NINTH ANNUAL LEADERSHIP EVENT

CYBER SECURITY SUMMIT

Security solutions through collaboration.™

October 28–30, 2019 | Minneapolis Convention Center

cybersecuritysummit.org | #cybersummitmn

OPERATIONALIZED END TO END ENTERPRISE AI Remains a Complete Mystery

Simon Crosby, CTO @swim

Cyberwar

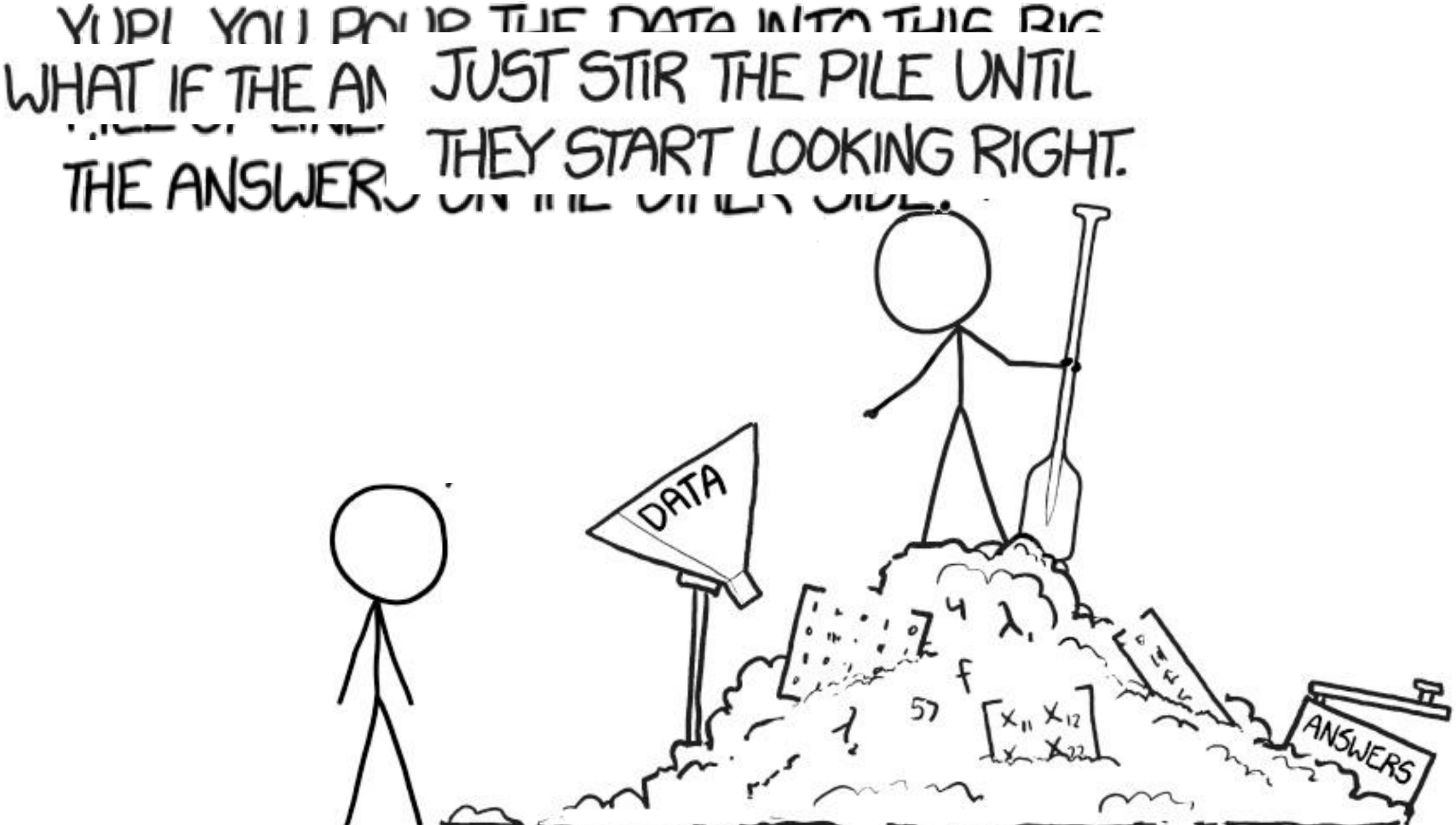
War in the fifth domain

Is AI the new weapon of choice?

Sure…

What can you do to prepare for it?

SWIM


WHAT IF THE AN
JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.
DATA
ANSWERS

AI/ML in Practice

Statistics

- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

AI/ML Use Cases

- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure that allows it to be learned.

Source: www.cs.toronto.edu/~hinton

SWIM

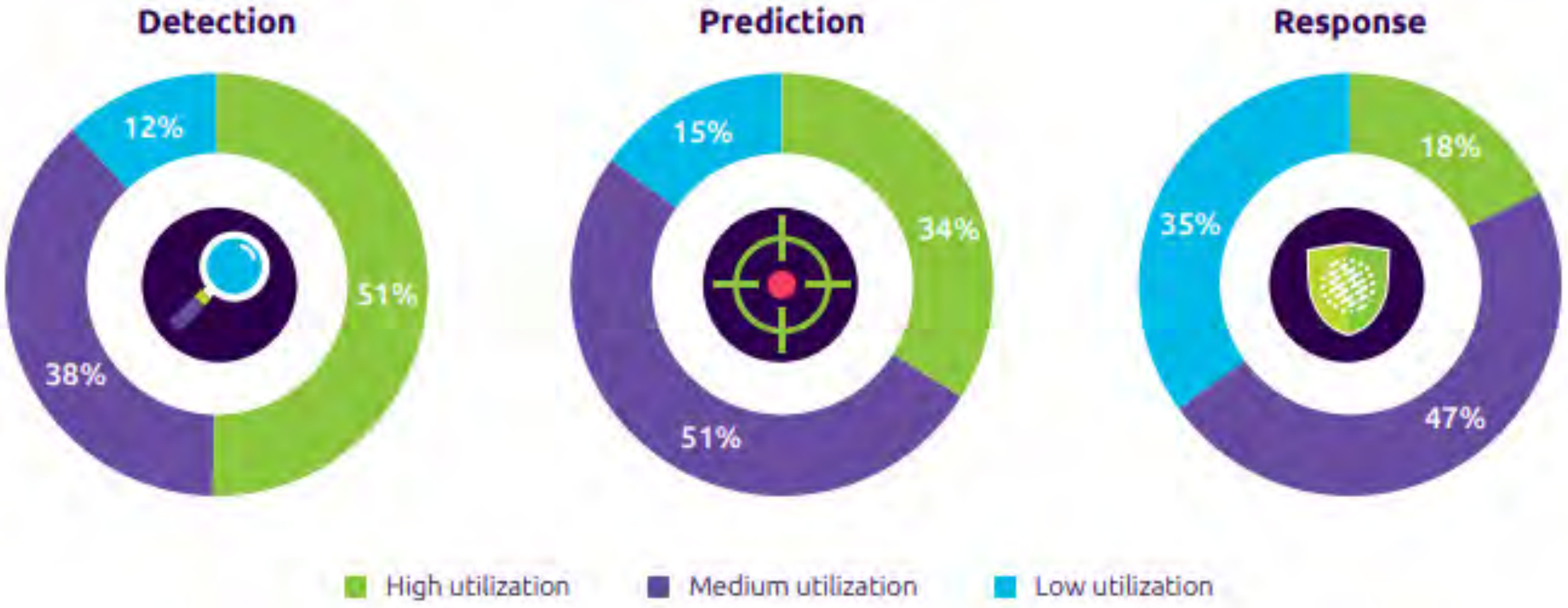
Example

Each document is converted to a vector of word counts.

This vector is then mapped to two coordinates and displayed as a dot.

Let classes be represented by colors.

When the documents are laid out in 2-D, the classes are not useful. So we can judge how good the algorithm is by seeing if the classes are separated.

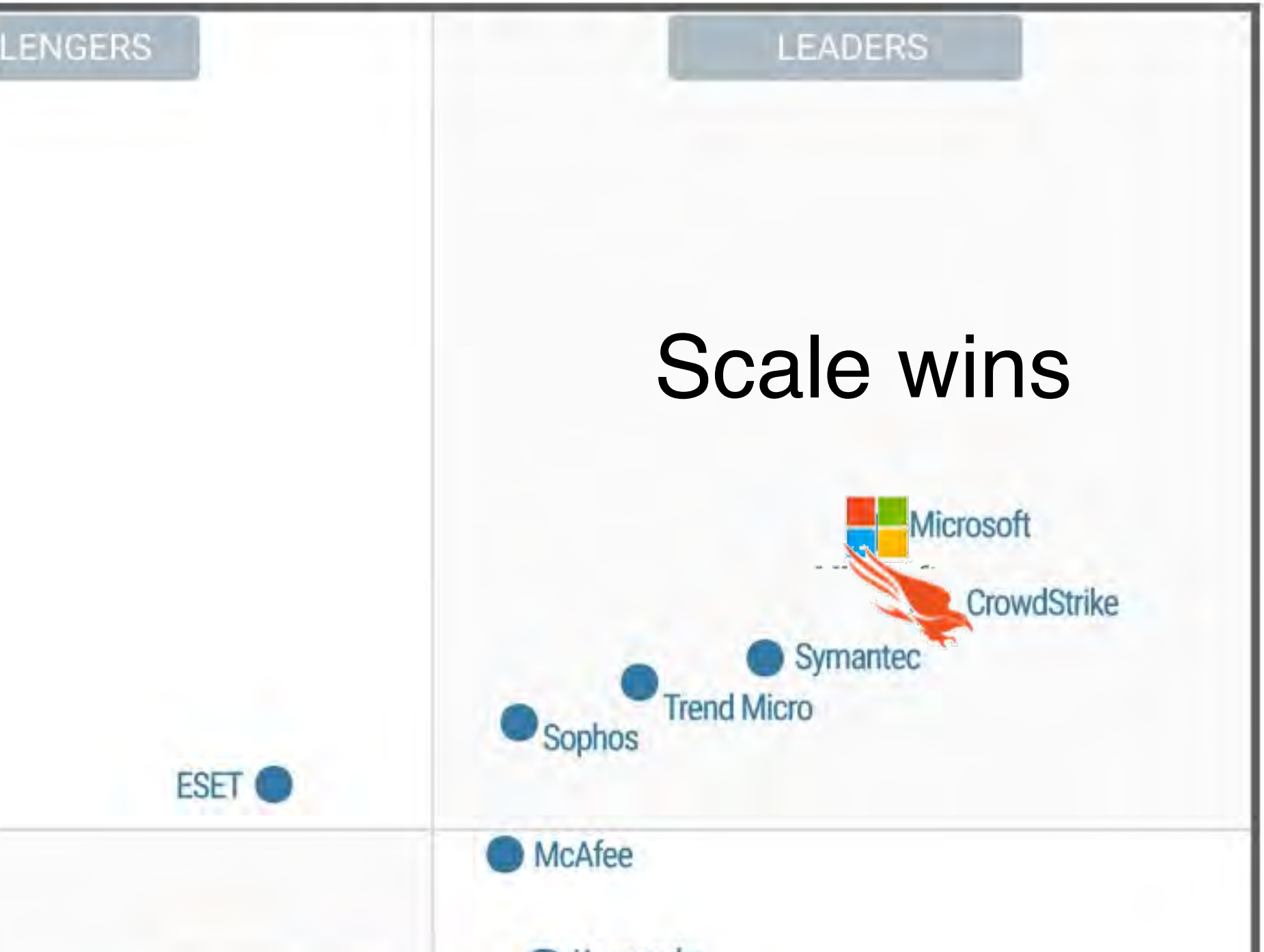


Source: Capgemini Research Institute, AI in Cybersecurity executive survey, N = 850 executives

Figure 1. Magic Quadrant for Endpoint Protection Platforms

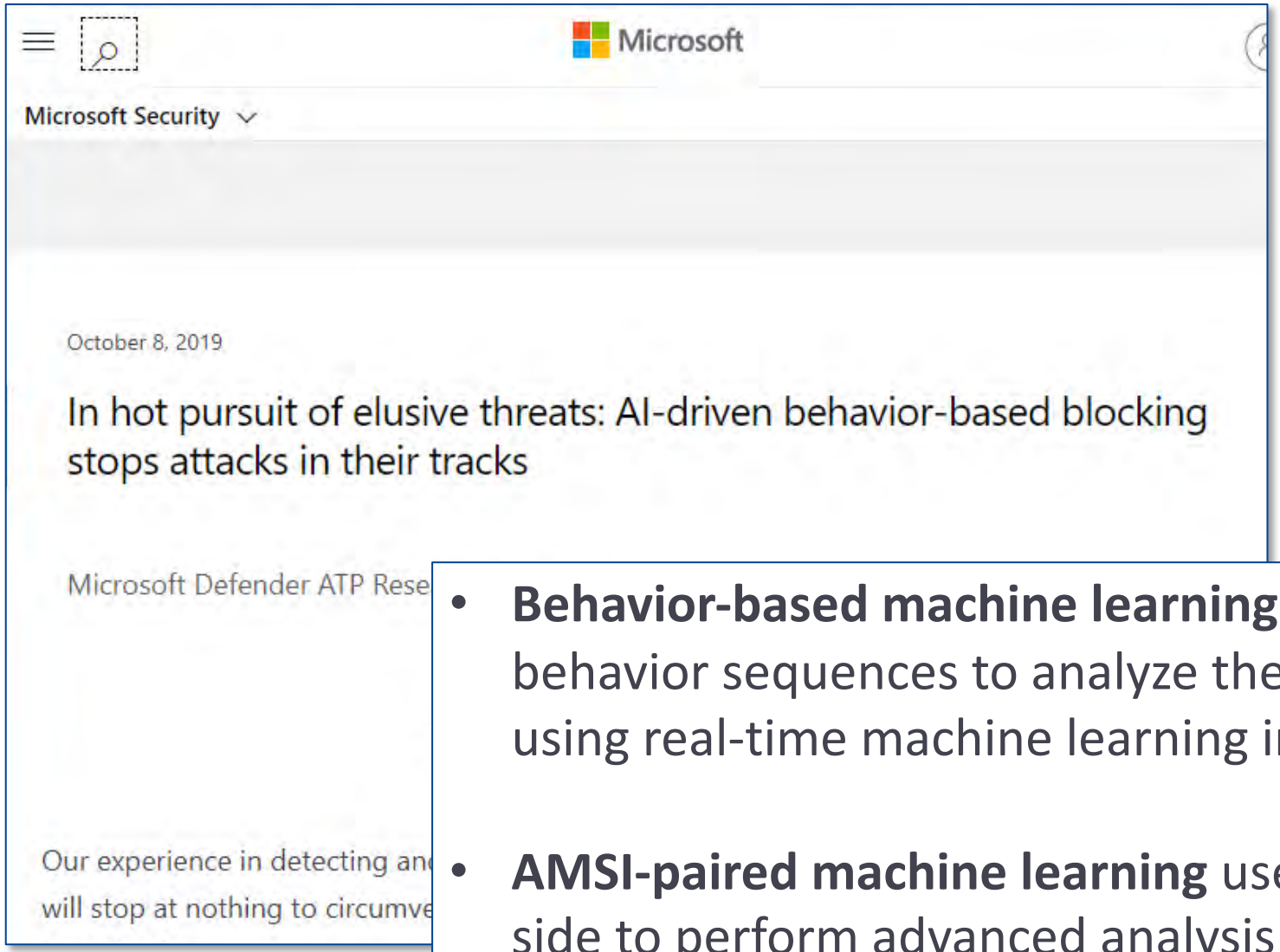
Source: Gartner (August 2019)

SWIM



Scale wins

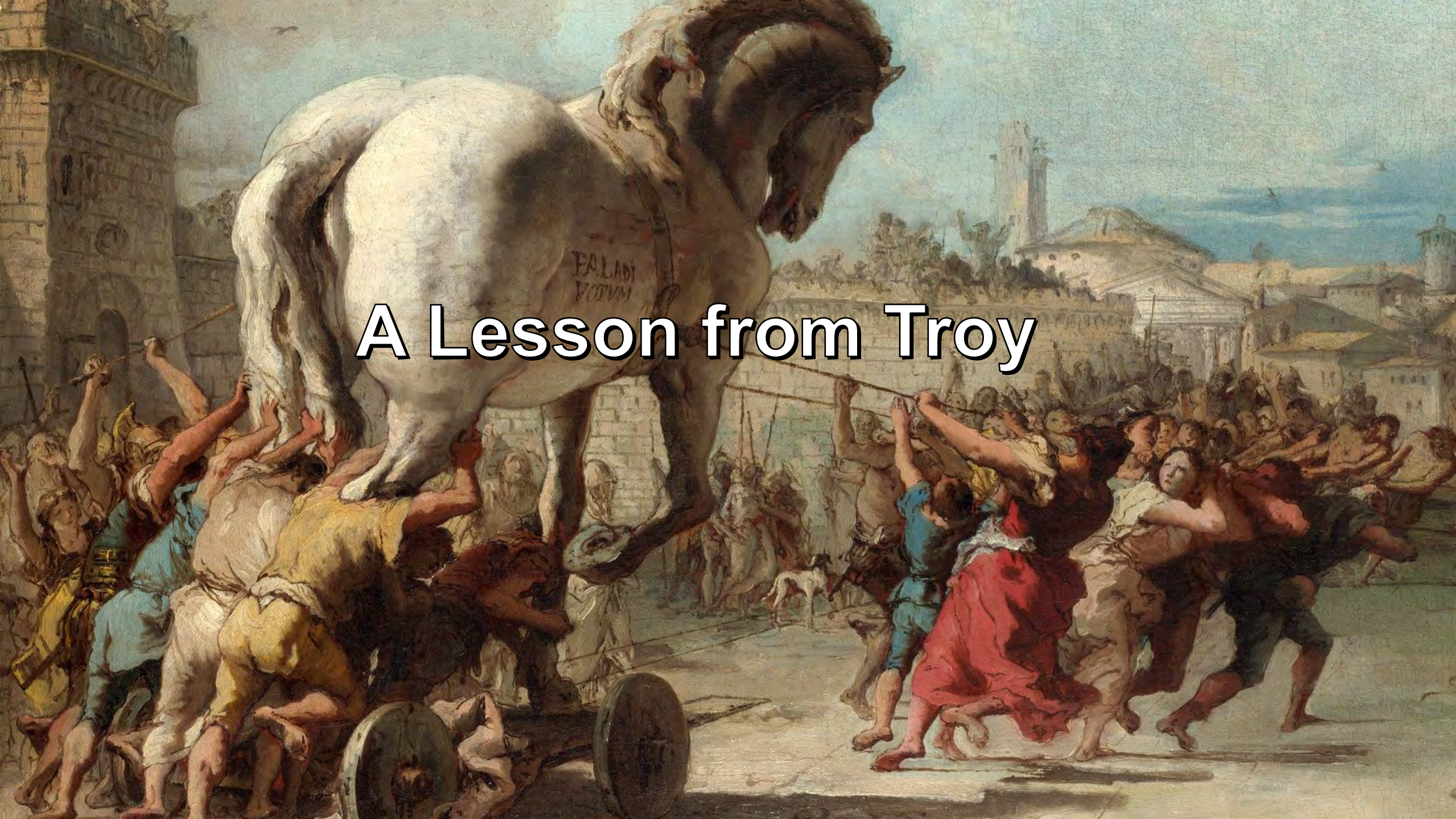
SWIM



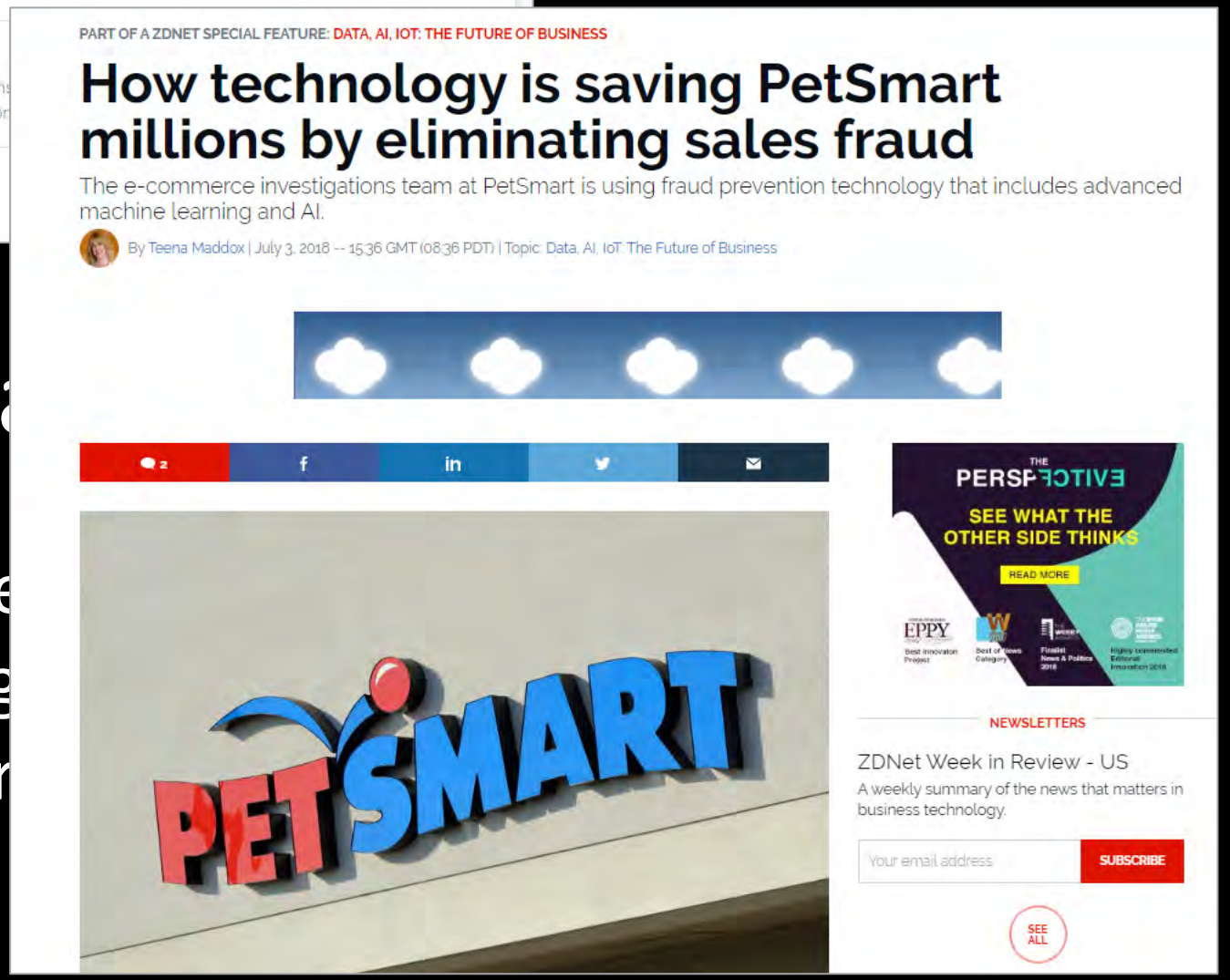
- **Behavior-based machine learning** identifies suspicious process behavior sequences to analyze the process tree on the client, using real-time machine learning in the cloud

- **AMSI-paired machine learning** uses pairs of client-side and cloud-side to perform advanced analysis of scripting pre- and post-execution to catch fileless and in-memory attacks

SWIM

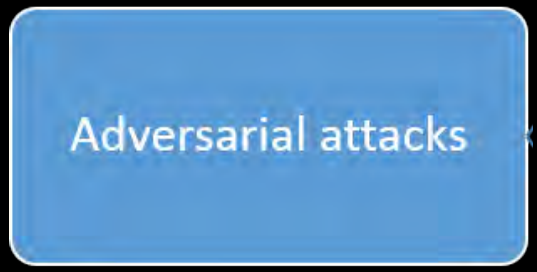


A Lesson from Troy



Offensive AI: a pa

- Impersonation of truste
- Blending into the backg
- Faster attacks with mor
- Finding vulnerabilities



Source: Axionable

NEWS

Researchers fool Cylance AI antimalware with 'simple' bypass

Security researchers developed a method to make "pure AI" antimalware products classify malware as benign, but it is unclear what antimalware solutions could be considered "pure AI."

Exploit Generation

- Analyzed 14 OSS projects
- Generated 16 control flow hijacking exploits.
- Two of them are 0-days
- General approach for generating working exploits once a bug is found

AEG: Automatic Exploit Generation

Thanassis Avgerinos, Sang Kil Cha, Brent Lim Tze Hao and David Brumley
Carnegie Mellon University, Pittsburgh, PA
{thanassis, sangkilc, brentlim, dbrumley}@cmu.edu

Abstract

The automatic exploit generation challenge is given a program, automatically find vulnerabilities and generate exploits for them. In this paper we present AEG, the first end-to-end system for fully automatic exploit generation. We used AEG to analyze 14 open-source projects and successfully generated 16 control flow hijacking exploits. Two of the generated exploits (expect-5.43 and htget-0.93) are zero-day exploits against unknown vulnerabilities. Our contributions are: 1) we show how exploit generation for control flow hijack attacks can be modeled as a formal verification problem, 2) we propose preconditioned symbolic execution, a novel technique for targeting symbolic execution, 3) we present a general approach for generating working exploits once a bug is found, and 4) we build the first end-to-end system that automatically finds vulnerabilities and generates exploits that produce a shell.

1 Introduction

Control flow exploits allow an attacker to execute arbitrary code on a computer. Current state-of-the-art in control flow exploit generation is for a human to think very hard about whether a bug can be exploited. Until now, automated exploit generation where bugs are automatically found and exploits are generated has not been shown practical against real programs.

In this paper, we develop novel techniques and an end-to-end system for automatic exploit generation (AEG) on real programs. In our setting, we are given the potentially buggy program in source form. Our AEG techniques find bugs, determine whether the bug is exploitable, and, if so, produce a working control flow hijack exploit string. The exploit string can be directly fed into the vulnerable application to get a shell. We have analyzed 14 open-source projects and successfully generated 16 control flow hijacking exploits, including two zero-day exploits for previously unknown vulnerabilities.

Our automatic exploit generation techniques have several immediate security implications. First, practical AEG fundamentally changes the perceived capabilities of attackers. For example, previously it has been believed that it is relatively difficult for untrained attackers to find novel vulnerabilities and create zero-day exploits. Our research shows this assumption is unfounded. Understanding the capabilities of attackers informs what defenses are appropriate. Second, practical AEG has applications to defense. For example, automated signature generation algorithms take as input a set of exploits, and output an IDS signature (*aka* an input filter) that recognizes subsequent exploits and exploit variants [3, 8, 9]. Automated exploit generation can be fed into signature generation algorithms by defenders without requiring real-life attacks.

Challenges. There are several challenges we address to make AEG practical:

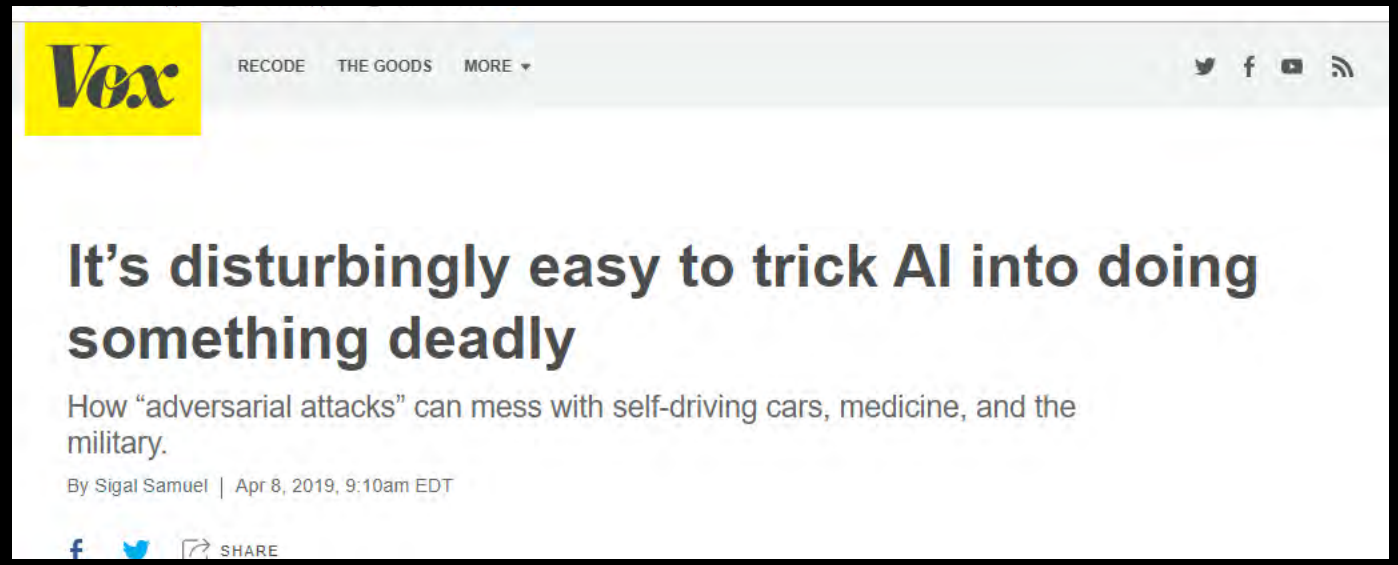
A. Source code analysis alone is inadequate and insufficient. Source code analysis is insufficient to report whether a potential bug is exploitable because errors are found with respect to source code level abstractions. Control flow exploits, however, must reason about binary and runtime-level details, such as stack frames, memory addresses, variable placement and allocation, and many other details unavailable at the source code level. For instance, consider the following code excerpt:

```
char src[12], dst[10];
strncpy(dst, src, sizeof(src));
```

In this example, we have a classic buffer overflow where a larger buffer (12 bytes) is copied into a smaller buffer (10 bytes). While such a statement is clearly wrong [1] and would be reported as a bug at the source

[1] Technically, the C99 standard would say the program exhibits undefined behavior at this point.

Deepfakes

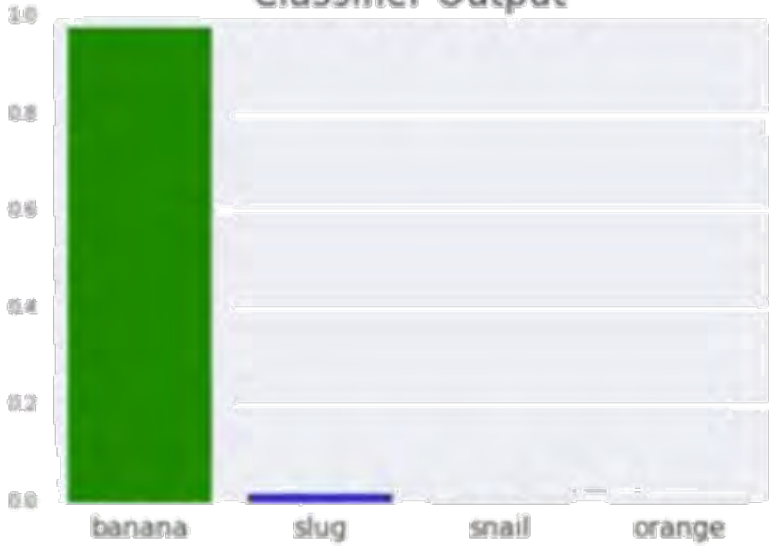


"There are no fixes for the fundamental brittleness of DNNs,"

François Chollet, Google

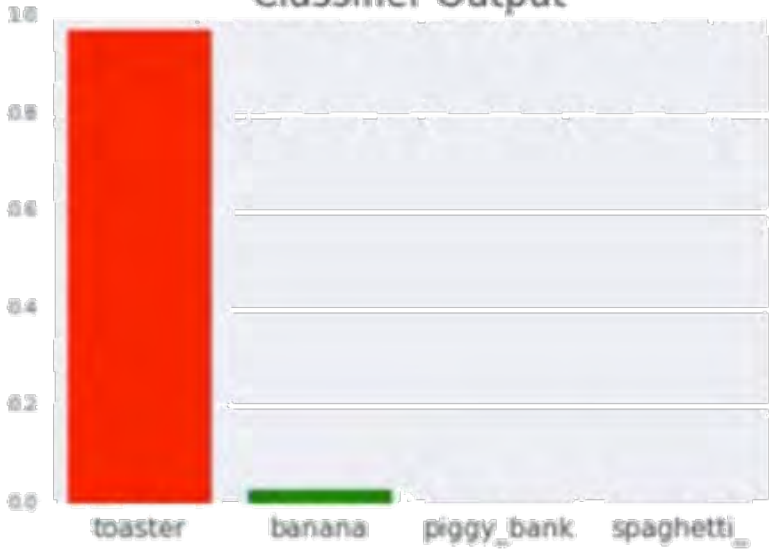
Classifier Input

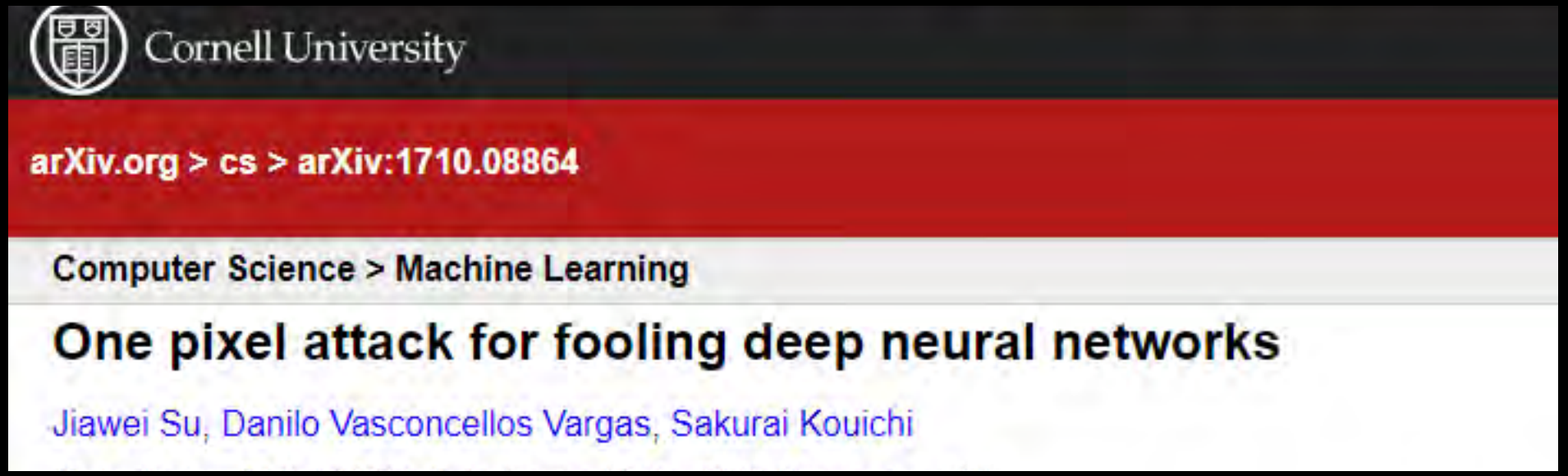
Classifier Output



Classifier Input

Classifier Output





"68.36% of images in CIFAR-10 and 16.04% of ImageNet images can be successfully attacked by modifying just one pixel"

Facial identifier that works through (some) masks

- Face liveness detection
- Contextual information
- Texture analysis
- Target interaction

… deep learning for "Disguised Facial Identification",

AKA "how to identify people at protests who have covered their faces"

To Appear in the IEEE International Conference on Computer Vision Workshops (ICCVW) 2017

Disguised Face Identification (DFI) with Facial KeyPoints using Spatial Fusion Convolutional Network

Amarjot Singh
Department of Engineering
University of Cambridge, U.K.
as2436@cam.ac.uk

Devendra Patil
National Institute of Technology
Warangal, India
pdevendra@student.nitw.ac.in

G Meghana Reddy
National Institute of Technology
Warangal, India
rmeghana@student.nitw.ac.in

SN Omkar
Indian Institute of Science
Bangalore, India
omkar@aero.iisc.ernet.in


A Lesson from Byzantium


Br
Bromium®
Microsoft


Microsoft
Bromium®
Device Guard uses micro-virtualization to hardware-isolate two Windows Services
Credential Guard isolates credential hashes to reduce pass-the-hash attacks
Hypervisor enforced Code Integrity enforces white-listing for applications and the OS kernel
Windows Defender Application Guard isolates tabs in Edge
CG
Credential Guard
HVCI
Code Integrity

Basic integrity protection
Secure boot
TPM2.0
BitLocker

Protection from kernel attacks
Virtualization Based Security (VBS)
Hypervisor protected Code Integrity (HVCI)
Kernel DMA protection

Protection from firmware attacks
System Guard Secure Launch
System Guard SMM Protections

SWIM