Kitsune

AN ENSEMBLE OF AUTOENCODERS FOR ONLINE NETWORK INTRUSION DETECTION

Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai
Neural Networks (NN) are great at detecting malicious packets

- Great results in literature (NNs can learn nonlinear complex patterns and behaviors)
- But, not so common in practice *(where is my SNORT plugin?)*

Existing NN solutions use supervised learning (e.g., classification):

1. Collect packets
2. Label packets: malicious or normal
3. Train deep NN on labeled data
4. Deploy the NN model to the device
5. Execute the model on each packet
6. When a new attack is discovered, go to #1
Neural Networks (NN) are great at detecting malicious packets

- Great results in literature (NNs can learn nonlinear complex patterns and behaviors)
- But, not so common in practice (where is my SNORT plugin?)

Existing NN solutions use supervised learning (e.g., classification):

1. Collect packets
2. Label packets: malicious or normal
3. Train deep NN on labeled data
4. Deploy the NN model to the device
5. Execute the model on each packet
6. When a new attack is discovered, go to #1
A *Kitsune*, in Japanese folklore, is a mythical fox-like creature that has a number of tails, can mimic different forms, and whose strength increases with experience.

So too, *Kitsune* has an ensemble of small neural networks (autoencoders), which are trained to mimic (reconstruct) network traffic patterns, and whose performance incrementally improves overtime.

**Unsupervised**: Anomaly detection, no labels!

**Online**: Incremental learning, incremental feature extraction

**Plug-and-Play**: On-site training, unsupervised learning

**Light-weight**: The NN uses a hierarchal architecture

Enables NN on network traffic

Enables realistic deployments e.g., routers
Kitsune Framework

NIDS

NIDS are Located on:
- Gateways/Routers
- Servers
- Dedicated Devices (e.g., PI attached to a mirror port)
Kitsune Framework

- Feature Extractor (FE)
- Anomaly Detector (AD)
- Packet Parser
- Packet Capturer

External Libs
- Kitsune
- KitNET
- Network
- Packet++
- scapy
- ...
**Kitsune Feature Extractor (FE)**

- FE uses **damped incremental statistics** to efficiently measure recent traffic patterns.

An unbounded stream of values \( S = \{x_1, x_2, \ldots \} \)

**Objective:** Compute the stats (\( \mu, \sigma, \ldots \)) over the recent history of \( S \), given limited memory and non-uniform sample rates (timestamps).

**Decay Factor:**
\( d \downarrow \lambda (t) = 2^t - \lambda t \)

### System Diagram

#### Decay Factor

\[
d \downarrow \lambda (t) = 2^t - \lambda t
\]

#### Incremental Statistic Object

\[
\mathcal{I}_S := (N, L_S, S_S, S_R)
\]

#### Basic Stats

\[
\mu = \frac{L_S}{N}, \quad \sigma = \sqrt{\frac{S_S}{N} - \left(\frac{L_S}{N}\right)^2},
\]

#### Update IS with \( x \downarrow i \):

\[
\mathcal{I}_S \leftarrow (N + 1, L_S + x \downarrow i, S_S + (x \downarrow i)^2, S_R + r \downarrow i r \downarrow j)
\]

### Calculation Table

<table>
<thead>
<tr>
<th>Type</th>
<th>Statistic</th>
<th>Notation</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D</td>
<td>Weight</td>
<td>( w )</td>
<td>( w )</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>( \mu_{S_i} )</td>
<td>( L_S/w )</td>
</tr>
<tr>
<td></td>
<td>Std.</td>
<td>( \sigma_{S_i} )</td>
<td>( \sqrt{SS/w - (LS/w)^2} )</td>
</tr>
<tr>
<td>2D</td>
<td>Magnitude</td>
<td>( |S_i, S_j| )</td>
<td>( \sqrt{\mu_{S_i}^2 + \mu_{S_j}^2} )</td>
</tr>
<tr>
<td></td>
<td>Radius</td>
<td>( R_{S_i, S_j} )</td>
<td>( \sqrt{(\sigma_{S_i}^2)^2 + (\sigma_{S_j}^2)^2} )</td>
</tr>
<tr>
<td></td>
<td>Approx. Covariance</td>
<td>( \text{Cov}_{S_i, S_j} )</td>
<td>( \frac{SR_{ij}}{w_i + w_j} )</td>
</tr>
<tr>
<td></td>
<td>Correlation Coefficient</td>
<td>( P_{S_i, S_j} )</td>
<td>( \frac{\text{Cov}<em>{S_i, S_j}}{\sigma</em>{S_i} \sigma_{S_j}} )</td>
</tr>
</tbody>
</table>
Kitsune Feature Extractor (FE)

5 Types of Streams:
- Potentially thousands of streams...
- 5 inc-stats each \( \lambda = \{5, 3, 1, 0.1, 0.01\} \)

Packet Sizes from a MAC-IP [3]
Packet Sizes from an IP [3]
Jitter of the traffic from an IP [3]

Packet Sizes between two IPs [7]
...between two Sockets [7]

Source Y
TCP
UDP

Dest. 1

Dest. 2

Dest. X
TCP
UDP

\( x \in \mathbb{R} \times 23 \times 5 = 115 \)
The **KitNET** Anomaly Detector

Anomaly Detection with an Autoencoder

- An Autoencoder is a NN which is trained to reproduce its input after compression.
- There are two phases:
  - **Train**
  - **Execute**

![Diagram of an Autoencoder](image)

**Reconstruction Error**

\[
\text{RMSE} (\hat{x}, y) = \sqrt{\frac{\sum_{i=1}^{n} (x_i - y_i)^2}{n}}
\]

**Error:**

- **Low value:** $x$ is normal.
- **High value:** $x$ is abnormal (does not fit known concepts).
The KitNET Anomaly Detector

Why not one massive deep autoencoder?

- Curse of dimensionality!
- Train/Execute Complexity

Our Solution:

Each autoencoder receives a group of correlated features
How do you find the groupings online?
The KitNET Anomaly Detector

- For the first $N$ observations ($x$), **incrementally** update a correlation distance matrix $D = [D_{ij}] = 1 - (x_i - x_i) \cdot (x_j - x_j) / \| (x_i - x_i) \|_2 \| (x_j - x_j) \|_2$

- Perform **one-time** agglomerative hierarchical clustering on $D$ (fast)

- Cut the dendrogram so that no cluster is larger than $m$ (max autoencoder size)

- Each discovered cluster represents an autoencoder
Kitsune NIDS

External Libs

Raw Packet

Packet Capturer

Packet Parser

Kitsune

Damped Incremental Statistics

Feature Extractor (FE)

Feature Mapper (FM)

Ensemble Layer

Output Layer

Log

KitNET

No more that one instance (packet) is stored in memory at a time.
Experimental Results

- **Networks:**
  - Surveillance
  - IoT

- **Algorithms:**
  - **Signature-based:** Suricata with over 13,465 emerging threat rules
  - **Anomaly-based:**
    - **Batch:** GMM, Isolation Forest
    - **Online:** pcStream & iGMM
## Experimental Results

### Attacks

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Attack Name</th>
<th>Tool</th>
<th>Description: The attacker...</th>
<th>Violation</th>
<th>Vector</th>
<th># Packets</th>
<th>Train [min.]</th>
<th>Execute [min.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recon.</td>
<td>OS Scan</td>
<td>Nmap</td>
<td>...scans the network for hosts, and their operating systems, to reveal possible vulnerabilities.</td>
<td>C</td>
<td>1</td>
<td>1,697,851</td>
<td>33.3</td>
<td>18.9</td>
</tr>
<tr>
<td>Fuzzing</td>
<td>SFuzz</td>
<td></td>
<td>...searches for vulnerabilities in the camera’s web servers by sending random commands to their egis.</td>
<td>C</td>
<td>3</td>
<td>2,244,139</td>
<td>33.3</td>
<td>52.2</td>
</tr>
<tr>
<td>Man in the Middle</td>
<td>Video Injection</td>
<td>Video Jack</td>
<td>...injects a recorded video clip into a live video stream.</td>
<td>C, I</td>
<td>1</td>
<td>2,472,401</td>
<td>14.2</td>
<td>19.2</td>
</tr>
<tr>
<td>ARP MitM</td>
<td>Ettercap</td>
<td></td>
<td>...intercepts all LAN traffic via an ARP poisoning attack.</td>
<td>C</td>
<td>1</td>
<td>2,504,267</td>
<td>8.05</td>
<td>20.1</td>
</tr>
<tr>
<td>Active Wiretap</td>
<td>Raspberry PI 3B</td>
<td></td>
<td>...intercepts all LAN traffic via active wiretap (network bridge) covertly installed on an exposed cable.</td>
<td>C</td>
<td>2</td>
<td>4,554,925</td>
<td>20.8</td>
<td>74.8</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>SSDP Flood</td>
<td>Saddam</td>
<td>...overloads the DVR by causing cameras to spam the server with UPnP advertisements.</td>
<td>A</td>
<td>1</td>
<td>4,077,266</td>
<td>14.4</td>
<td>26.4</td>
</tr>
<tr>
<td>SYN DoS</td>
<td>Hping3</td>
<td></td>
<td>...disables a camera’s video stream by overloading its web server.</td>
<td>A</td>
<td>1</td>
<td>2,771,276</td>
<td>18.7</td>
<td>34.1</td>
</tr>
<tr>
<td>SSL Renegotiation</td>
<td>THC</td>
<td></td>
<td>...disables a camera’s video stream by sending many SSL renegotiation packets to the camera.</td>
<td>A</td>
<td>1</td>
<td>6,084,492</td>
<td>10.7</td>
<td>54.9</td>
</tr>
<tr>
<td>Botnet Malware</td>
<td>Mirai</td>
<td>Tcnet</td>
<td>...infected IoT with the Mirai malware by exploiting default credentials, and then scans for new vulnerable victims network.</td>
<td>C, I</td>
<td>X</td>
<td>764,137</td>
<td>52.0</td>
<td>66.9</td>
</tr>
</tbody>
</table>
### Experimental Results

#### Area Under the Curve (AUC) - *Higher is better*

<table>
<thead>
<tr>
<th></th>
<th>ARP</th>
<th>Fuzzing</th>
<th>Mirai</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Scan</td>
<td>0.59835</td>
<td>0.97670</td>
<td>0.58423</td>
</tr>
<tr>
<td>SSL F.</td>
<td>0.99648</td>
<td>1.00000</td>
<td>0.99648</td>
</tr>
<tr>
<td>SSL R.</td>
<td>0.69096</td>
<td>0.73100</td>
<td>0.69096</td>
</tr>
<tr>
<td>SYN DoS</td>
<td>0.69560</td>
<td>0.58948</td>
<td>0.69560</td>
</tr>
<tr>
<td>Video Inj.</td>
<td>0.72964</td>
<td>0.57875</td>
<td>0.72964</td>
</tr>
<tr>
<td>Wiretap</td>
<td>0.50000</td>
<td>0.50000</td>
<td>0.50000</td>
</tr>
</tbody>
</table>

#### Equal Error Rate (EER) - *Lower is better*

<table>
<thead>
<tr>
<th></th>
<th>ARP</th>
<th>Fuzzing</th>
<th>Mirai</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Scan</td>
<td>0.59826</td>
<td>0.29056</td>
<td>0.69560</td>
</tr>
<tr>
<td>SSL F.</td>
<td>0.43381</td>
<td>0.65969</td>
<td>0.45329</td>
</tr>
<tr>
<td>SSL R.</td>
<td>0.00016</td>
<td>0.00016</td>
<td>0.00016</td>
</tr>
<tr>
<td>SYN DoS</td>
<td>0.45876</td>
<td>0.23274</td>
<td>0.45876</td>
</tr>
<tr>
<td>Video Inj.</td>
<td>0.00016</td>
<td>0.00016</td>
<td>0.00016</td>
</tr>
<tr>
<td>Wiretap</td>
<td>0.62166</td>
<td>0.62166</td>
<td>0.62166</td>
</tr>
</tbody>
</table>

### Graphs

- **AUC (Area Under the Curve)**: The AUC is a measure used in machine learning to evaluate the performance of a classifier. A higher AUC indicates better performance. The graphs show the AUC values for different techniques and scenarios.
- **EER (Equal Error Rate)**: The EER is a measure of the error rate where the false positive rate equals the false negative rate. A lower EER is better. The graphs display EER values for various conditions.

The graphs illustrate the performance metrics for different techniques and their implications in cybersecurity.
Experimental Results

- ~20,000 packets/sec on a PI
- ~140,000 packets/sec on a desktop PC
In the past, NNs on NIDS were used for the task of **classification**.

We propose using NNs for the task of **anomaly detection**
- Eliminates the need for labeling data (endless traffic & unknown threats)
- Enables plug-and-play

**Kitsune Achieves this by,**
- Efficient feature extraction
- Efficient anomaly detection (KitNET)
Thank you!

Source code: https://github.com/ymirsky/KitNET-py

Contact: yisroel@post.bgu.ac.il