Attacking Machine Learning
The Cylance Case Study

BSides Sydney 2019
About Us

Wears T-Shirts in Corporate Headshots
[yes, I am wearing it now too!]

Heavy on the offensive cyber side (Government)

Adi Ashkenazy
CEO, Skylight Cyber

Category 5 stage fright

Red-team automation

Shahar Zini
CTO, Skylight Cyber

Sydney based consultancy

Help companies navigate cyber security

Not even AI power users...
In a nutshell

Why is this important?

What are we looking to achieve?

AI in Cyber for people who understand quickly

How we approached the problem and reversing the product

Results!

Publication and Feedback

Questions
Challenge

Assess Technology

Silver Bullet Hunting
WHY CYLANCE?

AI Centric, can buy it off the shelf, consistently ranks high

Their marketing didn’t help!

“For the product is as close as you can get to a silver bullet in our space. Greater than 99% efficacy and protection against nearly every zero-day malware.”

Zero-Day Attacks
AI model prevents zero-day payloads from executing

Forrester Report: Cylance Provides 251% ROI
The What

- A five finger death punch to the heart of the product - a “universal passive bypass”.
- Proving that an ML model itself presents a new attack surface.
- Show that no, AI did not “solve security”.
Classification with AI/ML - The Basics

“In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.”

Wikipedia

What is THAT?

Cat

Dog
**Classification with AI/ML**

**Training - Lab**
- Large Data Set
- Feature Selection
- Training [Black Magic]
- Model!

Lengthy, resource intensive

**Classifying - Field**
- One Sample
- Feature Extraction
- Model application
- Decision and confidence

Quick and easy to replicate
An offensive mindset

- Classification is innately naive
- A model is only as good as its data
- How would we fool the bird vs. human classifier?
The OpSec Paradox

Productivity, Marketing & Legal

OpSec

Good for sales, bad for security

Good for security, bad for sales
How CylanceINFINITY Works

CylanceINFINITY collects data, trains, and learns from the data, then calculates likely outcomes based on what it sees. It’s constantly getting smarter from environmental feedback and a constant stream of new data from all around the world.

To achieve its magic, CylanceINFINITY first collects vast amounts of data from every conceivable source. Next, it extracts features that we have defined to be uniquely atomic characteristics of the file depending on its type (.exe, .dll, .com, .pdf, .java, .doc, .xls, .ppt, etc.). Then, it constantly adjusts to the real-time threatscape and trains the machine learning system to make better decisions. Finally, for each query to CylanceINFINITY, we classify the data as good or bad.
Extracting the Model

```
namespace Cylance.Engine.Core.Ensemble
{
    public class EnsembleReader : IlogAccess, IEnsembleHeader, IDisposable
    {
        protected const int RandomHeaderSize = 1024;
        protected const string KeyAndIV = "I am decrypting Cylance's intellectual property!";
        protected Stream _stream;
        protected byte[] _activeKey;
        protected byte[] _activeIV;
        protected bool _loadSectionData;
    }
}
```
Our own classifier

Let's build our own classifier so we can dynamically debug and follow the code

```csharp
static void Main(string[] args)
{
    SampleScoreFactory2ME factory = new SampleScoreFactory2ME();
    SampleScoring2ME scorer = factory.Create("test_model.bin") as SampleScoring2ME;
    Stream test_file = File.Open("minikatz_with_slight_modification.exe", FileMode.Open);
    Dictionary<string, object> extraData;
    double score = scorer.ComputeScore(test_file, out extraData);
}
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>-0.85276468809127071</td>
</tr>
</tbody>
</table>
Anti-Tampering & Obfuscation

ERROR

Protected module cannot disassemble

USELESS

ANNOYING

```
if (true) {
    int i = 10;
    byte b = (byte) 8;
    this.int1 = i;
    this.byte1 = b;
}
else {
    this.int2 = 20;
}
```
### Parsing the Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linker version</td>
<td>5.1</td>
</tr>
<tr>
<td>Num sections</td>
<td>5</td>
</tr>
<tr>
<td>Section casing</td>
<td>Uppercase</td>
</tr>
<tr>
<td>Entropy</td>
<td>0.2315</td>
</tr>
<tr>
<td>Timestamp</td>
<td>13102382120</td>
</tr>
<tr>
<td>Max section size</td>
<td>827Kb</td>
</tr>
<tr>
<td>CLR version</td>
<td>4.0</td>
</tr>
<tr>
<td>#UI imports</td>
<td>98</td>
</tr>
<tr>
<td>#Process imports</td>
<td>14</td>
</tr>
<tr>
<td>#imports</td>
<td>412</td>
</tr>
</tbody>
</table>
# Building the Feature Vector

The diagram illustrates the process of building the feature vector from various property types. Each property type is associated with a sequence of actions, which are specified in the provided table.

<table>
<thead>
<tr>
<th>Property Type #1</th>
<th>Value Range #1</th>
<th>Sequence of actions #1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value Range #2</td>
<td>Sequence of actions #2</td>
</tr>
<tr>
<td></td>
<td>Value Range #3</td>
<td>Sequence of actions #3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property Type #N</th>
<th>Value Range</th>
<th>Sequence of actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The diagram also highlights the relationship between the property type and the feature converter, indicating a mapping process that converts property types into a feature vector.

**Property->Feature Converter**
Building the Feature Vector

Determine type handler and select action

CLR Property Handler

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>[203]++</td>
</tr>
<tr>
<td>1.1</td>
<td>[102]++</td>
</tr>
<tr>
<td>2.0</td>
<td>[100]++</td>
</tr>
<tr>
<td>4.0</td>
<td>[105]--</td>
</tr>
</tbody>
</table>

This process is repeated for every harvested property.
Linear Algebra, How I Missed You

\[
\begin{pmatrix}
\text{feature}_1 & \ldots & \text{feature}_{7000}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
A_{1,1} & \ldots & A_{1,256} \\
\vdots & \ddots & \vdots \\
A_{7000,1} & \ldots & A_{7000,256}
\end{pmatrix}
\rightarrow\tanh
\rightarrow
\begin{pmatrix}
B_{1,1} & \ldots & B_{1,256} \\
\vdots & \ddots & \vdots \\
B_{256,1} & \ldots & B_{256,256}
\end{pmatrix}
\rightarrow\tanh
\rightarrow
\begin{pmatrix}
C_{1,1} \\
\vdots \\
C_{256,1}
\end{pmatrix}
\rightarrow\text{sigmoid}
\rightarrow
(\text{score})
\]

Populated by property processing

The Model Core

[-1, 1]

Finally!
White/Black-Listing

Reduce and normalize features

Check Distance from Centroids

Reverse score if below defined threshold

Feature Y

Feature X

Centroid #n

Sample

Threshold
Rocket What?

Hmmm… This could be interesting, hold that thought
Let’s Pause and Hypothesise

- Attack the core mechanism
- Search for easily influenced properties

- Anything you can infer

- Attack the whitelisting mechanism
- Craft the PE to be “close enough” to a centroid
Strings Galore

```
for (int index = 0; index < this.imagePEFile_0.Strings.Length; ++index)
{
    if (!this.method_26(this.imagePEFile_0.Strings[index].S, 95088, 854066, 0))
        this.method_14(this.list_0[0], 15166118410741992125UL, 2847678, 0);
}
```

Location of the String Type handler

WOAH, that's a large handler!

Process property function
### Strings Galore, Contd.

<table>
<thead>
<tr>
<th>Hash (String)</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH(Str #1)</td>
<td>[203++] [423]--</td>
</tr>
<tr>
<td>HASH(Str #2)</td>
<td>[1020]++</td>
</tr>
<tr>
<td>HASH(Str #3)</td>
<td>[866]++</td>
</tr>
<tr>
<td>HASH(Str #4)</td>
<td>[53]++</td>
</tr>
<tr>
<td>HASH(Str #5)</td>
<td>[4500]++</td>
</tr>
<tr>
<td>HASH(Str #6)</td>
<td>[10]++</td>
</tr>
<tr>
<td>HASH(Str #7)</td>
<td>[453]++</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hash (String)</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH(Str #854063)</td>
<td>[23]++</td>
</tr>
<tr>
<td>HASH(Str #854064)</td>
<td>[6088]++</td>
</tr>
<tr>
<td>HASH(Str #854065)</td>
<td>[100]++</td>
</tr>
<tr>
<td>HASH(Str #854066)</td>
<td>[1335]++</td>
</tr>
<tr>
<td>HASH(Str #854067)</td>
<td>[64]++</td>
</tr>
<tr>
<td>HASH(Str #854068)</td>
<td>[12]++</td>
</tr>
<tr>
<td>HASH(Str #854069)</td>
<td>[6778]++</td>
</tr>
</tbody>
</table>
The Hypothesis

Strings have the potential for disproportionate impact on the feature vector.

The Whitelist provides a hint as to what type of executables are “good” (e.g. Rocket League) and may have been used to retrain the model at a later stage.

If we strip the strings from the good PEs and carefully inject them into a malicious payload, we may be able to fool the model, as they will overpower the effect of “negative” properties. Note that the model does not regard “attacker economics”.

Note that we are NOT aiming to fool the whitelisting mechanism, rather the main model!
<table>
<thead>
<tr>
<th>char1</th>
<th>char2</th>
<th>char3</th>
<th>char4</th>
<th>char5</th>
<th>char6</th>
<th>char7</th>
<th>char8</th>
<th>char9</th>
<th>char10</th>
</tr>
</thead>
<tbody>
<tr>
<td>66 1B</td>
<td>C9 66</td>
<td>23 C2</td>
<td>49 8B</td>
<td>D7 66</td>
<td>81 E1</td>
<td>00 04</td>
<td>66 0B</td>
<td>C8 0F</td>
<td>BF 45</td>
</tr>
<tr>
<td>24 28</td>
<td>4B CE</td>
<td>48 89</td>
<td>44 24</td>
<td>30 48</td>
<td>8B 45</td>
<td>00 00</td>
<td>66 89</td>
<td>4C 24</td>
<td>2B A0</td>
</tr>
<tr>
<td>89 4B</td>
<td>8B D8</td>
<td>48 3B</td>
<td>C1 0F</td>
<td>84 97</td>
<td>0C 00</td>
<td>00 00</td>
<td>6F B7</td>
<td>40 30</td>
<td>45 0A</td>
</tr>
<tr>
<td>07 04</td>
<td>45 0A</td>
<td>45 33</td>
<td>ED 41</td>
<td>3A FD</td>
<td>3B 45</td>
<td>3A 0F</td>
<td>B6 4B</td>
<td>3A 0F</td>
<td>B6 84</td>
</tr>
<tr>
<td>24 89</td>
<td>00 00</td>
<td>00 00</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This would never work, right?
Let’s have a look...
## Summon the Malware Hordes

### Malware Score Before Score After

<table>
<thead>
<tr>
<th>Malware</th>
<th>Score Before</th>
<th>Score After</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoinMiner</td>
<td>-826</td>
<td>884</td>
</tr>
<tr>
<td>Dridex</td>
<td>-999</td>
<td>996</td>
</tr>
<tr>
<td>Emotet</td>
<td>-923</td>
<td>625</td>
</tr>
<tr>
<td>Gh0stRAT</td>
<td>-975</td>
<td>998</td>
</tr>
<tr>
<td>Kovter</td>
<td>-999</td>
<td>856</td>
</tr>
<tr>
<td>Nanobot</td>
<td>971</td>
<td>999</td>
</tr>
<tr>
<td>Pushdo</td>
<td>-999</td>
<td>999</td>
</tr>
<tr>
<td>Qakbot</td>
<td>-998</td>
<td>991</td>
</tr>
<tr>
<td>Trickbot</td>
<td>-973</td>
<td>774</td>
</tr>
<tr>
<td>Zeus</td>
<td>-997</td>
<td>997</td>
</tr>
</tbody>
</table>

Tests on 384 samples from the Zoo repository:

- **88.54%** of malware passed as benign
- Average score before treatment = -0.92 (min is -1)
- Average score after mutation = 0.75 (max is 1)
- Average change in score = +1.67 (out of a range of 2).
Publication & Cylance’s Response

July 21st, Cylance’s Threat Vector

...researchers publicly disclosed a specific bypass of CylancePROTECT®. We verified the issue was not a universal bypass as reported, but rather a technique that allowed for one of the anti-malware components of the product to be bypassed in certain circumstances. The issue has been resolved for cloud-based scoring and a new agent will be rolled out to endpoints in the next few days.

We are still waiting for a fix for the SmartAV product...
Questions?
Thank You!

adi@skylightcyber.com