black hat USA 2021

Generating YARA Rules by Classifying Malicious Byte Sequences

Andrew Davis

bio

- andrew davis / @gradientjanitor
- principal data scientist at elastic
- teaching computers to detect malware since 2014

intro & motivation

intro & motivation

- yara: great first line of defense against malware
- deep learning: effective, but decisions are usually incomprehensible
- but model architectures are really really flexible!
- set up the model so it is interpretable from the get-go, perhaps at the expense of model performance
- wouldn't it be neat if we turned this interpretability into a yara rule generator?

related work

- Raff et al, [Automatic Yara Rule Generation Using Biclustering]

Automatic Yara Rule Generation Using Biclustering

Edward Raff
Laboratory for Physical Sciences
Booz Allen Hamilton
edraff@lps.umd.edu

Richard Zak
Laboratory for Physical Sciences
Booz Allen Hamilton
rzak@lps.umd.edu

Gary Lopez Munoz
Laboratory for Physical Sciences
Booz Allen Hamilton
dlmgary@lps.umd.edu

William Fleming
U.S. Navy
william.r.fleming1@navy.mil

Hyrum S. Anderson*
Microsoft
hyruma@microsoft.com

Bobby Filar
Elastic NV
robert.filar@elastic.co

Charles Nicholas
Univ. of Maryland, Baltimore County
nicholas@umbc.edu

James Holt
Laboratory for Physical Sciences
holt@lps.umd.edu

- Marcelli & Squillero, [YaYaGen]

```
__ __  __ __    _____
\ \/ /__ \ \/ /__ _/ ___/__ ___   YaYaGen -- Yet Another Yara Rule Generator
 \  / _ `/\  / _ `/ (_ / -_) _ \  (!) v0.5_summer18
 /_/\_,_/ /_/\_,_/\___/\__/_//_/  by Andrea Marcelli & Giovanni Squillero

usage: YaYaGen [-h] [-d] [-ndb] [-dry] [-a ALGORITHM] [-opt OPTIMIZER]
               [-u URL] [-dir DIRECTORY] [-f FILTER] [-o OUTPUTDIR]
               [-name RULENAME]
               [sha256 [sha256 ...]]
```

- Joshua Saxe, [YaraML]

```
rule Generic_Powershell_Detector
{
strings:
...
$s4 = "DownloadFile"       fullword // weight: 3.257
$s5 = "WOW64"              fullword // weight: 3.232
$s6 = "bypass"             fullword // weight: 3.021
$s7 = "meMoRYSTrEaM"       fullword // weight: 2.68
$s8 = "obJEct"             fullword // weight: 2.679
$s9 = "OBJecT"             fullword // weight: 2.659
$s10 = "ReGeX"              fullword // weight: 2.592
$s11 = "samratashok"        fullword // weight: 2.548
$s12 = "Dependencies"       fullword // weight: 2.494
$s13 = "TVqQAAMAAAAEAAAA"   fullword // weight: 2.428
$s14 = "CompressionMode"    fullword // weight: 2.366
...
condition:
...
((#s0 * 5.567) + (#s1 * 4.122) + (#s2 * 3.904) + (#s3 * 3.820) +
(#s4 * 3.257) + (#s5 * 3.232) + (#s6 * 3.021) + (#s7 * 2.680) +
(#s8 * 2.679) + (#s9 * 2.659) + (#s10 * 2.592) + (#s11 * 2.548) +
...
> 0
}
```

making an interpretable model

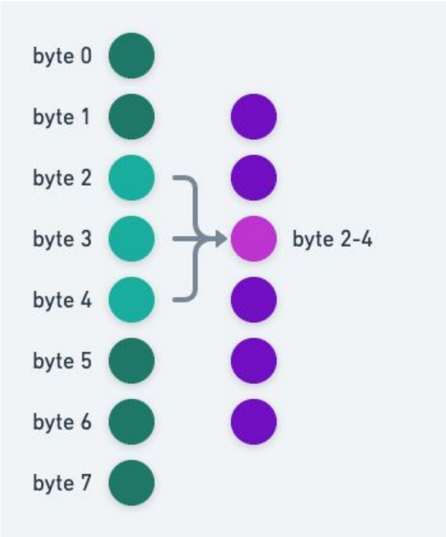
- deep learning models look at the whole of the sample to get a score
- what if we set up the model so we get a score for any contiguous series of bytes?
- we could feed in malicious samples and get exactly what ranges of bytes the model considered to be malicious
- then, we can create yara rules based on the byte sequences the model thought were malicious

$f(\text{"}you\ have\ been\ pwned\text{"}) = 0.99$ $f(\text{"}!This\ program\ cannot\ be\ run\ in\ DOS\ mode.\text{"}) = 0.00$

convolutional neural networks: a primer

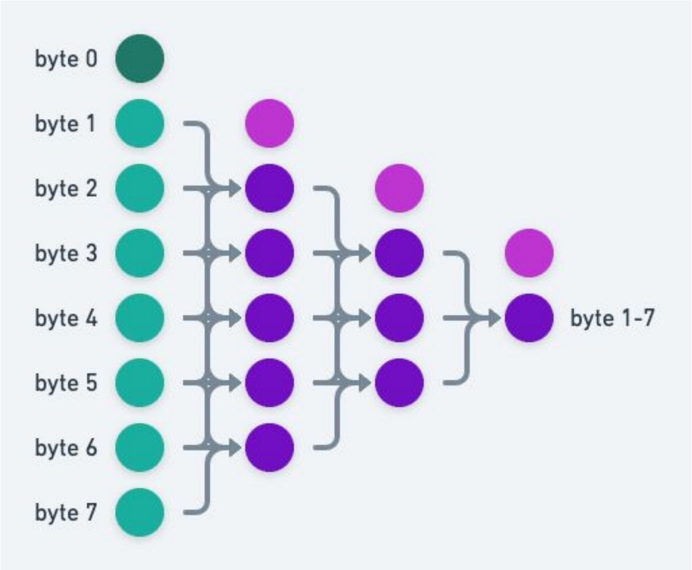
convolution

- sweep over chunks of contiguous bytes, applying the same function each time

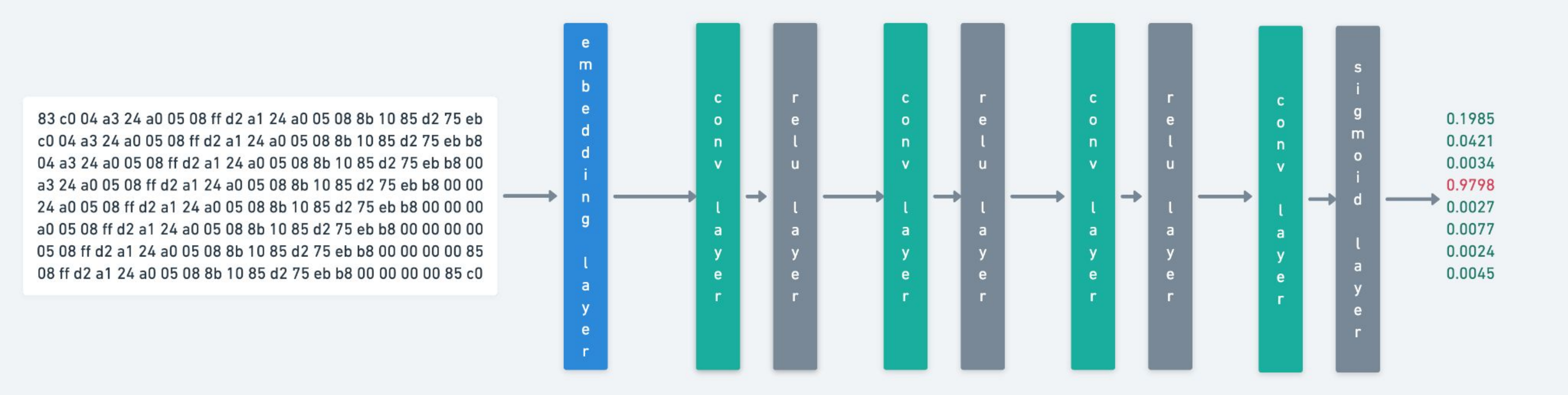


stacking convolutions

- more depth: wider receptive field



how the model works

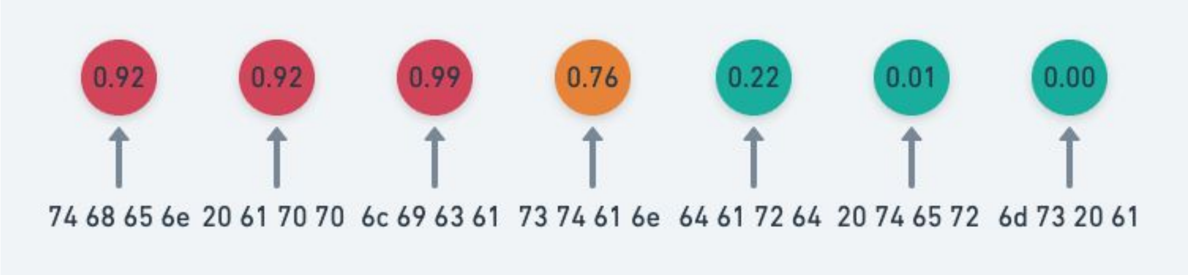


- just stacks of convolution/nonlinearity. we don't want to reduce dimensionality
- we want each sequence of input bytes to eventually get a score
- deeper architecture → larger receptive field → longer strings for yara rules
- feed-forward 1000 bytes → get (1000 - receptive field size) scores

training the model

model training - finding needles in a haystack

- "malicious" string: a string seen ONLY in malicious samples
- "benign" string: a string that can be seen in either malicious or benign samples
- use direct interpretability of output scores to assess benignness/maliciousness
- how to get the model to output zeros for almost everything except for strings associated with maliciousness?



model training - top-*k* selection

- when training the model, select the top-*k* valued scores to update

- allows malicious samples to have sparse outputs while avoiding updating garbage back through “benign” strings

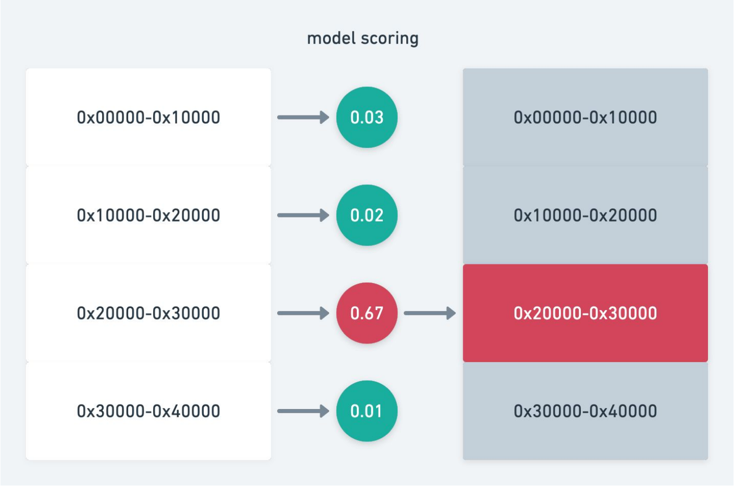
model training - top-*k* selection

- when updating the model, select the top-*k* valued scores to backpropagate through

- allows malicious samples to have sparse outputs while avoiding backpropagating garbage to “benign” strings
- while also forcing benign sample outputs to be very close to zero

training the model - fitting onto a gpu

- gpus only have so much memory - need to be sparing
- break each sample into 64kb segments
- for each sample:
  - feed each segment through the model
  - keep the segment associated with the max seen score and discard everything else



training the model - reducing FPs

- neural nets work very hard to find shortcuts that solve the problem in unexpected ways
- without correction, the model fixates on strings seen infrequently in benign samples, but frequently in malicious samples
- keep a rolling buffer of FPs from the last ~10 minutes to throw into each training minibatch
- sample FPs with more malicious scores more frequently than FPs with less malicious scores

signature generation

signature generation

per sample

in bulk

signature generation - sample by sample

terminal time

signature generation - in bulk

- run model over a corpus of malicious and benign samples
- dump out signature associated with the max score for each sample
- banish signatures from benign samples with high scores
- sort signatures by prevalence
- cluster signatures together based on hamming distance
- replace differing bytes of signatures in a cluster with wildcards to increase signature generality

```
d2 48 8b 05 04 73 21 00 48 8b
d2 48 8b 05 04 46 21 00 48 8b
d2 48 8b 05 04 75 21 00 48 8b
d2 48 8b 05 04 9f 21 00 48 8b
d2 48 8b 05 04 33 21 00 48 8b
d2 48 8b 05 04 9a 21 00 48 8b
d2 48 8b 05 04 87 21 00 48 8b
d2 48 8b 05 04 11 21 00 48 8b
d2 48 8b 05 04 21 21 00 48 8b
d2 48 8b 05 04 58 21 00 48 8b
d2 48 8b 05 04 19 21 00 48 8b
d2 48 8b 05 04 4a 21 00 48 8b
d2 48 8b 05 04 78 21 00 48 8b
d2 48 8b 05 04 6d 21 00 48 8b
d2 48 8b 05 04 2d 21 00 48 8b
d2 48 8b 05 04 a8 21 00 48 8b
-----------------------------
d2 48 8b 05 04 ?? 21 00 48 8b
```

signature efficacy

	ELF	Macho	PE
Collection Date	2017-2021	20xx-2021	2020-2021
Sample Breakdown	84k bad, 5.5mil good (4.5mil Ubuntu, 1mil VT)	1mil bad, 9mil good	10mil bad, 10mil good
TPR/FPR	81.6% TPR / 0% FPR (Ubuntu) / 0.15% FPR (VT)	90% TPR / 0.01% FPR	79.9% TPR / 0.07% FPR
Rule Count	950 rulesules	11 rules (!!)	700 rules

future work

- utilizing more yara functionality:
    - string offset
    - string count
    - complex combinations of strings and logical statements in yara rules
- model-driven string wildcarding
    - use input sensitivity to determine bytes the model doesn’t care about
- integrate tool with parsing libraries
    - provide context with section, surrounding opcodes, …

thank you!!