Machine Learning for Malware Analysis

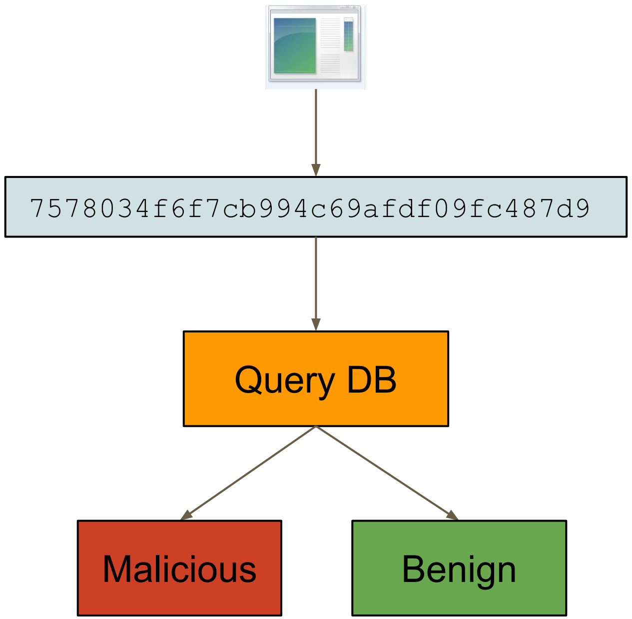
Andrew Davis
Data Scientist
CYLANCE

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

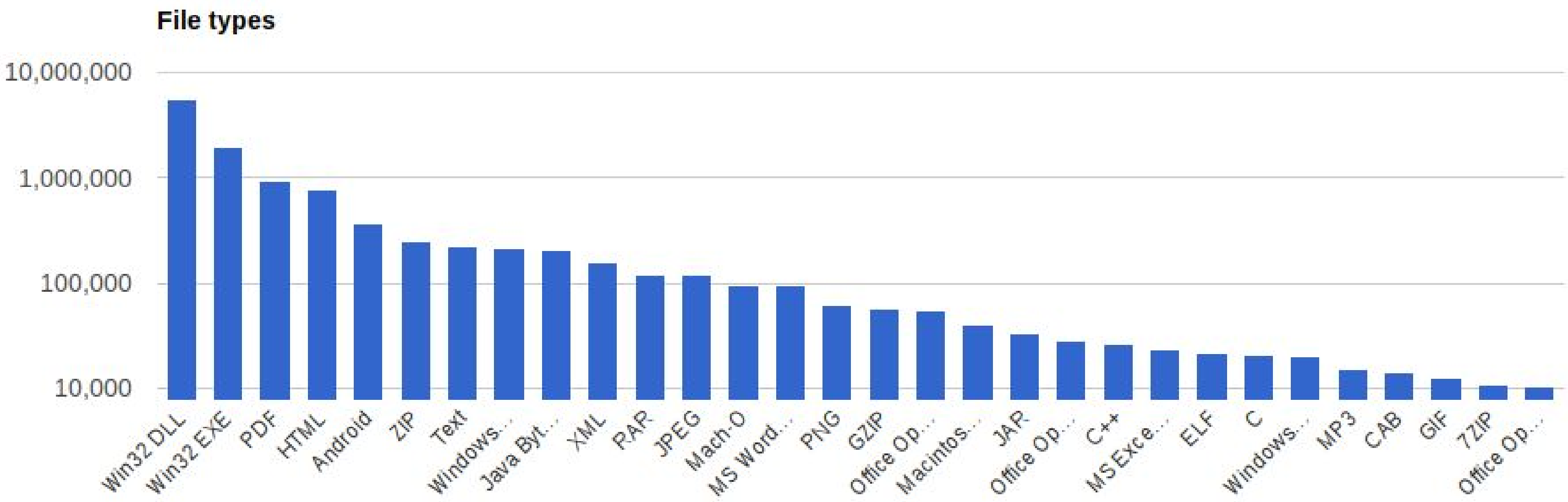
Malware Detection - Behavioral Methods

- 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

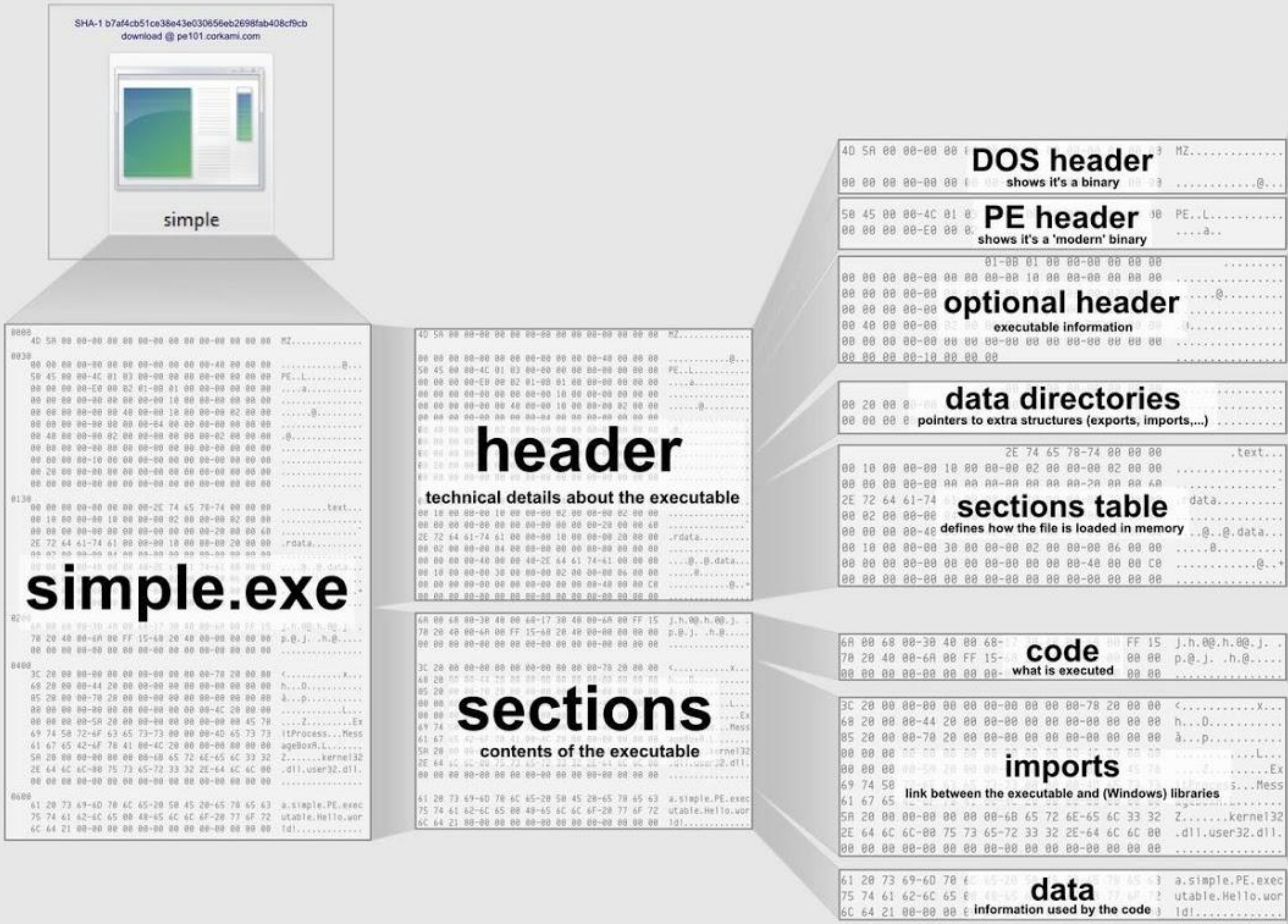
- Previously mentioned approaches have difficulty generalizing to new malware
- New kinds of malware require humans in the loop to reverse-engineer and create new signatures and heuristics for adequate detection
- Can we automate this process with machine learning?

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

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

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/

PE-Specific Features

[+] ADVAPI32.dll

[+] COMCTL32.dll

[+] GDI32.dll

GetTextMetricsW

SetMapMode

TextOutW

CreateFontIndirectW

GetTextExtentPoint32W

EnumFontsW

LPtoDP

GetDeviceCaps

DeleteDC

SetBkMode

EndDoc

StartPage

DeleteObject

GetObjectW

CreateDCW

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

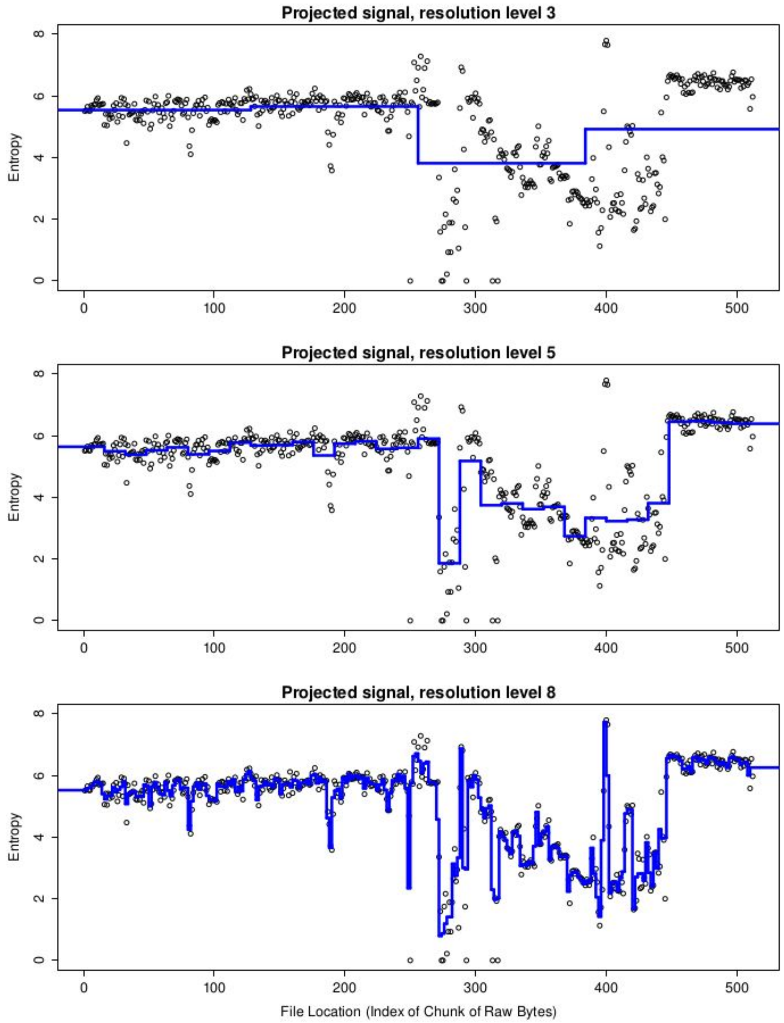
Feature Engineering - String Features

- 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

```
→  notepad strings Notepad.exe | head -n 25
!This program cannot be run in DOS mode.
Rich
.text
`.data
.rsrc
comdlg32.dll
SHELL32.dll
WINSPOOL.DRV
COMCTL32.dll
msvcrt.dll
ADVAPI32.dll
KERNEL32.dll
NTDLL.DLL
GDI32.dll
USER32.dll
@wARAw
j=v?
RegisterPenApp
notepad.chm
hhctrl.ocx
CLSID\{ADB880A6-D8FF-11CF-9377-00AA003B7A11}\InprocServer32
```

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

```
01001000 <.text>:
 1001000:       65 1b dd                gs sbb %ebp,%ebx
 1001003:       77 9a                   ja     0x1000f9f
 1001005:       18 dd                   sbb    %bl,%ch
 1001007:       77 ce                   ja     0x1000fd7
 1001009:       5f                      pop    %edi
 100100a:       dd 77 ca                fnsave -0x36(%edi)
 100100d:       60                      pusha
 100100e:       df 77 d7                fbstp  -0x29(%edi)
 1001011:       23 dd                   and    %ebp,%ebx
 1001013:       77 ea                   ja     0x1000fff
 1001015:       22 dd                   and    %ch,%bl
 1001017:       77 0b                   ja     0x1001024
 1001019:       58                      pop    %eax
 100101a:       dd 77 00                fnsave 0x0(%edi)
 100101d:       00 00                   add    %al,(%eax)
 100101f:       00 0d 77 96 71 00       add    %cl,0x719677
 1001025:       00 00                   add    %al,(%eax)
 1001027:       00 9a 86 c8 77 b7       add    %bl,-0x4888377a(%edx)
 100102d:       20 ca                   and    %cl,%dl
 100102f:       77 1d                   ja     0x100104e
 1001031:       87 c8                   xchg   %ecx,%eax
 1001033:       77 6b                   ja     0x10010a0
 1001035:       2c c7                   sub    $0xc7,%al
 1001037:       77 1e                   ja     0x1001057
 1001039:       88 c8                   mov    %cl,%al
 100103b:       77 1d                   ja     0x100105a
 100103d:       51                      push   %ecx
 100103e:       c7                      (bad)
 100103f:       77 68                   ja     0x10010a9
 1001041:       6a c7                   push   $0xffffffc7
```

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)

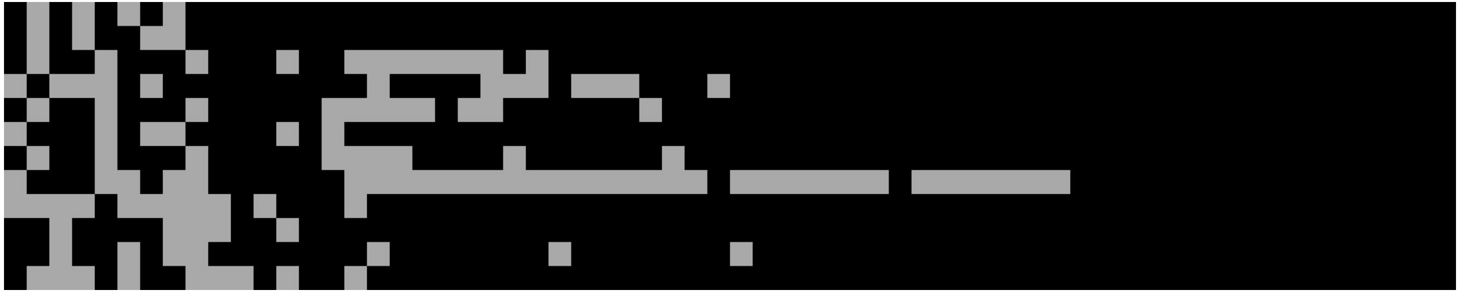
Modelling - Training on ~600 million samples

- 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

Convolutional Methods on Disassembly

```
push    %rbp
push    %rbx
mov     %rdi,%rbp
mov     $0x718700,%edx
sub     $0x8,%rsp
mov     (%rdx),%ecx
add     $0x4,%rdx
lea     -0x1010101(%rcx),%eax
not     %ecx
and     %ecx,%eax
and     $0x80808080,%eax
je      41aa4e <__sprintf_chk@plt+0x18b3e>
```

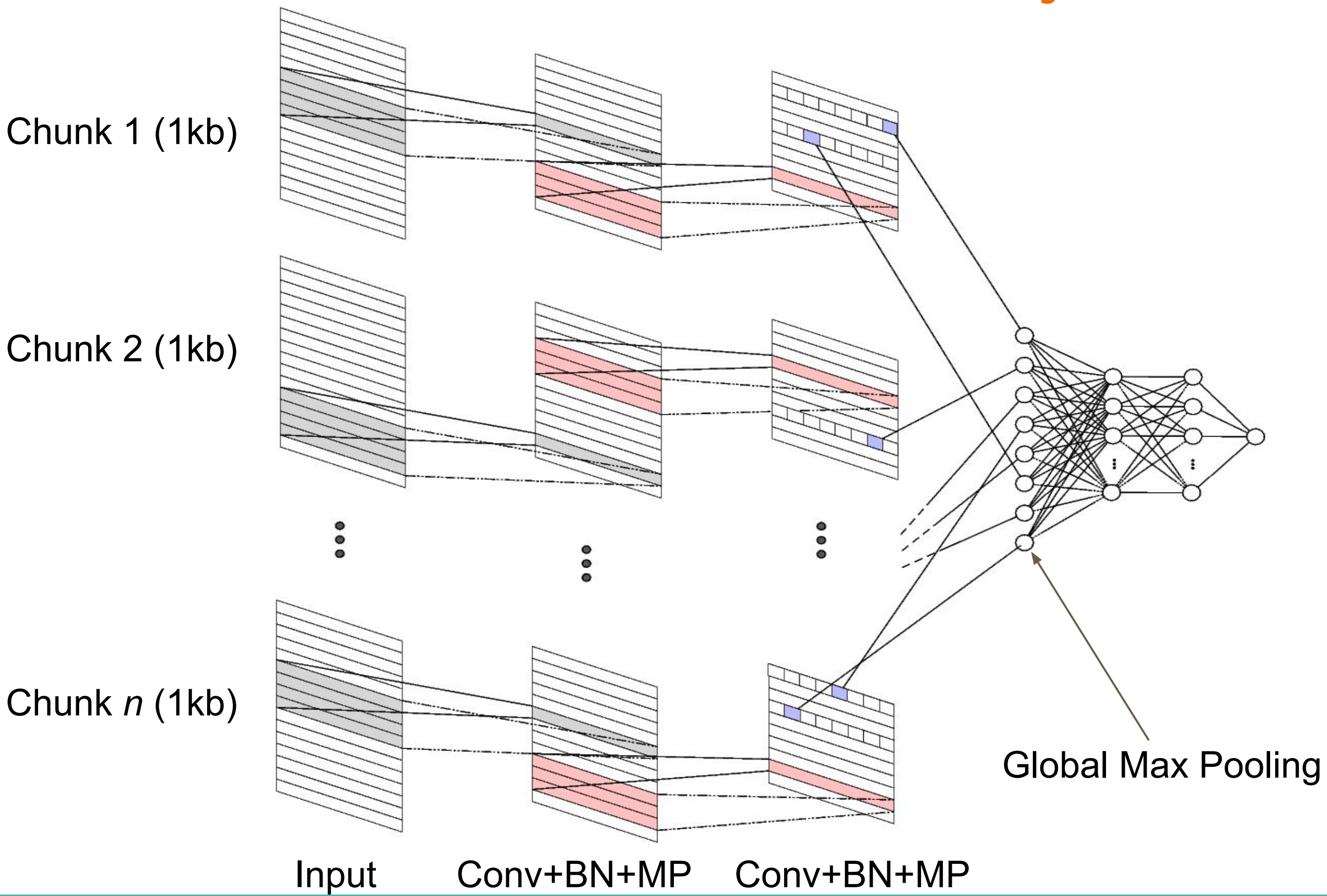
```
55
53
48 89 fd
ba 00 87 71 00
48 83 ec 08
8b 0a
48 83 c2 04
8d 81 ff fe fe fe
f7 d1
21 c8
25 80 80 80 80
74 e9
```



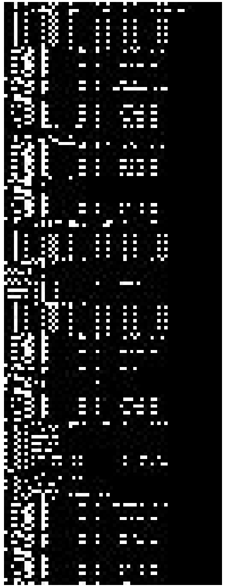

```
push    %rbp
push    %rbx
mov     %rdi,%rbp
mov     $0x718700,%edx
sub     $0x8,%rsp
mov     (%rdx),%ecx
add     $0x4,%rdx
lea     -0x1010101(%rcx),%eax
not     %ecx
and     %ecx,%eax
and     $0x80808080,%eax
je      41aa4e <__sprintf_chk@plt+0x18b3e>
```

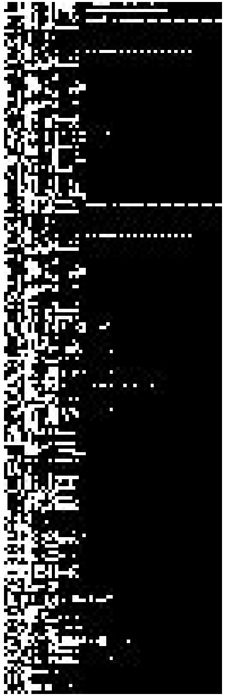

https://www.blackhat.com/docs/us-15/materials/us-15-Davis-Deep-Learning-On-Disassembly.pdf

Convolutional Methods on Disassembly

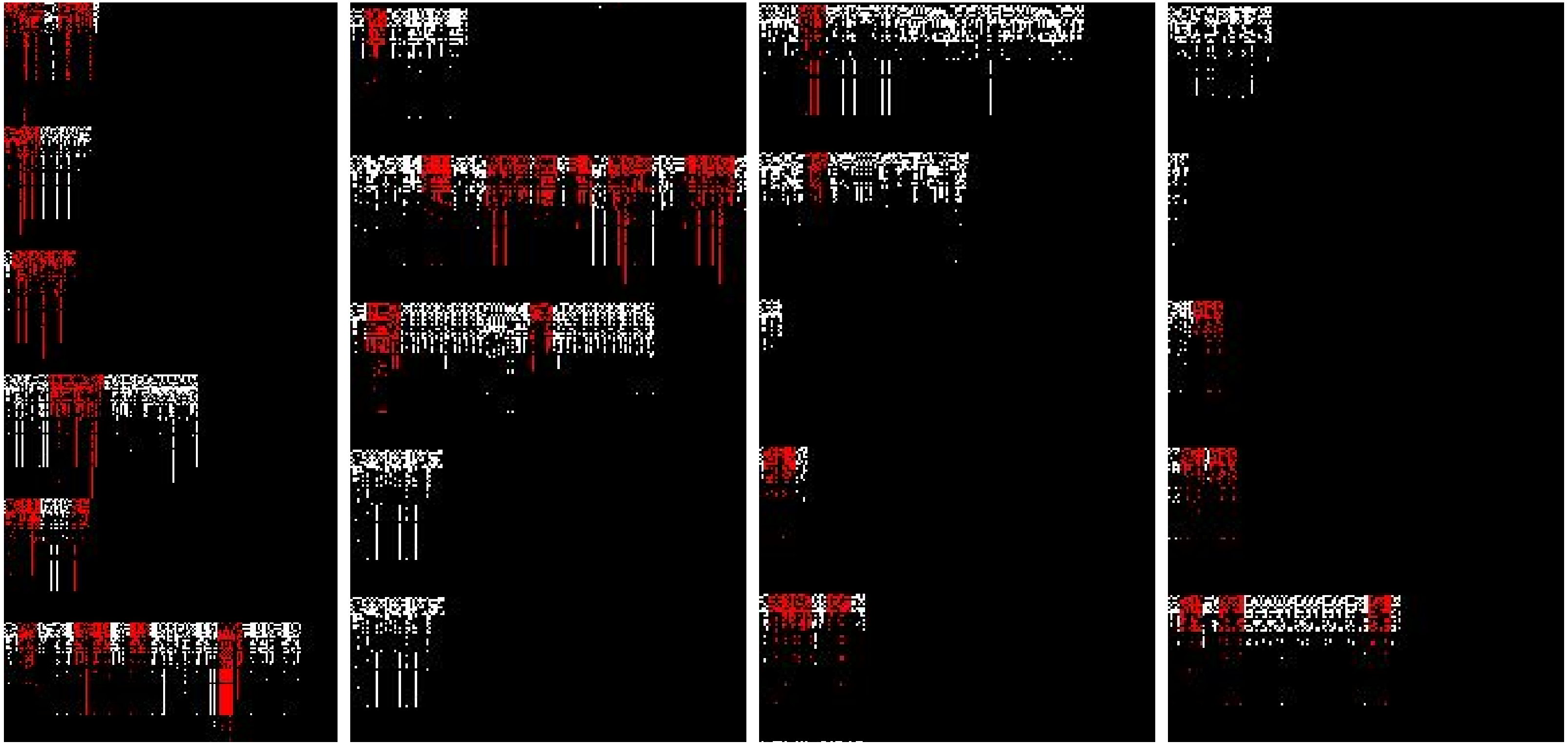


Spatial Structure in Instruction Visualizations





Global Max Pooling → Interpretability



MS Malware Kaggle Dataset

- 9 malware family classes:

Ramnit	Lollipop	Kelihos_ver3	Vundo	Simda	Tracur	Kelihos_ver1	Obfuscator.ACY	Gatak
1541	2478	2942	475	42	751	398	1228	1013

- ~10k training, ~10k testing
- Provides Ida disassembly and raw bytes, minus the PE header

Methodology:

- Separate training data into 90% training, 10% validation
- Use 10k testing samples to generate “pseudo-labels” (semi-supervision)

Model Definition

```
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 8, None, 1024) 0

____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 32, None, 512) 2080        input_1[0][0]
____________________________________________________________________________________________________
batchnormalization_1 (BatchNorma (None, 32, None, 512) 128         convolution2d_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 32, None, 512) 0           batchnormalization_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 32, None, 256) 0           activation_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 64, None, 128) 16448       maxpooling2d_1[0][0]
____________________________________________________________________________________________________
batchnormalization_2 (BatchNorma (None, 64, None, 128) 256         convolution2d_2[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 64, None, 128) 0           batchnormalization_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 64, None, 64)  0           activation_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 96, None, 32)  49248       maxpooling2d_2[0][0]
____________________________________________________________________________________________________
batchnormalization_3 (BatchNorma (None, 96, None, 32)  384         convolution2d_3[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 96, None, 32)  0           batchnormalization_3[0][0]
____________________________________________________________________________________________________
maxpooling2d_3 (MaxPooling2D)    (None, 96, None, 16)  0           activation_3[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 128, None, 8)  98432       maxpooling2d_3[0][0]
```

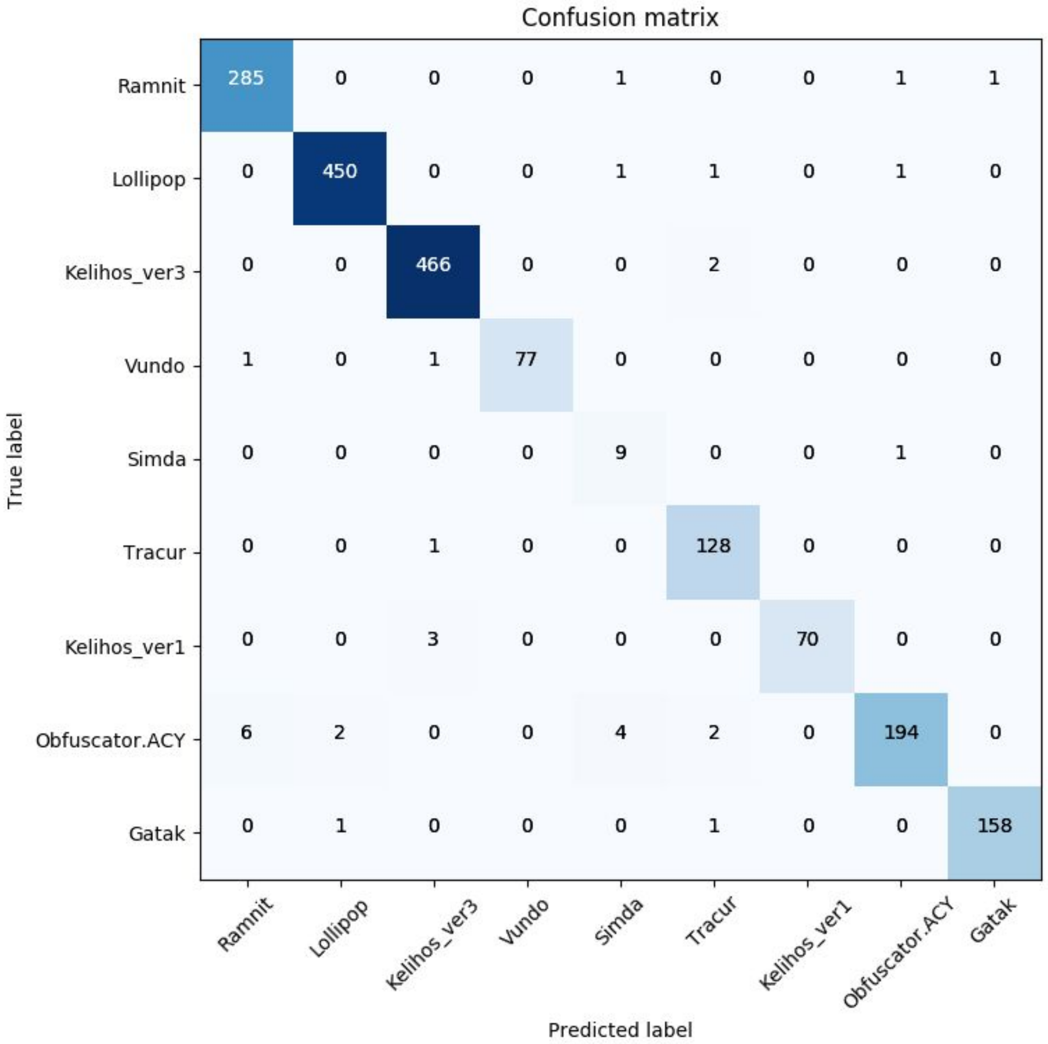
Model Definition

```
____________________________________________________________________________________________________
batchnormalization_4 (BatchNorma (None, 128, None, 8)  512         convolution2d_4[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)        (None, 128, None, 8)  0           batchnormalization_4[0][0]
____________________________________________________________________________________________________
maxpooling2d_4 (MaxPooling2D)    (None, 128, None, 4)  0           activation_4[0][0]
____________________________________________________________________________________________________
permute_1 (Permute)              (None, None, 128, 4)  0           maxpooling2d_4[0][0]
____________________________________________________________________________________________________
timedistributed_1 (TimeDistribut (None, None, 512)     0           permute_1[0][0]
____________________________________________________________________________________________________
globalmaxpooling1d_1 (GlobalMaxP (None, 512)           0           timedistributed_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 512)           0           globalmaxpooling1d_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 128)           65664       dropout_1[0][0]
____________________________________________________________________________________________________
batchnormalization_5 (BatchNorma (None, 128)           512         dense_1[0][0]
____________________________________________________________________________________________________
activation_5 (Activation)        (None, 128)           0           batchnormalization_5[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 128)           16512       activation_5[0][0]
____________________________________________________________________________________________________
batchnormalization_6 (BatchNorma (None, 128)           512         dense_2[0][0]
____________________________________________________________________________________________________
activation_6 (Activation)        (None, 128)           0           batchnormalization_6[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 9)             1161        activation_6[0][0]
====================================================================================================
```

Model: Results

Overall Acc	98.30%
Ramnit	98.96%
Lollipop	99.34%
Kelihos_v3	99.57%
Vundo	97.47%
Simda	90.00%
Tracur	99.22%
Kelihos_v1	95.89%
Obfusc	93.27%
Gatak	98.75%

#184 on Kaggle leaderboard



Thank You!

Questions?