
# Visualization Techniques for Efficient Malware Detection

RiskSense Technical White Paper Series

Written by John Donahue, Anand Paturi, and Dr. Srinivas Mukkamala

## Executive Summary

Traditional tools for reverse engineering of binary and portable executable (PE) files are limited to heavy text-based output, thus requiring skilled analysts to use them. In this white paper, we illustrate techniques that will visualize portable executable files, which will help analysts with basic skills to quickly understand their underlying structure. We also present Markov Byte Plot-based techniques to identify packed malware and discuss how the presented techniques can aid the cyber security community for better enumeration of malicious software.

## About RiskSense

RiskSense®, Inc., is the pioneer and market leader in pro-active cyber risk management. The company enables enterprises and governments to reveal cyber risk, quickly orchestrate remediation, and monitor the results. This is done by unifying and contextualizing internal security intelligence, external threat data, and business criticality across a growing attack surface.

The company's Software-as-a-Service (SaaS) platform transforms cyber risk management into a more pro-active, collaborative, and real-time discipline. The RiskSense Platform™ embodies the expertise and intimate knowledge gained from real world experience in defending critical networks from the world's most dangerous cyber adversaries. As part of a team that collaborated with the U.S. Department of Defense and U.S. Intelligence Community, RiskSense founders developed Computational Analysis of Cyber Terrorism against the U.S. (CACTUS), Support Vectors Intrusion Detection, Behavior Risk Analysis of Vicious Executables (BRAVE), and the Strike Team Program.

By leveraging RiskSense cyber risk management solutions, organizations can significantly shorten time-to-remediation, increase operational efficiency, strengthen their security programs, heighten response readiness, reduce costs, and ultimately minimize cyber risks. For more information, please visit www.risksense.com or follow us on Twitter at @RiskSense.

# Table of Contents

## 1.0 Introduction

Reverse engineering plays a vital role in malware (malicious software) behavior enumeration and incident response. Reverse engineering is considered as an initial step, which allows security professionals to identify the undercover motive of the target malware and eventually identify the threat vector involved. During the reverse engineering process, a target malicious binary file or a malicious portable executable file is dissected to detect and analyze its underlying code structure, API interactions, determine authorship, among other tasks.

From past experiences, it is evident that a majority of malware is targeted towards banks and other financial institutions (for monetary gain) and hospitals (to extract confidential information) and some of the malware often evade anti-virus software in place. Cyber security professionals of respective organizations are often required to reverse engineer and analyze suspicious files, which could be potential malware. Additionally, security professionals belonging to anti-virus companies and leading incident response companies often face a challenge of reverse engineering substantial amounts of malicious software at a very quick pace. In order to accommodate above scenarios, several tools are used for reverse engineering, which unfortunately do not offer processed visual analytics to help security professionals for quickly drawing conclusions. One such example is Hex editor.

Hex editors are tools that display files in hexadecimal and ASCII formats. These editors are used to examine a target file in its raw format, as a hex editor does not assume any underlying structure for the file under analysis. However, they have been heavily text-based and visualization has been underused in them. Hence it requires a trained security professional to understand the representation of a binary or a portable executable file in a hex editor in order to enumerate the underlying section headers, pointers to raw data, virtual address, etc. while trying to find their behavioral motive. We believe that references to all sections and attributes of a portable executable file can be graphically represented in a hex editor, thereby eliminating the need of a special expertise to manually navigate through different sections of a portable executable file while trying to enumerate its behavior.

### 1.1 Packed Malware

Malware authors perform code packing in a portable executable file, thus thwarting an analyst's capability to detect the malicious nature of underlying code in a portable executable file (potential malware sample). The most common packing method used by malware authors is either compressing or encrypting code in a portable executable file, thus evading its detection through static analysis. Advanced skilled malware authors further employ techniques like shifting the decode frame (blocks of memory are decrypted as required and encrypted subsequently), thus preventing an automated detection mechanism to access hidden code.

Wide availability of automated packers like UPX and others to perform above mentioned packing methods has made it easy for malware authors to create malware that requires a high-level, complex skill set to discover and interpret packed portions of a malicious portable executable file.

### 1.2 Contribution

In this paper, we present a set of novel visualization techniques to help security analysts to perform a quick analysis of a given portable executable file, thus quickly identifying any malicious nature it might be holding. Our motive is to present visualizations that will help security personnel with less experience to extract knowledge of underlying details of a given portable executable or a binary file:

- We present a tree-based navigation mechanism as an alternative to the conventional text-based hex editor, using an approach which allows end users to quickly navigate through the underlying sections of a portable executable file. This will eliminate manual navigation which would otherwise be required to identify different sections of a portable executable file and can be useful to personnel with beginner's knowledge to quickly enumerate sections and pointers in a portable executable file and carry on the analysis process from there.
- We visually represent packed portions of a portable executable file and its encodings, which will provide a quick guide to security analysts regarding the existence of packed code, packing techniques in underlying malware, and where different encodings are present.

Please note that below techniques can be applied both for a portable executable and a binary file alike, though we mention only portable executable file specific scenarios.

The rest of this paper is structured as follows: In Section 2, we describe our methodology using Markov Byte Plots to represent packed malware. In Section 3, we give a brief overview of related work and in Section 4 we provide our conclusion and future work.

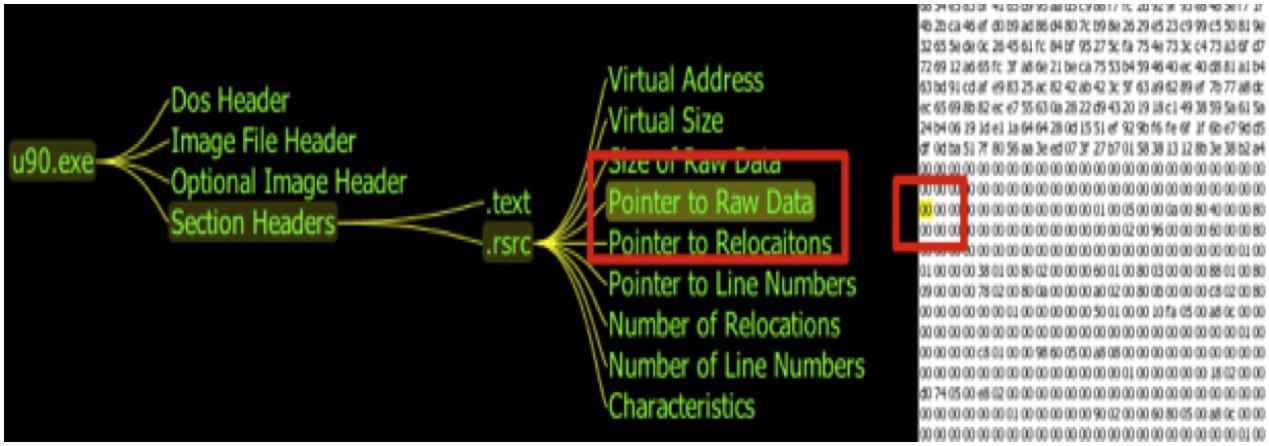


**FIGURE 1:** Interactive Hex Editor

## 2.0 Visualization Techniques

In this section we present recommended visualization techniques for an interactive hex editor and visualizing packed malware. We also discuss how these visualization techniques will help the security analyst community.

### 2.1 Filtered Hex Viewer

In this section we present our methodology to create an interactive Hex editor, which produces an interactive portable executable tree visualization on the left pane and the right pane has a classical hex editor that allows for colored filters.

To construct the portable executable tree, we first parse portable executable headers and data directories. Offsets for these are stored in their little endian format, but translated to big endian format when displayed. We used the Prefuse visualization toolkit [1] for the tree representation. Each leaf in the tree represents a piece of meta data or an offset in the portable executable header or data directory. All other nodes represent sections in the portable executable header or offsets to other portions of the portable executable header.

Security analysts - especially reverse engineers - can use this tree representation for better navigation of a portable executable file. While navigating, they will select a leaf, which will refer them to the appropriate offset in the hex viewer (on right the pane). While the hex viewer is displaying the file as it is mapped on disk, offsets in the portable executable file (with a few exceptions) are virtual addresses. The portable executable tree automatically calculates the appropriate virtual offset, as if it were already mapped into memory. This can produce some interesting artifacts when sections are allocated space on load or other similar scenarios. However, this is preferred for viewing a memory-mapped file in a hex editor, as lots of bytes tend to be zero.

Figure 1 represents the filtered hex editor, whereby the portable executable tree is on the left pane and the regular hex editor on the right pane. The highlighted areas in the red represent the leaf node selected (the resource section header) and its corresponding pointer in the hex file. By following this navigation, header sections of a portable executable file can be explored including the section header information. Additionally, this approach enhances the navigation by scrolling the hex editor to the appropriate location when a header entry references that specific region in the file. In practical scenarios, whenever security analysts need to examine a portable executable file, they can adapt this editor for quickly navigating to different sections of a portable executable header rather than depending on the heavy text-based nature of current hex editors.

### 2.2 Packed Malware Visualization

As mentioned above, packed malware poses a great challenge to security analysts by encrypting critical code sections in a portable executable file. In this scenario, a tool which visualizes packed portions of a portable executable file and represents the existence of different encodings in the portable executable file, will aid security analysts to quickly draw conclusions about properties of a packed portable executable file (which could be potential malware) like packed code, packers used, and different encodings present.

To visualize packed malware, we use a Markov Byte Plot (MBP) (based on a Markov random field). To create the MBP, we start with a complete, weighted, digraph $G=<V, E>$ on 256 nodes. Each vertex $V$ represents a unique byte value and each directional edge, $E_{i,j}$ represents a transition from byte $i$ to byte $j$. Each edge weight, $W(E_{i,j})$ is initialized to zero.

We start off with reading the entire binary / portable executable file and for each transition from $bt$ to $bt+1$, the corresponding edge $Ei,j$ is incremented. To calculate the transition probability $P(bt|bt-1)$ from the raw counts, we divide each edge weight by the sum of all out bound edges from the source node. This is represented in equation 1, where $i = bt$.

$$P(b_t|b_{t-1}) = \frac{w(e_{(i-1),i})}{\sum_{j=0}^{255} w(e_{(i-1),i})}$$

Each of the calculated transition probabilities is used to populate the 256x256 transition matrix $P$ *where* $Pi,j = P(bj|bi)$. To create a visualization of the transition matrix, each entry $Pi,j$ is assigned to a pixel. Each pixel is assigned a color by first normalizing the transition probabilities and then mapping them to an RGB spectrum.

Figure 2 represents an example of this visualization, whereby the left side pictorial represents the packed malware sample of *Beagle.exe* (packed using *UPX* packer), and the pictorial on the right side represents the un-packed version of the same sample.

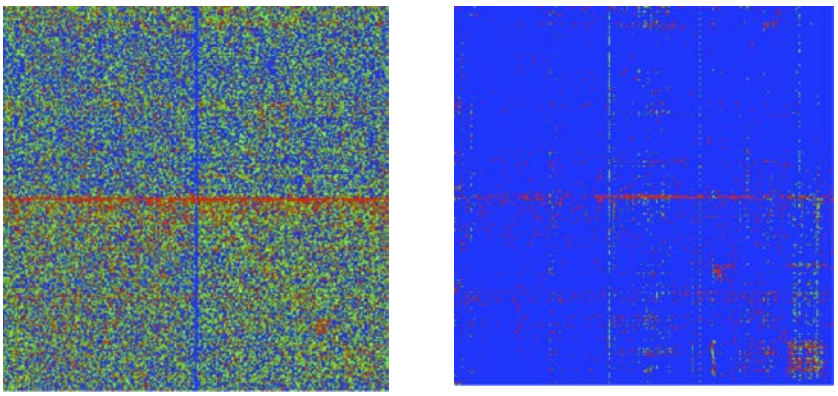

**FIGURE 2:** Packed and unpacked versions of Beagle malware

It is clear from our visualization in Figure 2 how a packed portable executable file can be detected by representing it using MBPs. Since the packed version of a portable executable file results in an entropy-based RGB representation (having a red dotted line in the visual in our case), we can quickly detect the packed portion of the portable executable file. Additionally, our representation of the unpacked version also helps to identify the encodings present in the target portable executable file. For example, in Figure 2 the unpacked representation clearly highlights the portion of text

encodings in the source portable executable file (in red dots). This will help security analysts to identify various encodings present in each portable executable file.

## 3.0 Related Work

Conti et. al. in [2] introduce several interesting visualization tools into a hex editor-like environment. The three visualizations work in concert to improve the workflow of an analyst. The 'Byteview' visualization provides an at-a-glance view of an entire file, whereby each byte is represented as a pixel. The intensity of each pixel is dependent of the hex value of the byte the pixel represents. The 'Byte Presence' display works side-by-side with the 'Byteview' display. Each row of pixels in the 'Byte Presence' display summarizes the existence of bytes in the 'Byteview' display. The 'Dot Plot' visualization is a concept borrowed from biology. In biology a dot plot is used to align genome sequences. In this instance, the dot plot is used to compare two files. This visualization shows the presence of similar byte sequences between to files. However, this visualization can only be used on a subset of each file because of memory and display issues.

In our day-to-day security analytics, we are using the 'Byte Plot' representation to represent and detect meaningful sections of a given portable executable or binary file.

## 4.0 Future Work and Conclusion

We envision a visualization framework, which will aid security analysts with less experience in reverse engineering to analyze any portable executable file and detect the underlying malicious nature (if any). The work described above is going to be part of such framework. We anticipate to further enhance the 'Byte Plot' representation to detect which packing mechanism has been used to produce the packed portions of a portable executable file. This will help security analysts to save the time they spend on finding the packer used by adversaries to pack underlying code.

In this paper, we shared two visualization techniques implemented on portable executable file. One technique produces a navigational portable executable file structure, thereby enhancing the capability of a hex editor. The second technique is using the Markov Byte Plot to detect packed portions of portable executable file while trying to reverse engineer it.

## 5.0 References

[1] prefuse. http://www.prefuse.org.

[2] Conti, Gregory and Dean, Erik and Sinda, Matthew and Sangster, Benjamin. Visual reverse engineering of binary and data files. In *Proceedings of the 5th international workshop on Visualization for Computer Security*, VizSec '08, pages 1–17, 2008.


**RiskSense New Mexico**
4200 Osuna Road NE, Suite 3-300
Albuquerque, NM 87109
United States

**RiskSense Silicon Valley**
530 Lakeside Drive, Suite 170
Sunnyvale, CA 94085
United States

**General Inquiries**
+1 505.217.9422
+1 844.234.RISK (toll free)
info@risksense.com

**Media Inquiries**
media.relations@risksense.com