Machine Learning for Malware Analysis

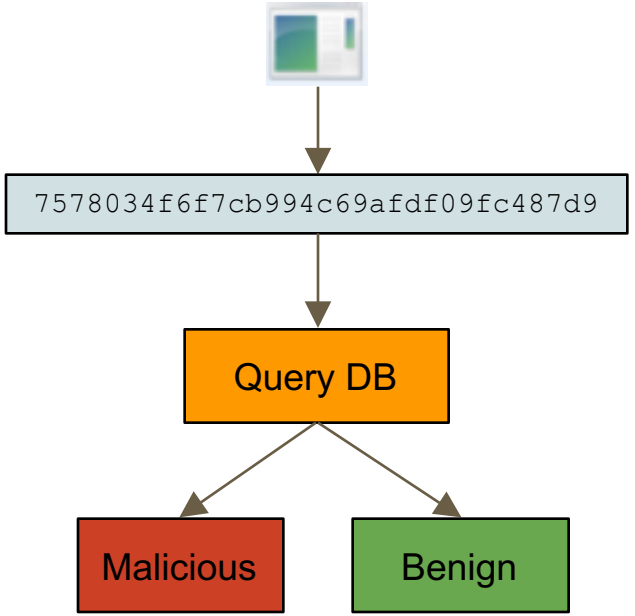
Mike Slawinski
Data Scientist

Introduction - What is Malware?

- Software intended to cause harm or inflict damage on computer systems
- Many different kinds:
  - Viruses
  - Trojans
  - Worms
  - Adware/Spyware
  - Ransomware
  - Rootkits
  - Backdoors
  - Botnets
  - ...

Malware Detection - Hashing

- Simplest method:
  - Compute a fingerprint of the sample (MD5, SHA1, SHA256, …)
- Check for existance of hash in a database of known malicious hashes
- If the hash exists, the file is malicious
- Fast and simple
- Requires work to keep up the database



Malware Detection - Signatures

Look for specific strings, byte sequences, … in the file.

If attributes match, the file is likely the piece of malware in question

Signature Example

```
rule Stuxnet_Malware_3
{

    meta:
        description = "Stuxnet Sample - file ~WTR4141.tmp"
        author = "Florian Roth"
        reference = "Internal Research"
        date = "2016-07-09"
        hash1 = "6bcf88251c876ef00b2f32cf97456a3e306c2a263d487b0a50216c6e3cc07c6a"
        hash2 = "70f8789b03e38d07584f57581363afa848dd5c3a197f2483c6dfa4f3e7f78b9b"

    strings:
        $x1 = "SHELL32.DLL.ASLR." fullword wide
        $s1 = "~WTR4141.tmp" fullword wide
        $s2 = "~WTR4132.tmp" fullword wide
        $s3 = "totalcmd.exe" fullword wide
        $s4 = "wincmd.exe" fullword wide
        $s5 = "http://www.realtek.com0" fullword ascii
        $s6 = "{%08x-%08x-%08x-%08x}" fullword wide

    condition:
        ( uint16(0) == 0x5a4d and filesize < 150KB and ( $x1 or 3 of ($s*) ) ) or ( 5 of them )
}
```

Problems with Signatures

- Can be thought of as an overfit classifier
- No generalization capability to novel threats
- Requires reverse engineers to write new signatures
- Signature may be trivially bypassed by the malware author

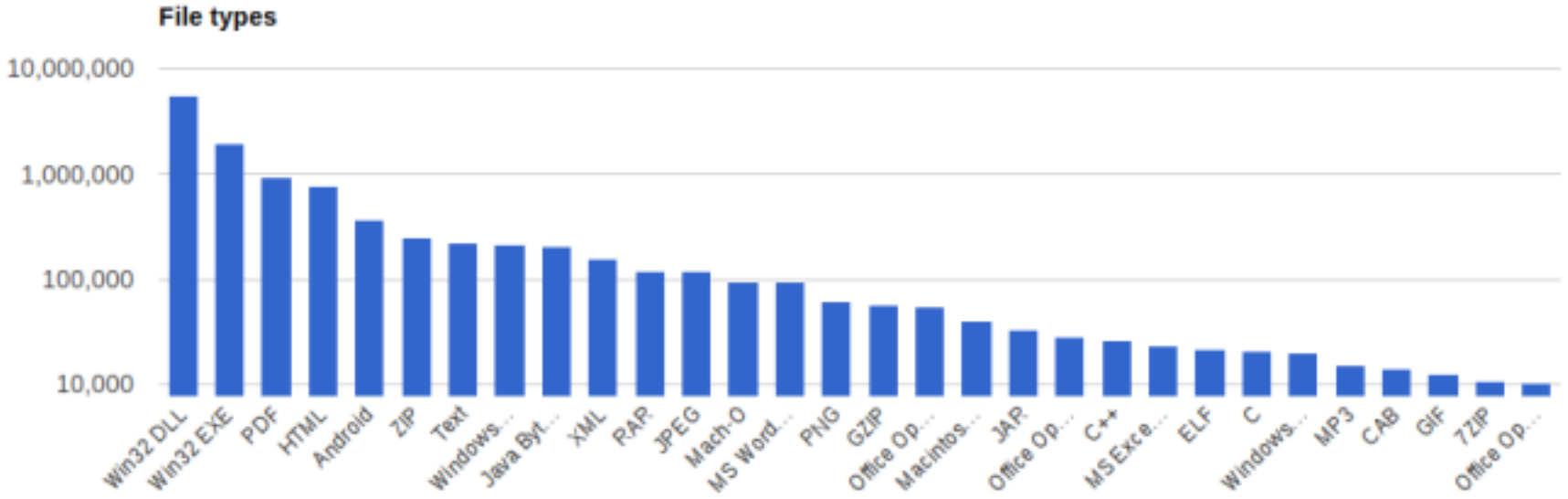
Malware Detection - Behavioral Methods

- Instead of scanning for signatures, examine what the program does when executed
- Very slow - AV must run the program and extract information about what the sample does
- Malicious samples can “run out the clock” on behavior checks

Scaling Malware Detection

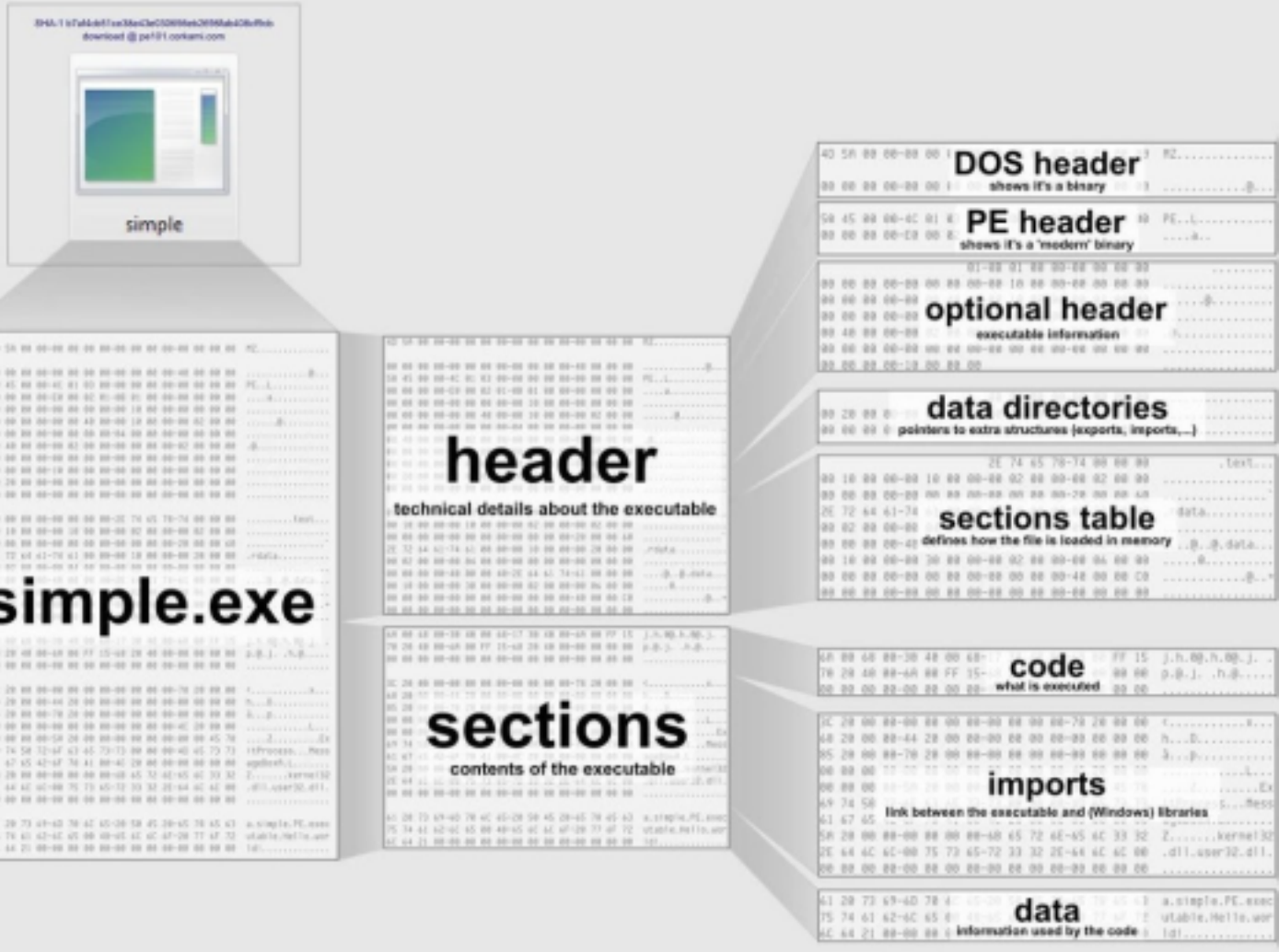
- Previously mentioned approaches have difficulty generalizing to new malware
- New kinds of malware require humans in the loop to reverse-engineer and create new signatures and heuristics for adequate detection
- Can we automate this process with machine learning?

Focus: Windows DLL/EXEs (Portable Executable)



Number of samples submitted to VirusTotal, Jan 29 2017

Portable Executable (PE) Format



Feature Engineering - Static Analysis

- What kinds of features can we extract for PE files?
- Objective: extract features from the EXE without executing anything
- PE-Specific features
  - Information about the structure of the PE file
- Strings
  - Print off all human-readable strings from the binary
- Entropy features
  - Extract information about the predictability of byte sequences
  - Compressed/encrypted data is high entropy
- Disassembly features
  - Get an idea of what kind of code the sample will execute

PE-Specific Features

FileVersionInfo properties

Copyright	© Microsoft Corporation. All rights reserved.
Product	Microsoft® Windows® Operating System
Original name	NOTEPAD.EXE
Internal name	Notepad
File version	5.1.2600.0 (xpclient.010817-1148)
Description	Notepad

PE header basic information

Target machine	Intel 386 or later processors and compatible processors
Compilation timestamp	2001-08-17 20:52:29
Entry Point	0x00006AE0
Number of sections	3

https://virustotal.com/en/file/e328b2406d8784e54e77ccc7dbe8e3731891a703e6c21cf7e2f924fa8a42ea5c/analysis/

PE-Specific Features (cont.)

PE sections

Name	Virtual address	Virtual size	Raw size	Entropy	MD5
.text	4096	28018	28160	6.28	ccf25baa681168e6396609387910d90a
.data	32768	7080	1536	1.40	cf692e5fbaebba02c2ad95f4ba0e60be
.rsrc	40960	35144	35328	5.41	c65b2250b8dd3870595004ca95f8f8b3

PE imports

[+] ADVAPI32.dll

[+] COMCTL32.dll

[+] GDI32.dll

[+] KERNEL32.dll

[+] SHELL32.dll

[+] USER32.dll

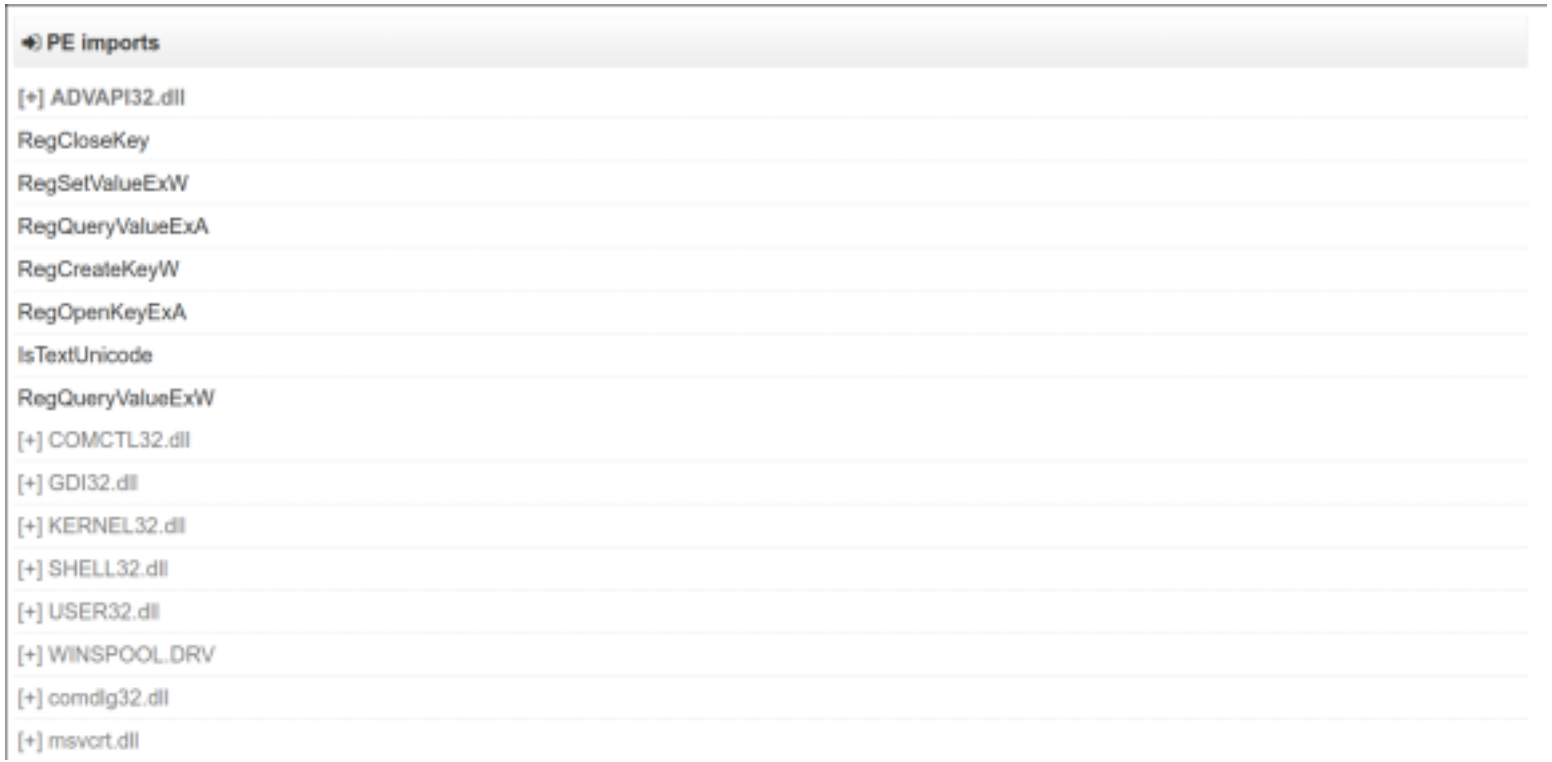
[+] WINSPOOL.DRV

[+] comdlg32.dll

[+] msvcrt.dll

https://virustotal.com/en/file/e328b2406d8784e54e77ccc7dbe8e3731891a703e6c21cf7e2f924fa8a42ea5c/analysis/

PE-Specific Features (cont.)



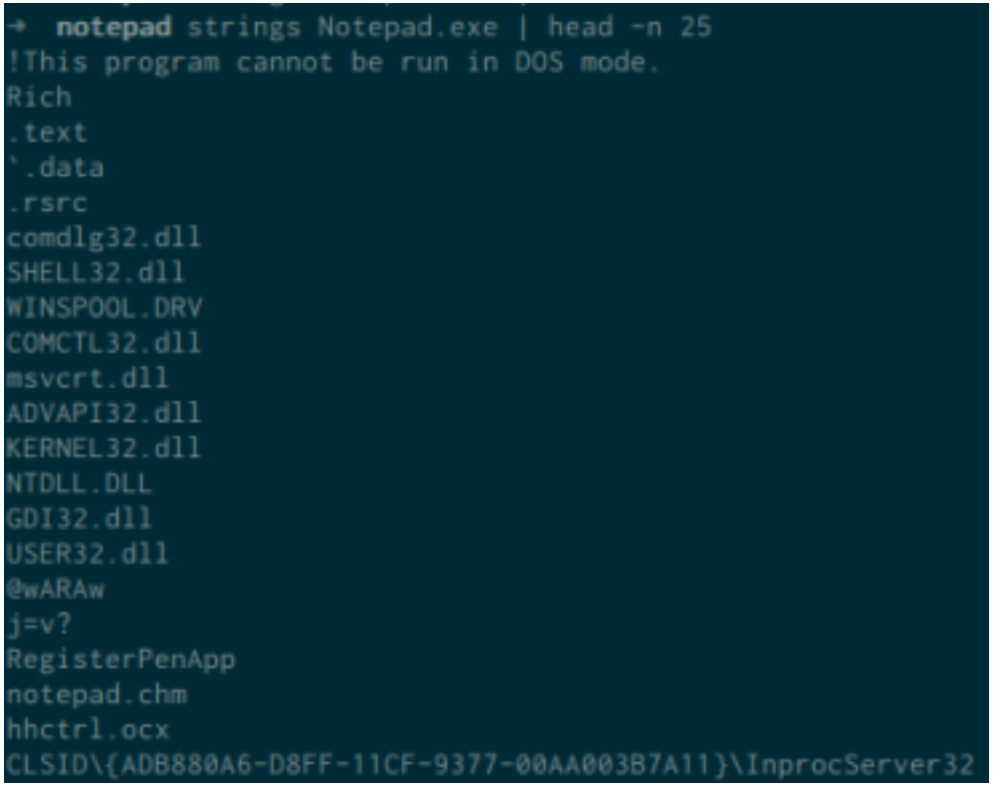
PE imports

[+] ADVAPI32.dll
RegCloseKey
RegSetValueExW
RegQueryValueExA
RegCreateKeyW
RegOpenKeyExA
IsTextUnicode
RegQueryValueExW
[+] COMCTL32.dll
[+] GDI32.dll
[+] KERNEL32.dll
[+] SHELL32.dll
[+] USER32.dll
[+] WINSPOOL.DRV
[+] comdlg32.dll
[+] msvcrt.dll

https://virustotal.com/en/file/e328b2406d8784e54e77ccc7dbe8e3731891a703e6c21cf7e2f924fa8a42ea5c/analysis/

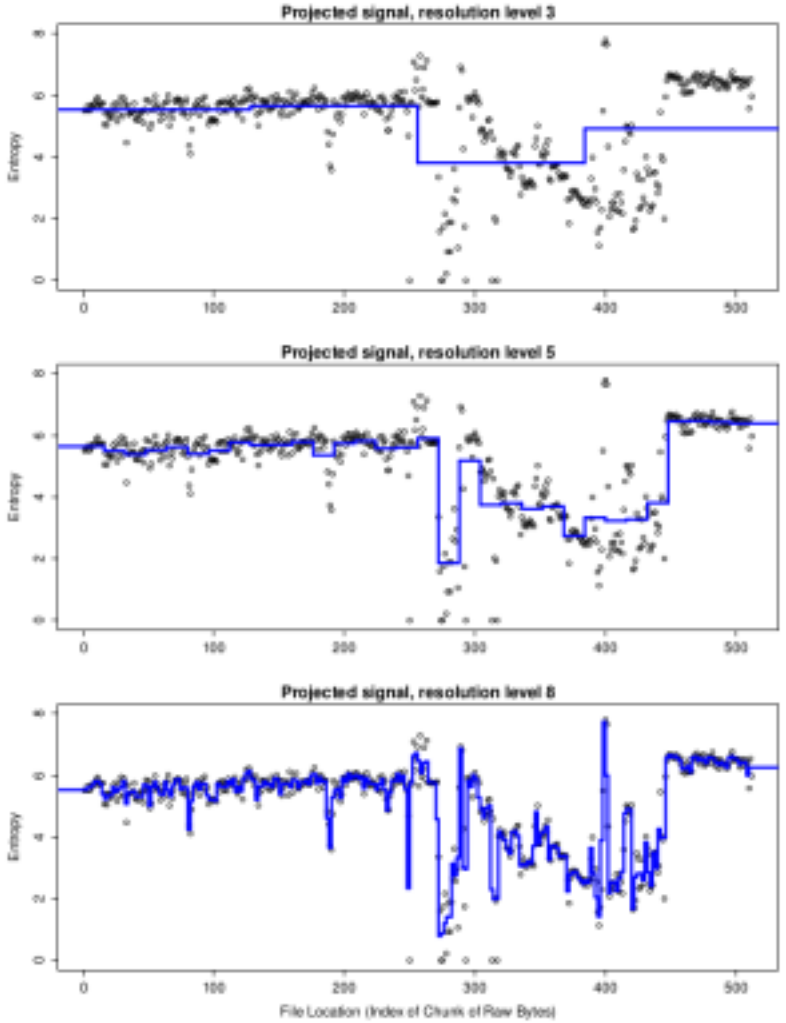
Feature Engineering - String Features

- Extract contiguous runs of ASCII-printable strings from the binary
  - Can see strings used for dialog boxes, user queries, menu items, ...
  - Samples trying to obfuscate themselves won't have many strings



Entropy Features

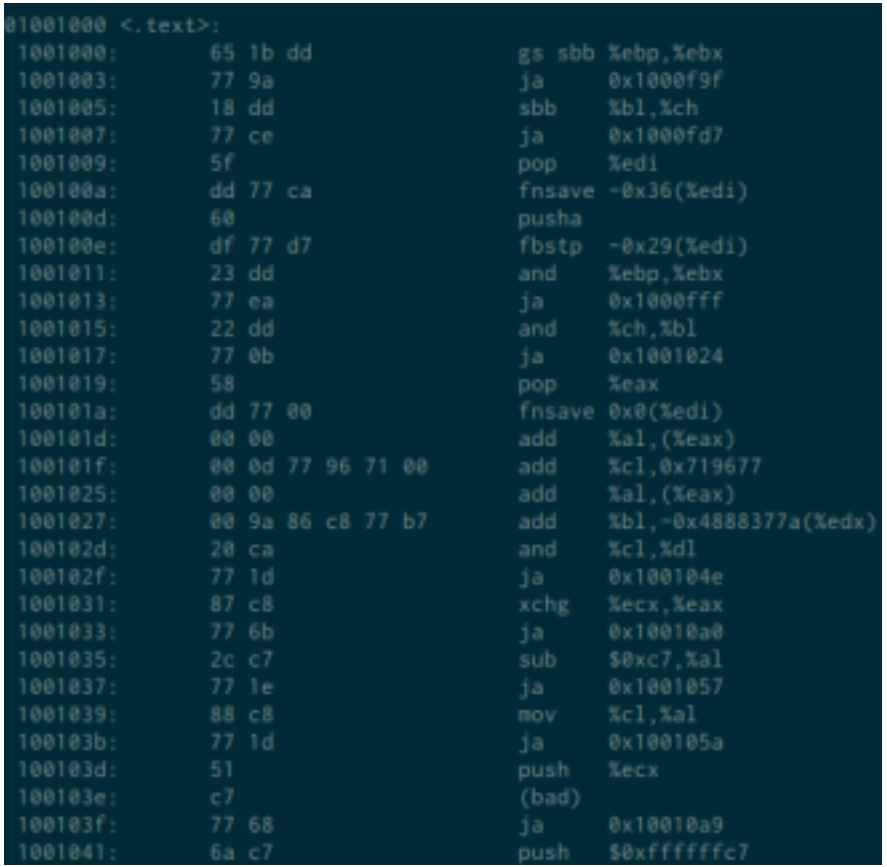
- Interpret the stream of bytes as a time-series signal
- Compute a sliding-window entropy of the sample
- Information can determine if there are compressed, obfuscated, or encrypted parts of the sample



"Wavelet decomposition of software entropy reveals symptoms of malicious code". Wojnowicz, et. al. https://arxiv.org/abs/1607.04950

Disassembly Features

- Contains information about what will actually execute
- Disassembly is difficult:
  - Hard to get all of the compiled instructions from a sample
  - x86 instruction set is variable-length
  - Ambiguity about what is executed depending on where one starts interpreting the stream of x86 instructions



Difficulties for Static Analysis

- Polymorphic code
  - Code that can modify itself as it executes
- Packing
  - Samples that compress themselves prior to execution, and decompress themselves while executing
  - Can hide malicious behavior in a compressed blob of bytes
  - Can obscure benign code as well
  - Requires expensive implementation of many unpackers (UPX, ASPack, Mew, Mpress, …)
- Disassembly
  - Malware authors can intentionally make the disassembly difficult to obtain

Modelling - Malicious versus Benign

- Boils down to a binary classification task
- N: hundreds of millions of samples
- P: millions of highly sparse features (s=0.9999)

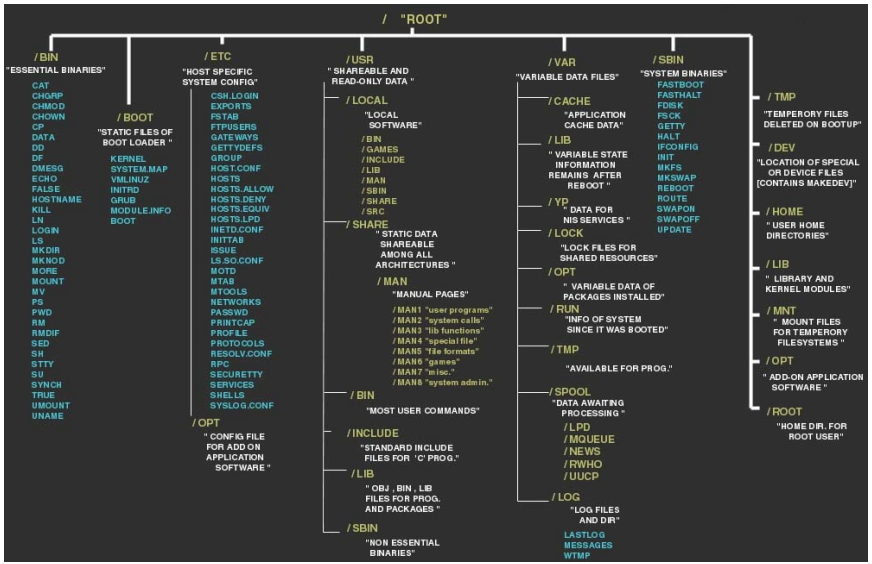
Modelling - Training on ~600 million samples

- Strong preference for minibatch methods and fast, compact models
- Logistic regression works very well
- Neural networks coupled with dimensionality reduction techniques are the workhorse
- Tend to combine lasso, dimensionality reduction, and neural networks

Files to Filesystems

Question: How else can we leverage hardware optimized for matrix operations?

Answer: Graph Kernels applied to filesystems



Filesystems – interesting topological structure

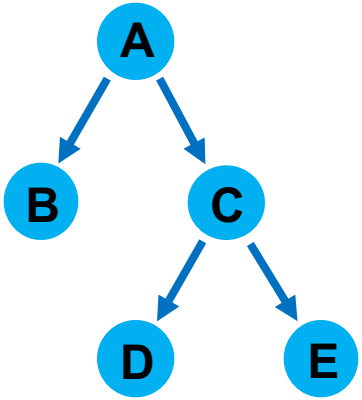
Idea: construct a map which measures the similarity between graphs G and H, which takes into account both the topological differences of the trees **and** the label differences.

$$K : \Gamma \times \Gamma \rightarrow \mathbb{R}$$

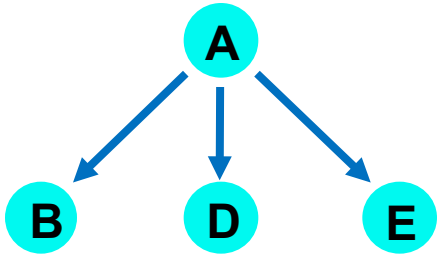
$K(G, H)$ measures the similarity between G and H, taking into account both the topological structure of the trees and their labels.

Upshot: We can measure the similarity between two file systems A and B by measuring the similarity between their labeled tree structure.

Graph Comparison and Vectorization



X



$\mathbb{R}$

$$\begin{pmatrix} 0 & ab & ac & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & cd & ce \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

X

$$\begin{pmatrix} 0 & ab & ad & ae \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$\mathbb{R}$

Filesystems – interesting topological structure

Can leverage GPU hardware in two ways:

- Kernel computations $K : \Gamma \times \Gamma \rightarrow \mathbb{R}$
- Neural Network training on features derived from these kernels

Upshot: The framing a given problem/procedure in terms of matrix algebra translates to massive computational advantages (GPU).

Selected Hardware

AWS P2 instances - up to 16 NVIDIA K80 GPUs

AWS G3 instance - four NVIDIA Tesla M60 GPUs

Thank You!

Questions?