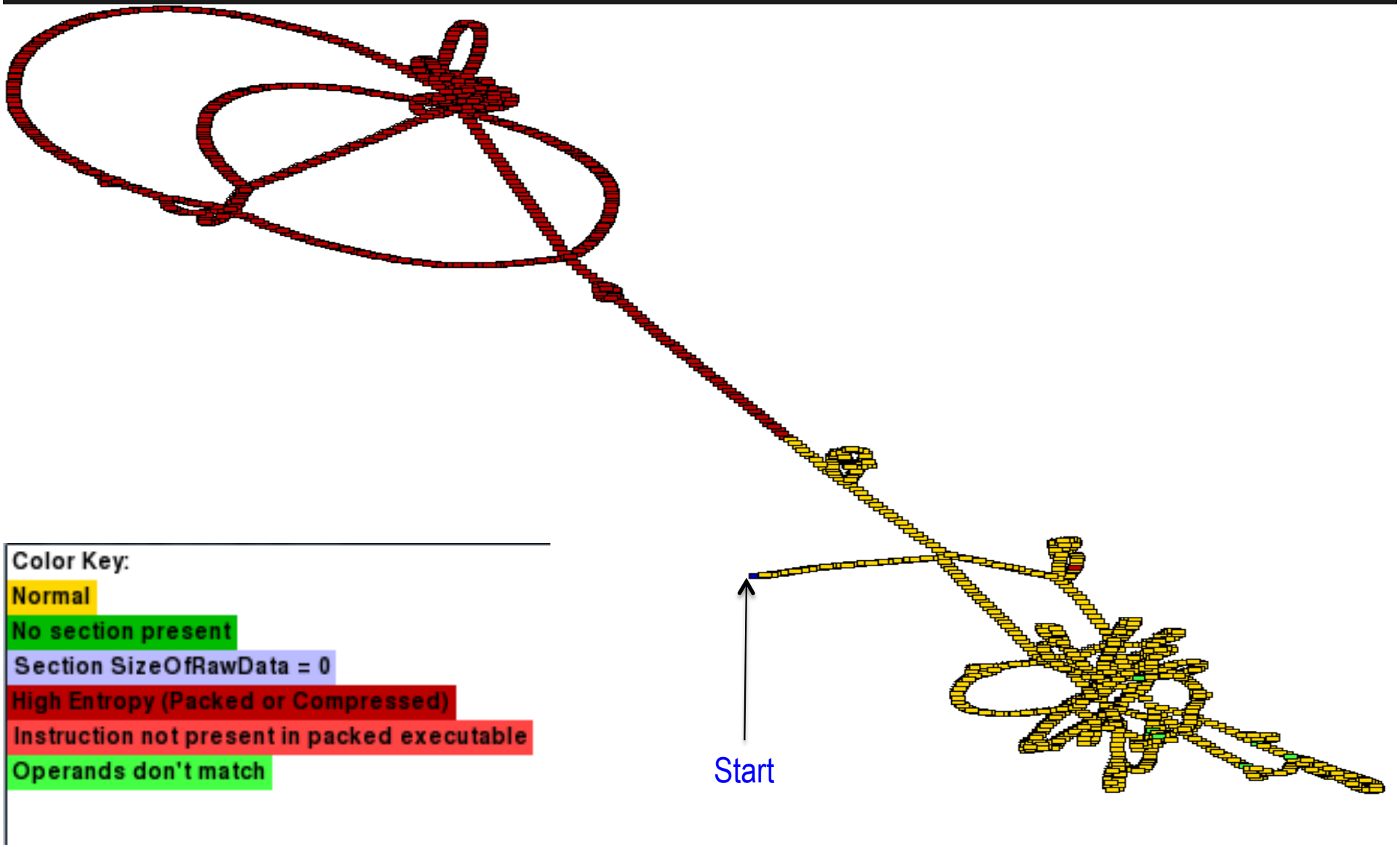
UPX


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

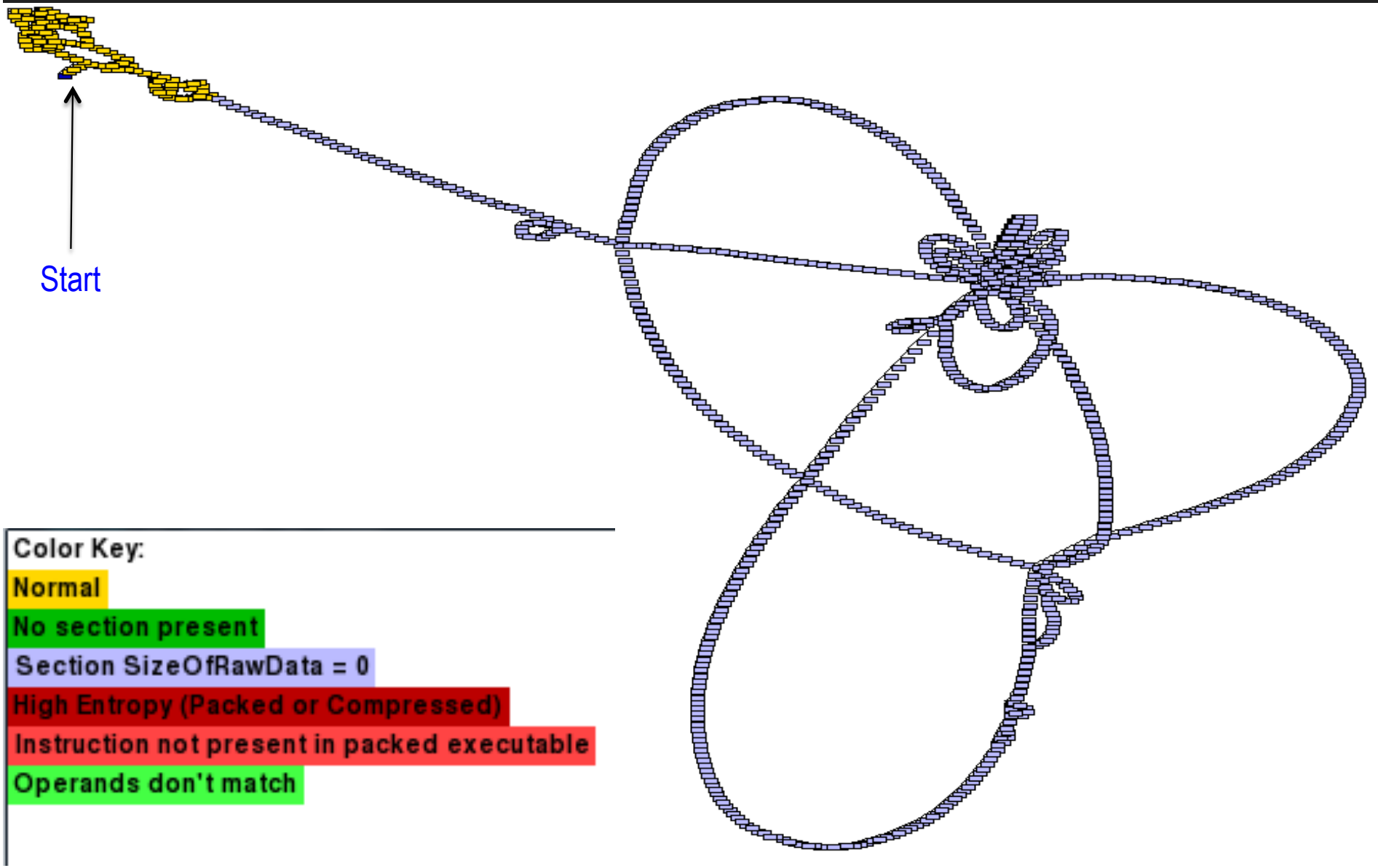
UPX - OEP


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

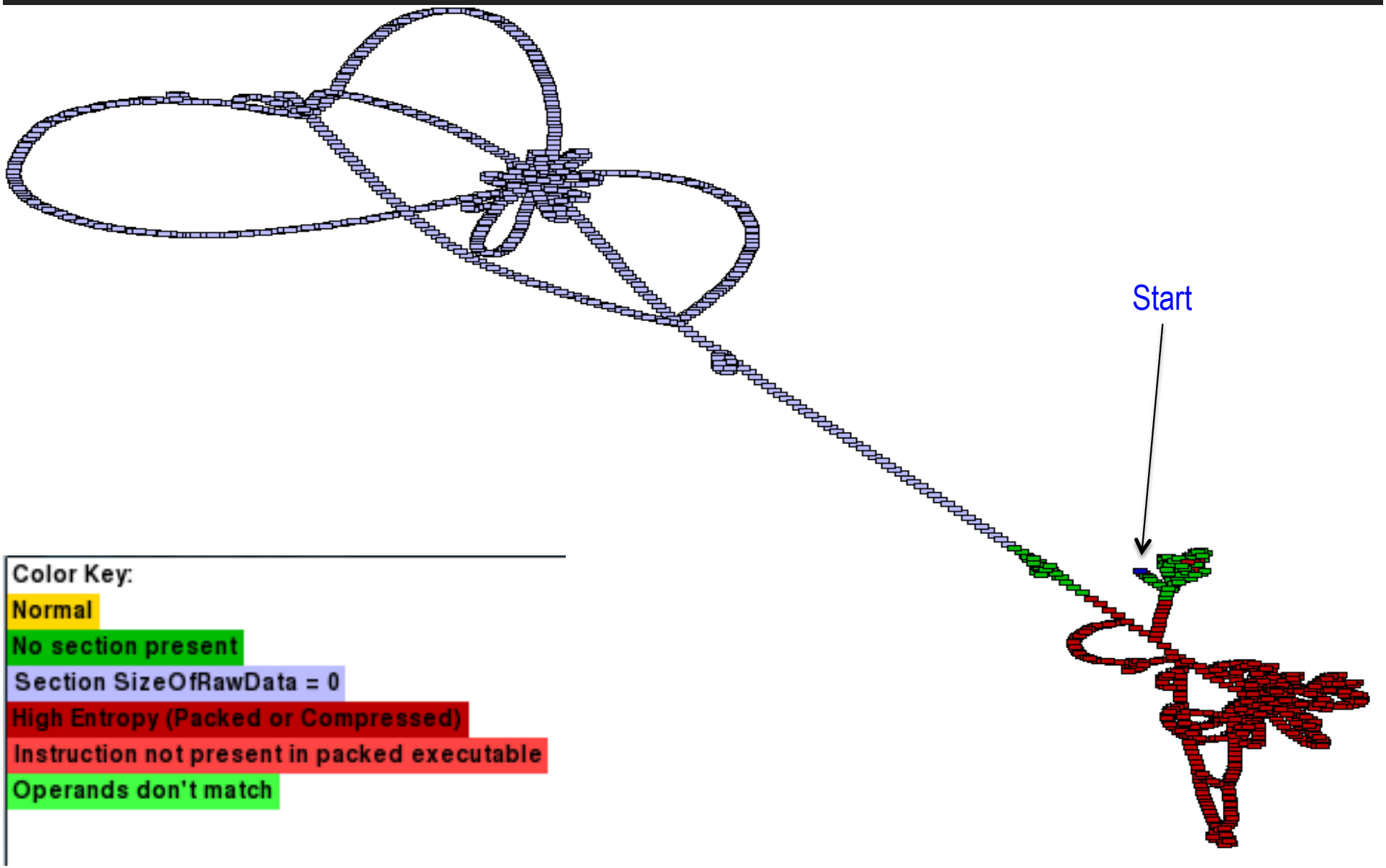
ASPACK


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

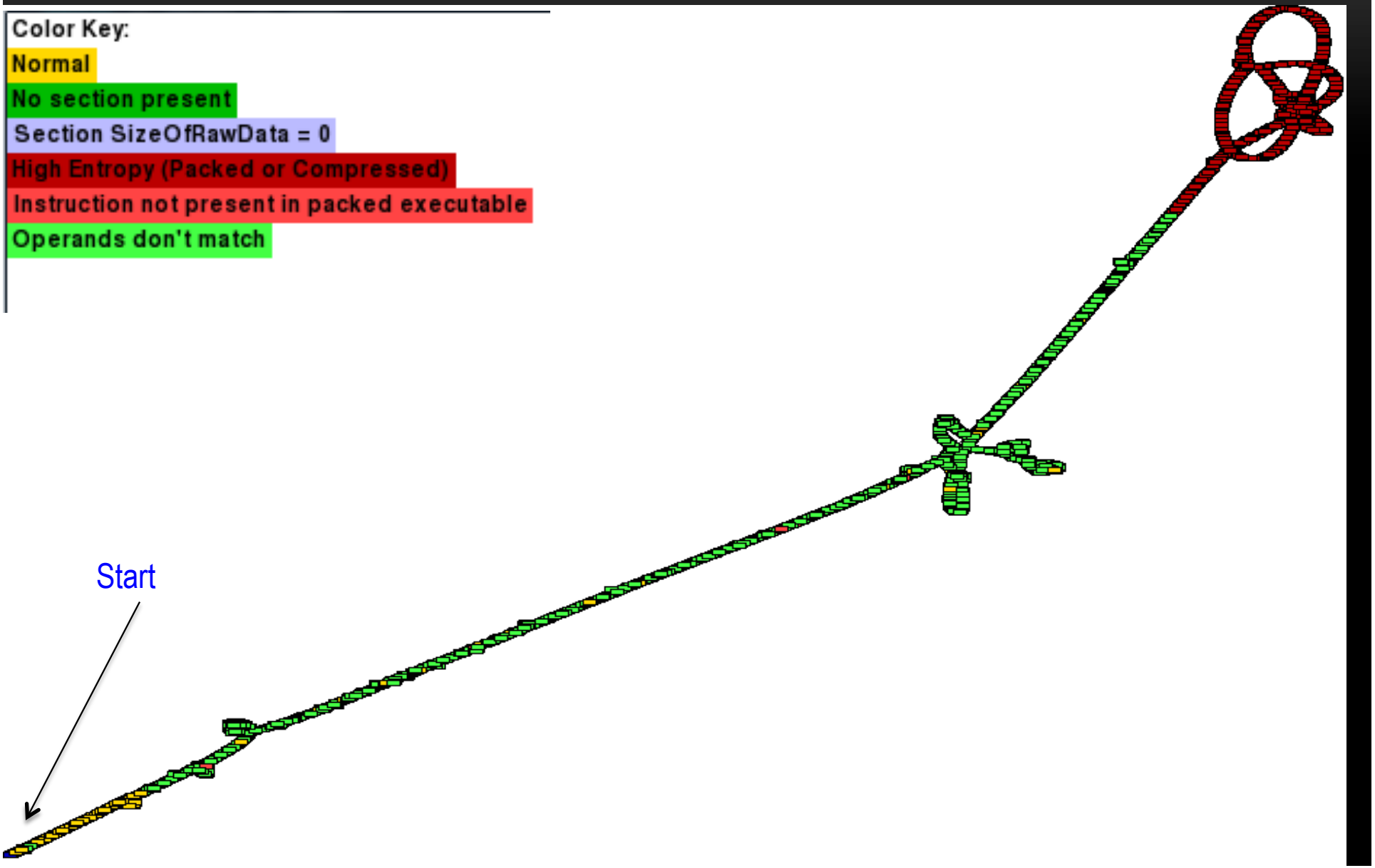
FSG


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

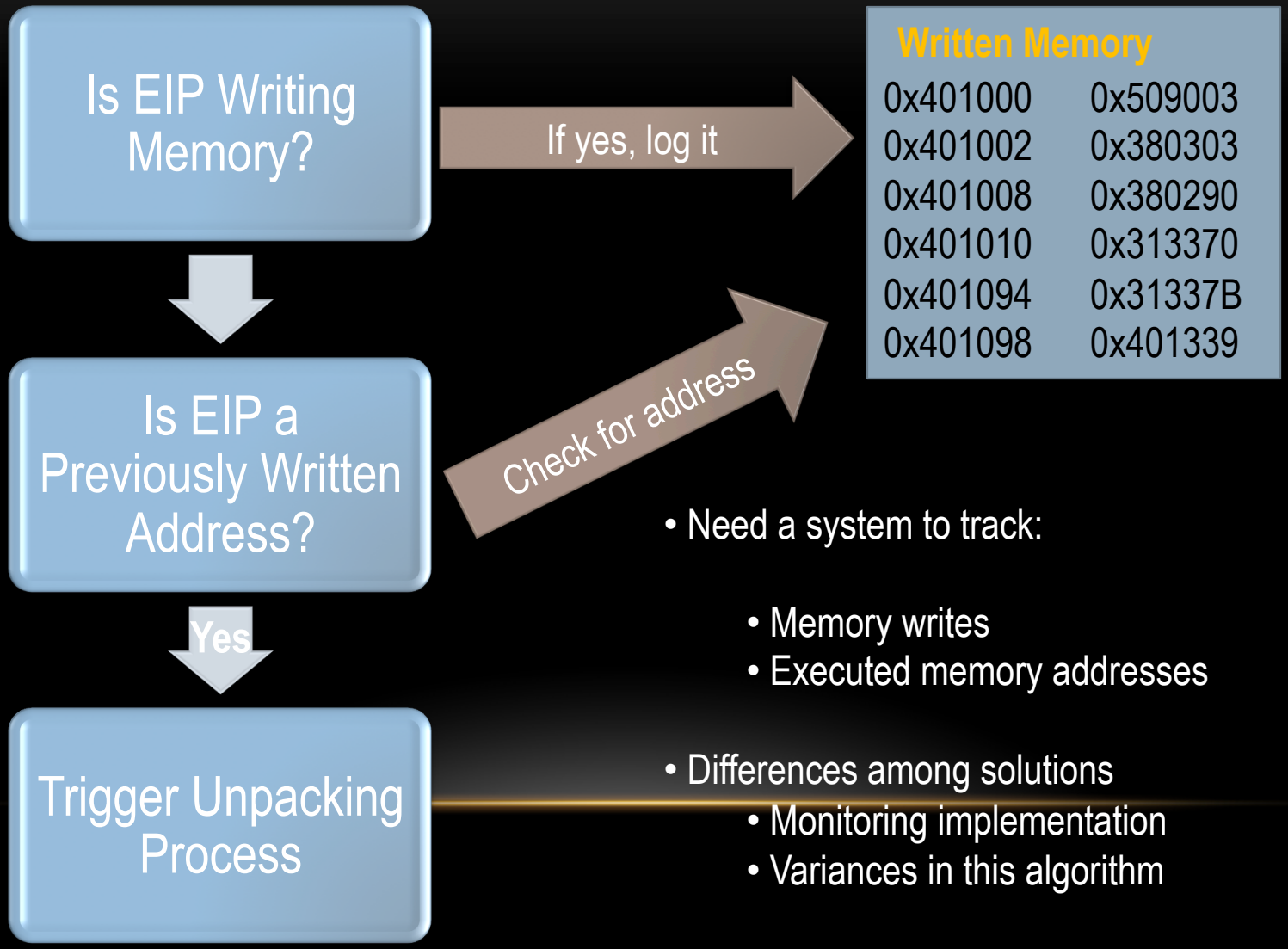
MEW


Start
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
Start

AUTOMATED UNPACKING



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

VISUAL UNPACKING DEMO

SUMMARY

- 3D Force Directed Visualizations
- Searching inside Visualization
- Temporal Animation
- Visual Reverse Engineering

RELEASE NOTES

- Timeframe for release in the next two weeks
  - Government code release bureaucracy
  - Version 0.50 will contain new features
- Videos on Youtube soon
- Download VERA, presentation, high quality videos at:
  - http://csr.lanl.gov/vera

THANKS AND ACKNOWLEDGEMENT

- Nathan Brown
  - Development of Ether automation, tracing tools
- Erin Ochoa
  - Testing and Packaging
  - GUI Consistency
- Josh Neil, Mike Fisk, Alex Kent
- ShmooCon Staff

CONTACT INFORMATION

Danny Quist

@ocomputing

csr.lanl.gov/vera

offensivecomputing.net