



# Malware Detection in the AI Era: Attacks and Defenses on Machine Learning Classifiers

Dmitrijs Trizna

Security Researcher | Microsoft Corporation, University of Genova

Luca Demetrio

Assistant Professor | University of Genova

# WhoAreWe

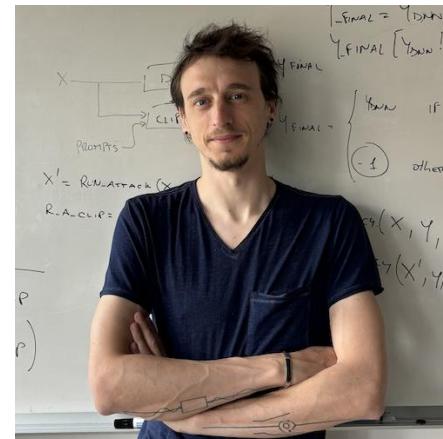


**Security Researcher @ Microsoft**  
Tech-lead in M365/Azure back-end:  
AI/ML for Linux & Kubernetes cyber-threat detection

**Doctoral Researcher @ UNIGE**  
AI/ML based malware detectors

In Past:

- Red Teaming
- NATO events
- OSCP, SANS/GIAC certs
- ...



**Assistant Professor @ UNIGE**

Doing research for fun and profit since 2019

Main contribution in offensive security  
with Adversarial EXEmples

Reviewer for top-tier journals and conferences

Working in EU projects

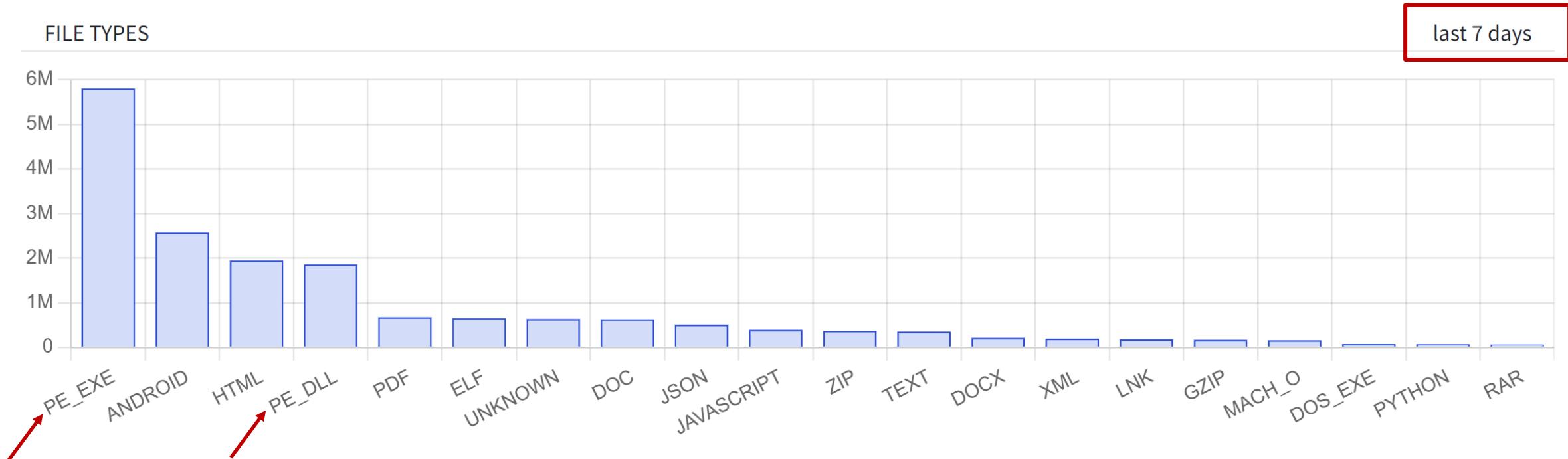


# Agenda

- **Part 1: Building AI-Based Malware Detector**
  - Intro and Threat Model
  - Static Malware Analysis with AI/ML
  - Dynamic Malware Analysis with AI/ML
- **Part 2: Attacking AI-Based Malware Detector**
  - Intro to Adversarial Machine Learning
  - Adversarial EXEmples
  - Defenses against Adversarial Manipulations

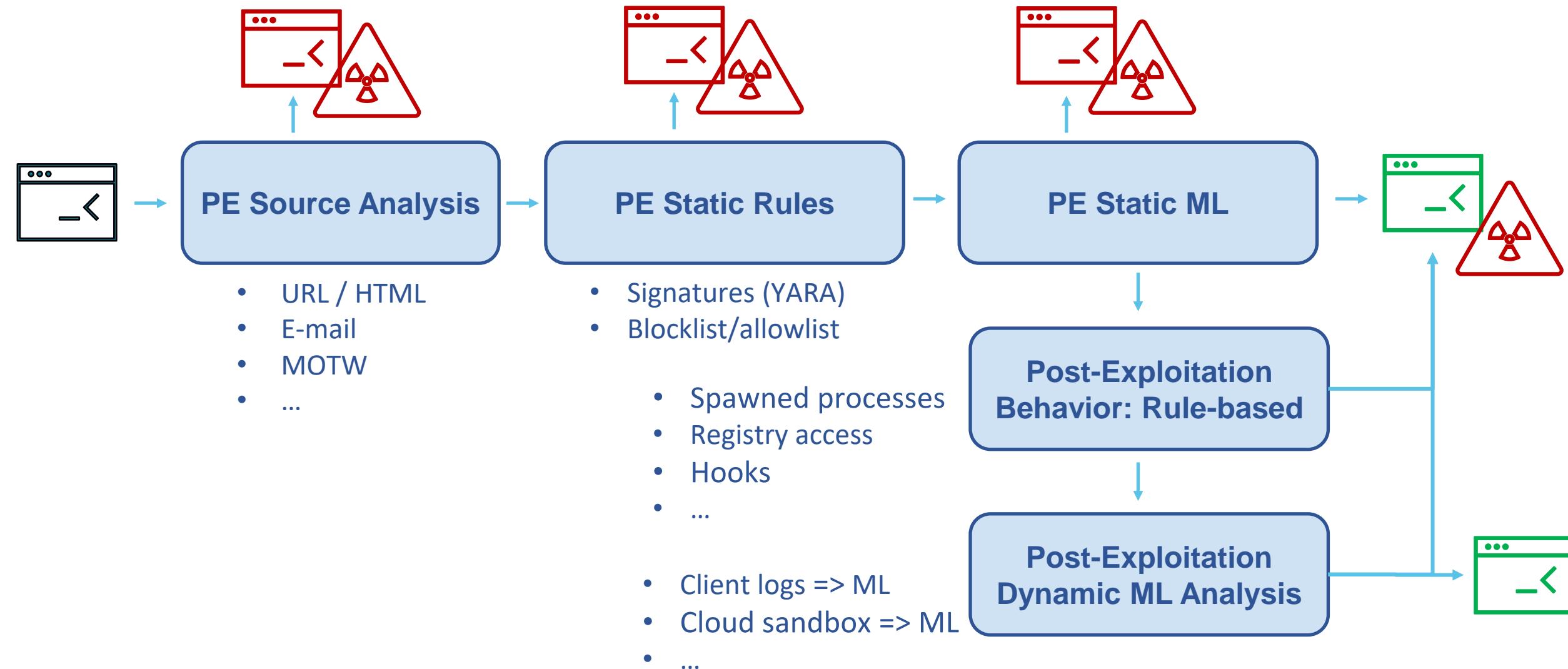
# Threat Model

# PE Files Are Still The Most Common Malware Type



[Source: <https://www.virustotal.com/gui/stats> ]

# Modern Malware Detector Stack



# Holistic Evasion of Modern Stack

**Analysis of Domain Fronting Technique: Abuse and Hiding via CDNs**

KRISHNA KONA, LIDOR PERGAMENT  
March 22, 2022 – 7 min read



**The Problem With YARA: Evading Google Elastic Security EDR with a NOP instruction**

Feb. 10, 2024 | Categories: Research

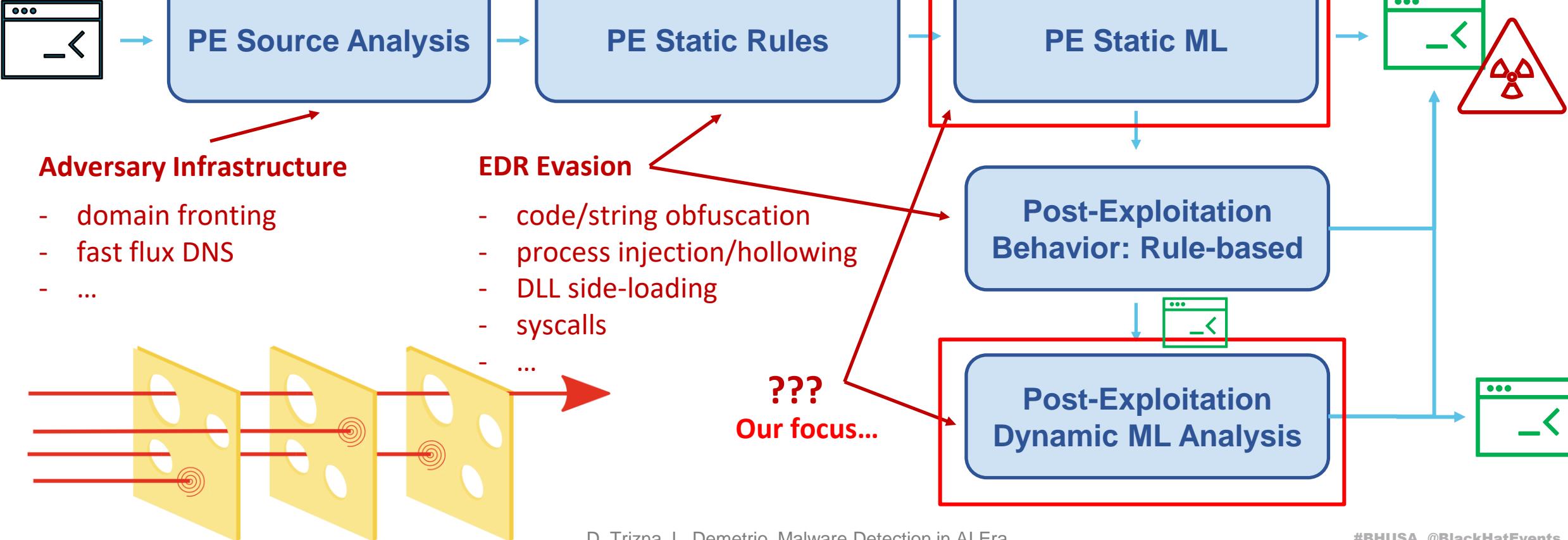


edr evasion

Past year ▾

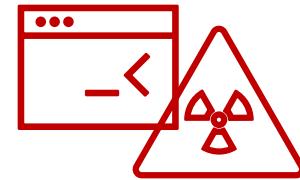
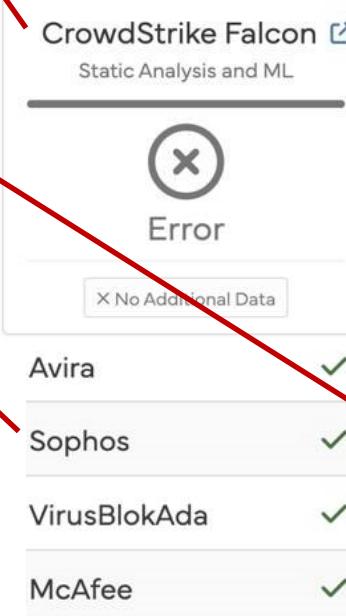
About 10,900 results (0.26 seconds)

**Evading Windows Defender Detections with a combination of MiniDumpWriteDump, Donut, and Process Injection Techniques**



# Attack Just on Static Module

Figure 1: Magic Quadrant for Endpoint Protection Platforms



Petya Ransomware

PE Static ML

#petyavirus	#petya	#apt
#blackenergy	#quedagh	
#ransomware	#sandworm	
#sandwormteam	#temp.noble	
#voodoobear	#jigsaw	#shamoon

CMC	✓ Undetected
Cylance	✓ Undetected
DeepInstinct	✓ Undetected
Elastic	✓ Undetected
Jiangmin	✓ Undetected
Kingsoft	✓ Undetected
Lionic	✓ Undetected
Palo Alto Networks	✓ Undetected
Panda	✓ Undetected

# Limitations and Strengths of Static Rules

## YARA

# YARA Basics

- Rules have straightforward syntax:
- Easy to use CLI tool:

```
yara <rule> <file>
```

- Features:

```
dtrizna@tanuki:/data/aponte/repos/mal-pipeline$ yara -X yara_rules/gen_
SUSP_XORed_UPD_In_EXE /data/datasets/pe/pe_validation/malware/e430acd25
0x68650:$s1:xor(0x01,http://)
0x68964:$s1:xor(0x01,http://)
```

```
rule Example
{
    strings:
        $a = "This program cannot" xor
        $b = { 41 42 ( 43 | 44 )}

    condition:
        $a or $b
}
```

- Python API:

```
def check_sample(self, sample_bytes, rules) -> yara.Match:
    rules = yara.compile(source=rules)
    self.matches = rules.match(data=sample_bytes)
    if self.matches:
        print(f"[+] Match found: {self.matches}")
```

# YARA Is Awesome: Do's

- While looking for additional implants that used the same loader shellcode as the 3CX implants, we discovered a sample on a multiscanner service (MD5: [933508a9832da1150fcfdbc1ca9bc84c](#)) loading a payload that uses the wirexpro[.]com C2 server. The same server is [listed](#) as an IoC for an AppleJeus campaign by Malwarebytes.

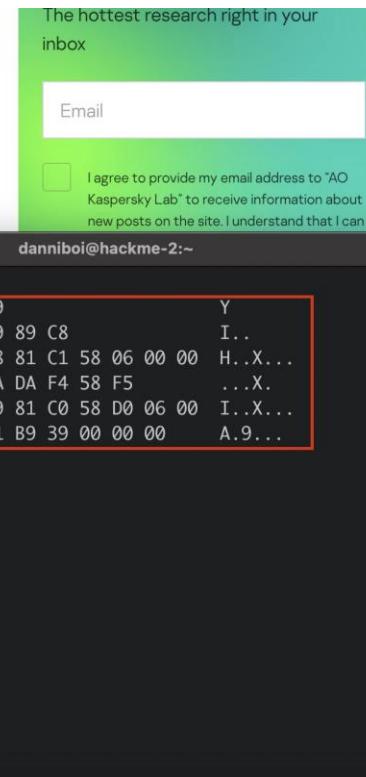
```

sub_0      proc near
var_18      = dword ptr -18h
              . . .
call    $+5
pop    rcx
mov    r8, rcx
add    rcx, 658h
mov    edx, 0F558F4DAh
add    r8, 6D058h
mov    r9d, 39h ; '9'
push   rsi
mov    rsi, rsp
and    rsp, 0FFFFFFFh
sub    rsp, 30h
mov    [rsp+38h+var_18]
call   refl_load
mov    rsp, rsi
pop    rsi
ret
ret

First bytes of the loader shellcode use

```

@dan\_mayer 100DY 2023/Stairwell



[Source:<https://securelist.com/gopuram-backdoor-deployed-through-3cx-supply-chain-attack/109344/>]

```

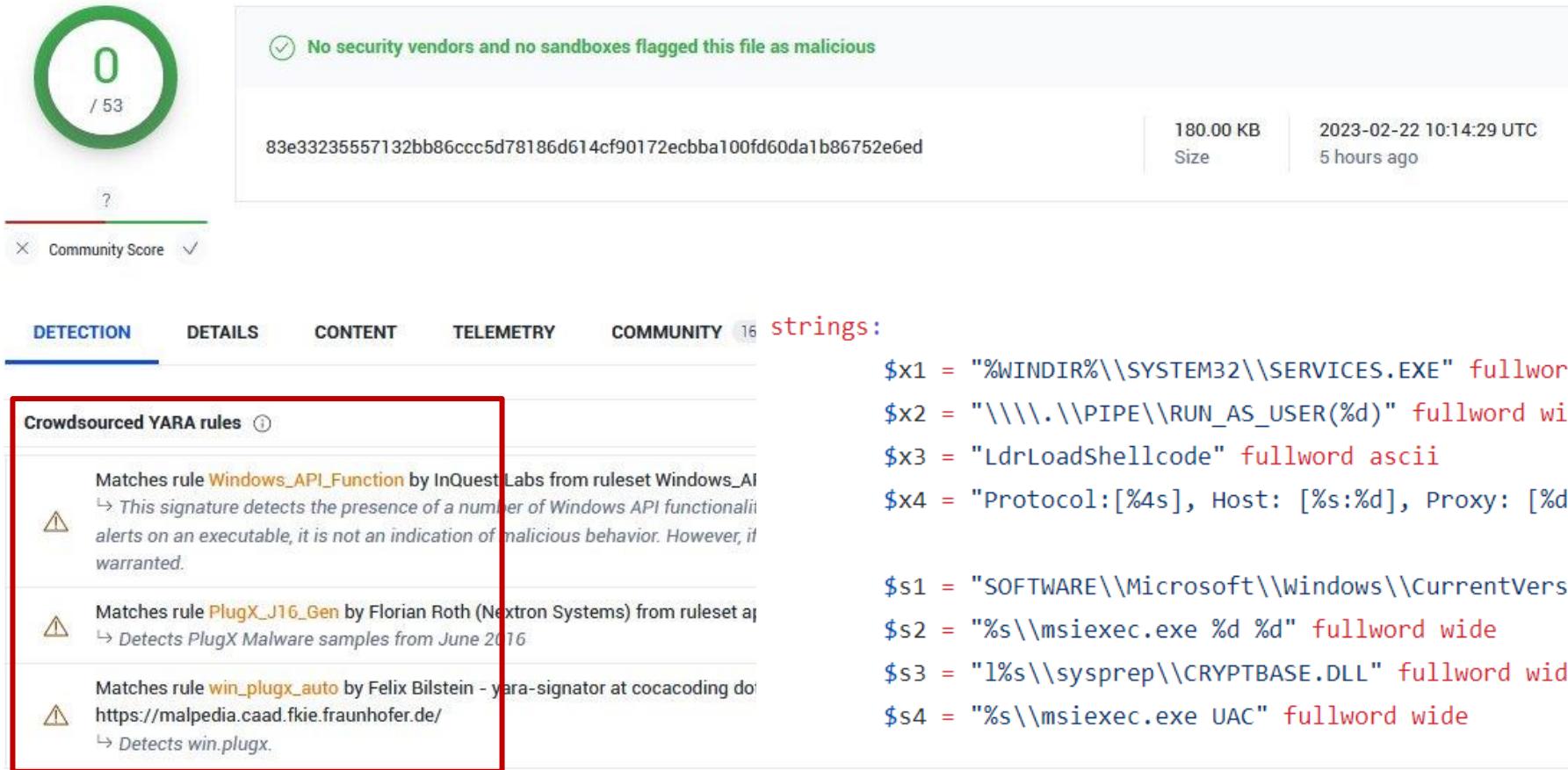
rule dev_3CX_Gopuram_AppleJeus_Shellcode_Hunt
{
  meta:
    author = "Daniel Mayer (daniel@stairwell.com)"
    description = "A rule for Gopuram shellcode based off of a screenshot of ida in Kaspersky"
    version = "1.0"
    date = "2023-04-03"
    reference1="https://securelist.com/gopuram-backdoor-deployed-through-3cx-supply-chain-at
  strings:
    $test_hunt = {
      59                      // pop rcx
      49 89 C8                // mov r8, rcx
      48 81 C1 58 06 00 00    // add rcx, 0x658
      BA DA F4 58 F5          // mov edx, 0xf558f4da
      49 81 C0 58 D0 06 00    // add r8,0x6d058
      41 B9 39 00 00 00        // mov r9d, 0x39
    }

  condition:
    all of them
}

```

[Source:[https://github.com/MayerDaniel/100DY\\_2023/blob/main/dan/dev\\_3CX\\_Gopuram\\_AppleJeus\\_Shellcode\\_Hunt.yara](https://github.com/MayerDaniel/100DY_2023/blob/main/dan/dev_3CX_Gopuram_AppleJeus_Shellcode_Hunt.yara)]

# YARA Is Awesome: Do's



0 / 53

✓ No security vendors and no sandboxes flagged this file as malicious

83e33235557132bb86ccc5d78186d614cf90172ecba100fd60da1b86752e6ed

180.00 KB      2023-02-22 10:14:29 UTC  
Size              5 hours ago

?

Community Score ✓

**DETECTION**   **DETAILS**   **CONTENT**   **TELEMETRY**   **COMMUNITY** 16

**Crowdsourced YARA rules** ⓘ

- ⚠️ Matches rule [Windows\\_API\\_Function](#) by InQuest Labs from ruleset Windows\_AI
  - ↳ This signature detects the presence of a number of Windows API functionality alerts on an executable, it is not an indication of malicious behavior. However, it warranted.
- ⚠️ Matches rule [PlugX\\_J16\\_Gen](#) by Florian Roth (Nextron Systems) from ruleset aj
  - ↳ Detects PlugX Malware samples from June 2016
- ⚠️ Matches rule [win\\_plugx\\_auto](#) by Felix Bilstein - yara-signator at cocacoding do
  - ↳ https://malpedia.caad.fkie.fraunhofer.de/
  - ↳ Detects win.plugx.

**strings:**

```
$x1 = "%WINDIR%\SYSTEM32\SERVICES.EXE" fullword wide
$x2 = "\\\.\PIPE\RUN_AS_USER(%d)" fullword wide
$x3 = "LdrLoadShellcode" fullword ascii
$x4 = "Protocol:[%4s], Host: [%s:%d], Proxy: [%d:%s:%d:%s]" fullword ascii

$s1 = "SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\User Age"
$s2 = "%s\msiexec.exe %d %d" fullword wide
$s3 = "1%sysprep\CRYPTBASE.DLL" fullword wide
$s4 = "%s\msiexec.exe UAC" fullword wide
```

**condition:**

```
( uint16(0) == 0x5a4d and filesize < 600KB and ( 1 of ($x*) or 4 of ($s*) ) ) or ( 8 of them )
```

[Source: [https://github.com/Neo23x0/signature-base/blob/6b8e2a00e5aafcfc767f3f53ae986cf81f968a/yara/apt\\_win\\_plugx.yar#L10](https://github.com/Neo23x0/signature-base/blob/6b8e2a00e5aafcfc767f3f53ae986cf81f968a/yara/apt_win_plugx.yar#L10)]

# YARA Is Awesome: Do's and Don'ts

```

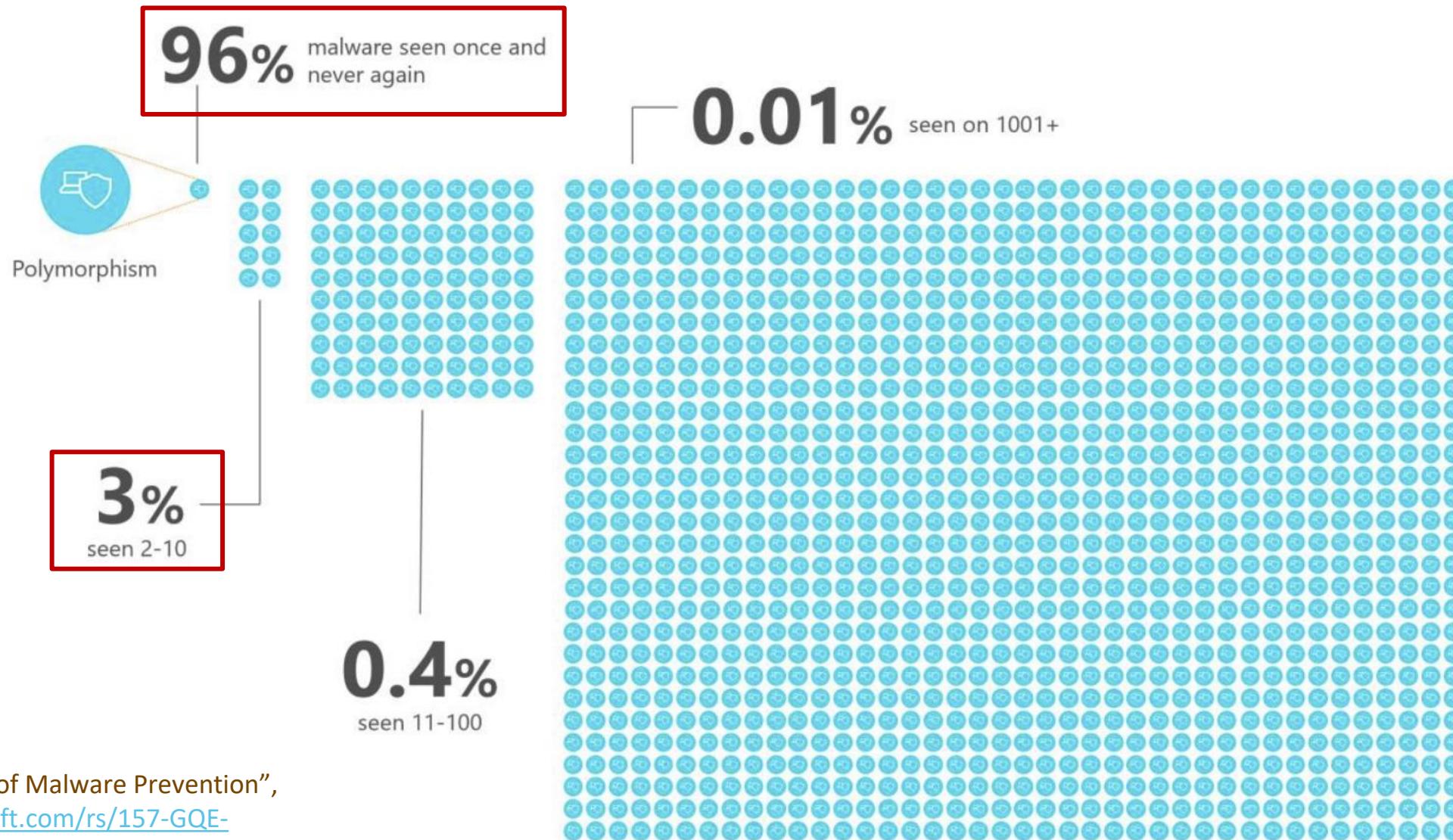
2437
2438 rule XProtect_MACOS_6eaea4b
2439 {
2440     meta:
2441         description = "MACOS.6eaea4b"
2442     condition:
2443         Macho and filesize < 10MB and (
2444             hash.sha1(0, 454544) == "8bda23d6fe3c5f"
2445             hash.sha1(0, 478384) == "eff0e86a0c1fdb"
2446             hash.sha1(0, 888976) == "3bddee4293c423dcf791872e214c364b89df558b"
2447             hash.sha1(0, 465728) == "ab859e350bca96ed8ab4d3ee87ecbdaad42cbf76"
2448             hash.sha1(0, 462816) == "175a12023d4de5"
2449             hash.sha1(0, 465536) == "c91343995496fc20c853d177411338cef954994f"
2450             hash.sha1(0, 465536) == "231b970b66af08780b6fbaf07367d1c8d73df8e"
2451             hash.sha1(0, 888976) == "674493bd15f6fd9476d832d42ffdb00197a05a9"
2452             hash.sha1(0, 482416) == "034b36682182f83390899db99de1c409db2d1"
2453             hash.sha1(0, 482416) == "2fa6c3b9ebcfa66eda50621dbf1bb1e52d3e07"
2454             hash.sha1(0, 465936) == "f058b8f6f82e306ca0f3c43b405536ec9efaf13a"
2455             hash.sha1(0, 482688) == "cadcf5a2618893e6477ddc8162a5c0b971ad7"
2456             hash.sha1(0, 922656) == "b6cd41a0b199a131572e1985805a523aaaf285b5"
2457             (hash.sha1(23728, 267226) == "e15f33dc0ab04e560b25a2548f7698c46d7a64" and hash.sha1(492920, 235168) == "5134edfe096a6ff12f803"
2458             (hash.sha1(21528, 269322) == "82bc3b1a31e95917aa5e0b976b620349e130" and hash.sha1(490900, 237068) == "b0bd8de3512a5924deda465"
2459             hash.sha1(45444, 277834) == "1812f9c0c91ca31fb21549ba3345b36887be8" and hash.sha1(522084, 238548) == "a7634edcd5541fa5f75"
2460             (hash.sha1(45440, 278122) == "572efd09090128cb2b7cf03a1ef95852b8003d61" and hash.sha1(521784, 238816) == "a29914816fa671c5ee1a"
2461             (hash.sha1(45456, 278106) == "01677a4751a3667t0285350a0325405d06df94b" and hash.sha1(521784, 238816) == "a10f16e40f8b8f0a6d1"
2462             (hash.sha1(36704, 270282) == "0212d6febddb9915ef5e48dc8672e8c49efaf695d" and hash.sha1(508868, 235172) == "0cdba314624dd23045bf4"
2463             (hash.sha1(53088, 270282) == "0212d6febddb9915ef5e48dc8672e8c49efaf695d" and hash.sha1(525252, 235172) == "0cdba314624dd23045bf4"
2464             hash.sha1(0, 478384) == "d7d7f9407f678a8ee04ab4b26c0490d4871"
2465             hash.sha1(0, 474224) == "166f3d5b8e9cd7e03bf0a22fb3b365l13d13b1c34"
2466             hash.sha1(0, 458256) == "6bee6b632a140904a648fc9a9dbd14d73dc94c"
2467             hash.sha1(0, 474224) == "09b03db51357d5a67439d10ie81c16374eab4b0"
2468             hash.sha1(0, 888976) == "e386145673963ebfed96e6586106ec00e23607"
2469             hash.sha1(0, 922480) == "b1d322995d77938c508886e59f3a079143cb0f2"
2470             (hash.sha1(21618, 264618) == "25ce0a0b70630409c7439ad0832d141a999c" and hash.sha1(495056, 232672) == "bdc7c63c903907d737c"
2471             (hash.sha1(27404, 260608) == "e52e255f91c3fe140791fa142bf0ff2c1209657c9" and hash.sha1(493628, 234068) == "aa386881e0f0c210e8c"
2472             (hash.sha1(474298, 275578) == "e2936b6f699f51c70891216574bf83757f5f6cc" and hash.sha1(524636, 235708) == "f87ee1e0488bf3c64ab"
2473             (hash.sha1(47616, 275866) == "e2d9be92730b6e093793d5270b21ab7151ba3504" and hash.sha1(524348, 235988) == "d43f3b4125270b21ab7151ba3504"
2474             (hash.sha1(47632, 275850) == "5fb180aeffd1a3c7163b3d7facb19e3b526" and hash.sha1(524352, 235984) == "0a59d04c27ca3a7617e13"
2475             (hash.sha1(22208, 268314) == "757f1e6b991c2e91f5b9fcebacf35edfccc9c315" and hash.sha1(493612, 232732) == "639f0fa300ed26656ac2"
2476             (hash.sha1(22208, 268314) == "757f1e6b991c2e91f5b9fcebacf35edfccc9c315" and hash.sha1(493612, 232732) == "639f0fa300ed26656ac2"
2477             hash.sha1(0, 454448) == "e4b48e2224486b5b7a3f3a81fb5d4dd163b0b0"
2478             hash.sha1(0, 474144) == "5c448f6272d63a57c0e7965d99d93e23a15e80"
2479             hash.sha1(0, 888964) == "a15f39c5c0507e25e742d071d15c8e38658165e5a"
2480             hash.sha1(0, 922384) == "cbf08fae71fc46c852fad7502685466c40e168"
2481             (hash.sha1(25576, 264906) == "8e71e1581ce30793a6393b7e3045494753" and hash.sha1(494844, 232940) == "dddb9c37fa949c7e17"
2482             (hash.sha1(24128, 266330) == "d4b503675bb8d91f6e70d76917324ada1300ff" and hash.sha1(493444, 232484) == "d8fae0947ab4d51278bc"
2483             (hash.sha1(46928, 276282) == "51b6595a807c480d304c6148ecc605457b7ad" and hash.sha1(524176, 236216) == "8586c6458530652b575"
2484             (hash.sha1(46624, 276570) == "326e5a9e393ced730e548a515b231" and hash.sha1(523872, 236496) == "55efafedf76814a75f1e"
2485             (hash.sha1(46656, 276554) == "77b726c10e6d3814567940be3c76b5963ff9625" and hash.sha1(523876, 236492) == "4420f8ed63ed01f9df20"
2486             (hash.sha1(37392, 269162) == "77756fb4720bc7ea364a947df659495a473f15d" and hash.sha1(509160, 234424) == "dfdab2704a010782a4b0c"
2487             (hash.sha1(53776, 269162) == "55efafedf76814a75f1e" and hash.sha1(525544, 234424) == "dfdab2704a010782a4b0c"
2488             hash.sha1(0, 458112) == "21b63689d192a7d1309d98fa435d42f695987d" or
2489             hash.sha1(0, 474048) == "509db1a168fdeecf990784741e14cb17b2a31e" or

```

[Source: <https://github.com/SentinelLabs/XProtect-Malware-Families/blob/779534f1c966611e07d9d993906ffee3c57b5974/2192.yara#L2438>]



# Malware Is Too Variable For Manual Human Labor



[Source: "Evolution of Malware Prevention",  
<https://info.microsoft.com/rs/157-GQE-382/images/Windows%2010%20Security%20Whitepaper.pdf>]

# Static Malware Analysis With Machine Learning

Part 1:  
Feature Engineering

# Windows PE File Format

Format adapted for “modern” programs  
(from Windows NT 3.1 on)

Before there were other formats, one is the DOS  
(kept for retro compatibility)

## PE Header

Real metadata of the program  
Describes general information of the file

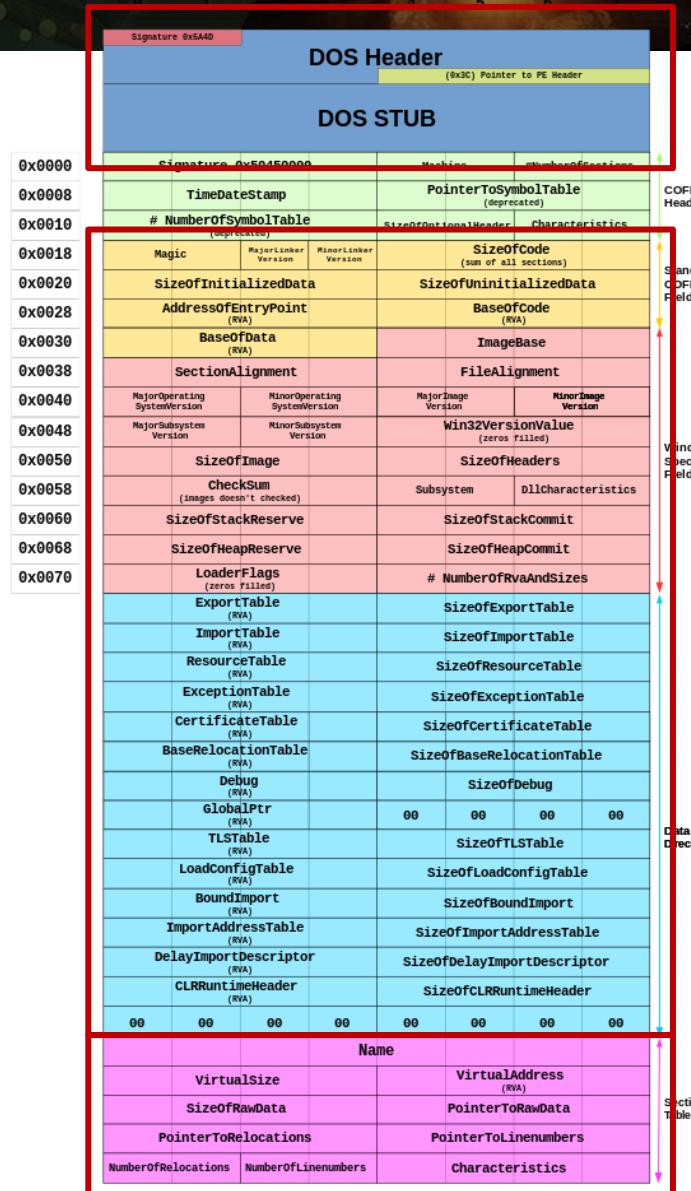
## Optional Header

Spoiler: not optional at all :)  
Instructs the loader where to find each object inside the file

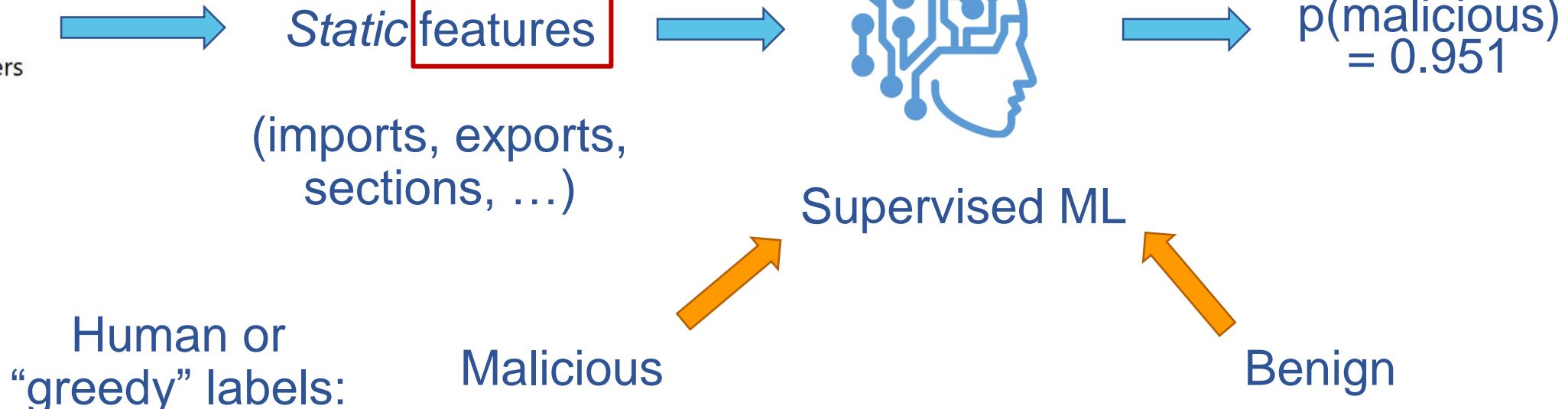
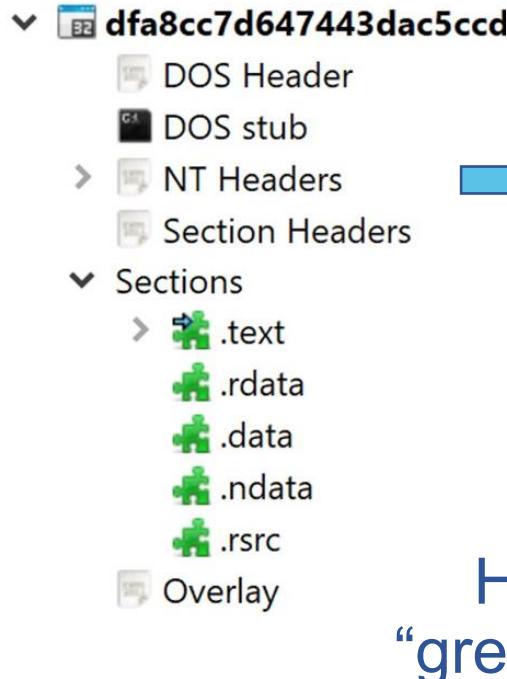
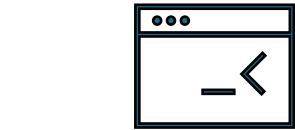
## Section Table and Sections

Describes where to find code, initialized data, resources, etc. to the loader

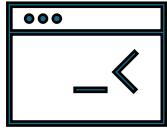
Examples: code is “.text”, read-only data is “.rodata”, resources are “.rsc”, and counting



# Static Malware Classification with ML Models



# Static Malware Classification with ML Models



```
    71   class ByteEntropyHistogram(FeatureType):  
    72       ''' 2d byte/entropy histogram based loosely on (Saxe and Berlin, 2015).  
    73       This roughly approximates the joint probability of byte value and local entropy.  
    74   125   class SectionInfo(FeatureType):  
    75       126       ''' Information about section names, sizes and entropy. Uses hashing trick  
    76       127       to summarize all this section info into a feature vector.  
    77       128       ...  
    78   195   class ImportsInfo(FeatureType):  
    79       196       ''' Information about imported libraries and functions from the  
    80       197       import address table. Note that the total number of imported  
    81       198       functions is contained in GeneralFileInfo.  
    82       199       ...
```

## Lab 1:

# Static Malware Detection using ML with Gradient Boosted Decision Trees (GBDT)

[aka.ms/malware-lab1](http://aka.ms/malware-lab1)

# Static Malware Analysis With Machine Learning

Part 2:  
Byte Level Analysis with Neural Networks

# Development of AI and Neural Nets



## Pre- “Deep Learning”: Feature Engineering Era

### Computer Vision (CV):

- SIFT
- SURF



# Development of AI and Neural Nets



## Pre- “Deep Learning”: Feature Engineering Era

### Computer Vision (CV):

- SIFT
- SURF

### Natural Language Processing (NLP):

- HMM & POS tags
- CRFs & NER tags



Back in 2000 , People Magazine PUBLISHER highlighted Prince Williams' PERSON style who at the time was a little more fashion-conscious , even making fashion statements at times .

Now-a-days the prince mainly wears navy COLOR suits ITEM ( sometimes double-breasted DESIGN ), light blue COLOR button-ups ITEM with classic LOOK pointed DESIGN collars PART , and burgundy COLOR ties ITEM .

But who knows what the future holds ...

Duchess Kate PERSON did wear an Alexander McQueen BRAND dress ITEM to the wedding OCCASION in the fall of 2017 SEASON .

# Development of AI and Neural Nets

1989: CNNs  
(LeCun et al.)

1999: SIFT

2006: SURF

2000: HMM for POS  
2001: CRF for NER

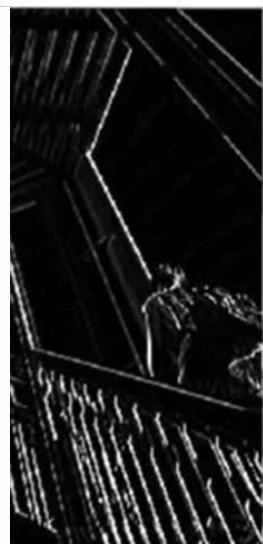
2010-2017:  
CNNs and RNNs dominate CV and NLP

2010s Deep Learning allows to scale up -- Neural Nets “learn” low level features

Computer Vision (CV):  
• Convolutional  
Neural Networks  
(CNNs)



this →	0.2	0.4	-0.3
movie →	0.1	0.2	0.6
has →	-0.1	0.4	-0.1
amazing →	0.7	-0.5	0.4
diverse →	0.1	-0.2	0.1
characters →	0.6	-0.3	0.8



# Development of AI and Neural Nets

1989: CNNs  
(LeCun et al.)



1997: LSTM  
(Hochreiter and Schmidhuber)

1999: SIFT

2000: HMM for POS  
2001: CRF for NER

2006: SURF

2010-2017:  
CNNs and RNNs dominate CV and NLP

**2010s Deep Learning allows to scale up -- Neural Nets “learn” low level features**

Computer Vision (CV):

- Convolutional Neural Networks (CNNs)

NLP:

- Recurrent Neural Networks (RNNs)



# CNNs for Malware Detection

**1989: CNNs**  
(LeCun et al.)

**1999: SIFT**

**2006: SURF**

**2017: CNN for Malware (MalConv)**



**1997: LSTM**  
(Hochreiter and Schmidhuber)

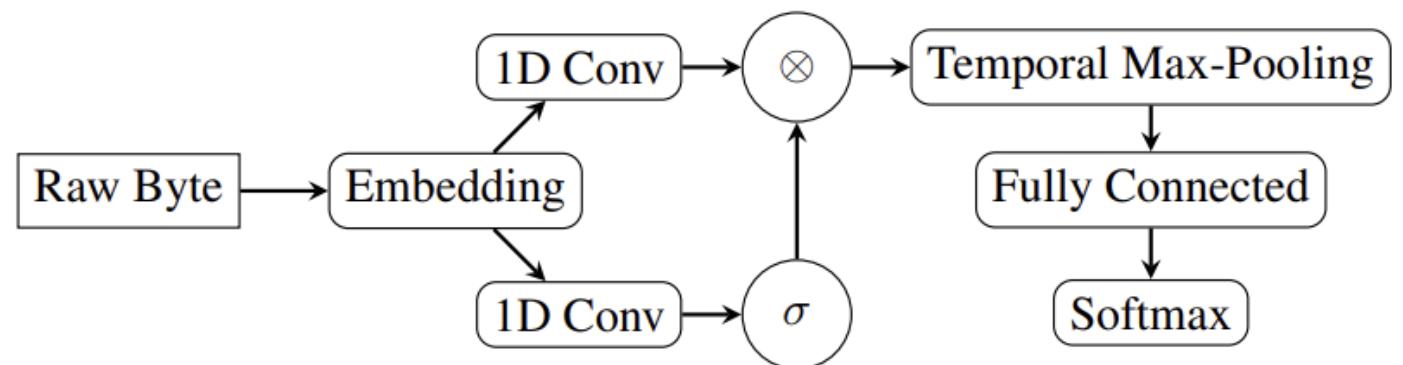
**2000: HMM for POS**  
**2001: CRF for NER**

**2010-2017:**  
**CNNs and RNNs dominate CV and NLP**

## Malware Detection by Eating a Whole EXE

**Edward Raff<sup>1,3,4</sup>, Jon Barker<sup>2</sup>, Jared Sylvester<sup>1,3</sup> Robert Brandon<sup>1,3,4</sup>**  
**Bryan Catanzaro<sup>2</sup>, Charles N**

<sup>1</sup>Laboratory for Physical Sciences, <sup>2</sup>NVIDIA, <sup>3</sup>Booz Allen Hamilton,  
{edraff,jared,rbrandon}@lps.umd.edu, {jbarker,bcatanzaro}@nvidia.com



*Figure 1.* Architecture diagram of MalConv model.

[Source: <https://arxiv.org/pdf/1710.09435>]

## Lab 2:

# Feature Extraction from Byte Level Data with Neural Networks and PyTorch

[aka.ms/malware-lab2](http://aka.ms/malware-lab2)

# Dynamic Malware Analysis

# Behavioral Properties

dfa8cc7d647443dac5ccd.
DOS Header
DOS stub
NT Headers
Section Headers
Sections
.text
.rdata
.data
.ndata
.rsrc
Overlay



Summary | 103 calls | 46 KB used | regsvr32.exe

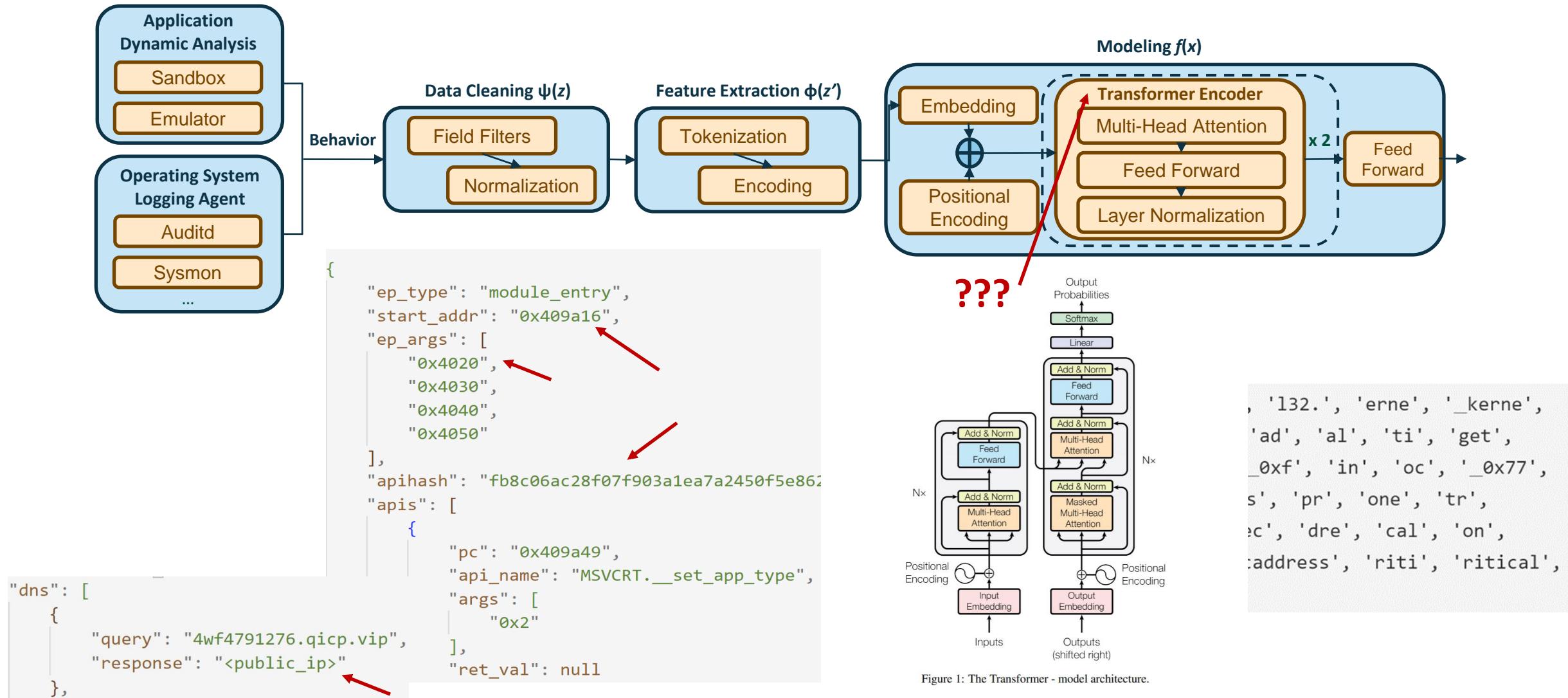
Process Hacker  
Hacker View Tool  
Refresh Opt  
Processes Services

Module	API	Return Value
KERNELBASE.dll	LdrLoadDll ( 1, 0x04daf790, 0x04daf7a0, 0x04daf794 )	STATUS_SUCCESS
regsvr32.exe	GetProcAddress ( 0x77310000, "ZwQuerySystemInformation" )	0x7737eae0
KERNELBASE.dll	LdrLoadDll ( 1, 0x04daf790, 0x04daf7a0, 0x04daf794 )	STATUS_SUCCESS
regsvr32.exe	GetProcAddress ( 0x77310000, "ZwQuerySystemInformation" )	0x7737eae0
KERNELBASE.dll	LdrLoadDll ( 1, 0x04daf790, 0x04daf7a0, 0x04daf794 )	STATUS_SUCCESS
regsvr32.exe	GetProcAddress ( 0x77310000, "ZwQuerySystemInformation" )	0x7737eae0
KERNELBASE.dll	LdrLoadDll ( 1, 0x04daf790, 0x04daf7a0, 0x04daf794 )	STATUS_SUCCESS
regsvr32.exe	GetProcAddress ( 0x77310000, "ZwQuerySystemInformation" )	0x7737eae0
explorer.exe	2684 ASLR Medium	Medium
Procmam.exe	2072 ASLR Medium	Medium
cmd.exe	6568 ASLR Medium	Medium
conhost.exe	6412 ASLR Medium	Medium
powershell.exe	5792 ASLR Medium	Medium
PDFXCview.exe	920 Medium	Medium

Process  
Command line: "C:\ProgramData\PDFXCview.exe"  
Current directory: C:\ProgramData\  
Started: a minute and 32 seconds ago (8:18:57 AM)  
PEB address: 0x214000 (32-bit: 0x215000)

Process Name	RID	Operation	Path
PDFXCview.exe	920	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\BAM
PDFXCview.exe	920	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager\BAM
PDFXCview.exe	920	Process Create	C:\WINDOWS\SysWOW64\regsvr32.exe
PDFXCview.exe	920	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlIs
PDFXCview.exe	920	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlIs

# Machine Learning Pipeline for Dynamic Analysis: Nebula



# CNNs for Malware Detection

1989: CNNs

(LeCun et al.)



1999: SIFT



2006: SURF



2017:

"Attention is All You Need"



1997: LSTM  
(Hochreiter and Schmidhuber)

2000: HMM for POS  
2001: CRF for NER

2010-2017:  
CNNs and RNNs dominate  
CV and NLP

Google introduces the  
"Transformer" model

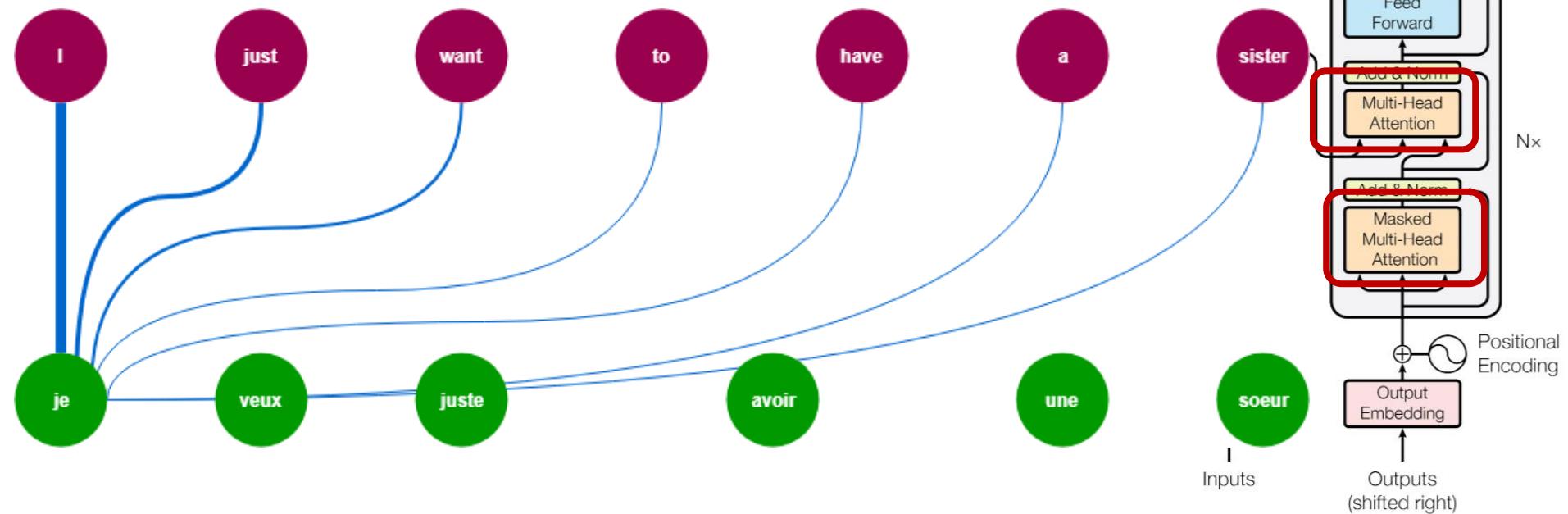
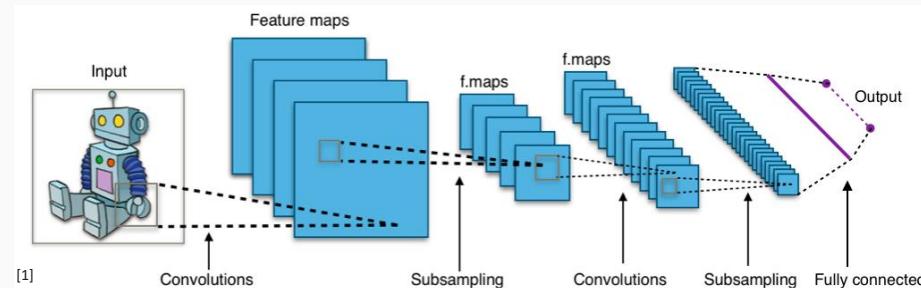


Figure 1: The Transformer - model architecture.

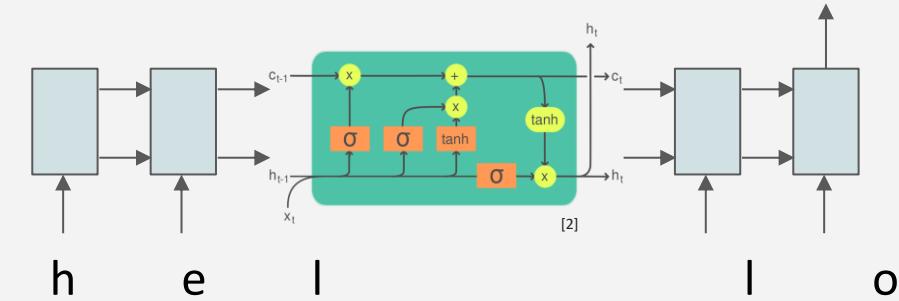
## Computer Vision

### Convolutional NNs (+ResNets)



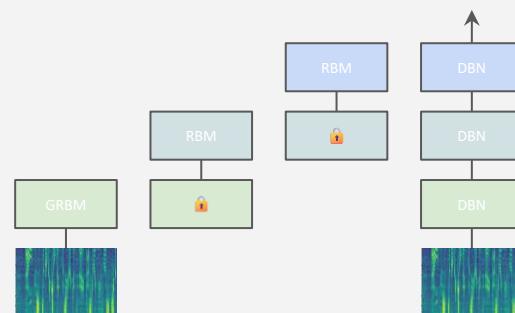
## Natural Lang. Proc.

### Recurrent NNs (+LSTMs)



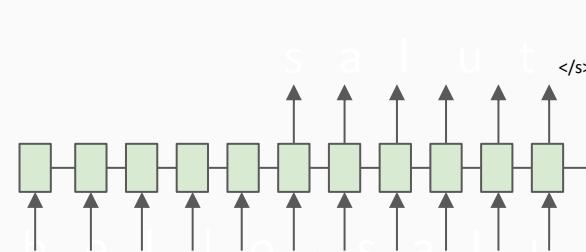
## Speech

### Deep Belief Nets (+non-DL)



## Translation

### Seq2Seq



[1] CNN image CC-BY-SA by Aphex34 for Wikipedia [https://commons.wikimedia.org/wiki/File:Typical\\_cnn.png](https://commons.wikimedia.org/wiki/File:Typical_cnn.png)  
[2] RNN image CC-BY-SA by GChe for Wikipedia [https://commons.wikimedia.org/wiki/File:The\\_LSTM\\_Cell.svg](https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg)

## RL

### BC/GAIL

#### Algorithm 1 Generative adversarial imitation learning

- 1: **Input:** Expert trajectories  $\tau_E \sim \pi_E$ , initial policy and discriminator parameters  $\theta_0, w_0$
  - 2: **for**  $i = 0, 1, 2, \dots$  **do**
  - 3:     Sample trajectories  $\tau_i \sim \pi_{\theta_i}$
  - 4:     Update the discriminator parameters from  $w_i$  to  $w_{i+1}$  with the gradient
- $$\hat{\mathbb{E}}_{\tau_i} [\nabla_w \log(D_w(s, a))] + \hat{\mathbb{E}}_{\tau_E} [\nabla_w \log(1 - D_w(s, a))] \quad (17)$$

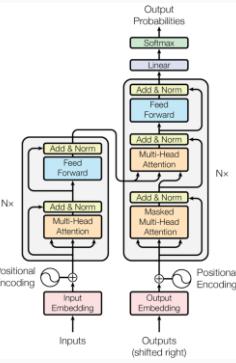
- 5:     Take a policy step from  $\theta_i$  to  $\theta_{i+1}$ , using the TRPO rule with cost function  $\log(D_{w_{i+1}}(s, a))$ . Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_\theta \log \pi_\theta(a|s) Q(s, a)] - \lambda \nabla_\theta H(\pi_\theta), \\ \text{where } Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i} [\log(D_{w_{i+1}}(s, a)) | s_0 = \bar{s}, a_0 = \bar{a}] \quad (18)$$

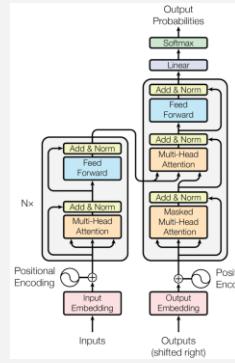
- 6: **end for**

[Source: Lucas Beyer, "Transformers."]

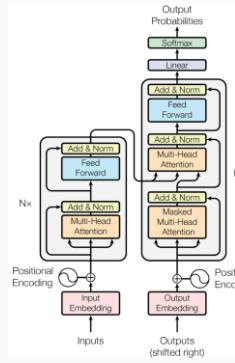
## Computer Vision



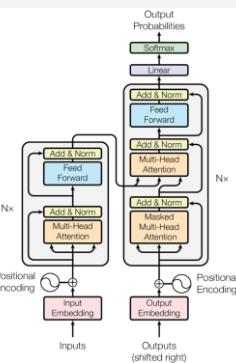
## Natural Lang. Proc.



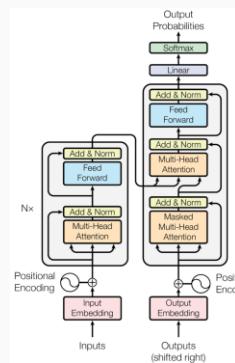
## Reinf. Learning



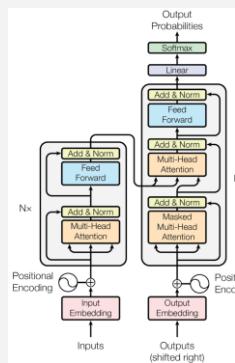
## Speech



## Translation



## Graphs/Science



## Lab 3:

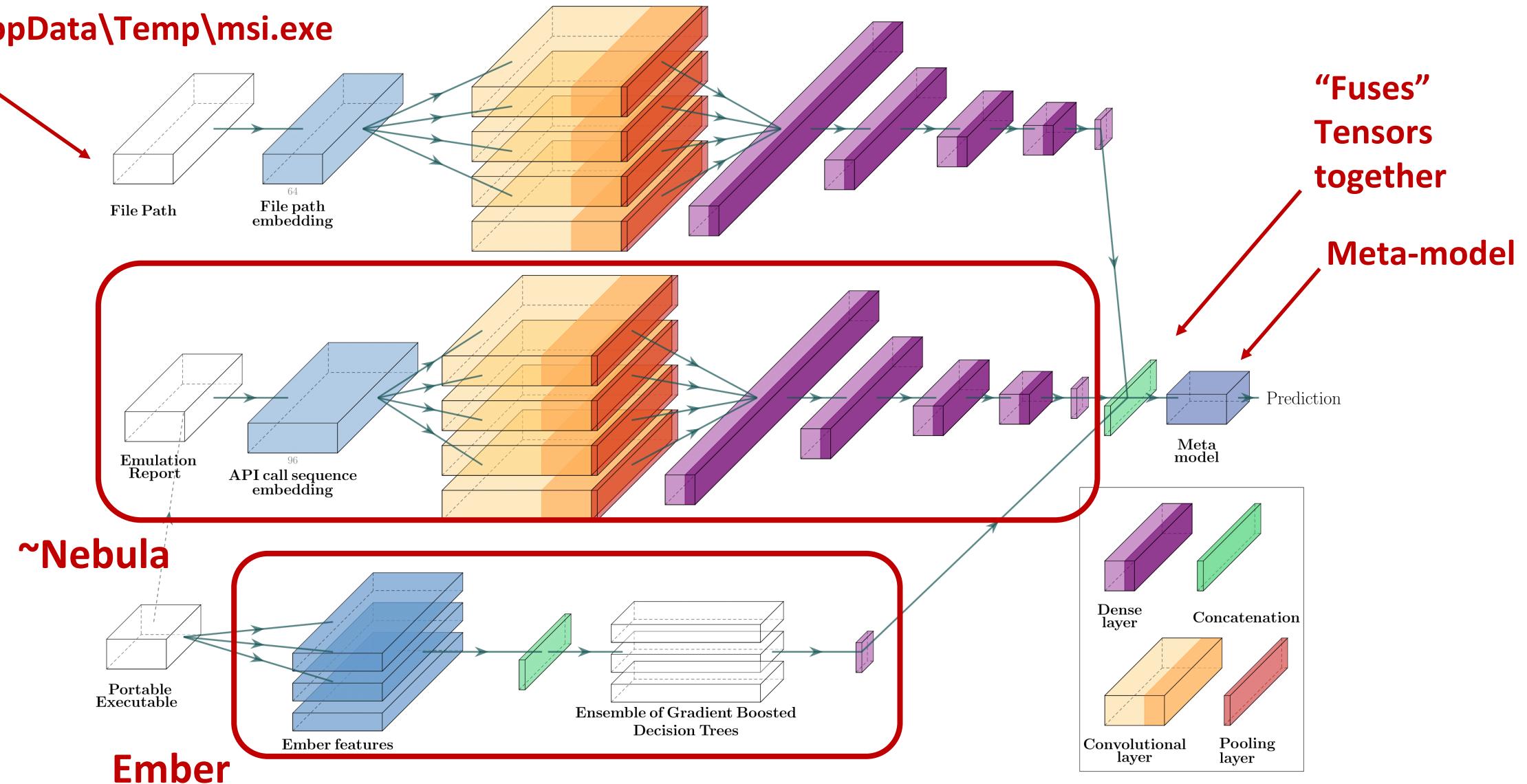
# Dynamic Malware Analysis using Transformer Neural Network

[aka.ms/malware-lab3](http://aka.ms/malware-lab3)

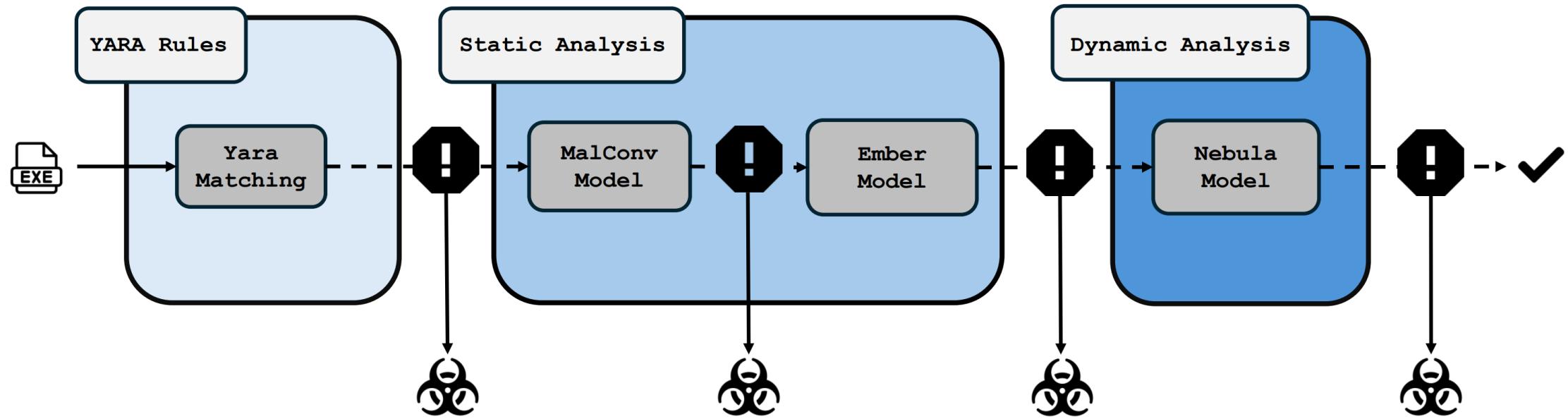
# Hybrid Malware Analysis

# End-to-End Monolithic Model

C:\...\AppData\Temp\msi.exe

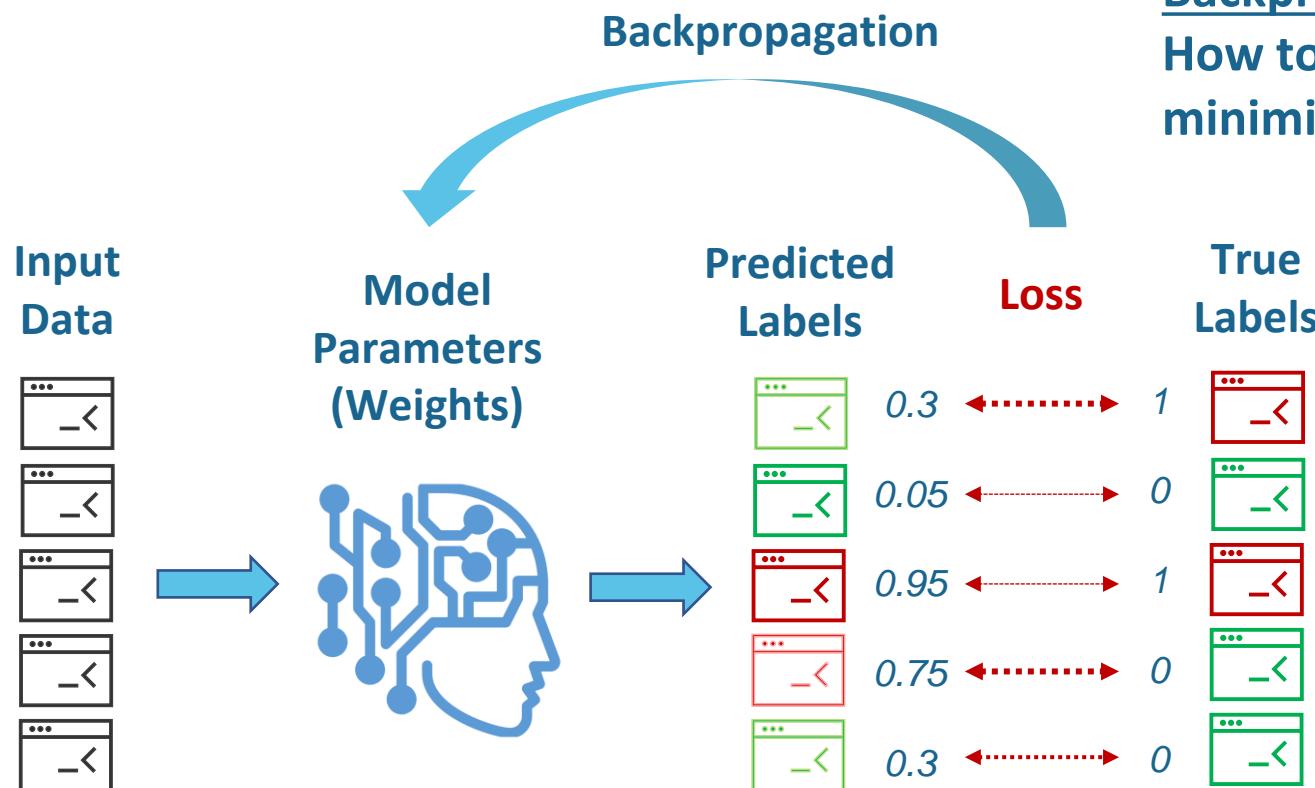


# Modular Sequential Processing



# Training Machine Learning Model

# Optimization: Loss

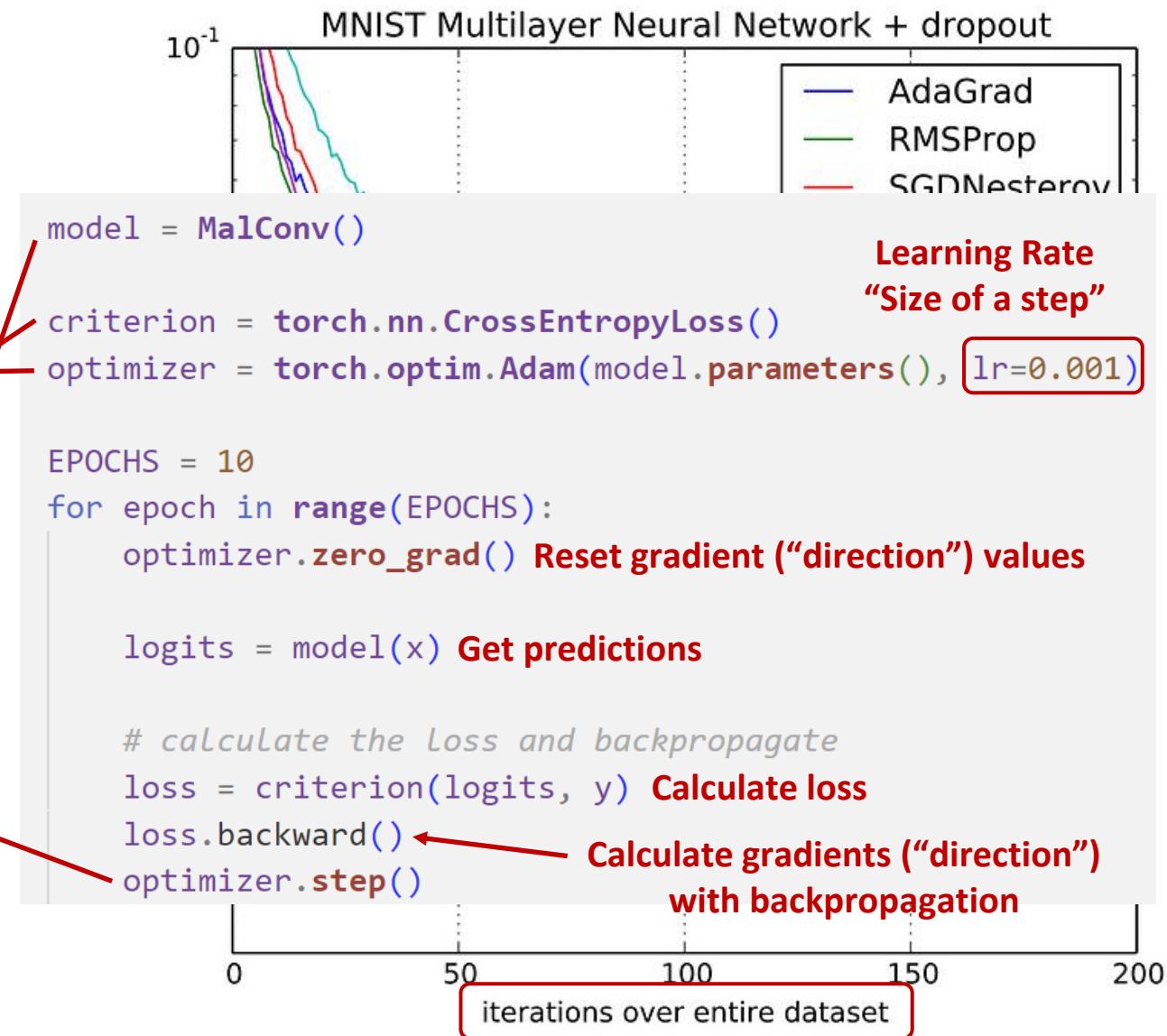
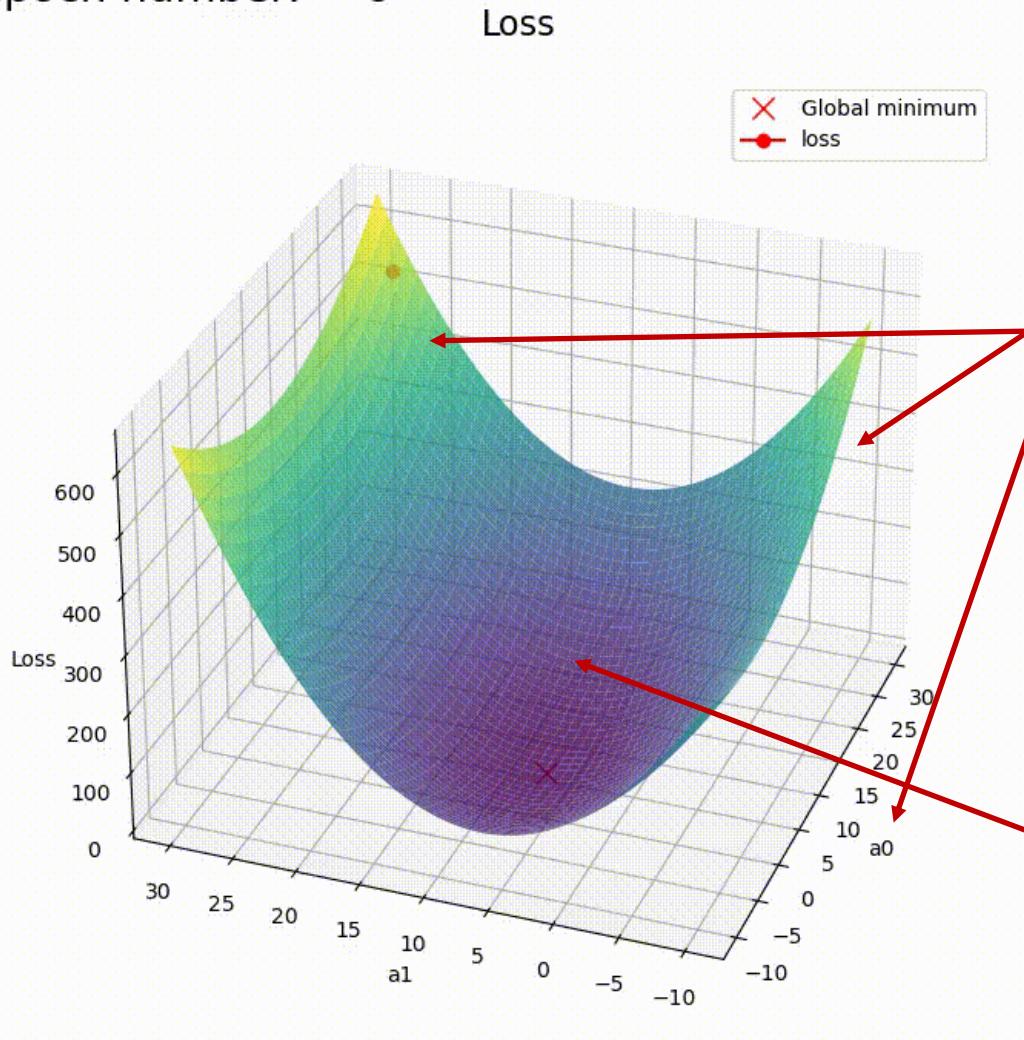


**Backprop Intuition:**  
How to update model parameters to minimize loss during next pass?

```
criterion = torch.nn.CrossEntropyLoss()  
  
preds = torch.Tensor([0.3, 0.05, 0.95, 0.75, 0.3])  
labels = torch.Tensor([1, 0, 1, 0, 0])  
  
loss = criterion(preds, labels)  
loss.item()  
✓ 0.0s  
3.0192079544067383
```

# Optimization: Gradient Descend

Batch Gradient Descent  
epoch number: = 0



# Misuse this process? Sure!

## Gradient -free and -full Attacks!

### Part 2: Attacking AI-Based Malware Detector

- Intro to Adversarial Machine Learning
- Adversarial EXEmple
- Defenses against Adversarial Manipulations

# Part 1 Survey

[aka.ms/malware-survey1](http://aka.ms/malware-survey1)