Malware Detection in Cloud Infrastructures using Convolutional Neural Networks

Mahmoud Abdelsalam, Ram Krishnan, Yufei Huang and Ravi Sandhu

Institute for Cyber Security,
Center for Security and Privacy Enhanced Cloud Computing,
Department of Computer Science,
Department of Electrical and Computer Engineering
University of Texas at San Antonio

June 11, 2018

© Mahmoud Abdelsalam

Outline

- ➢ Introduction and Motivation
- ➢ Mislabeling Problem
- ➢ Convolutional Neural Networks (CNN) Overview
- ➢ Methodology
- ➢ Experimental Setup
- ➢ Results

Methodology - CNN Input Issues

- ➢ Cloud malware injection is a threat where an attacker injects a malware to manipulate the victim’s Virtual Machine (VM).
- ➢ Static analysis (code is analyzed) vs **dynamic analysis** (behavior is analyzed).
- ➢ The underestimated **mislabeling problem** in a big challenge.

© Mahmoud Abdelsalam

Methodology - CNN Input Issues

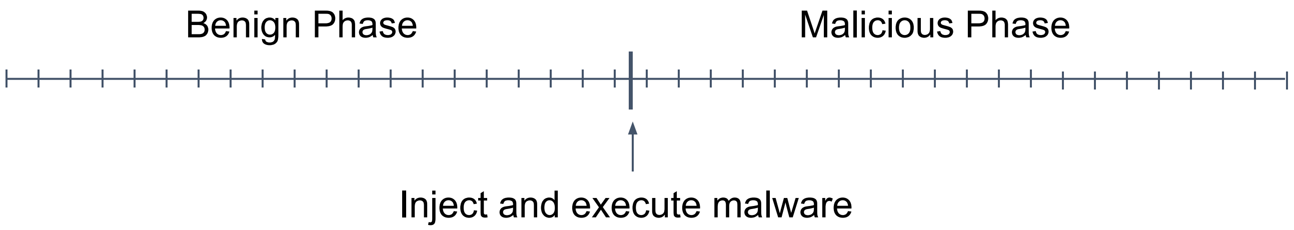
- Cloud malware injection is a threat where an attacker injects a malware to manipulate the victim's Virtual Machine (VM).
- Static analysis (code is analyzed) vs **dynamic analysis** (behavior is analyzed).
- The underestimated **mislabeling problem** in a big challenge.

Goals:

- The feasibility of applying CNN to VMs malware detection using fine-grained process performance metrics.

- Tackling the mislabeling problem by using 3d CNNs.

Mislabeling Problem



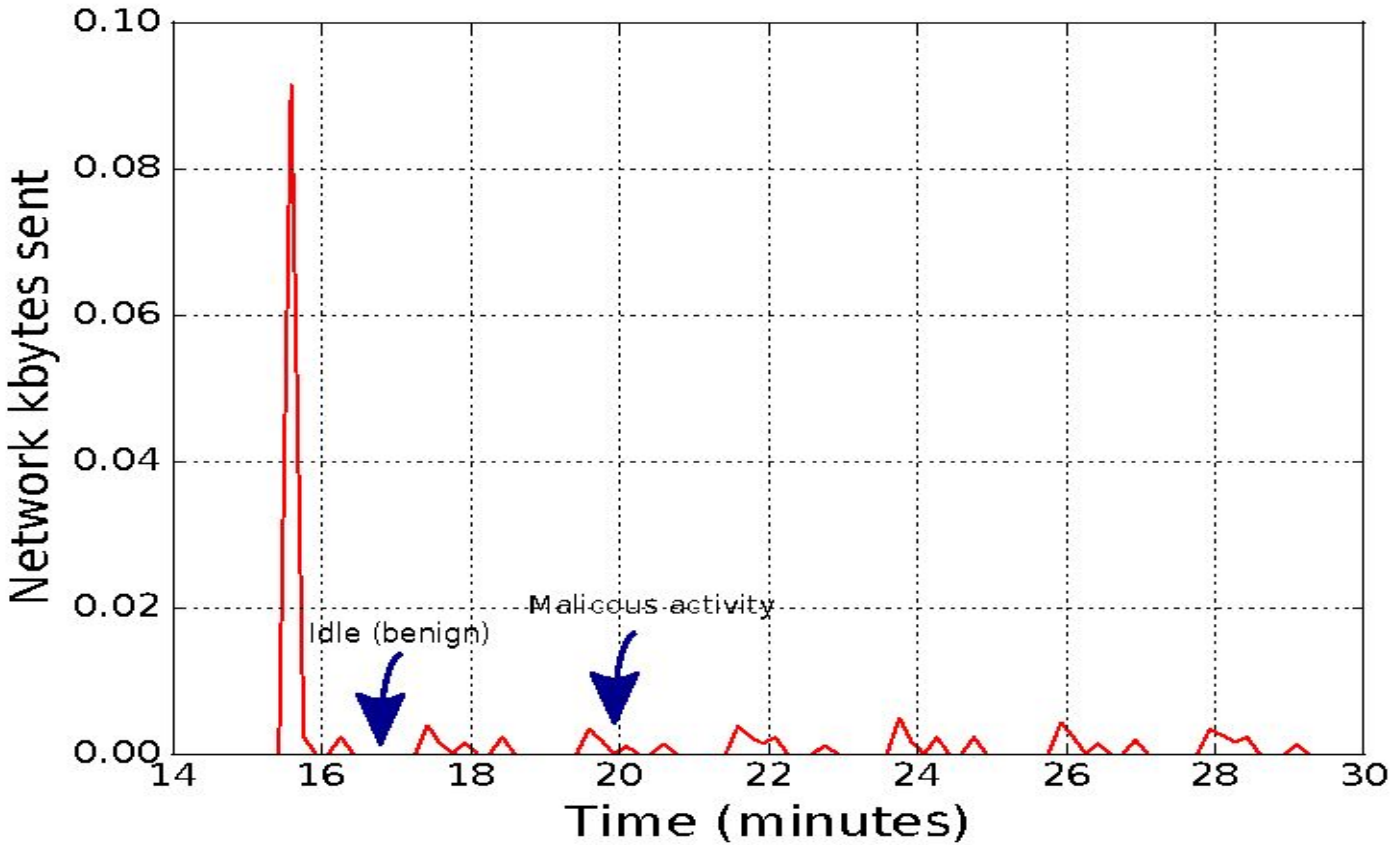
During the training phase, there is no guarantee that a malware exhibited malicious behavior.

- A malware may never show a malicious activity during the training phase at all.

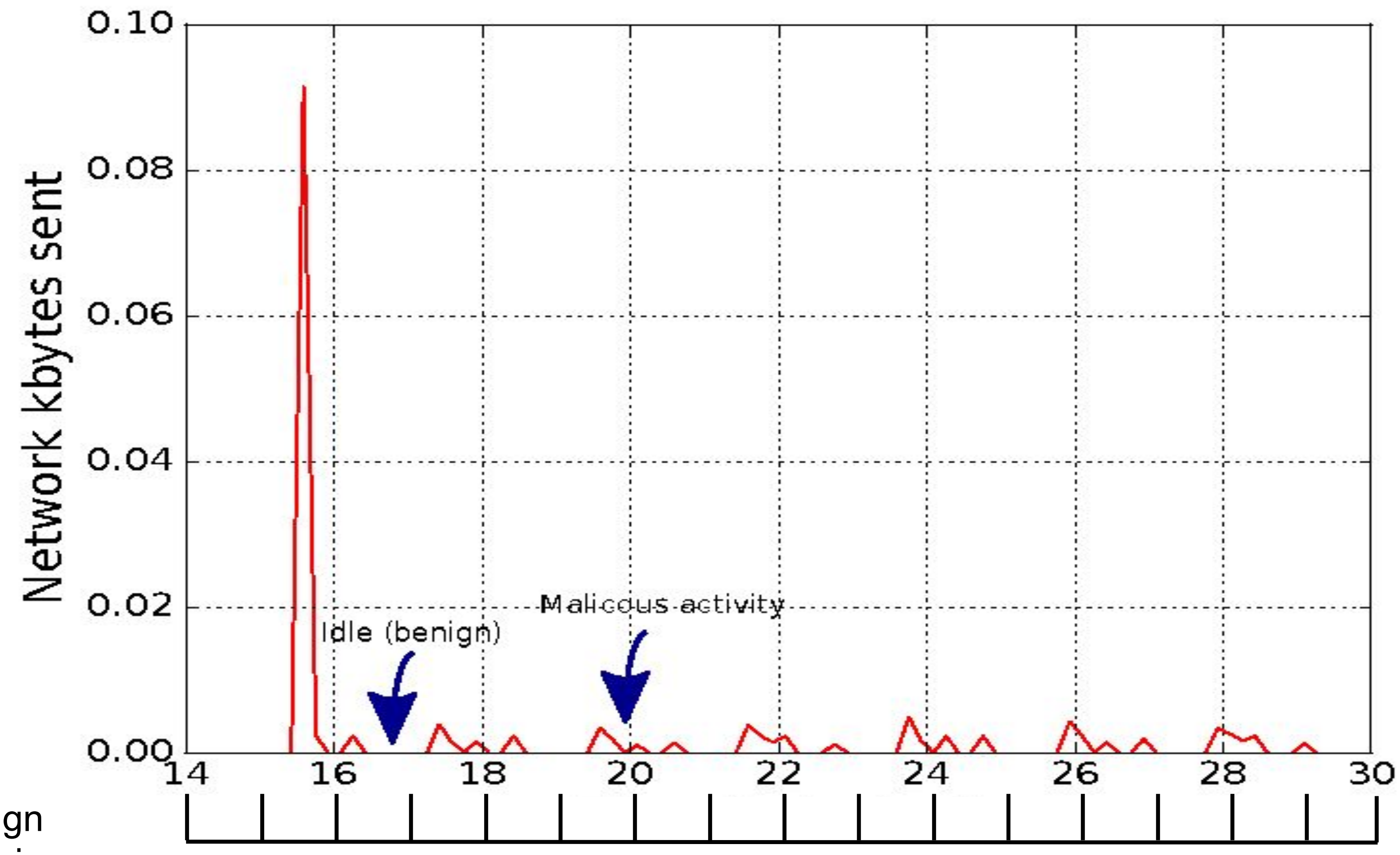
+ More common scenario is when a malware periodically (e.g., every 1 minute) performs malicious activities such as stealing and sending some information to its Command and Control servers (C&Cs).

© Mahmoud Abdelsalam

Mislabeling Problem



Mislabeling Problem

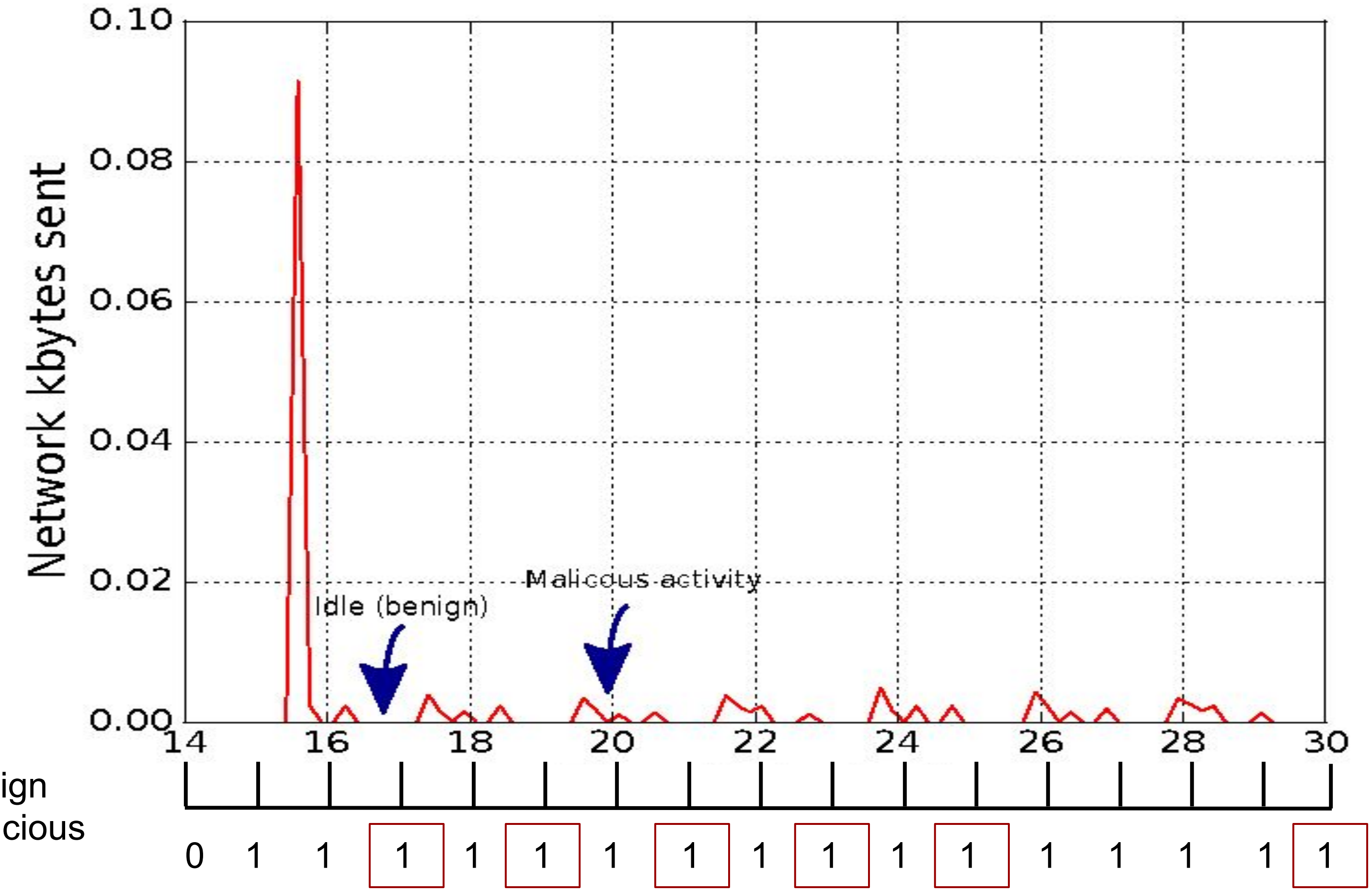


0 – Benign
1 – Malicious

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

© Mahmoud Abdelsalam

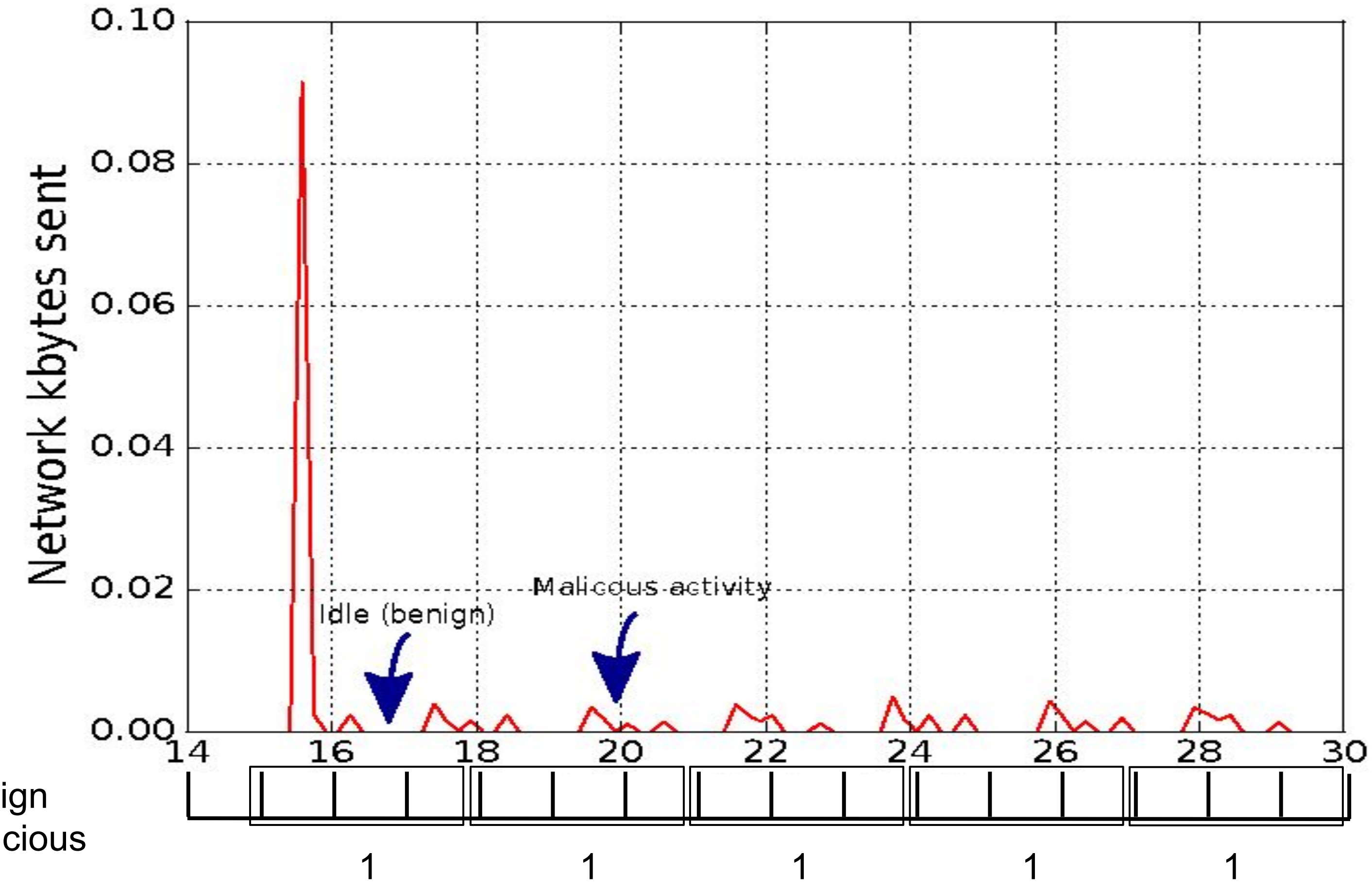
Mislabeling Problem



0 – Benign

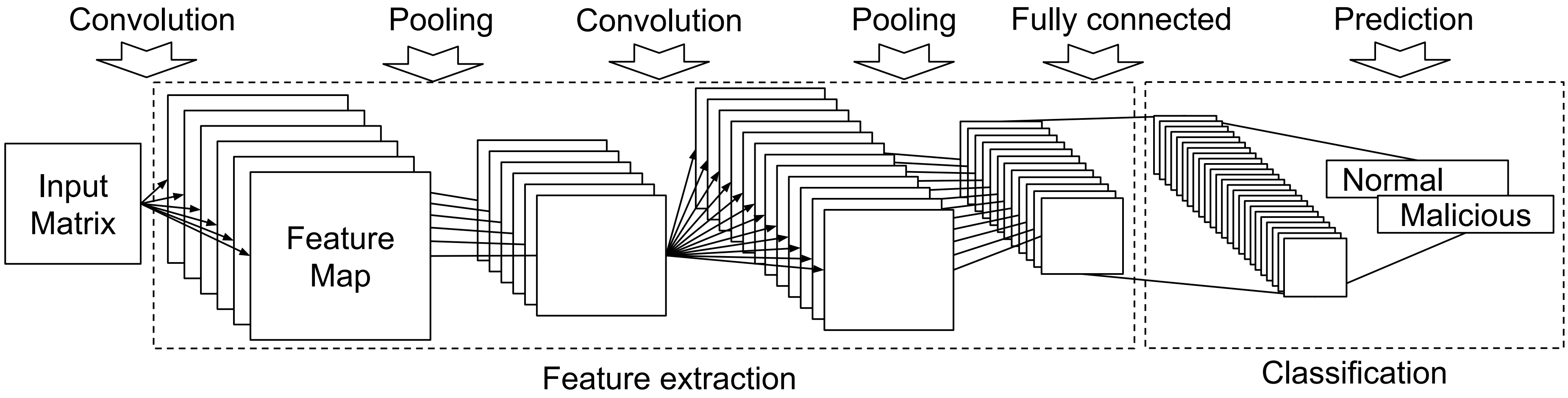
1 – Malicious

Mislabeling Problem



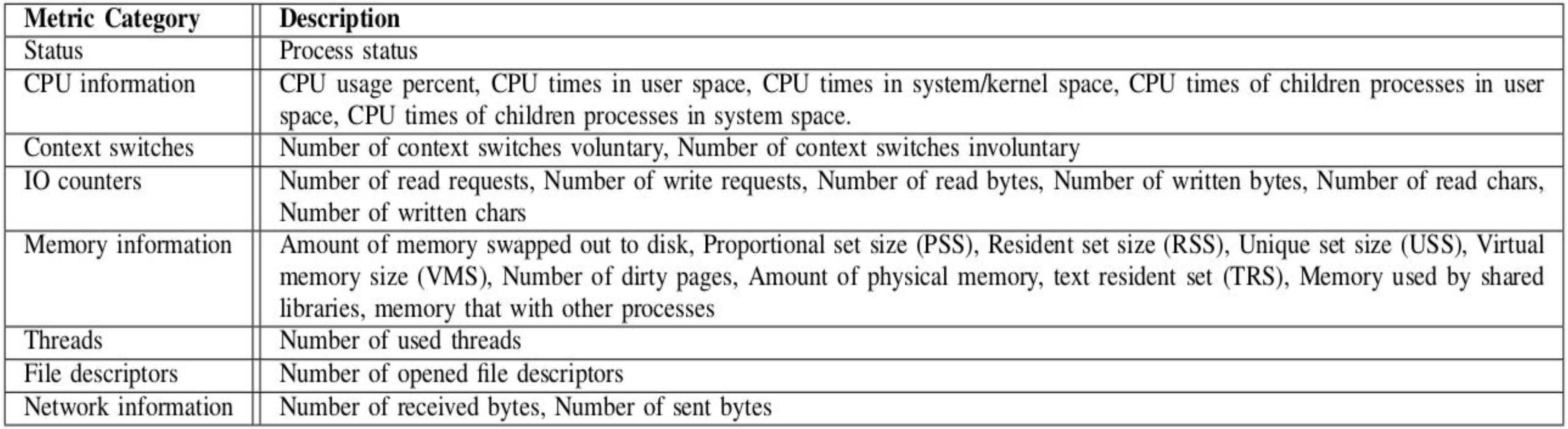
0 – Benign
1 – Malicious

CNN Overview



Methodology - Performance Metrics

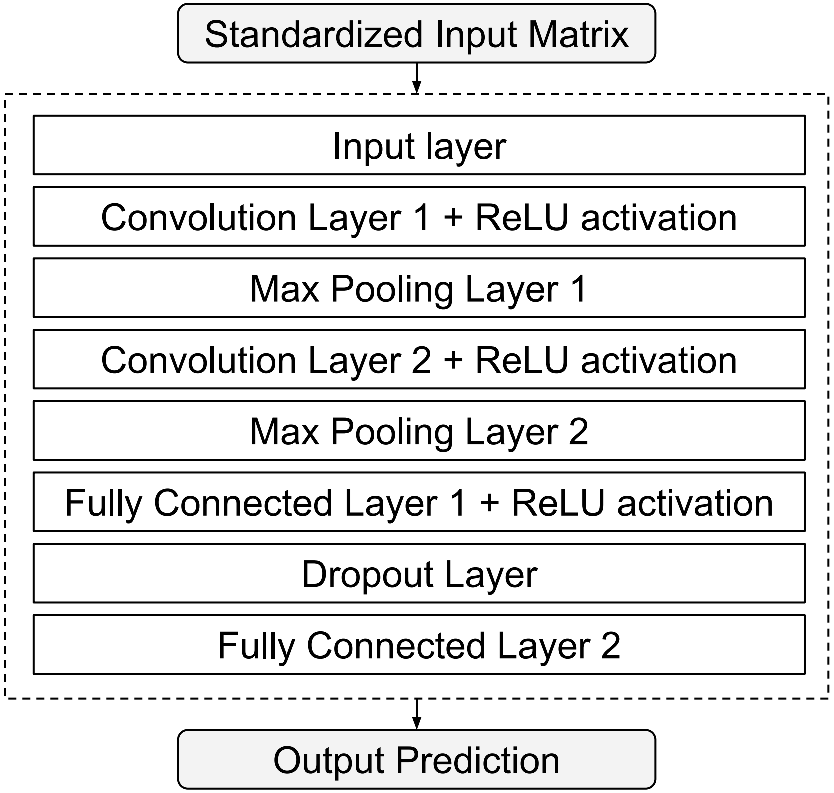
- ➢ We use performance metrics as a way of defining a process behavior.
- ➢ 28 process-level performance metrics.
- ➢ These metrics can easily be fetched through the hypervisor.



Metric Category	**Description**
Status	Process status
CPU information	CPU usage percent, CPU times in user space, CPU times in system/kernel space, CPU times of children processes in user space, CPU times of children processes in system space.
Context switches	Number of context switches voluntary, Number of context switches involuntary
IO counters	Number of read requests, Number of write requests, Number of read bytes, Number of written bytes, Number of read chars, Number of written chars
Memory information	Amount of memory swapped out to disk, Proportional set size (PSS), Resident set size (RSS), Unique set size (USS), Virtual memory size (VMS), Number of dirty pages, Amount of physical memory, text resident set (TRS), Memory used by shared libraries, memory that with other processes
Threads	Number of used threads
File descriptors	Number of opened file descriptors
Network information	Number of received bytes, Number of sent bytes

© Mahmoud Abdelsalam

Methodology - CNN Model



Standardized Input Matrix

Input layer

Convolution Layer 1 + ReLU activation

Max Pooling Layer 1

Convolution Layer 2 + ReLU activation

Max Pooling Layer 2

Fully Connected Layer 1 + ReLU activation

Dropout Layer

Fully Connected Layer 2

Output Prediction

Methodology - 2D CNN Input

We represent each sample as an image (2d matrix) which will be the input to the CNN.

Consider a sample x_t at a particular time t, that records n features (performance metrics) per process for m processes in a VM:

$$\mathbf{X}_t = \left[\begin{array}{ccccc} & f_1 & f_2 & \dots & f_n \\ p_1 & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_m & \vdots & \vdots & \dots & \vdots \end{array}\right]$$

Methodology - CNN Input Issues

- CNN requires the same process to remain in the same row in each sample.
- The CNN in computer vision takes fixed-size images as inputs, so the number of features and processes must be predetermined.

Use the **max** process identification number (PID) which is set by the OS?

- The limit (max number of PIDs) is defined in /proc/sys/kernel/pid_max which is usually 32k.
- Huge input matrix!
- Change the max PID number defined?
  - Kernel confusion if wrap around happened too often.

- there is no guarantee that, for instance, a process with a PID 1000 at a particular time is going to be the same process at a later time.

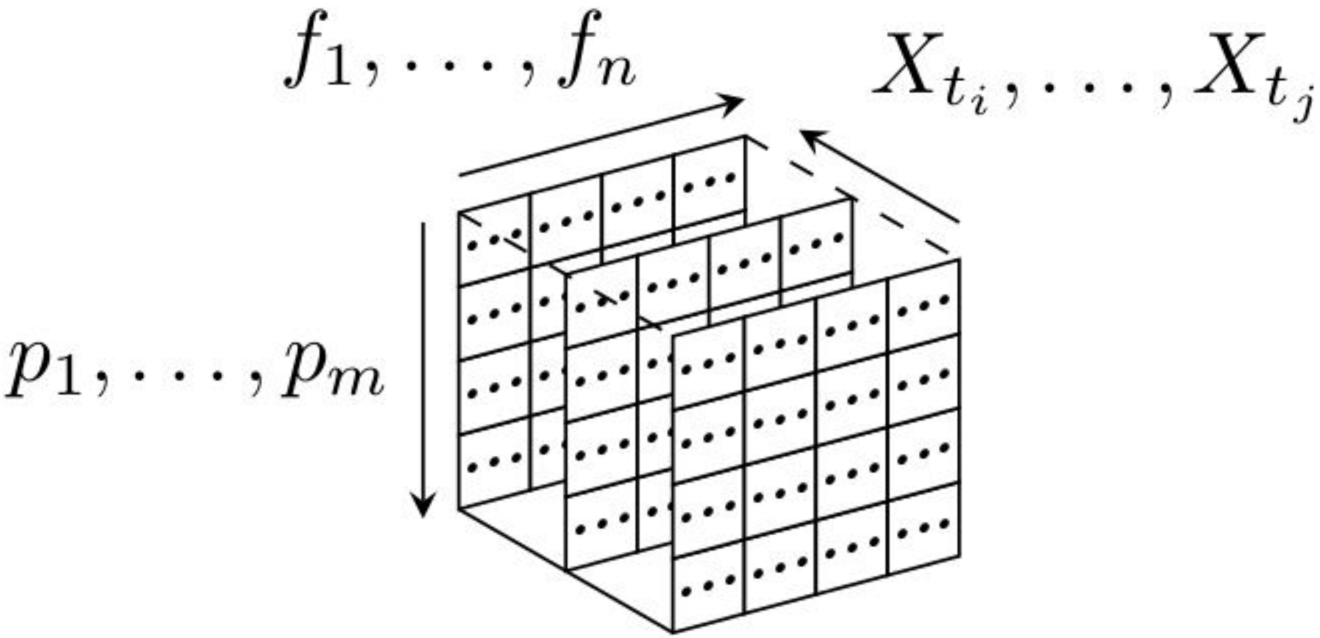
Methodology - CNN Input Issues

- We define a process, referred to as unique process, by a 3-tuple:
  - process name
  - command line used to run process
  - hash of the process binary file (if applicable)

- We set the maximum number of unique processes to 120 to accommodate for newly created unique processes.

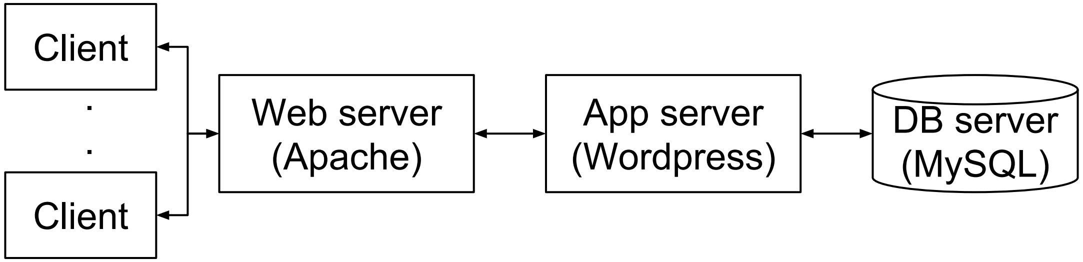
Methodology - 3D CNN Input

The 3d CNN model input includes multiple samples over a time window. The input matrix is:

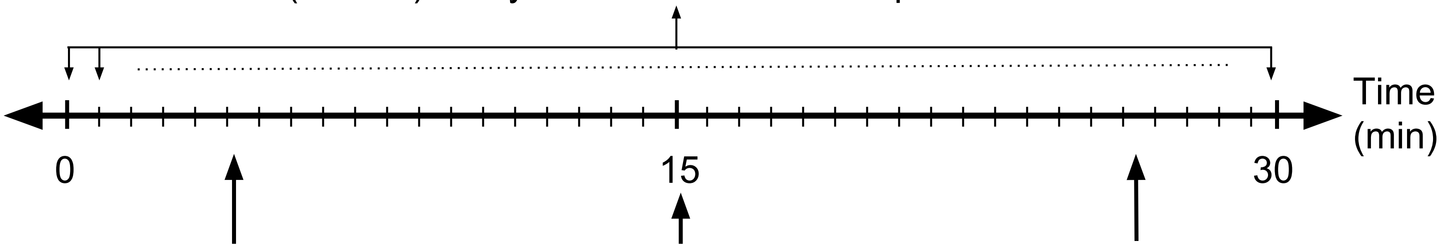


Experimental Setup

- Our experiments were conducted on Openstack.
- To simulate a real world scenario, we used a 3-tier web architecture and a self-similar traffic gen. (on/off Pareto) is used.



- Data collection:



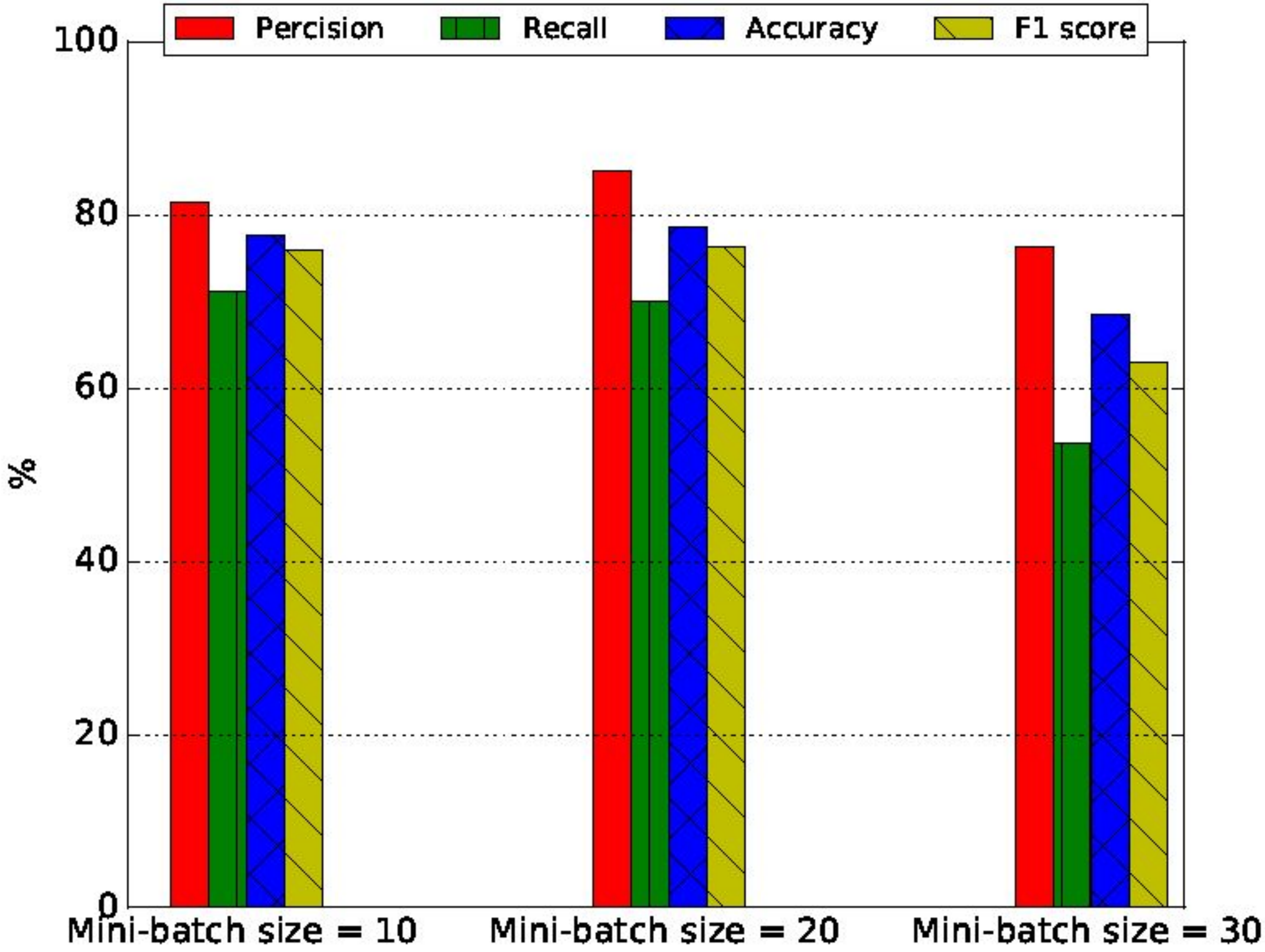
Collect 28 different process performance metrics (Table I) every 10 seconds for ≃ 100 processes

Clean phase

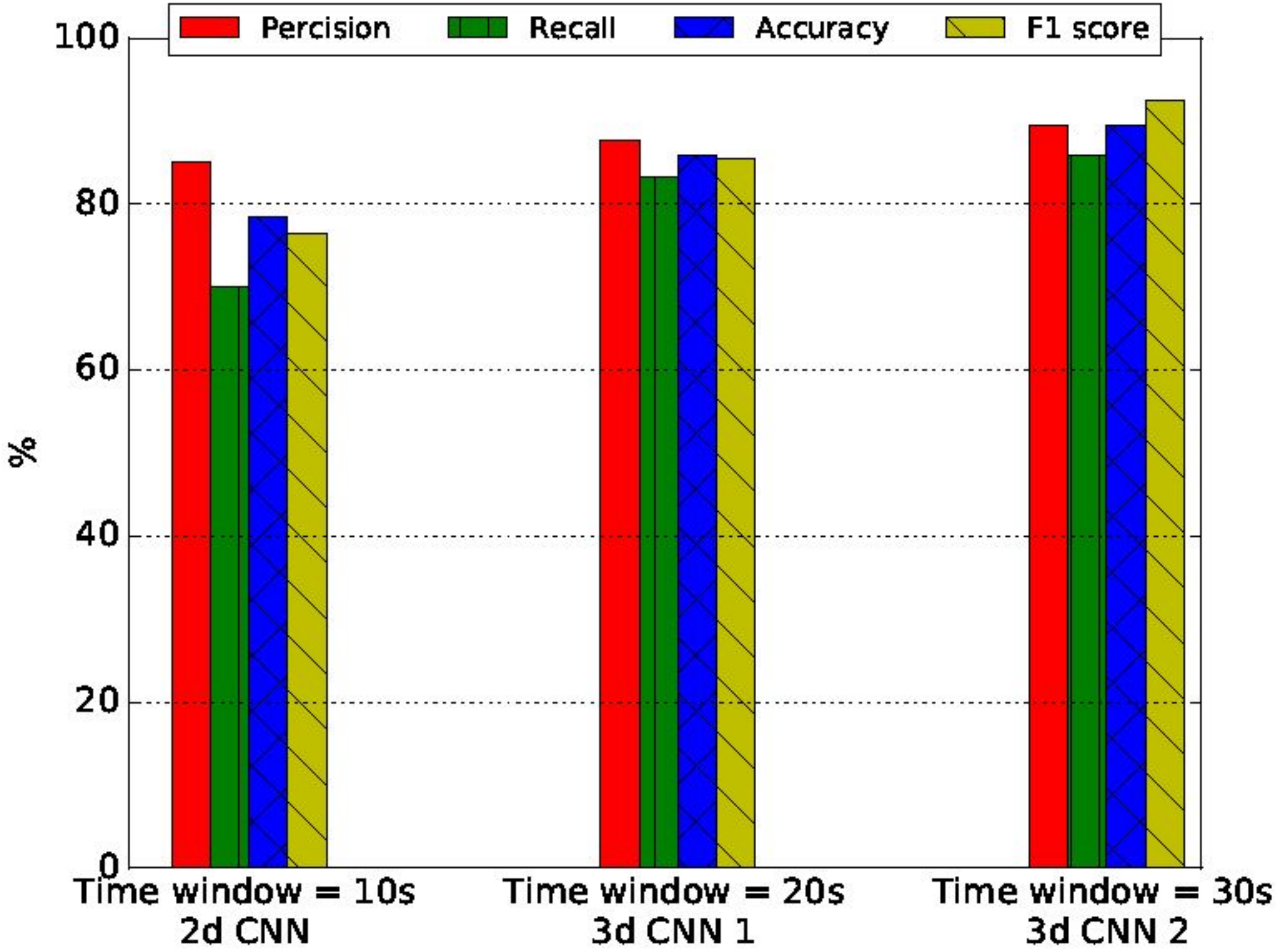
Malware injection Point.
25 Malware executables are injected (one per experiment).

Period of potential malware activity prone to mislabeling problem

2D CNN Results



3D CNN Results



Questions/Comments