Pimp My PE: Parsing Malicious and Malformed Executables

Virus Bulletin 2007

Sunbelt Software

Authors

- Sunbelt Software, Tampa FL
- Anti-Malware SDK team:
  - Casey Sheehan, lead developer
  - Nick Hnatiw, developer / researcher
  - Tom Robinson, developer / researcher
  - Nick Suan, developer / researcher

Purpose

- Chronicles the early development of our detection engine
  - Specifically, the PE parser
  - Building enterprise infrastructure to support development
- Technical issues:
  - Understand malformations prevalent in wild PE's
  - Methods for identifying malicious PE's
  - Reliably parsing PE's
    - Pietrek's article "An In-Depth Look into the Win32 Portable Executable File Format" [3] a great intro, but much more is needed to successfully process modern PE's
    - Virtually all commercial analysis tools have serious issues parsing malicious PEs

Overview

- Introduction
- Technical Background
- Infrastructure
- Image parsing in depth

Part 1: Introduction

The Need

- Ability to parse any PE into a robust internal representation
- Ability to detect and remediate threats

The Problem

- Initial assumption: parsing is easy
  - Simple parser should be able to cope with all samples.
- Reality: malicious samples break parser constantly
- Reaction: are these all corrupted PEs?
- Realization: Windows loader behavior a valuable comparison metric.
  - If Windows loads an image, we had better parse it
  - Corrupted images are, at very least, suspicious.
- In summary:
  - Implementations in the literature perform poorly versus threats in the wild; generally cope poorly with "malformed" images
  - A large percentage of images in the wild are malformed (68%)

The Problem (con't)

- The *actual* problem: building a parser to effectively process modern, malicious PEs
- Key hurdles:
  - Qualify behavior of Windows loader for comparison purposes
  - Analyze and categorize "anomalous" characteristics of sample images which identify malformed images
  - Iteratively improve parser performance (i.e., avoid performance regression)

The Solution

- Iteratively build and test parser
- Constant regression testing
  - Ensure new features don't cause overall performance to regress
- Verify performance vs. Windows loader
  - Gauge parser performance in absolute terms

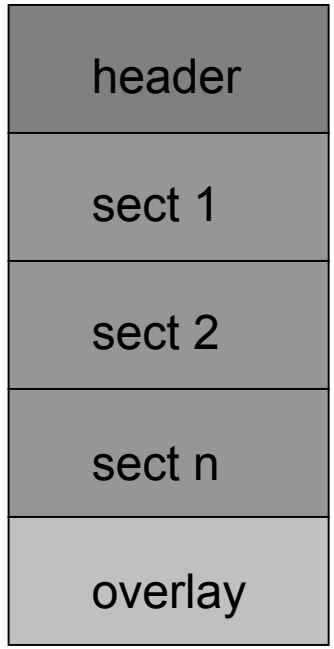
Image Anomalies

- Anomaly:
  - specific structural malformation; a particular field malformed a particular way
  - frequently inconsistent with PE specification, or just unusual or suspicious
- Analysis of anomalies and other structural characteristics provides key insight into common image malformations

Part 2: Background

Basic PE Structure

- PE Header
- PE Sections
- Overlay (optional)



Alignment

- Alignment applies to section mapping
- PE header specifies two sectional alignment values
  - *File alignment* specifies file mapped alignment
  - *Virtual alignment* specifies virtual mapped alignment

Image Mapping

- Windows loader performs "map and load" operation:
  - Map:
    - Size the view
    - Create view in process VA space
    - Allocate storage
  - Load image section by section
- Our parser mimics this behavior
  - "Source representation"
    - Frequently file mapped (linker output)
    - However we may be given memory mapped image with no corresponding file image
  - "Target representation"
    - Typically virtual mapped

Mapping Translation

- Need to handle both file- and virtual-mapped images
- cImageStream class
  - Accepts any source representation
  - Translates to requested target representation
  - Manages all stream-related details

Section Size

- Fundamental concept when dealing with sections due to variable section alignment
  - Applies to header and sections
- 3 unique size concepts:
  - Raw size: unpadded data size
  - File size: RoundUp(raw_size, file_align)
    - "File cave"; persistent
  - Virtual size: RoundUp(file_size, virtual_align)
    - "Virtual cave"; transient
  - Be precise!
    - Always explicitly state the size type in source code

Section Size (con't)

- Interesting (and annoying) that raw section size is unavailable
  - Important if you want size of *REAL* content!
  - E.g., when parsing structures in the header
  - … Or instructions (atoms) in a code section
- In practice, file aligned size is often treated as synonymous with raw size
- Demo:
  - Dump basic white file; identify raw, file, virtual sizes

PE Structure

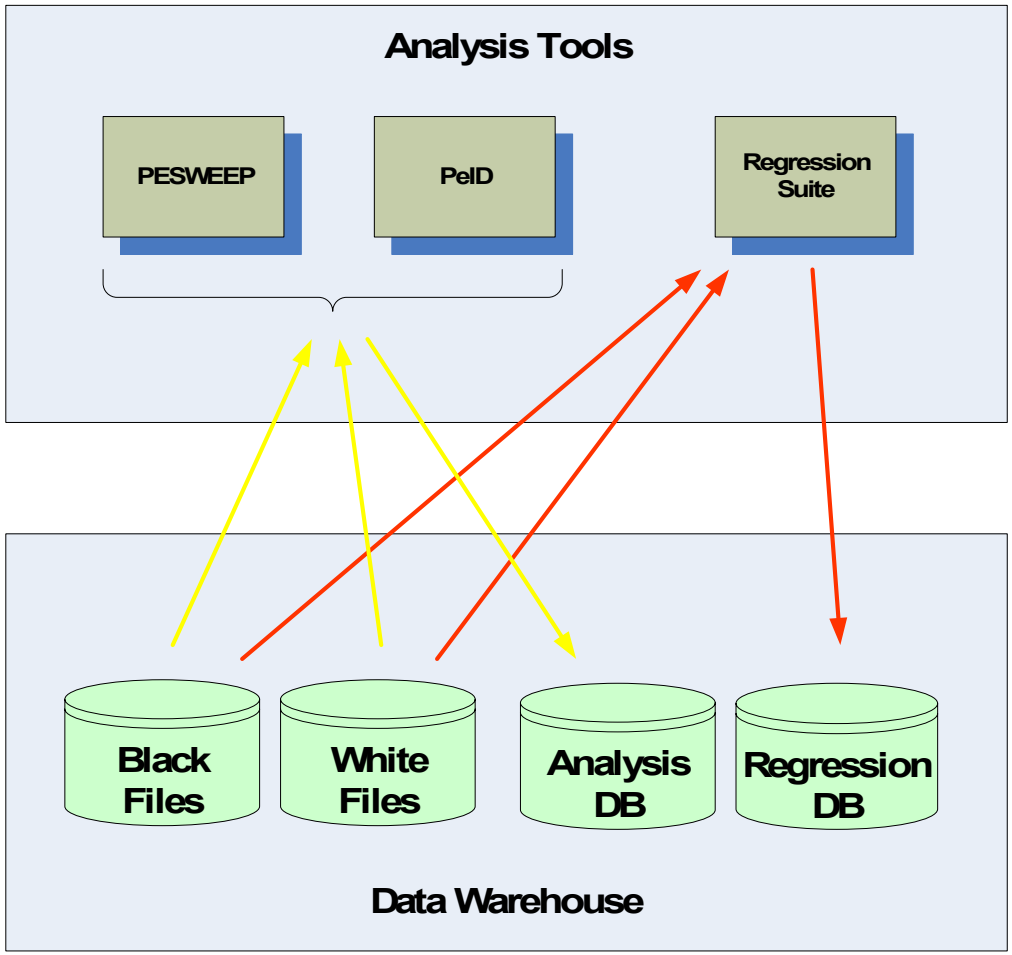
- PE header:
  - Documents “explicit” image structure
  - Vs. “implicit” structure
- PE section
  - Primary image content
  - Code, data, etc.
  - Described in header’s section table
- Overlay: non-loadable data, appended to PE image
  - Certificates
  - Debug info
  - Malware-specific payload
  - Demo Ganda

PE Structural Abstractions

- Metasection:
  - abstraction for header, section, overlay components
- Metadata:
  - predefined data types
  - enumerated in the Data Directory (“DD”)
  - scattered throughout the image (and overlay)

Part 3:
Enterprise Infrastructure:
Data Management & Analysis

Infrastructure Overview



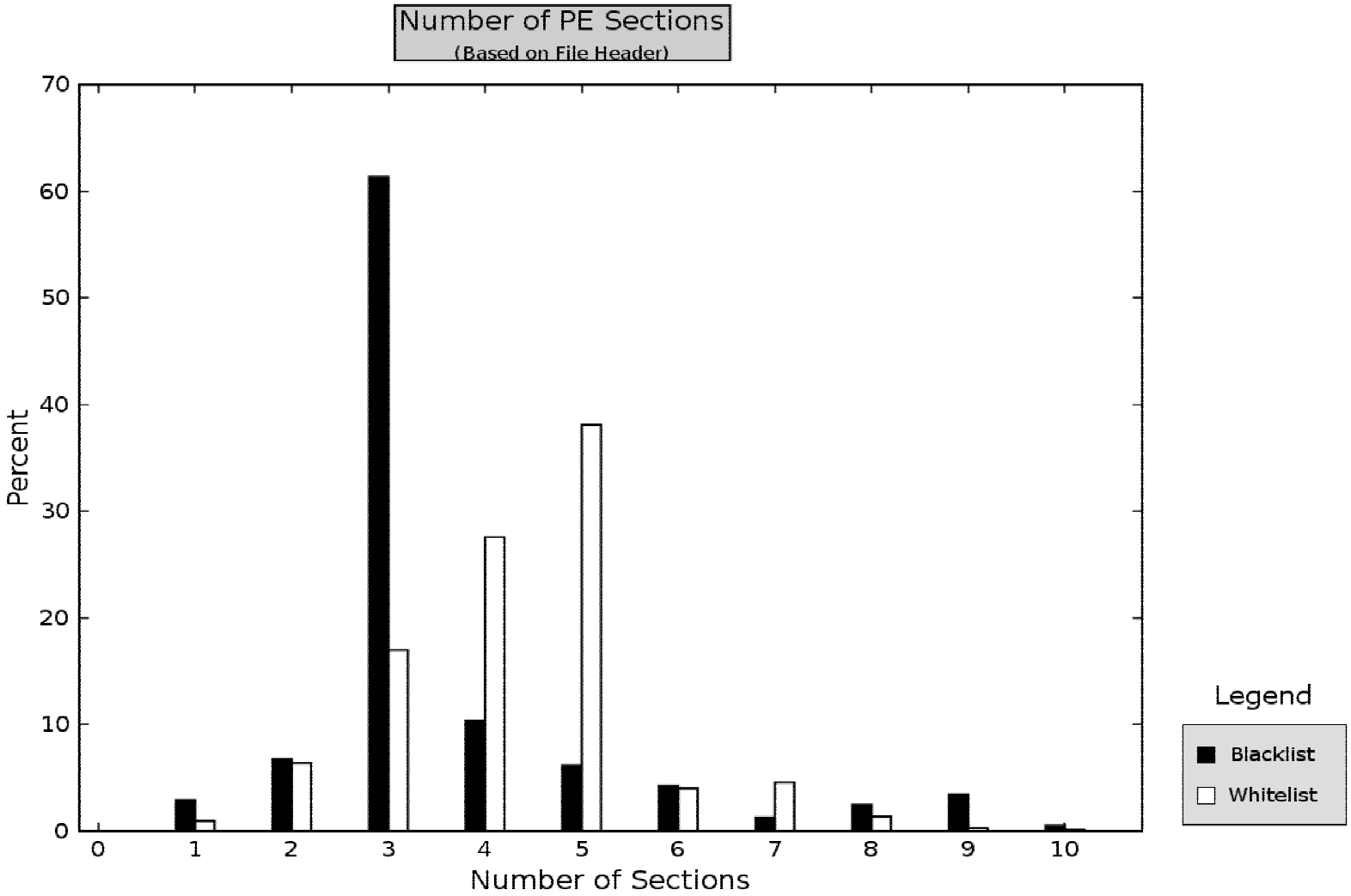
Data Repositories

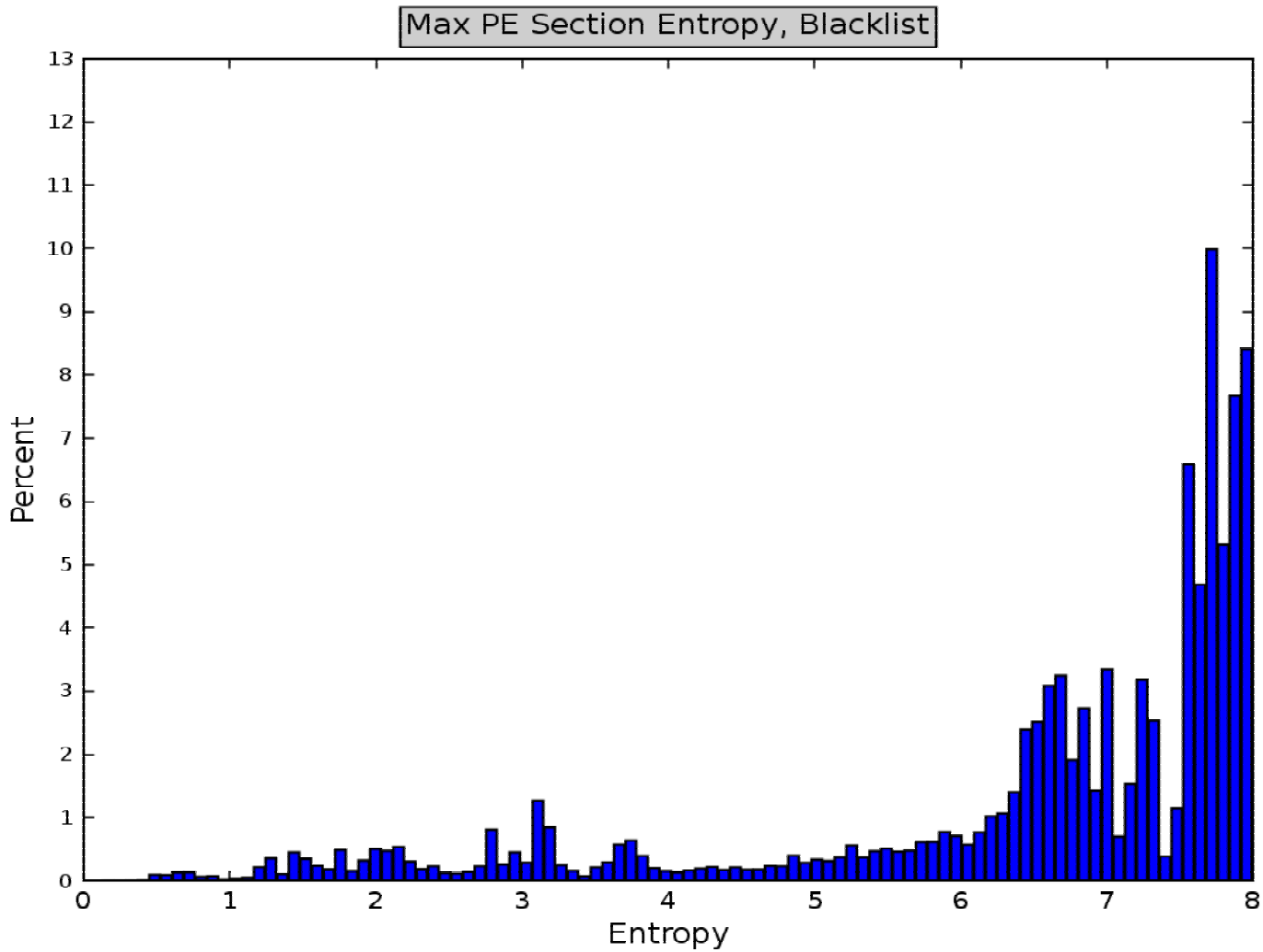
- PE repository consists of
  - ~9,000 known good PEs ("white collection")
  - ~70,000 known malicious PEs ("black collection")
- Images processed through two tools
  - PEiD packer identifier [1]
  - Proprietary static analyzer PeSweep
- Post-process tool output, import into DB
- Mine DB for interesting correlations
  - Data mining is speculative, iterative, time-consuming
  - Results shown here are tip of iceberg

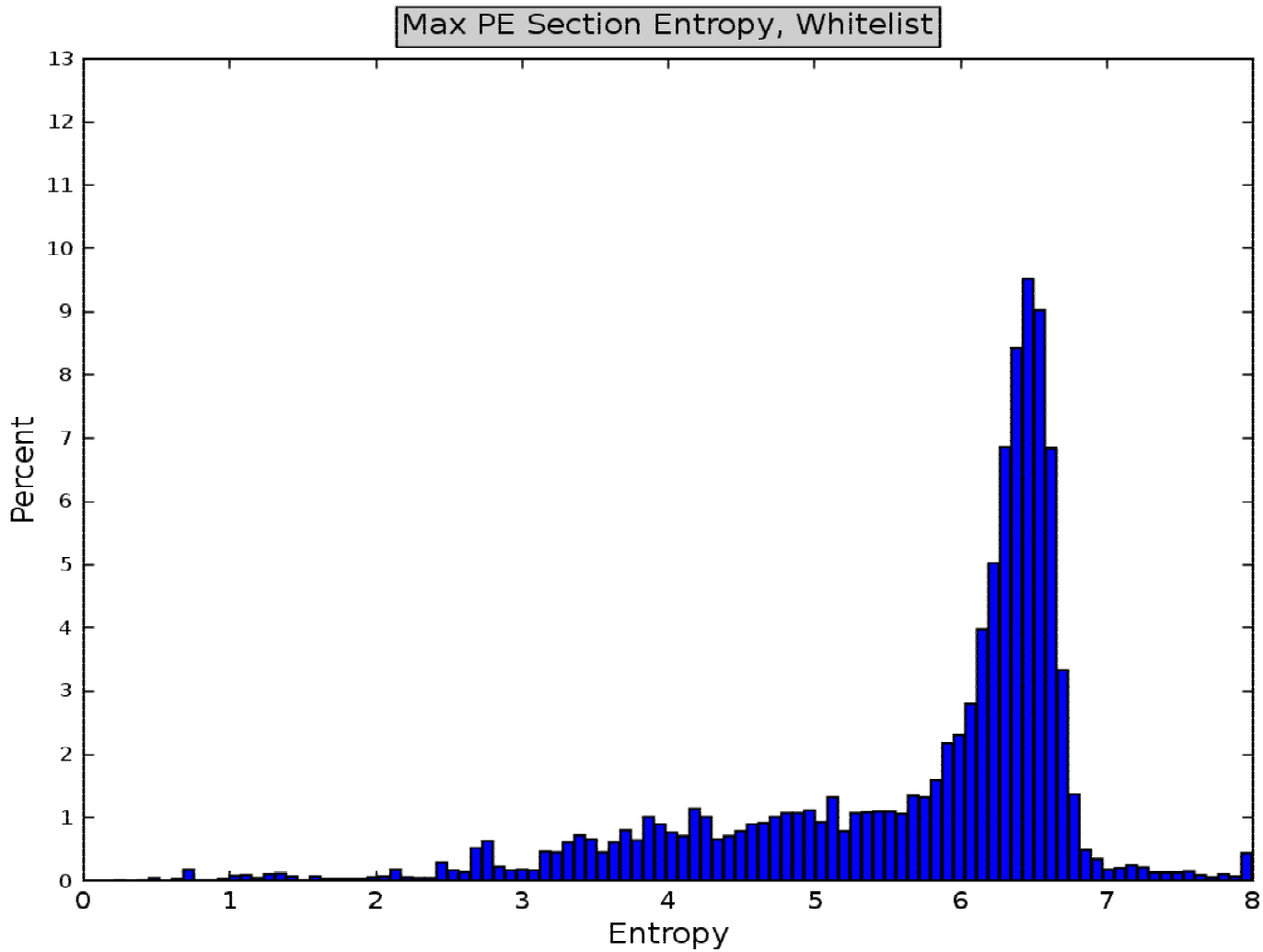
PeSweep Analysis

- Analyzes single file, directory, optional recursion
- For every file processed, generates info on:
  - Infer whether Windows is able to load it
  - Details on how much of the structure the parser is able to parse
  - Entropy values on a sectional basis
  - Header structure
  - Anomaly bits
- Able to create both file and virtually mapped target mappings of the image
- Fully parses "explicit" content (header+metadata) : import, export, relocation, resource, etc values

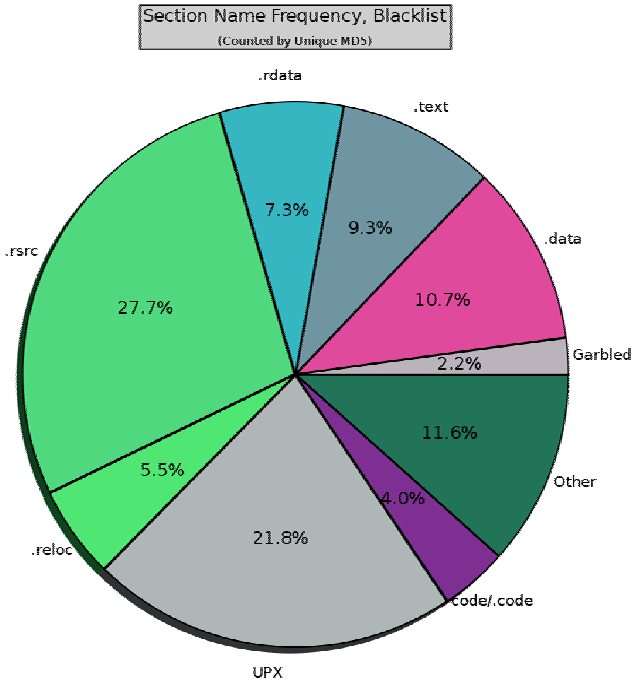
Sample Analysis Results

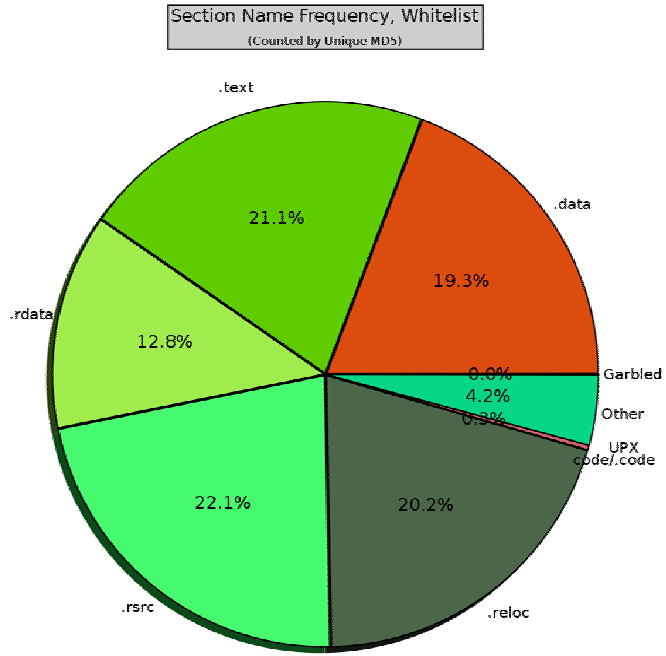






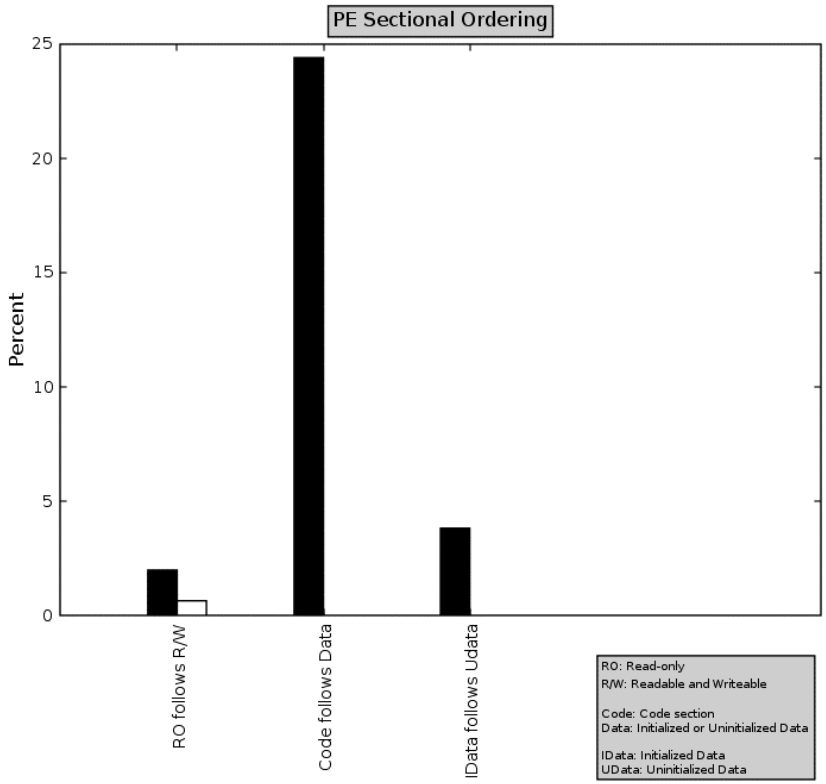
Section Name Frequency



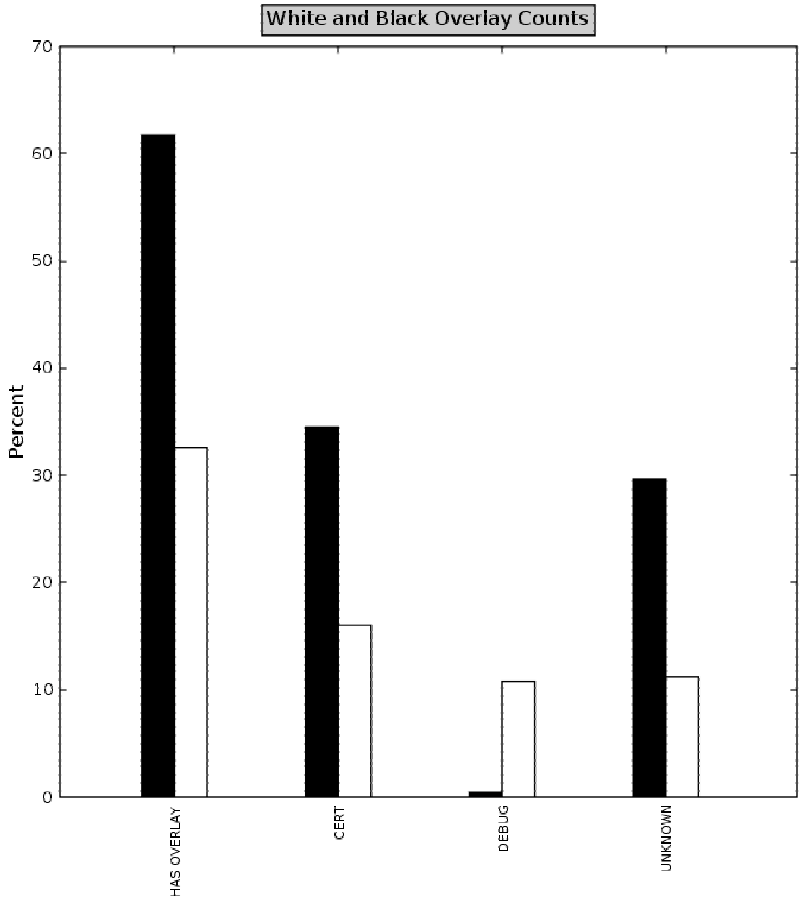


Sunbelt Software

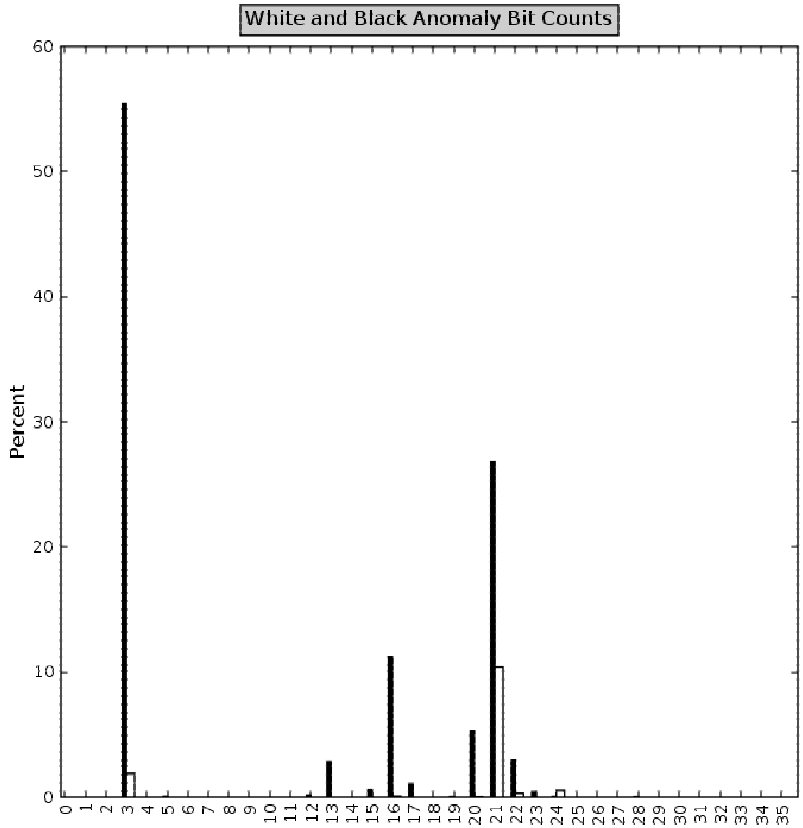
Sectional Analysis



Overlay Prevalence



Anomaly Frequency



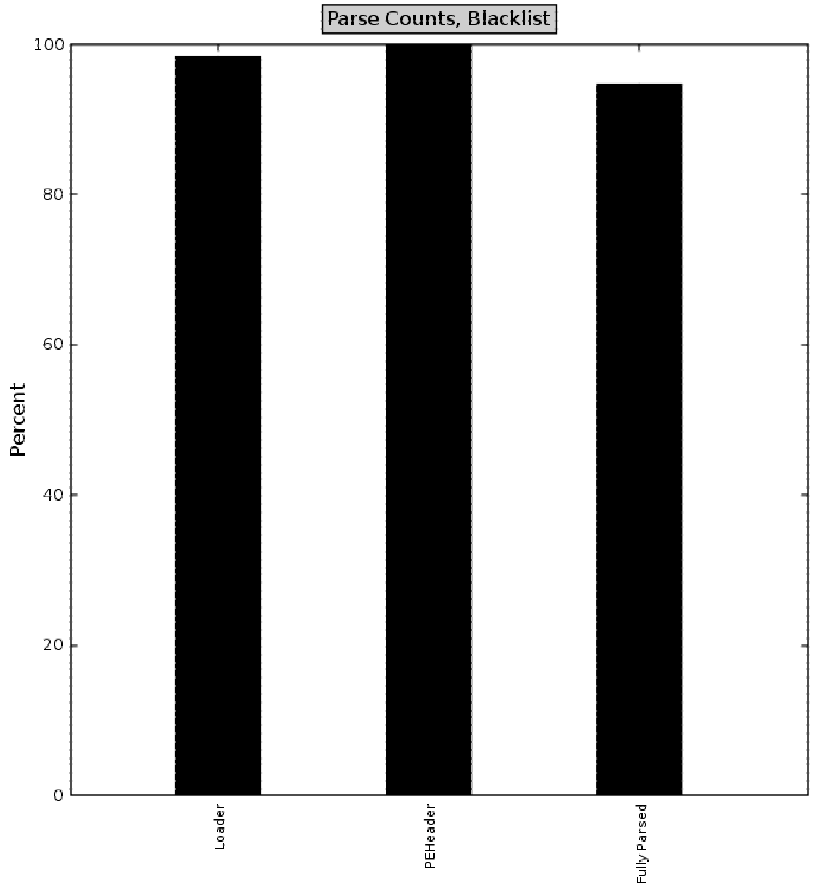
Analysis Summary

- We're profiling characteristics of known-bad and known-good images
- Distilling these results into general rules for filtering files at runtime
- These rules could help identify suspicious files
  - E.g., the more suspicious a file, the more analysis resources it receives

Analysis Use Case 1

- Goal: Identify Loadable PEs
  - Classify PEs as valid / invalid at runtime
- Approach: synthesized "loader test"
  - Indicates whether Windows will run the file
  - Comprised of CreateProcess/LoadLibraryEx
  - Run across NT, 2000, XP, Vista

Loader Test Results

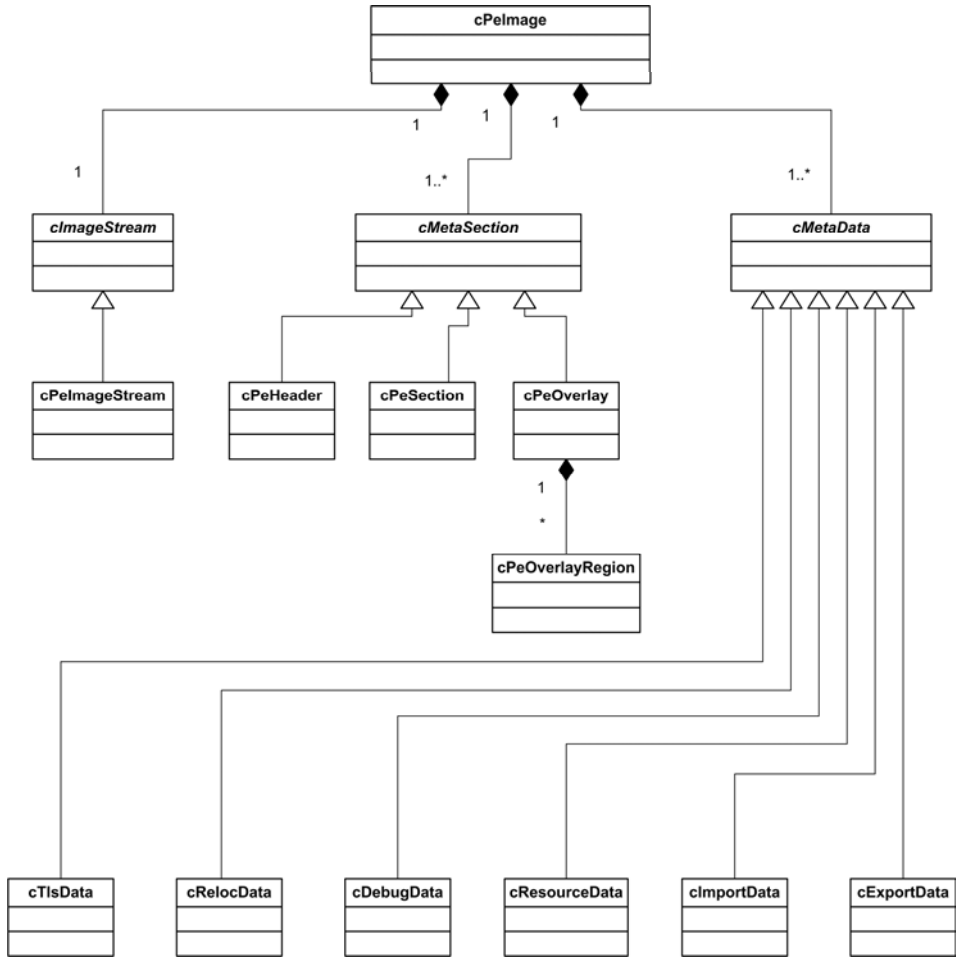


Analysis Data Use Case 2

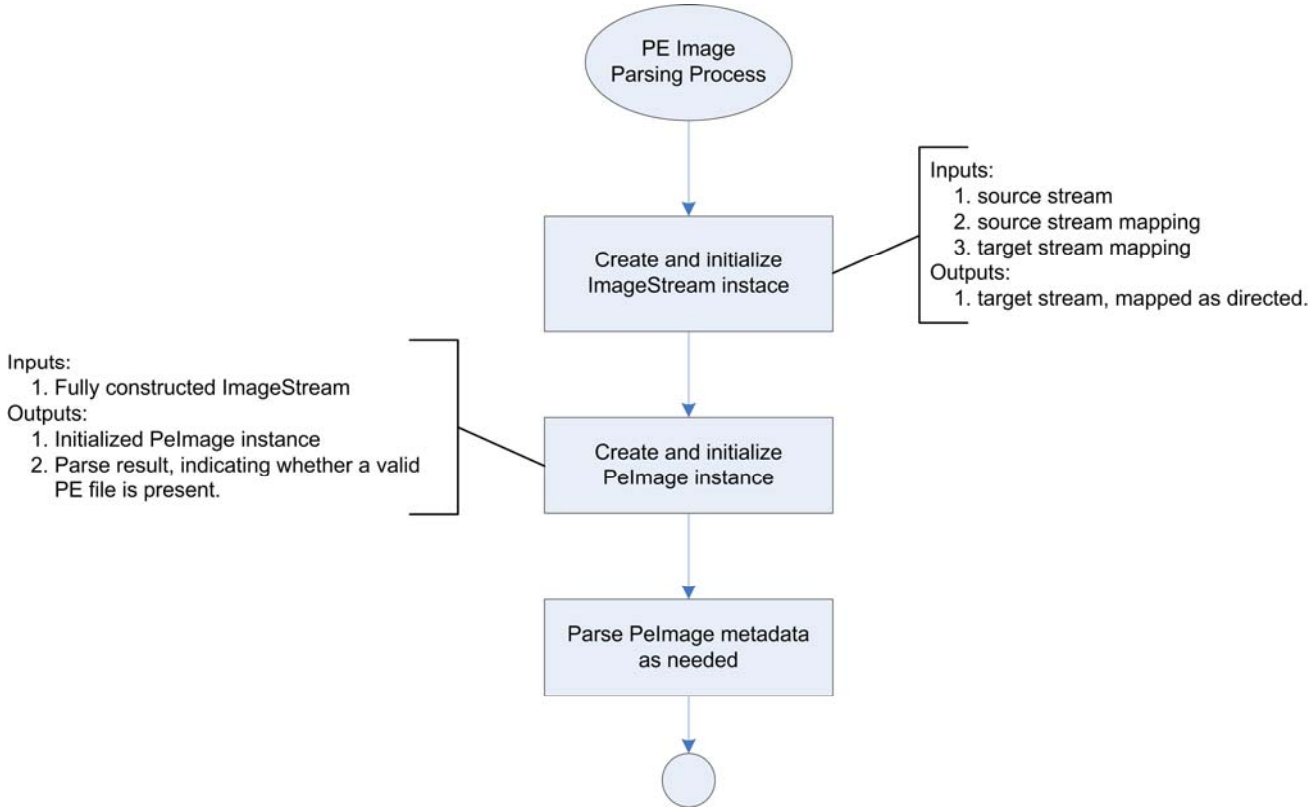
- Goal: Identify Malicious PEs
  - Obviously a runtime heuristic generating a reliable "Is Suspicious" flag is valuable
- Single query of anomaly bits
  - Identifies 67% of black list
  - Identifies 1.4% of white list
- This could be improved dramatically by increasing the sophistication of our query.

Part 4:
Image Parsing in Depth

PE Parser Class Organization



PE Parsing Flowchart



ImageStream Initialization

- Same MapAndLoad process as before
- Calculate target stream size
  - Sum source stream metasection sizes, according to target stream mapping
- Construct target stream
  - Copy each source metasection at computed offset in target stream
  - Delicate process due to possible structural anomalies
- Parse anomalies are tracked throughout entire parsing process

Stream Normalization

- Problem: MapAndLoad process is fragile
  - Image structure can be corrupted in a myriad of different ways
  - Non-validated fields can lead to crashes during mapping and loading
- Solution: preliminary scan of header
  - "Normalization" pass through the header to fix obviously illegal values
  - Guarantee subsequent parse pass succeeds
- Initial results were promising!

Stream Normalization (con't)

- Sample "illegal" values:
  - Section table entry RVA falls within the header
  - Section table entry wild RVA and sizes entry
  - Header structures overlap
  - Wild DD entries
- TinyPE breaks them all! [2]
  - File ends before nominal end of OptHdr!
- Demo
- Summary:
  - Normalization must allow many degenerate cases
  - Less is more
    - none is best ☺

In Summary

- Anomaly Mechanism
  - Useful source of info for analysis engine
- Parser Design
  - Hope there are some useful nuggets here..
- Infrastructure
  - Supports ongoing technology improvement and QA
  - Insight into malformations prevalent in the wild
  - Proven useful for technology refinement

Future Work

- Extend
  - Infrastructure
  - Analysis
- Refine heuristics for identifying malware and "suspicious" images
- Build additional tools
  - GUI version of PeSweep
- For now, SDK resources available at http://research.sunbelt-software.com/ViperSDK/
  - PeSweep (cmdline binary; no source ☺)
  - Presentation

Thanks!

caseys@sunbelt-software.com

References:

[1] PEiD homepage (http://peid.has.it/)

[2] TinyPE (http://www.phreedom.org/solar/code/tinype/)

[3] Matt Pietrek, Under The Hood, An In-Depth Look into the Win32 Portable Executable File Format, MSDN Magazine, April 2002, http://msdn.microsoft.com/msdnmag/issues/02/02/PE