

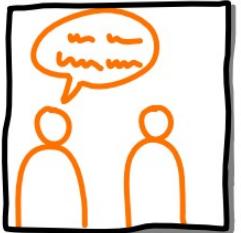
ABOUT ME



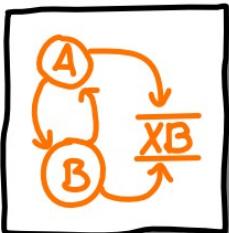
@ETZOLDIO

- IT SECURITY ARCHITECT

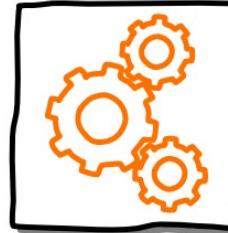
- 1&1 MAIL & MEDIA DEVELOPMENT & TECHNOLOGY GMBH



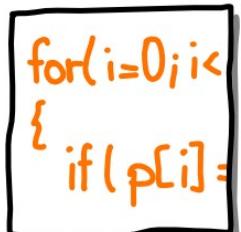
IN HOUSE
CONSULTING



THREAT
MODELING



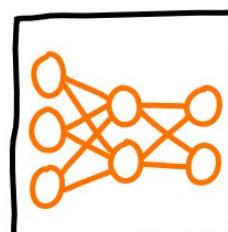
SECURE SOFTWARE
DEVELOPMENT
LIFECYCLE



CODE
REVIEW



PENETRATION
TESTING



MACHINE LEARNING &
SECURITY

DATA BREACHES



COMPANIES ARE
HACKED REGULARLY

$>10^8$ RECORDS PER BREACH
QUITE COMMON

Entity	Year	Records	Organization type	Method	Sources
Betsson Group	2020	unknown	gaming	unknown	[49]
CheckPeople	2020	56,000,000	background check	unknown	[74]
Clearview AI	2020	3,000,000,000 (Number of photos obtained)	information technology	hacked	[81]
Instagram	2020	200,000,000	social network	poor security	[181]
JailCore	2020	36,000	government	poor security	[186]
Koodo Mobile	2020	unknown	mobile carrier	hacked	[194]
Marriott International	2020	5,200,000	hotel	poor security/inside job	[211]
Nintendo (Nintendo Account)	2020	160,000	gaming	hacked	[238]
SlickWraps	2020	377,428	phone accessories	poor security	[268]
Tetrad	2020	120,000,000	market analysis	poor security	[301]
TikTok	2020	42,000,000	social media	poor security	[304]
Virgin Media	2020	900,000	mobile carrier	accidentally exposed	[338][339]
Wawa (company)	2020	30,000,000	retail	hacked	[345]
YouTube	2020	4,000,000	social media	poor security	[364]
Unknown agency (believed to be tied to United States Census Bureau)	2020	200,000,000	financial	accidentally published	[370]



FINANCIAL LOSS



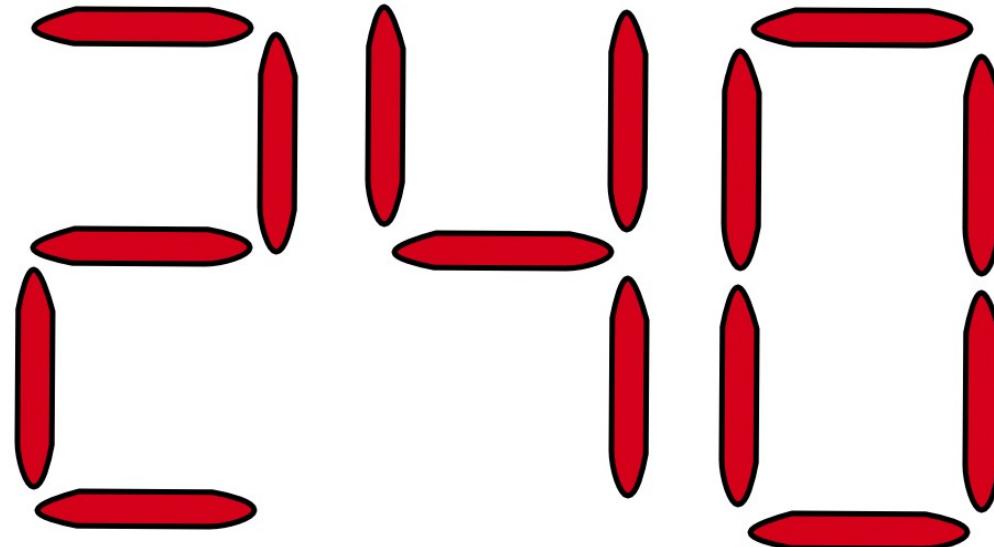
CUSTOMERS
FEAR MISUSAGE

STATUS QUO



MOST COMPANIES
ARE BAD AT DETECTING
INTRUSIONS

ON AVERAGE IT TAKES



DAYS UNTIL AN INTRUSION HAS BEEN DISCOVERED

AGENDA



ATTACK VECTORS



VULNERABILITY
PREVENTION
AND INTRUSION
DETECTION

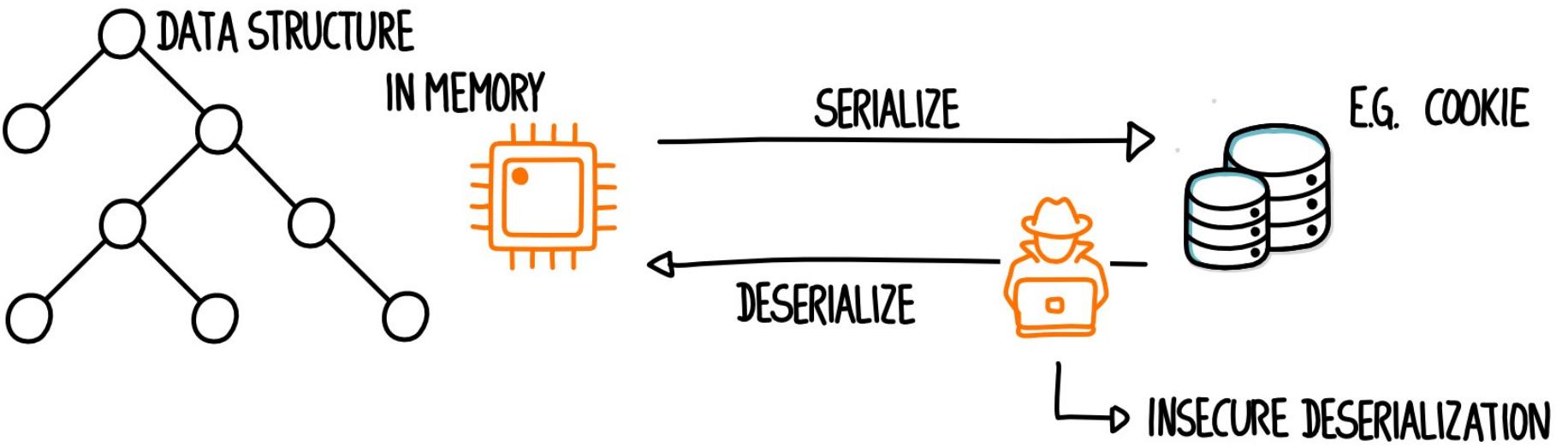


BASICS OF
MACHINE
LEARNING

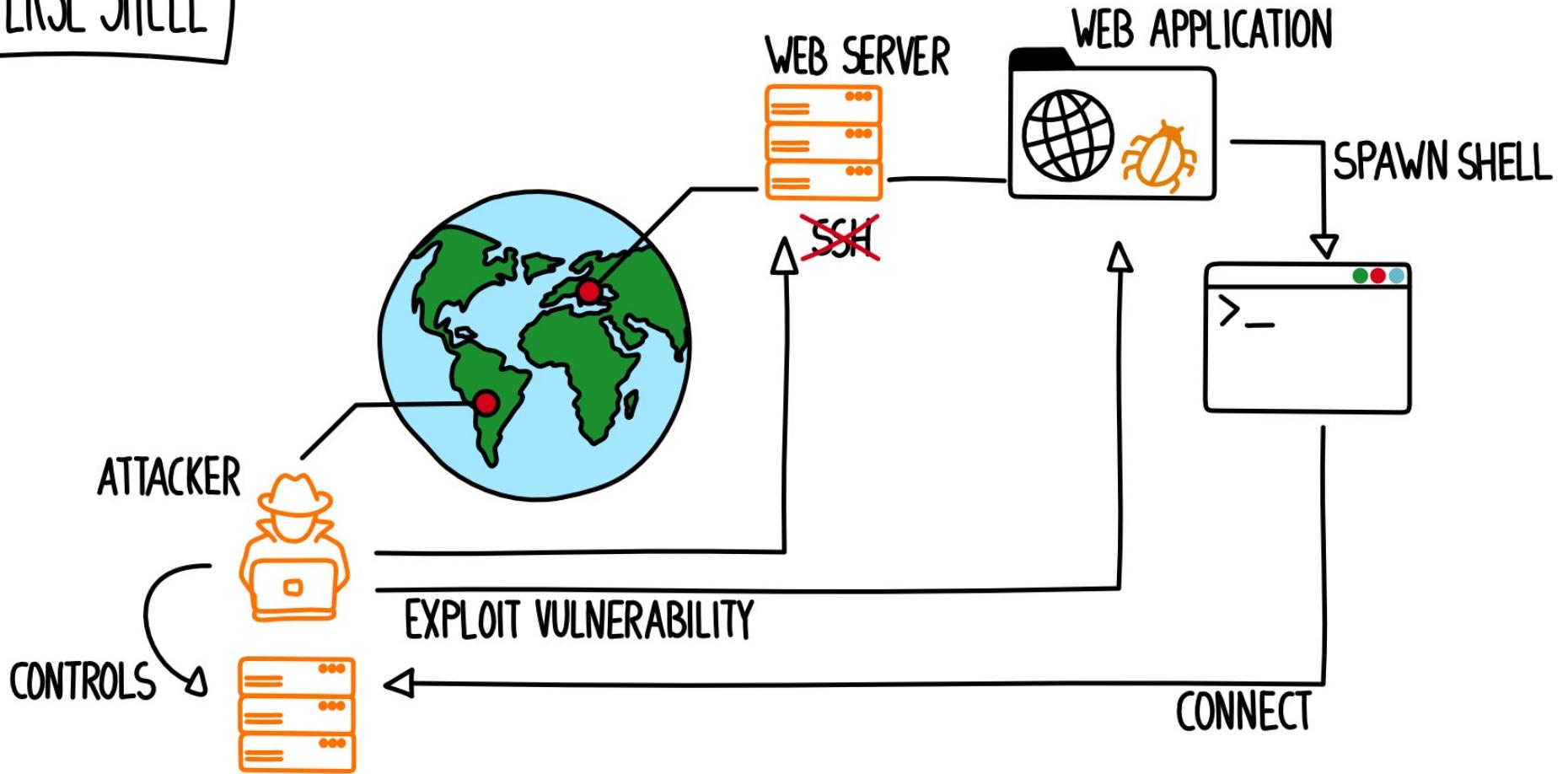


EXAMPLES:
SYSTEMS CALLS
LOG FILE VISUALIZATION
AUTOENCODER

EXAMPLE

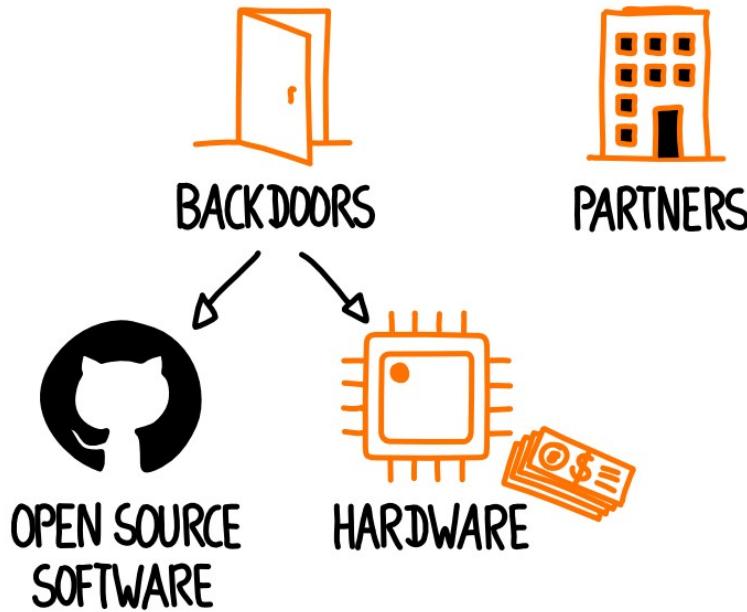
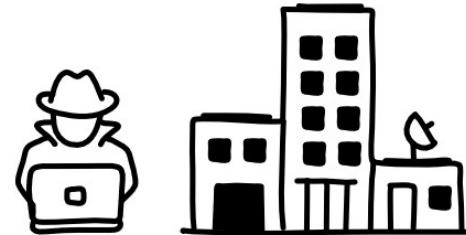


REVERSE SHELL



ATTACK VECTORS

HOW ATTACKERS GET ACCESS
TO A COMPANY NETWORK

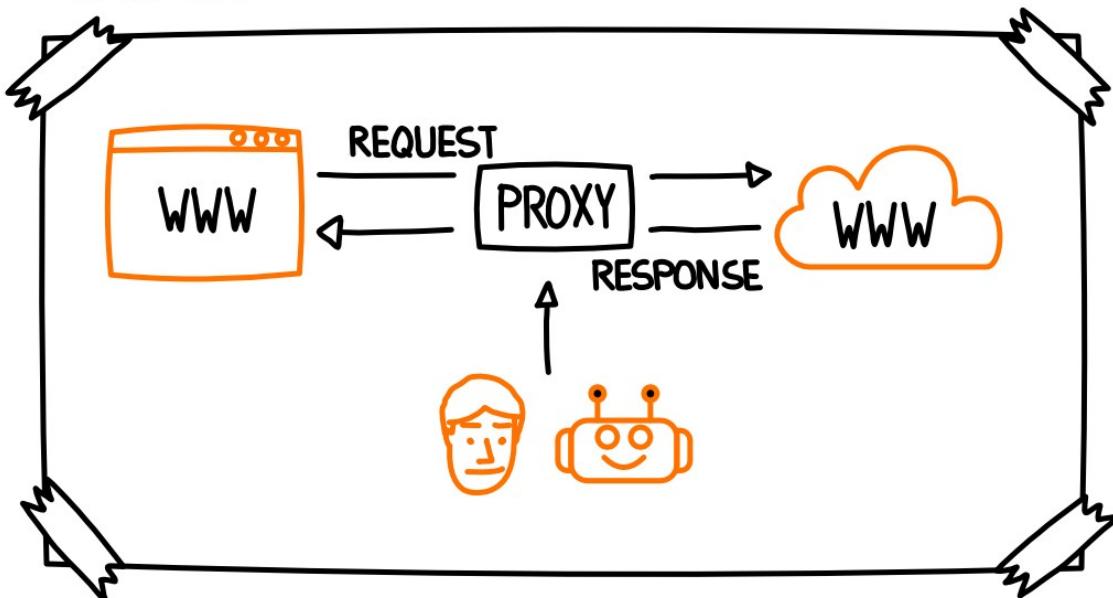


VULNERABILITIES
IN SOFTWARE

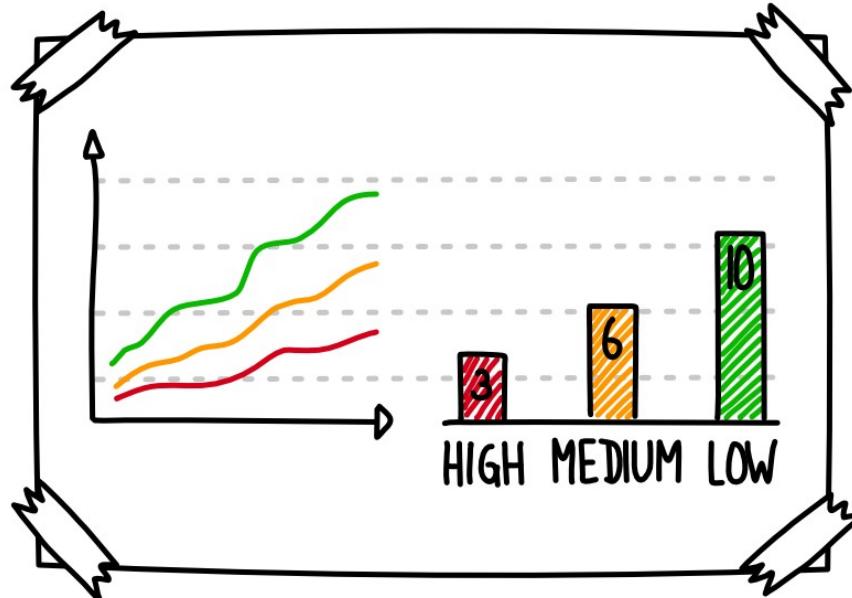


OPEN WEB APPLICATION SECURITY PROJECT

OWASP ZAP



OWASP DEPENDENCY TRACK



OWASP TOP 10



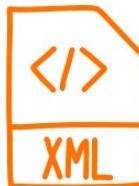
INJECTIONS



BROKEN
AUTHENTICATION



SENSITIVE DATA
EXPOSURE



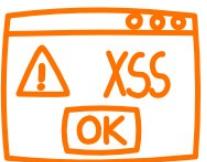
XML EXTERNAL
ENTITIES



BROKEN
ACCESS CONTROL



SECURITY
MISCONFIGURATION



CROSS SITE
SCRIPTING



INSECURE
DESERIALIZATION



COMPONENTS
WITH KNOWN
VULNERABILITIES



INSUFFICIENT LOGGING
AND MONITORING

DETECT SQL INJECTIONS



WEB APPLICATION
WITH LOGIN

VERIFIES
CREDENTIALS
VIA

WARNING

NEVER DO THIS

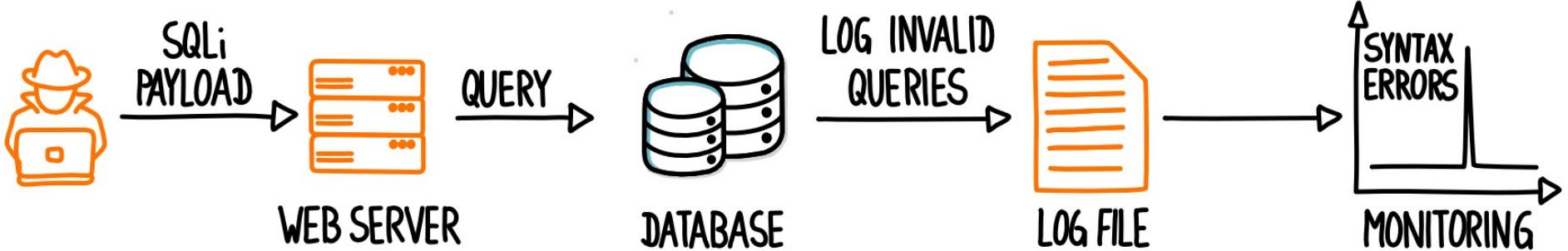
SELECT userid FROM users WHERE email='foo@example.com' AND pwd='password';

' OR 1=1;--

SELECT userid FROM users WHERE email=' OR 1=1;-- AND pwd='password';

DETECT SQL INJECTIONS

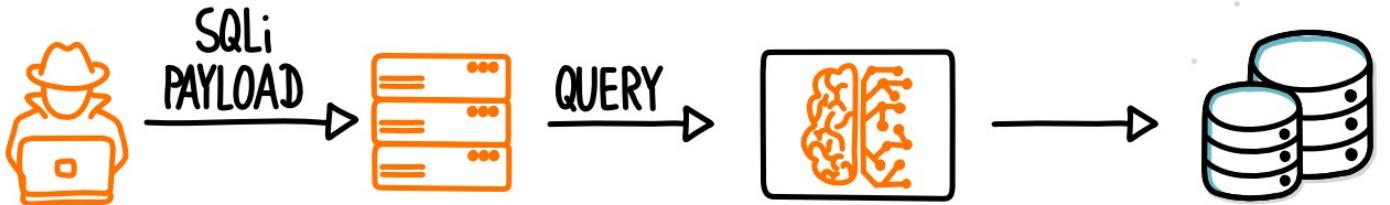
WITH LOGGING



DRAWBACK: PASSIVE APPROACH — SOLUTION → DEFENSE IN DEPTH

DETECT SQL INJECTIONS

WITH A SIMPLE LEARNING APPROACH



WEB SERVER

QUERY ANALYZING
COMPONENT

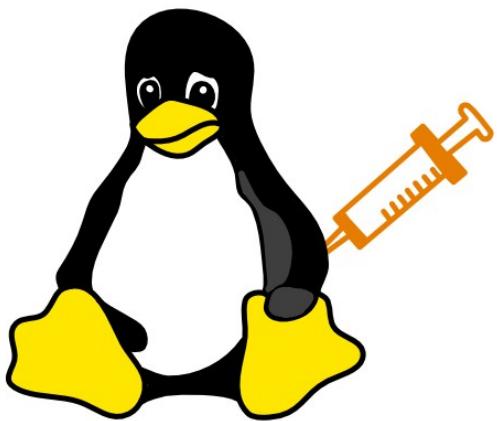
DATABASE

ADVANTAGE: CAN ACTIVELY BLOCK QUERIES

PHASE 1: BUILD MODEL

PHASE 2: USE MODEL TO CLASSIFY QUERIES

OS COMMAND INJECTION



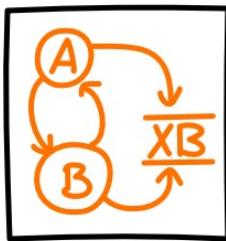
HTTPS://FOO.COM? ID=123

OS.SYSTEM("SCAN.SH "+\$REQUEST["ID"])

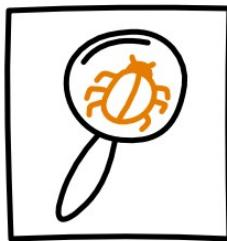
VULNERABILITY PREVENTION



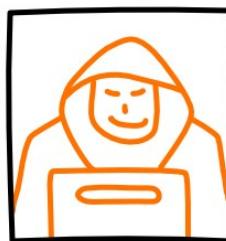
AWARENESS



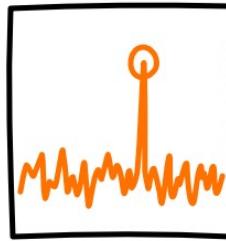
THREAT MODELING
(E.G. STRIDE)



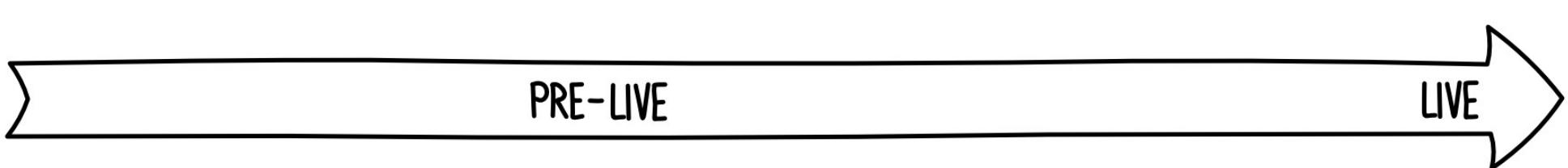
STATIC / DYNAMIC
APPLICATION
SECURITY TESTING



PENETRATION
TESTING



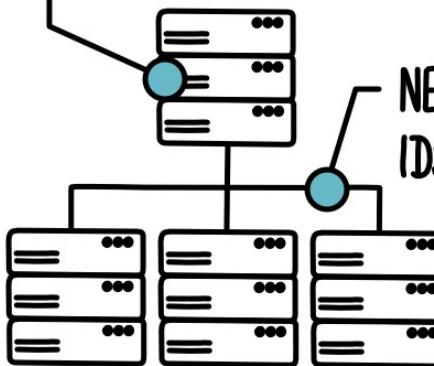
MONITORING



INTRUSION DETECTION SYSTEMS

VARIANTS

HOST-BASED IDS



TECHNOLOGIES USED FOR IDS

RULES/SIGNATURES

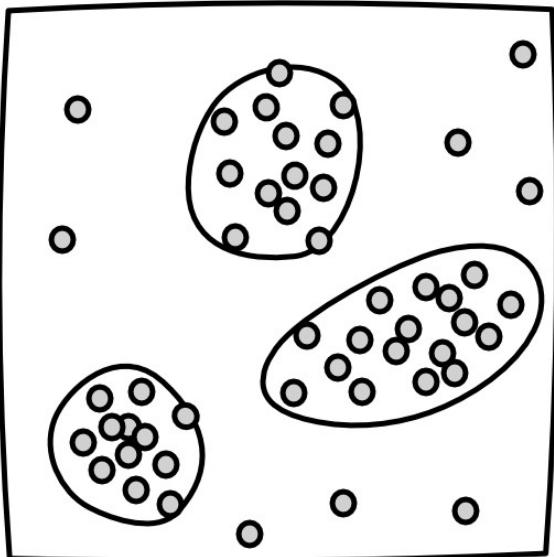
- + FAST
- + LOW FALSE POSITIVES
- CANNOT DETECT UNKNOWN ATTACKS
- NEED REGULAR UPDATES

MACHINE LEARNING

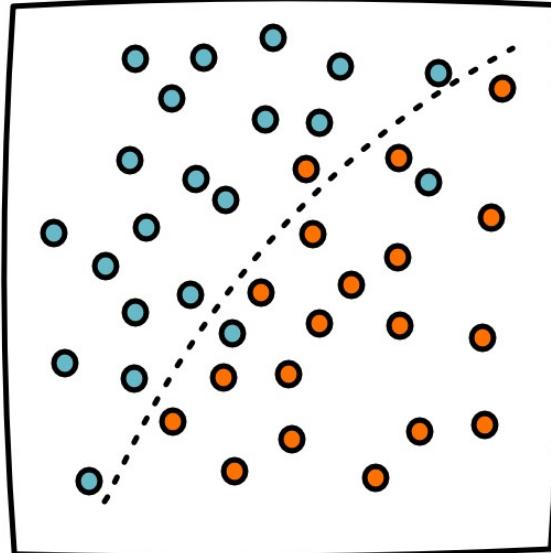
- + CAN DETECT UNKNOWN ATTACKS
- HAVE FALSE POSITIVES
- NEED RETRAINING
- BLACK BOX

MACHINE LEARNING

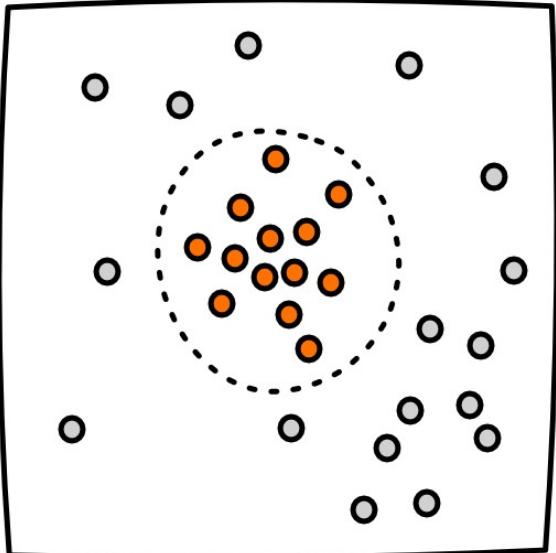
UNSUPERVISED LEARNING



SUPERVISED LEARNING



SEMI-SUPERVISED LEARNING



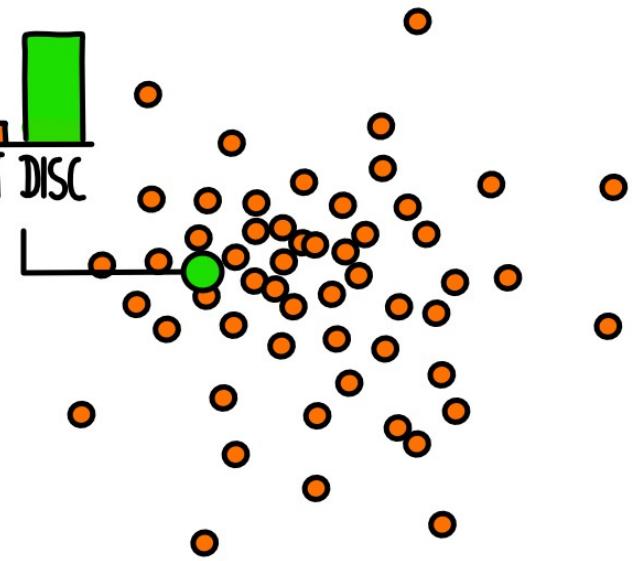
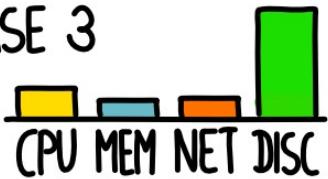
○ LABELS UNKNOWN

● ● LABELS KNOWN

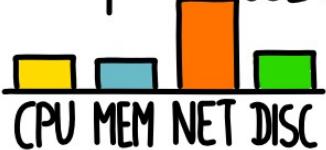
UNSUPERVISED LEARNING

CLUSTERING = GROUP SIMILAR DATA

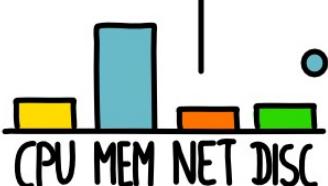
USE CASE 3



USE CASE 2



USE CASE 1



ATTACK

MAY RESULT IN DEVIATIONS IN:
CPU USAGE DISC USAGE
MEMORY USAGE NETWORK USAGE

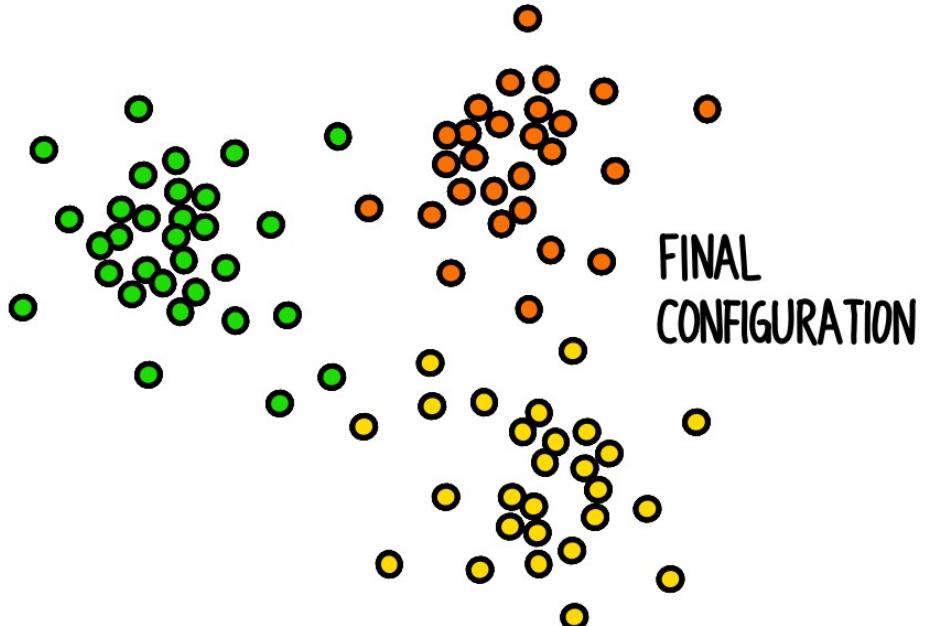
K-MEANS

STEPS

- ① INITIALIZE CLUSTER CENTERS
 - ② ASSIGN DATA POINTS TO CLUSTER CENTERS
 - ③ RECOMPUTE CLUSTER CENTERS
- REPEAT

DRAWBACKS

- NUMBER OF CLUSTERS MUST BE KNOWN
- NOT DETERMINISTIC
- ALL DATA POINTS ARE ASSIGNED TO CLUSTERS



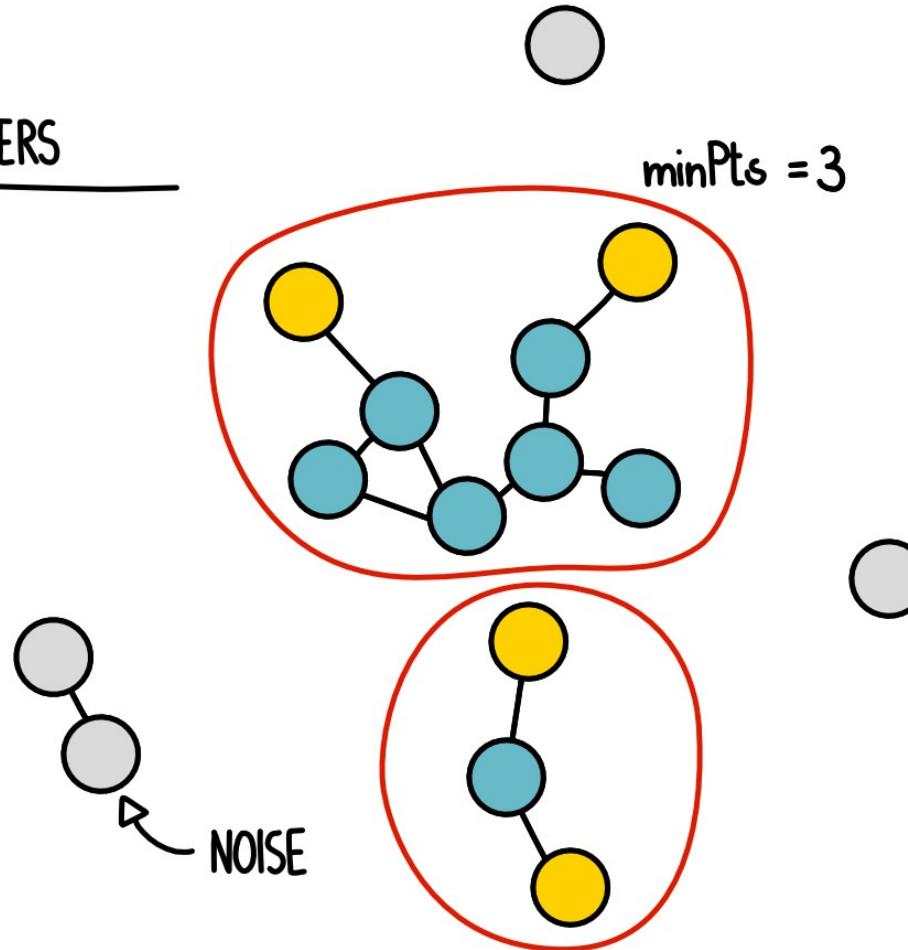
DBSCAN

ADVANTAGES

- MOSTLY DETERMINISTIC
- NUMBER IS DETERMINED AUTOMATICALLY
- CAN DETECT OUTLIERS
- FINDS CLUSTERS WITH ARBITRARY SHAPES

PARAMETERS

- ϵ
- minPts

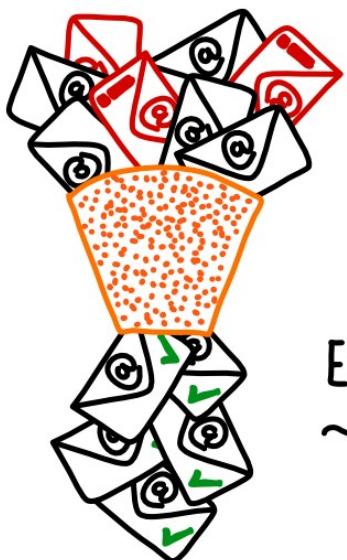


STEPS

- FOR EACH POINT FIND ALL POINTS IN ϵ NEIGHBORHOOD
- IDENTIFY THE CORE POINTS
- FIND CONNECTED COMPONENTS OF CORE POINTS
- ASSIGN NON-CORE POINTS TO CLUSTERS

SUPERVISED LEARNING

- LABELS OF DATA IS KNOWN
- TYPICAL USE CASE: CLASSIFICATION
E.G. SPAM FILTER



EXISTS FOR
~20 YEARS

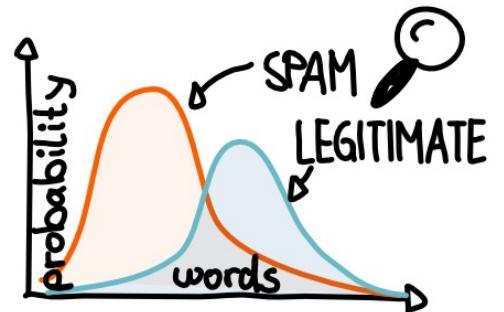
DATA FOR TRAINING



SPAM
OTHER



EXPLOIT DIFFERENCES
IN WORD DISTRIBUTIONS



DRAWBACKS

- CANNOT DETECT UNKNOWN PATTERNS
- DIFFICULT TO GET EXAMPLES OF ATTACKS (\rightarrow IMBALANCED DATA)

SEMI-SUPERVISED LEARNING

RECAP

SUPERVISED

- + SIMPLE
- + HIGH ACCURACY
- EXAMPLES FOR ATTACKS REQUIRED

UNSUPERVISED

- + NO LABELED DATA REQUIRED
- MAINLY FOR MANUAL ANALYSIS

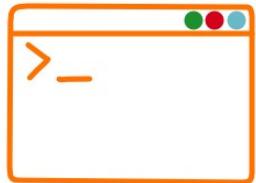


Detect RCEs

USEFUL DATA SOURCES

- NETWORK TRAFFIC
- USER ACTIVITIES
- SYSTEM LOGS
- APPLICATION LOGS

REVERSE SHELL



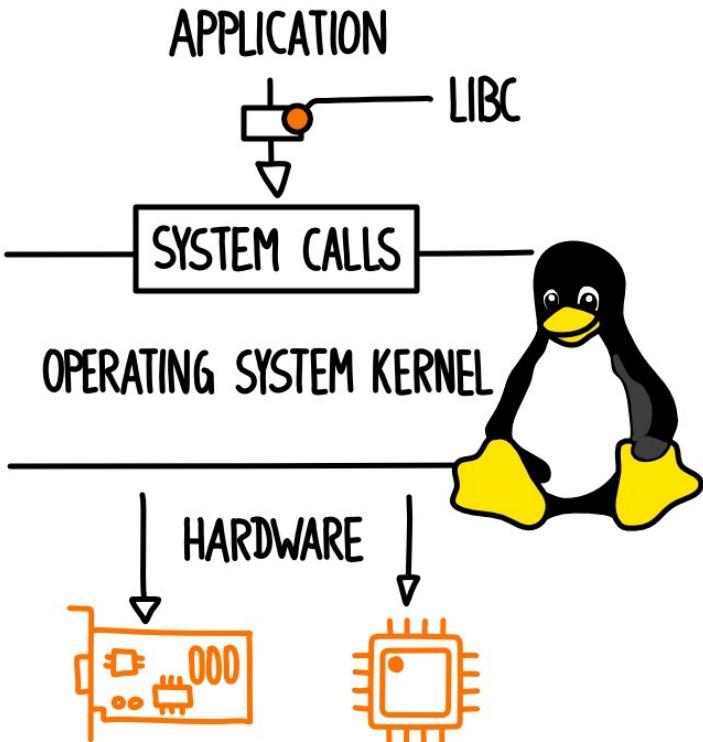
- CONNECTION OPEN FOR LONG TIME
- TYPICALLY IDLE
- SUSPICIOUS IP ADDRESS

INTERESTING

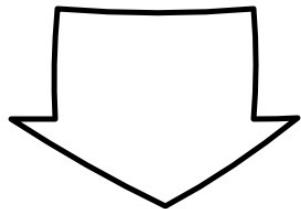
COLLECTING DATA WITH OSQUERY

- DEVELOPED BY FACEBOOK
- RUNS EVERYWHERE
- SQL API

SYSTEM CALLS



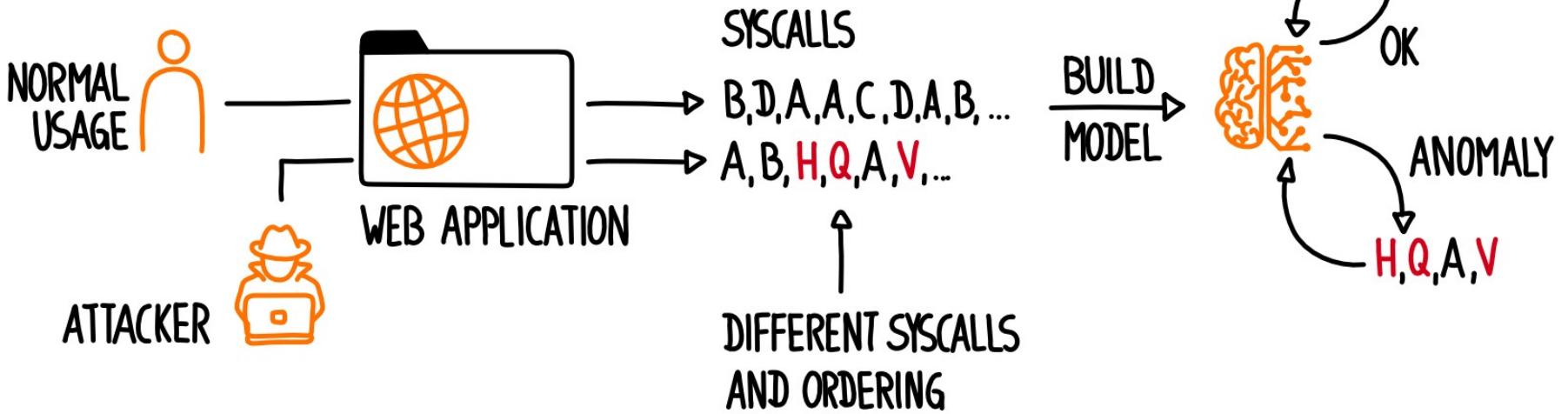
EXAMPLES



CATEGORY	SYSTEM CALLS
CONTROL PROCESS	KILL
FILE OPERATIONS	OPEN, READ, WRITE
NETWORK	ACCEPT, LISTEN, SOCKET

SYSTEM CALLS

ASSUMPTION: EACH APPLICATION USES ONLY A SUBSET OF SYSCALLS



SYSTEM CALLS

EXAMPLE OF A SEQUENCE

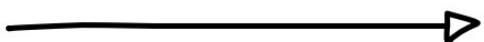
OPEN , READ , READ , OPEN , WRITE , WRITE , CLOSE , CLOSE , FORK , EXIT

SW 1 SW 2

SW 1 → OPEN , READ , READ

SW 2 → READ , READ , OPEN

BUILD A SUPER
SIMPLE MODEL



SEQUENCE WITH N SYSCALL
SLIDING WINDOW LENGTH M
⇒ N-M+1 SUBSEQUENCES

SET OF ALL
= UNIQUE
SUBSEQUENCES

FIRST APPROACH

- READ COMPLETE SEQUENCE
- CREATE FEATURE VECTOR

PROBLEM

WHAT TO DO WITH
UNLIMITED OR VERY LONG SEQUENCES

SECOND APPROACH

- SLIDING WINDOW

→ SUPER SIMPLE TRAINING

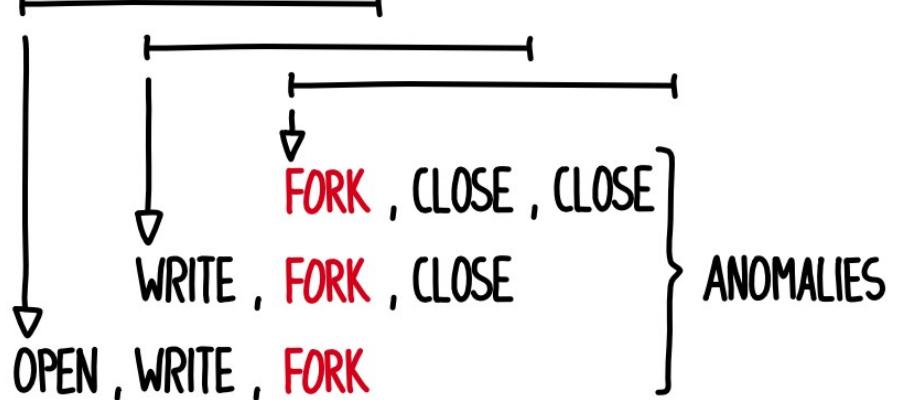
SYSTEM CALLS

EXAMPLE OF A SEQUENCE

OPEN , READ , READ , OPEN , WRITE , WRITE , CLOSE , CLOSE , FORK , EXIT

NEW SEQUENCE

OPEN , READ , READ , OPEN , WRITE , **FORK** , CLOSE , CLOSE , FORK , EXIT



REDUCING FALSE POSITIVES



NO ANOMALY

ANOMALY

↓
3 OF 4

SUBSEQUENCES
ARE ANOMALIES

DO SYSTEM CALLS WORK



ANALYZING 4 SCENARIOS

A) PHP APPLICATION

B) POSTGRESQL



C) POSTFIX

D) POP3



RECORDED MILLIONS
OF SYSTEM CALLS

RESULTS

- SMALL SUBSET OF SYSTEM CALLS IS CALLED
A,B,D: ~40 E: 13

- SEQUENCES OF LENGTH 1 DON'T WORK

WITH THIS
OBSERVATION



INTERSECTION

	B	C	D
A	9	4	1
B	8	3	
C	10		

IN PRACTICE

A B C D
250 250 150 88

IN THEORY

$\sim 330^3$ $\sim 36 \text{ MIO}$ 3-TUPLES
 $\sim 40^3$

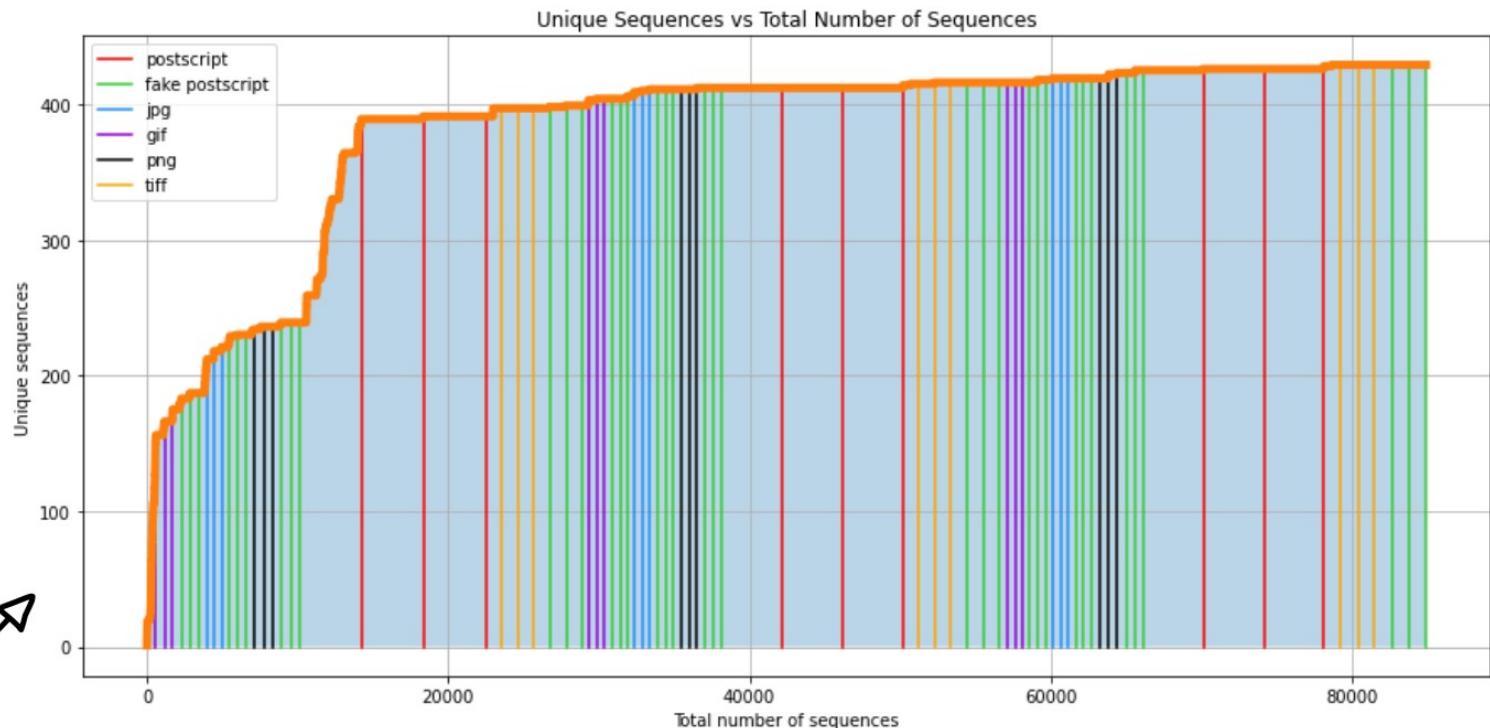
TESTS WITH LENGTH 3

EXAMPLE: CONVERT

WHEN CAN
WE STOP
LEARNING?



WHEN NO NEW
SEQUENCES
ARE OBSERVED



EXAMPLE: CONVERT

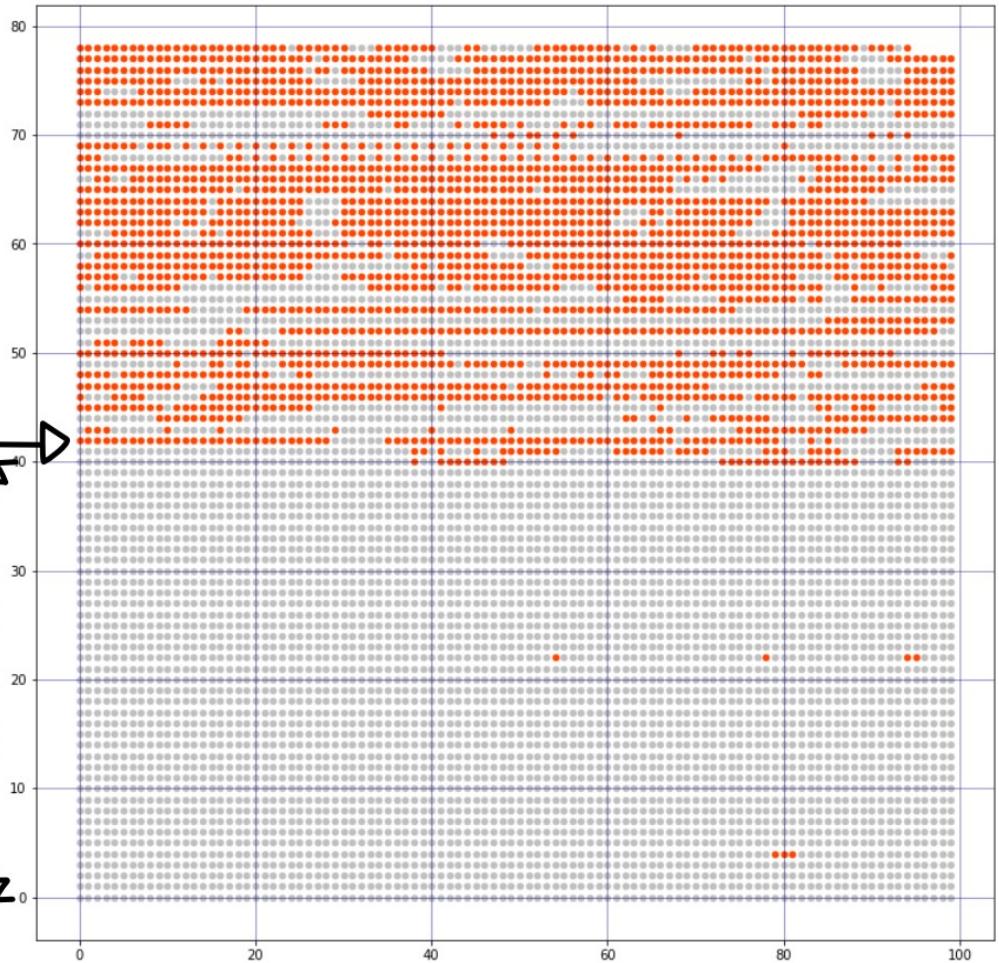
EXECUTED SEQUENCES WHEN REVERSE SHELL IS LAUNCHED

REVERSE SHELL LAUNCHED

COMMON PREFIX FOR POSTSCRIPT FILES

- UNKNOWN SEQUENCE

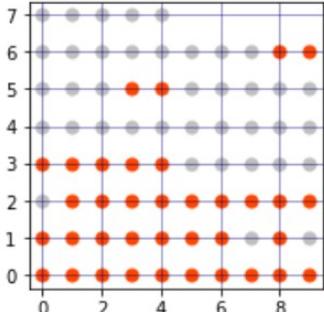
- KNOWN SEQUENCE (I.E. SEQUENCE IN MODEL)



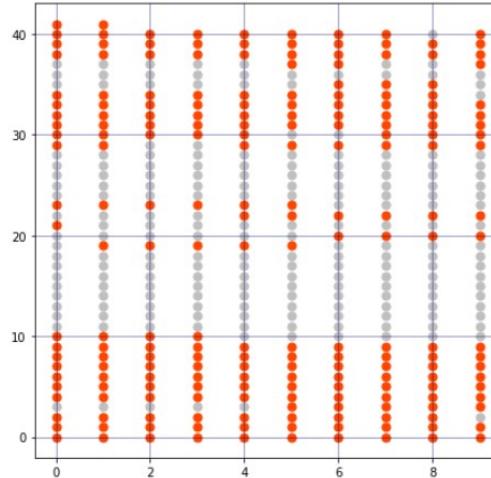
EXAMPLE: CONVERT

EXECUTED SEQUENCES FOR...

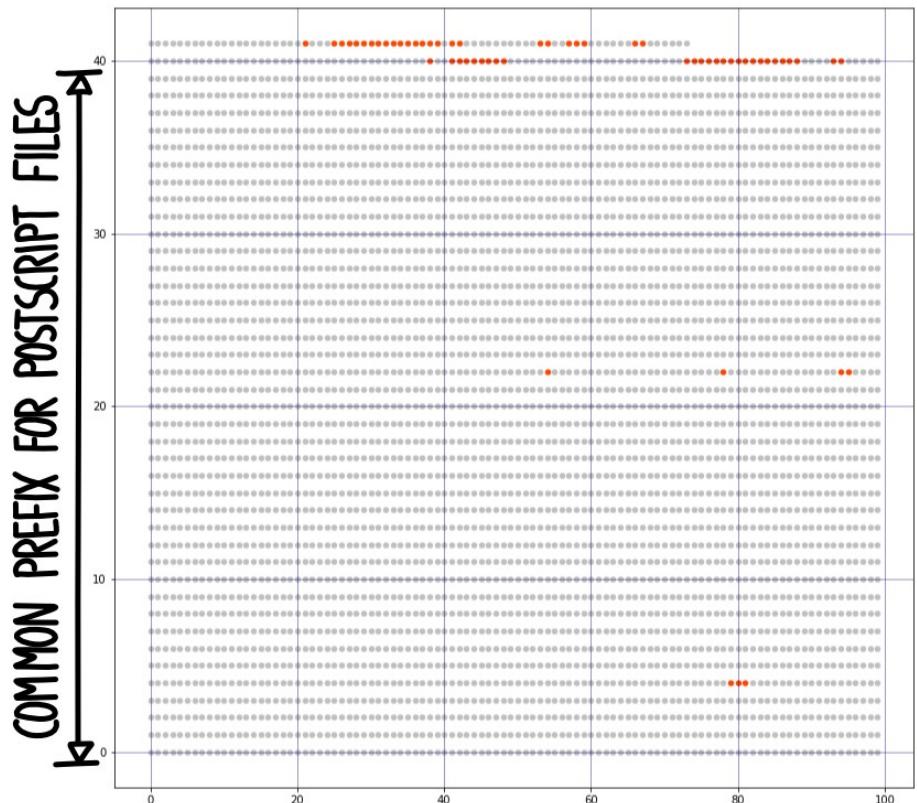
EXIT (EXECUTED IN RS.)



LS (EXECUTED IN RS.)



CAT /ETC/PASSWD



VISUALIZE LOG FILES

TYPICAL
CONTENT



CHALLENGES

- LOGFILES ARE UNSTRUCTURED



SEARCHING FOR ANOMALIES IS DIFFICULT

DYNAMIC CONTENT [X]

EXAMPLE

- ONE LINE = ONE EVENT
- ERROR, INFO, NOTICE
- DATE, TIME, IP, ...

```
2020-09-07 00:00:09,013 16859 [sid:68B9,ip:79.19.30.120] INFO session: [*****-**-**-**-*****ce80cd
2020-09-07 00:00:09,013 16859 [sid:68B9,ip:79.19.30.120] INFO IN: [EmailAddress [address=k*****@**.de],
    uid=null, session=*****-**-**-**-*****ce80cd]
2020-09-07 00:00:09,074 16920 [sid:68B9,ip:79.19.30.120] INFO RequestLog 4bd67537bcf4 - '' ''
    "DELETE https://server.foo/'' 204 () '' 60 millis ''
```

OUR GOAL →

- VISUALIZE LOGFILES W/O PREPROCESSING

- SHOULD WORK FOR MOST LOGFILES

VISUALIZE LOG FILES

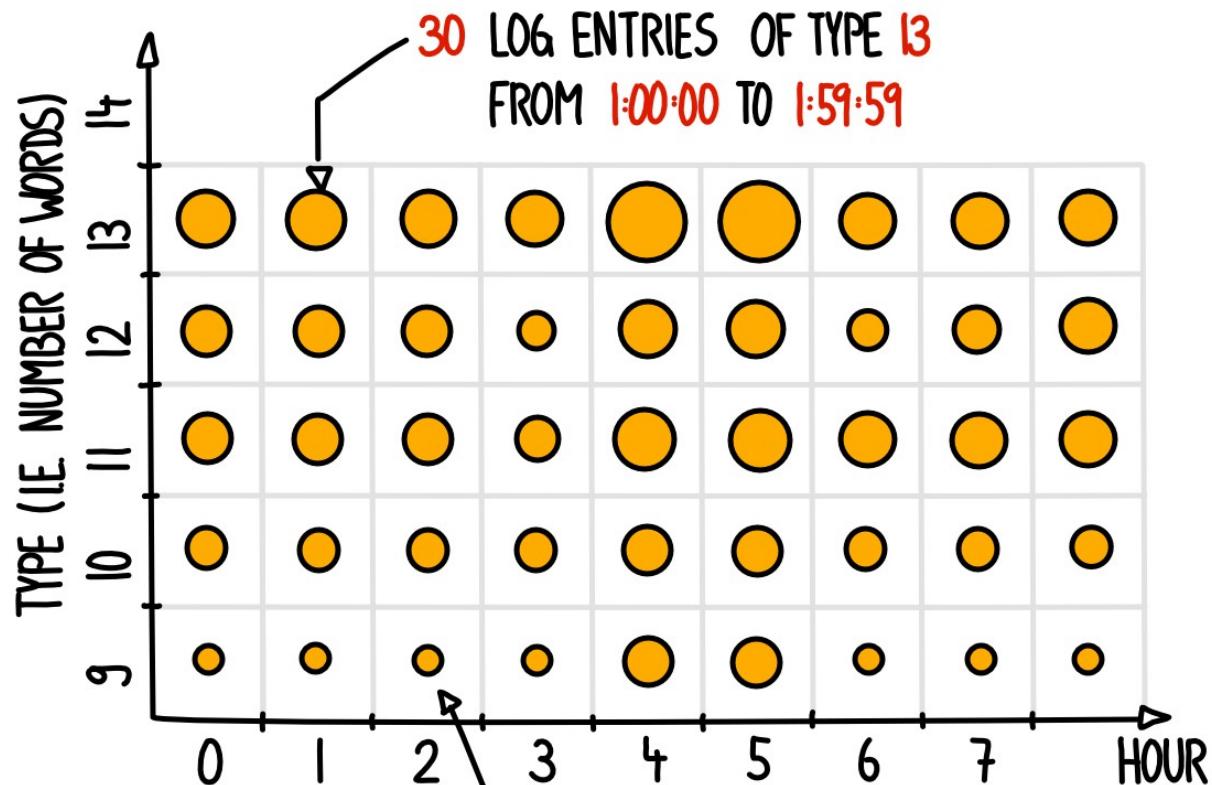
APPROACH BASED ON PAPER:

 "HISTOGRAM MATRIX:
LOG FILE VISUALIZATION FOR
ANOMALY DETECTION"
2007

- ① GROUP LOG ENTRIES BY TIME
- ② GROUP LOG ENTRIES BY TYPE

COUNT NUMBER
OF WORDS

→ SPLIT BY SPACE



$$\tilde{N}_{h,t} = \frac{N_{h,t}}{\sum_t N_{h,t}}$$

VISUALIZE LOG FILES

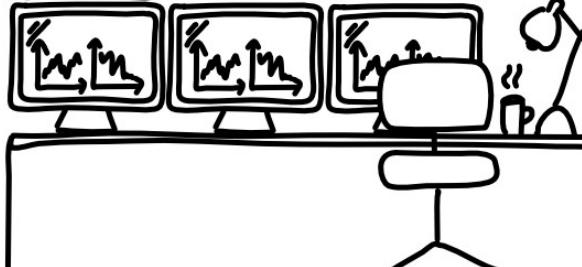
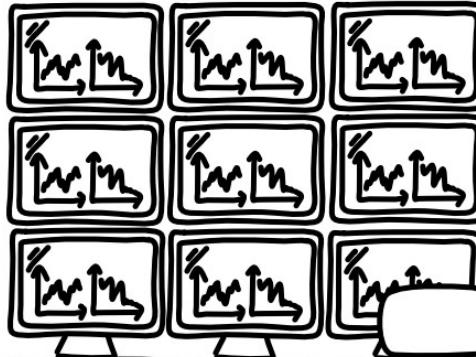
EXAMPLE HOW HMAT CAN HELP : WIZARD



OTHER EXAMPLES:

- INSECURE DIRECT OBJECT REFERENCE
- BRUTE FORCE ATTACKS

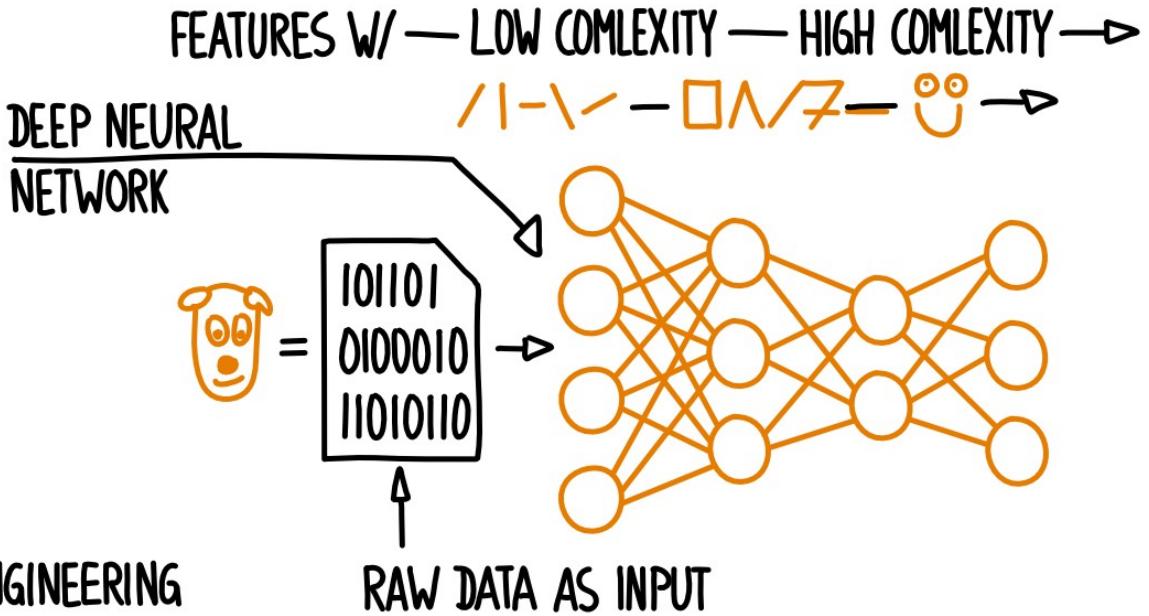
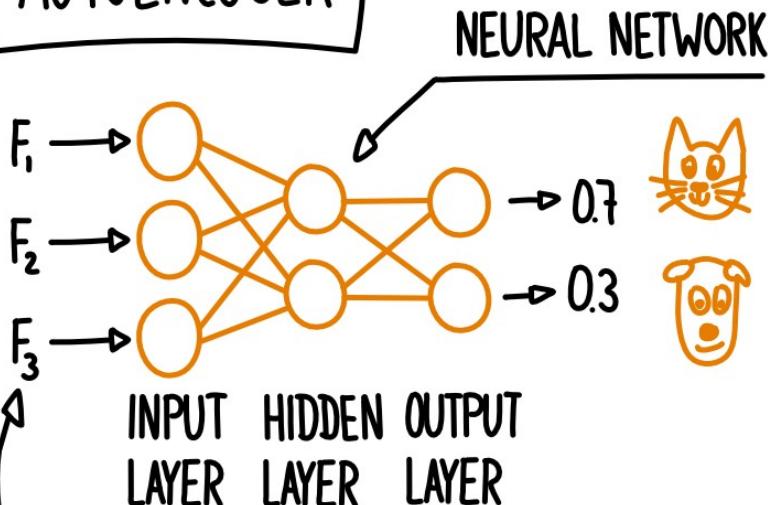
I HAVE
THOUSANDS
OF LOG FILES



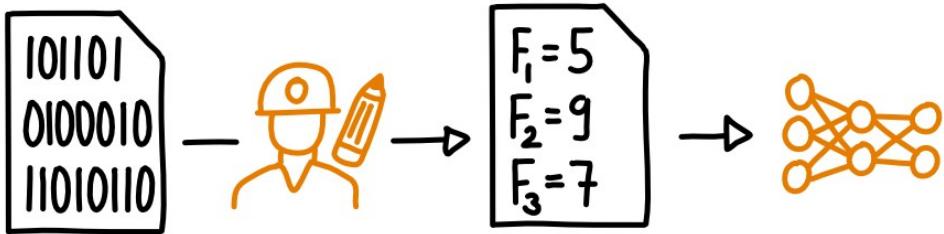
NO PROBLEM!
USE THRESHOLDS



AUTOENCODER

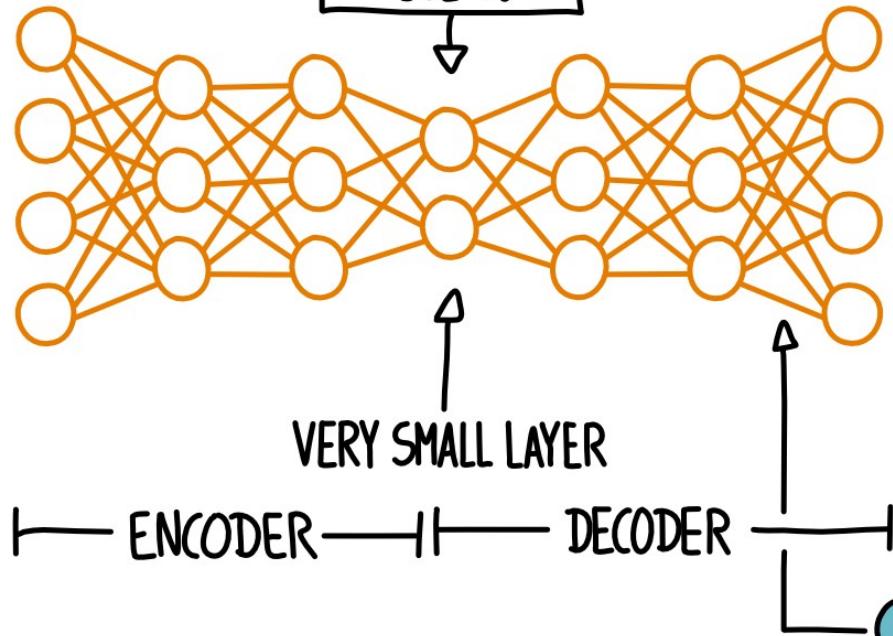


TRADITIONAL APPROACH: MANUAL FEATURE ENGINEERING



AUTOENCODER

- #OUTPUT NEURONS = #INPUT NEURONS
- GOAL: $f(x) = x$ — CHALLENGING DUE TO

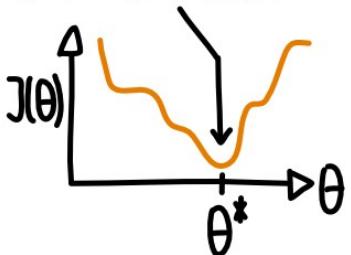


③ COMPUTE $J(\theta)$

COMPUTE $\frac{\partial}{\partial \theta_i} J(\theta)$

$J(\theta)$ = ERROR OF NETWORK

GOAL: FIND θ SUCH THAT
 $J(\theta)$ IS MINIMIZED



④ UPDATE PARAMETERS

CONCLUSION

SUMMARY



ATTACK VECTORS



BASICS OF MACHINE LEARNING



EXAMPLES:

- DETECT INTRUSION VIA SYSTEMS CALLS
- LOG FILE VISUALIZATION
- AUTOENCODER

QUESTIONS / FEEDBACK / CONTACT



HELLO@ETZOLD.IO



@ETZOLDIO



GITHUB.COM/DANIEL-E



GITHUB.COM/DETECTR

JUPYTER NOTEBOOKS

GITHUB.COM/DANIEL-E/2020-ML-SUMMIT