A Deep Learning Based Anomaly Detection Approach for Intelligent Autonomous Systems

Miguel Villarreal-Vasquez and Bharat Bhargava

Department of Computer Science, Purdue University, West Lafayette, IN, USA

MOTIVATION

Given this sequence

<table>
<thead>
<tr>
<th>Event</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low</td>
</tr>
<tr>
<td>2</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>High</td>
</tr>
</tbody>
</table>

Should this sequence occur?

<table>
<thead>
<tr>
<th>Event</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low</td>
</tr>
<tr>
<td>2</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>High</td>
</tr>
</tbody>
</table>

Legend:
Each color bar represents one of the all possible events (system calls) in the system.

- Advanced modern exploits characterize by their sophistication in stealthy attacks.
- Code-reuse attacks such as return-oriented programming and memory disclosure attacks allow attackers executing malicious instruction sequences on victim systems without injecting external code.
- This research proposes a new Deep Learning based anomaly detection technique that probabilistically models program control flows for behavioral reasoning and live monitoring.
- We aim to answer the binary classification problem of given a sequence of events \( e_1, e_2, \ldots, e_k \) whether or not the sequence should occur?
- An event \( e_i \) in the sequence is a function call in a given trace.

THREAT MODEL

The Eternal War

- Attack:
  - Code Injection (e.g., Buffer Overflow)
  - WiX (e.g., Windows DEP)
  - Code Reuse (e.g., ROP)
  - ASLR and variants
  - Memory Disclosure (JIT-ROP): Enables Code Reuse
  - Code Re-Randomization / Deep Learning based Anomaly Detection

- Defense:

ANOMALY DETECTION ALGORITHM

- We defined an event as a function call. Each possible function call must be identified as they will form the vocabulary of events.
- The dynamic code behaviors can be learned by training the model with non-malicious program traces.
- At any time \( t \) each possible event (system call or library call) in the system is assigned a probability estimated with respect to the sequence of events observed until time \( t-1 \).
- At classification, the decision is made with respect to a pre-defined threshold of the \( K \)-top most probably sequences.

\[
\text{Input: Sequence of events in the system} \quad \text{Output: normal or anomalous}
\]

1. Define a finite set \( E \) of events \( e_1, e_2, \ldots, e_N \) in the system. Events occur in a time-series fashion.
2. At time \( t-1 \), given an observed series of events \( \{e_1^i, e_2^i, \ldots, e_t^i\} \) (with \( i = 1, 2, \ldots, n \)) find the set \( K \) of the top \( k \) events to occur in time \( t \).
3. At time \( t \), the sequence \( \{e_1^t, e_2^t, \ldots, e_t^t, e_k^t\} \) is non-anomalous if \( e_i^t \in K \), otherwise anomalous.

CONTRIBUTIONS

- Systems can be trained with data of benign flows only in isolation (the data is full available.)
- Flow sensitive anomaly detection. Given the execution paths \( P_1, P_2 \) and \( P_3 \) our technique captures their occurrence probabilities, vital for high-precision anomaly detection.

IMPLEMENTATION

MODEL

- Model: Based on Recurrent Neural Networks (RNN): Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU)
- Programming Language: Python
- Computing Library: PyTorch
- Parallel Computing Platform: CUDA

CONTRIBUTIONS

- Code Injection
- Code Reuse
- Memory Disclosure

Can be caused by:
- Human error (e.g., unauthorized use or operation of the program)
- Software flaws (e.g., buffer overflow vulnerabilities)
- Attacks by remote attackers
- Malicious insiders (e.g., through drive-by downloads)

REFERENCE


Acknowledgement: This research is supported by NGC Research Consortium. We collaborated with Paul Conoval, Donald Steiner and Jason Kobes.