I·C·S
The Institute for Cyber Security

C·SPECC
Center for Security and Privacy Enhanced Cloud Computing

# Online Malware Detection in Cloud Auto-Scaling Systems Using Shallow Convolutional Neural Networks

Mahmoud Abdelsalam, Ram Krishnan 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**

**In Proceedings 31st Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec), Charleston, SC, July 15-17, 2019**



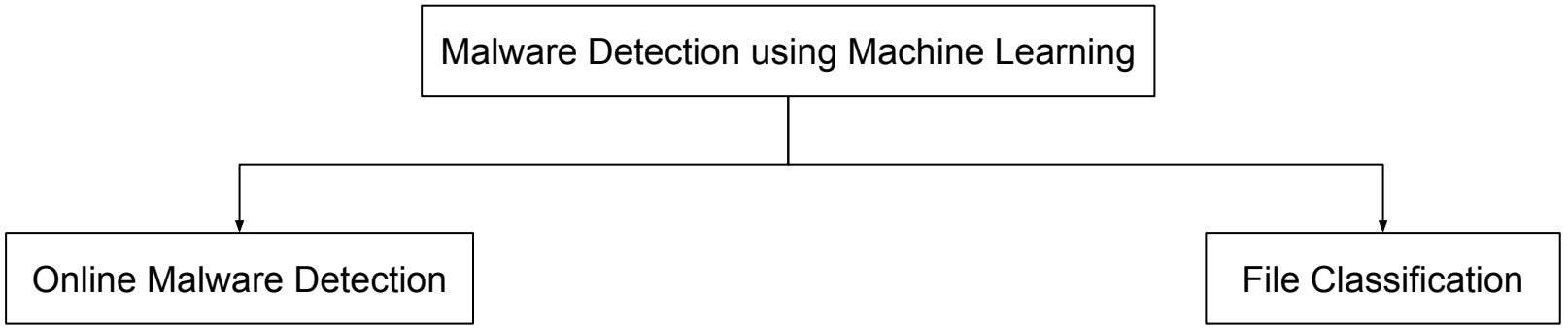
*World Leading Research with Real World Impact!*

UTSA Computer Science

# Introduction and Motivation

# Malware Detection Classification

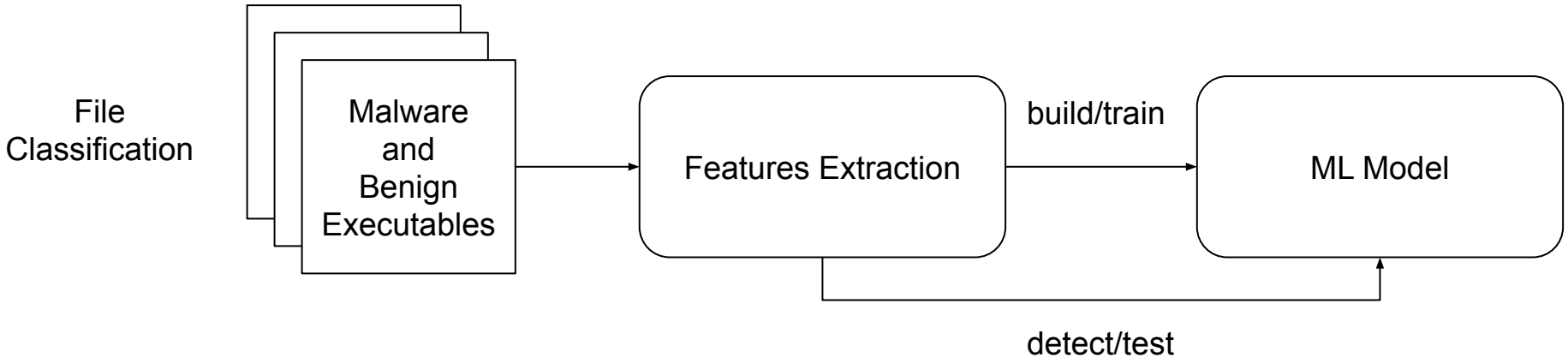





1. File classification:
   - Given a file/executable, classify if it's a malware or not by running it and observing its behavior.
   - You have a file as a suspect.
   - You don't keep monitoring them once they are clean.
2. Online malware detection:
   - Assume that the malware got into the system and is executing.
   - You keep monitoring the system's behavior for malware detection.
   - You don't just focus on a given file, but the entire system (processes).

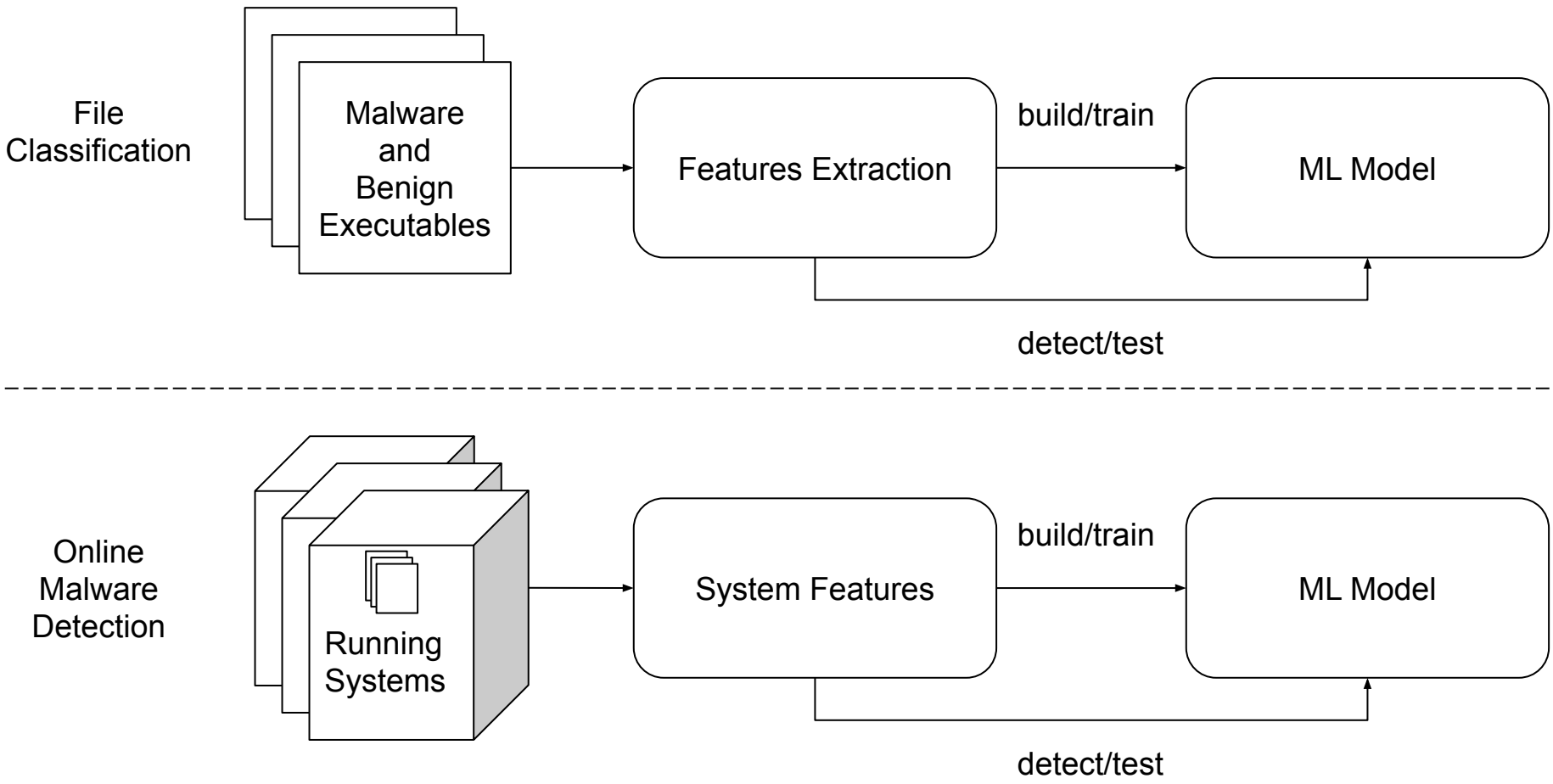
# Malware Detection using ML





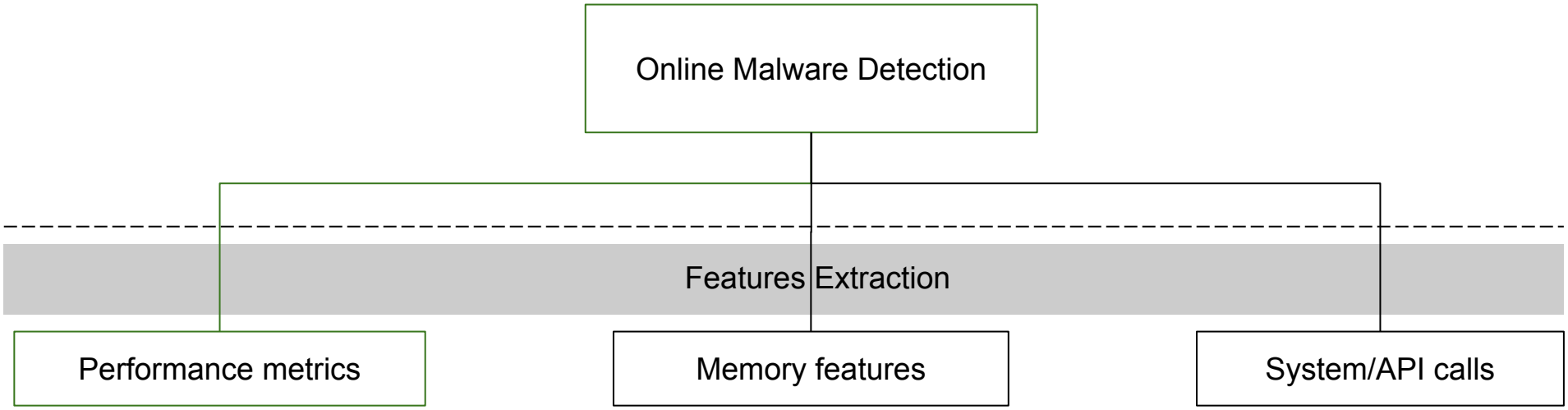
# Malware Detection using ML





# Motivation






What makes an approach cloud-specific?

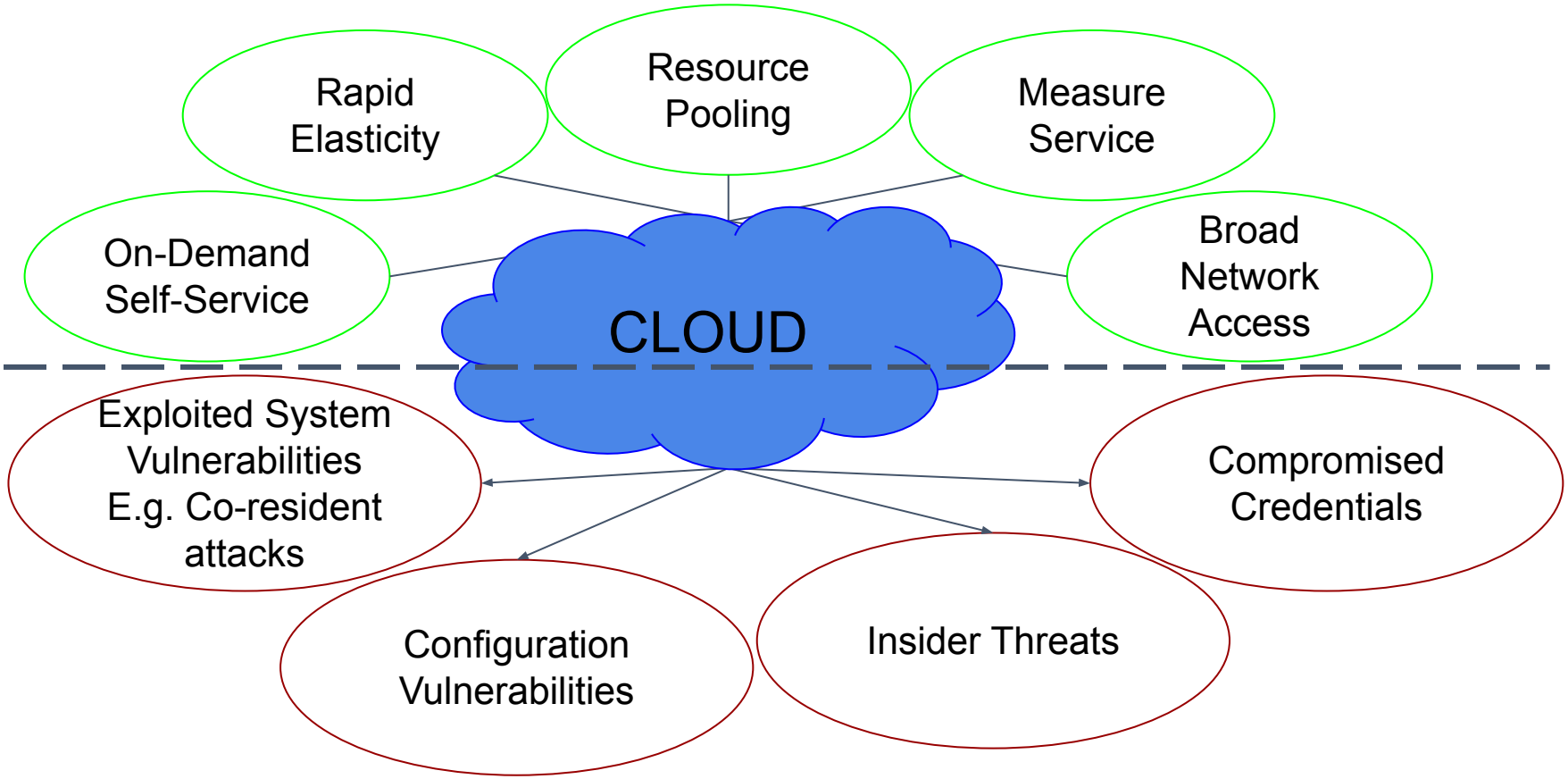
Most, if not all, **cloud-specific** research:

✔ Restrict the selection of features to those that can only be fetched through the hypervisor.

✘ Leverage cloud characteristics for online malware detection.

# Motivation (cont.)





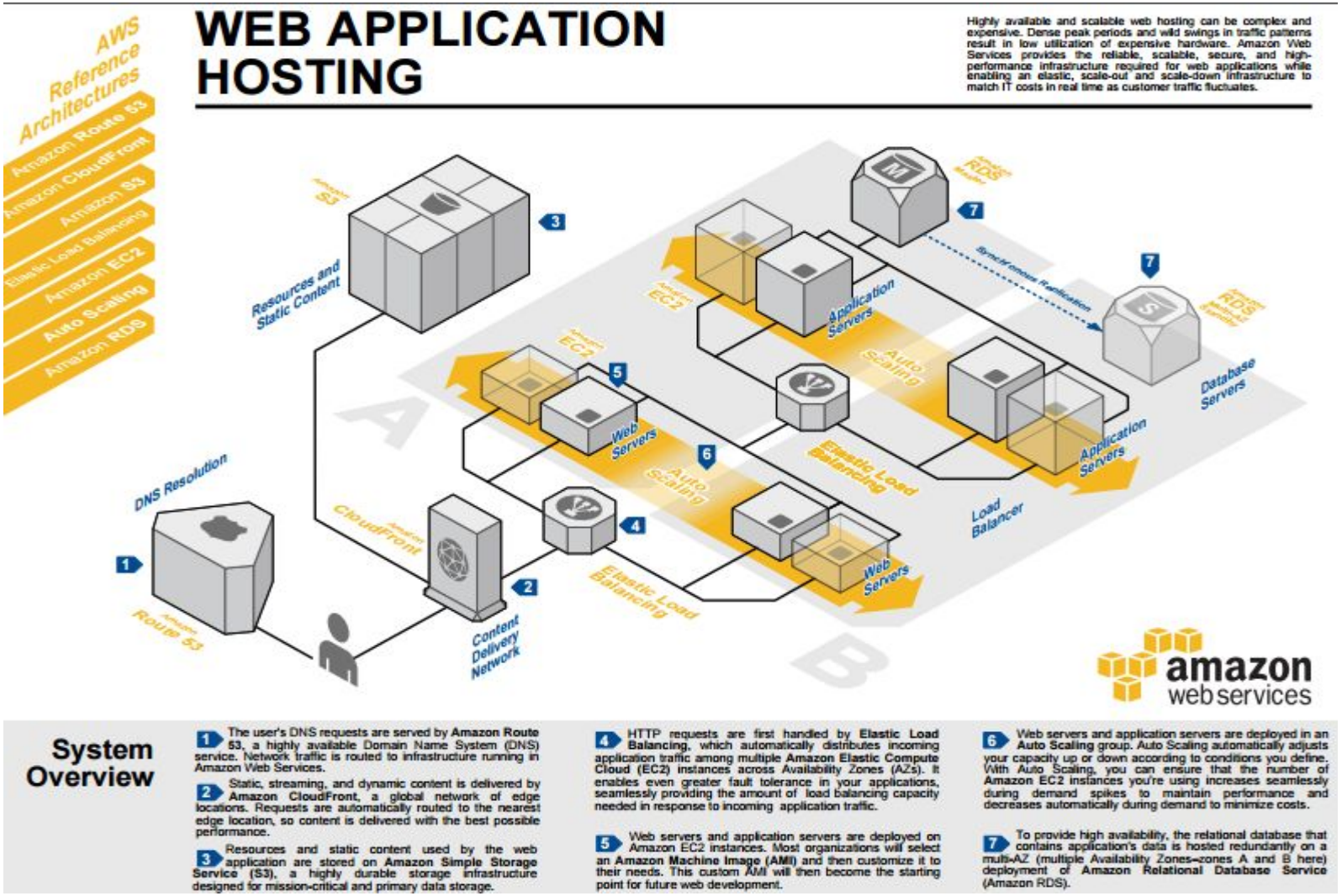

*Can we leverage cloud characteristics for online malware detection?* **"Auto-Scaling"**

**Goal: Leverage auto-scaling for online malware detection by:**

- Using 2d CNN to learn processes behavior of multiple VMs.
- Introducing a novel approach of pairing samples to accommodate for correlations between VMs.

# 3-tier example



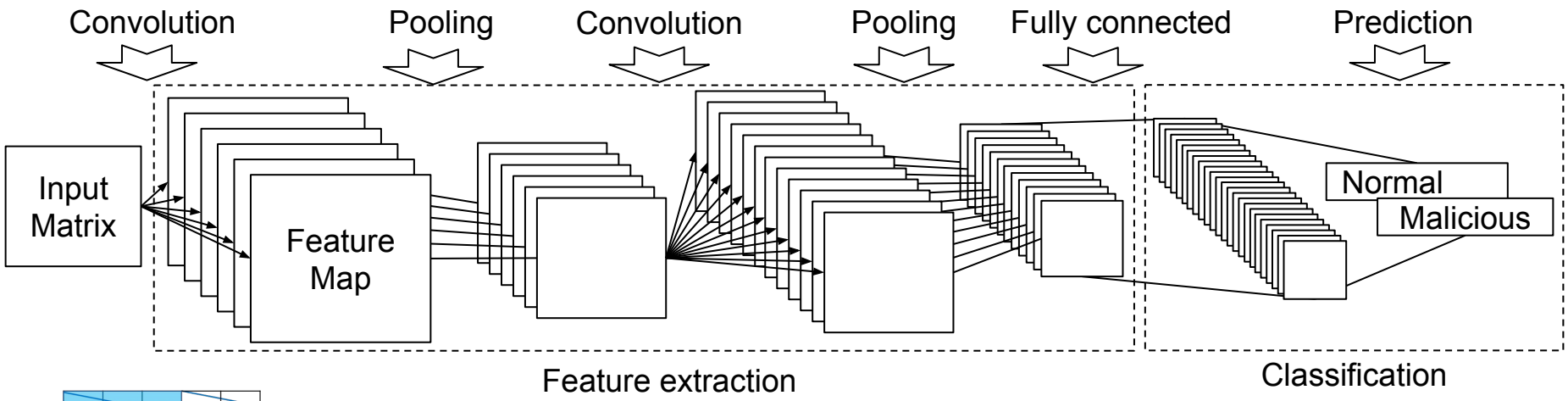


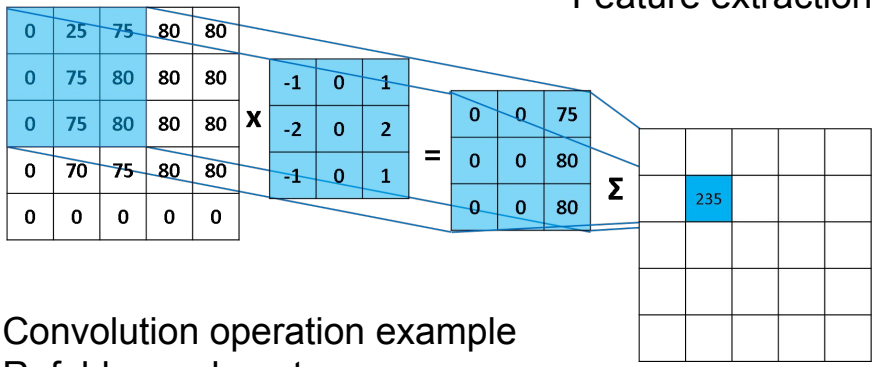



*World Leading Research with Real World Impact!*

# CNN Overview






Convolution operation example
Ref: blog.csdn.net

# Methodology

# Process-level Performance Metrics

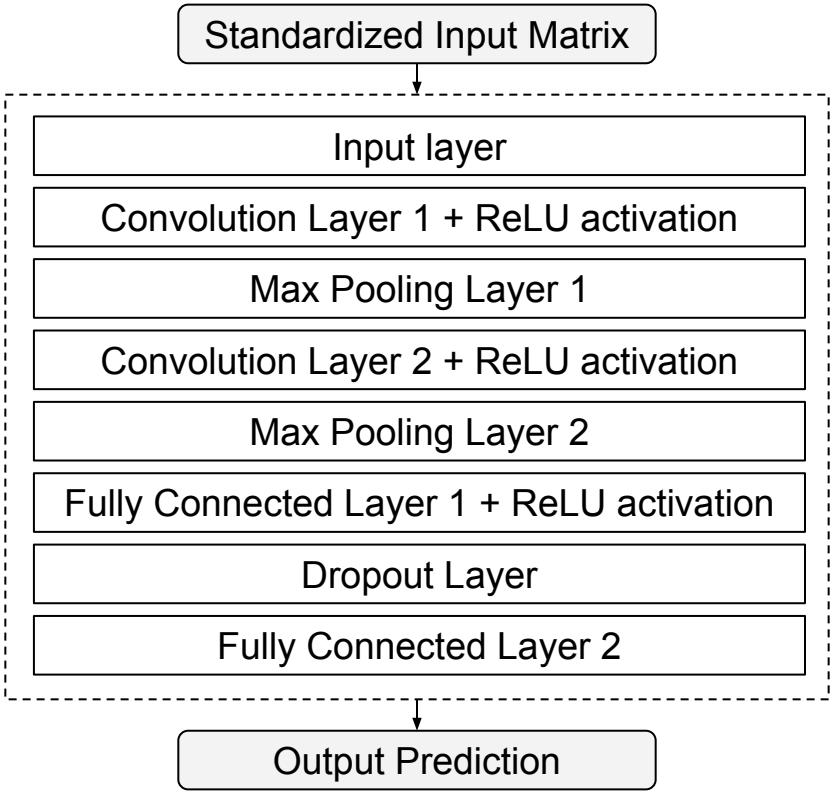


- We use performance metrics as a way of defining a process behavior.
- 28 process-level performance metrics.
- These metrics can easily be fetched through the hypervisor.

| 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 |

# Shallow CNN Model





# 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:

$$\mathbf{X}_t = \begin{bmatrix} & f_1 & f_2 & \dots & f_n \\ p_1 & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_m & \vdots & \vdots & \dots & \vdots \end{bmatrix}$$

# CNN Input (cont.)



- 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.

# Unique process



- 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.

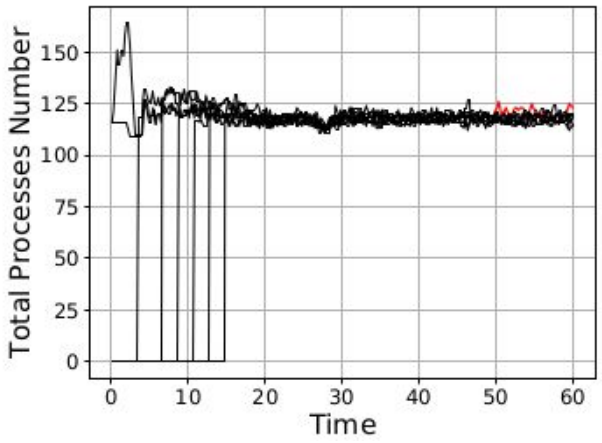
| pid | name | cmd | hash | kb_sent | cpu_user | sample_time |
|---|---|---|---|---|---|---|
| 1241 | php-fpm7.0 | php-fpm: pool www | 7eb8522425... | 33.61710 | 0.03000 | 2018-06-15 11:19:04 |
| 1240 | php-fpm7.0 | php-fpm: pool www | 7eb8522425... | 38.79308 | 0.00000 | 2018-06-15 11:19:04 |
| 1221 | php-fpm7.0 | php-fpm: master process (/etc/php/7.0/... | 7eb8522425... | 0.00000 | 0.02000 | 2018-06-15 11:19:04 |
| 1287 | python | python | 23eeeb4347… | 0.00000 | 0.15000 | 2018-06-15 11:19:04 |

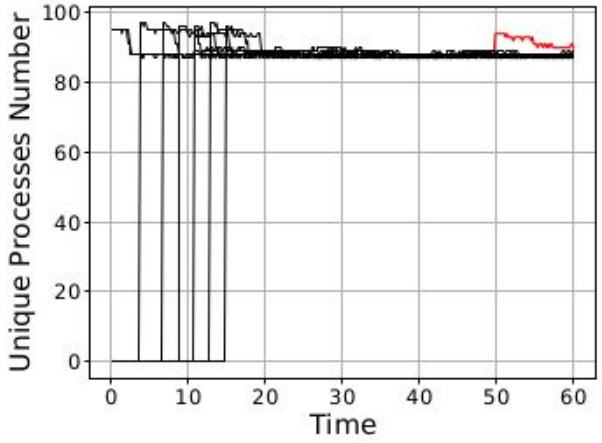
**Unique Process**

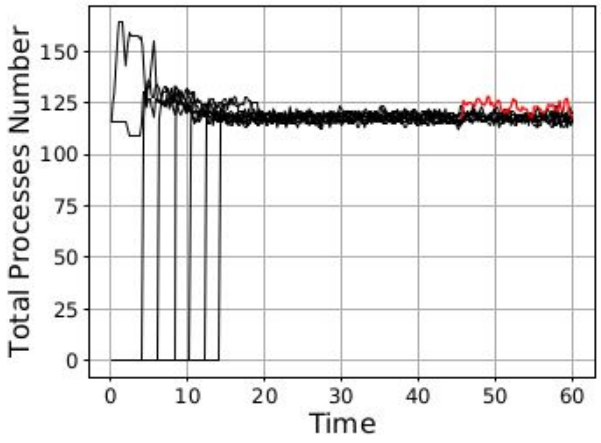
| **name** | **cmd** | **hash** | AVG(kb_sent) | AVG(cpu_user) | sample_time |
|---|---|---|---|---|---|
| php-fpm7.0 | php-fpm: pool www | 7eb8522425... | 36.2051 | 0.0150 | 2018-06-15 11:19:04 |
| php-fpm7.0 | php-fpm: master process (/etc/php/7.0/... | 7eb8522425... | 0.00000 | 0.0200 | 2018-06-15 11:19:04 |
| python | python | 23eeeb4347… | 0.00000 | 0.1500 | 2018-06-15 11:19:04 |

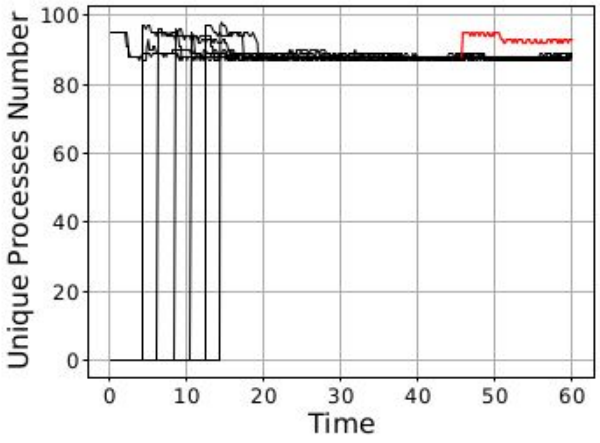
# Unique process (cont.)





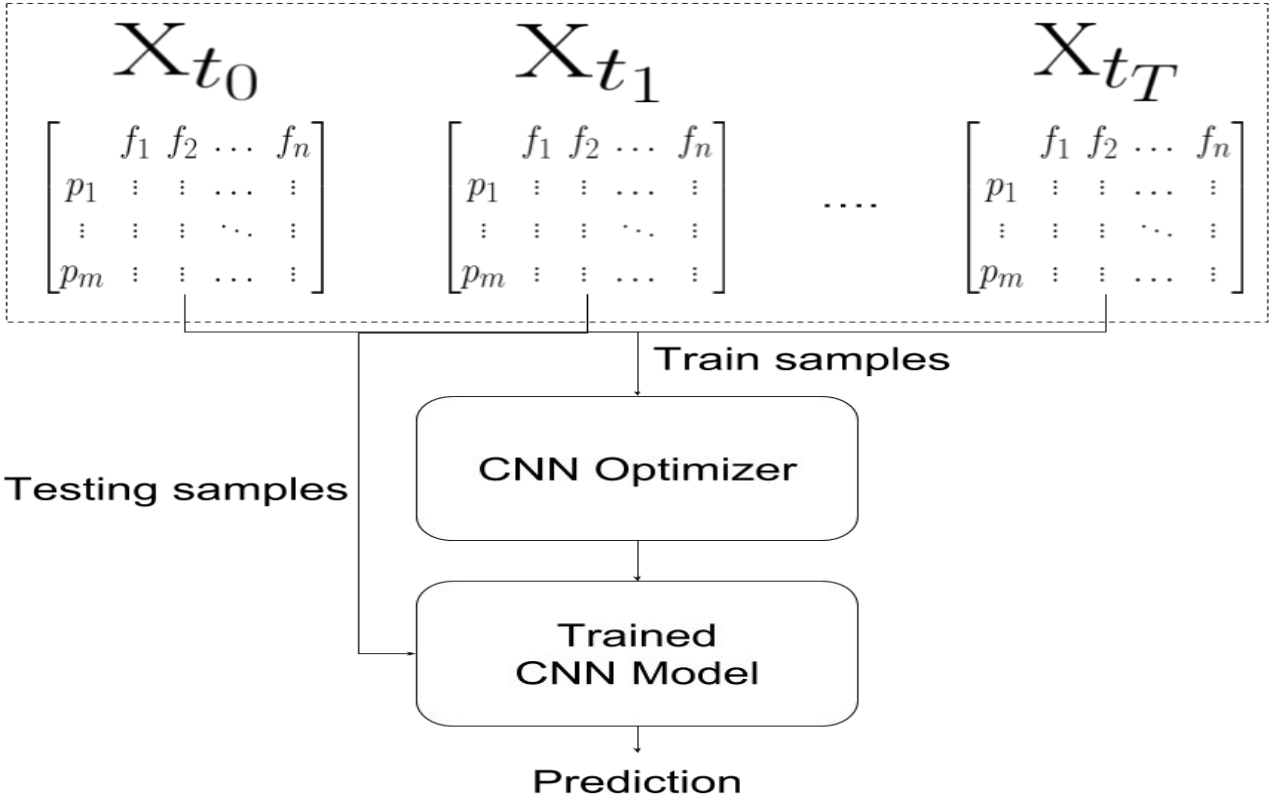




Two different experiments (each with a different malware) where the number of total standard processes are compared to the number of unique processes.

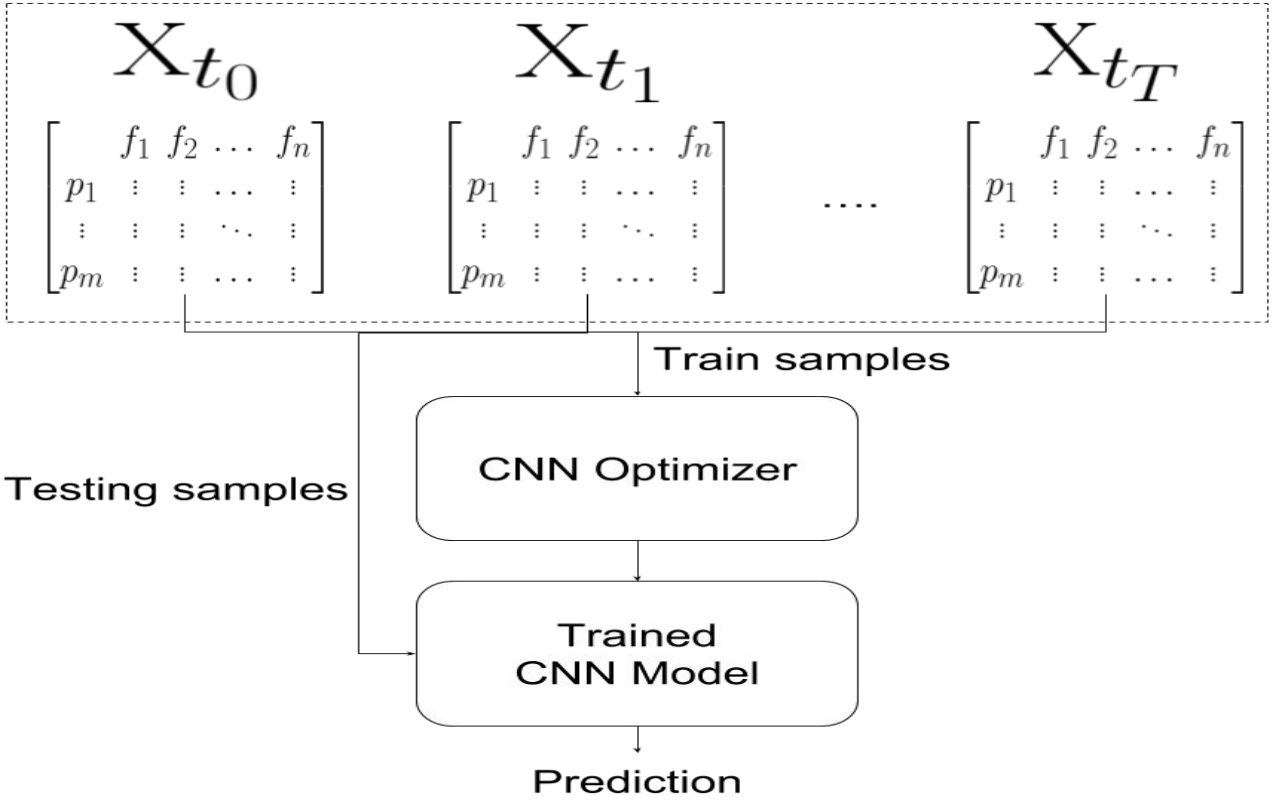
# Single VMs Single Samples (SVSS)





# Single VMs Single Samples (SVSS)

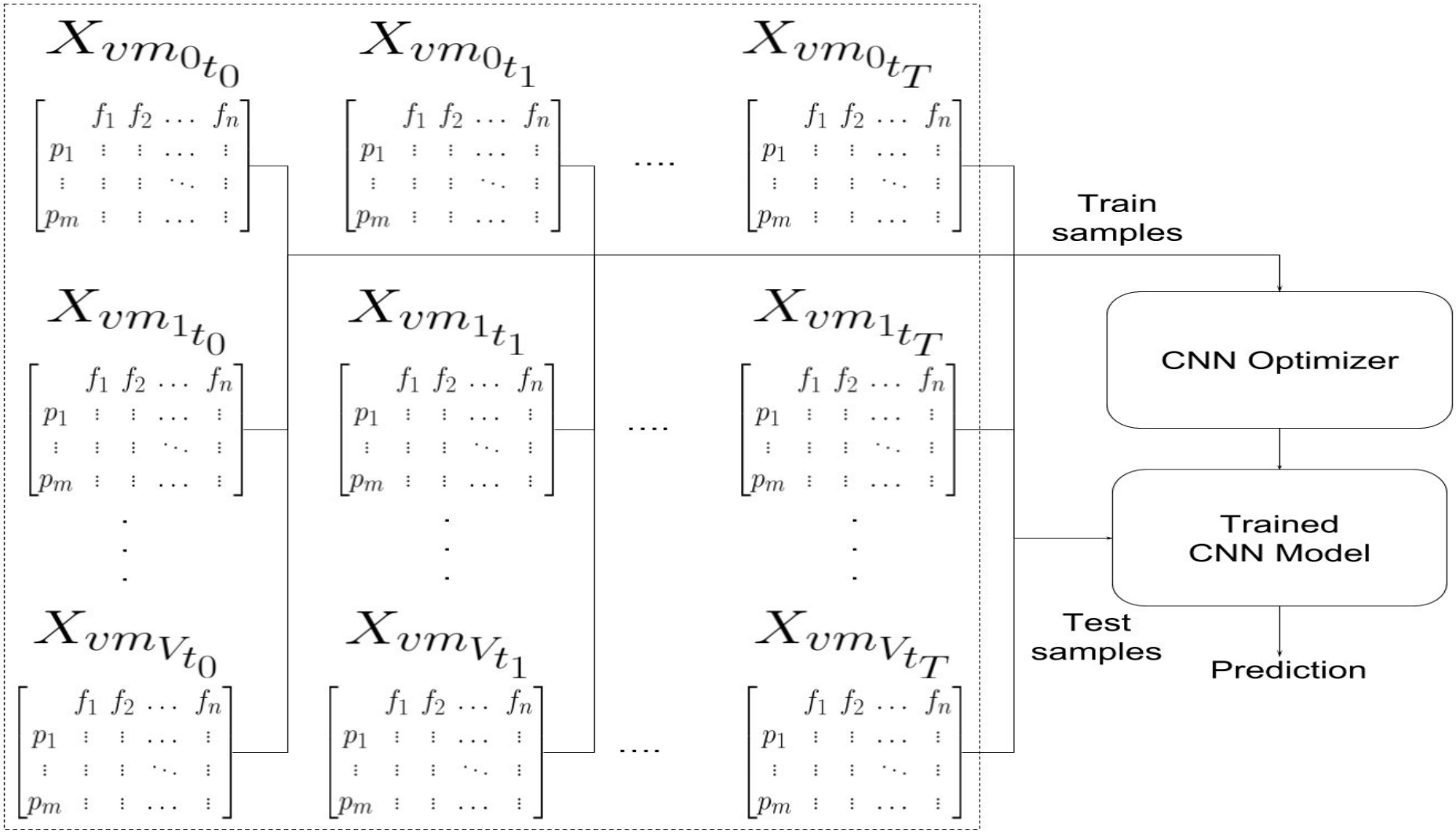





Disadvantage: Losing information if a VM has some effects on other VMs.

# Multiple VMs Single Samples (MVSS)







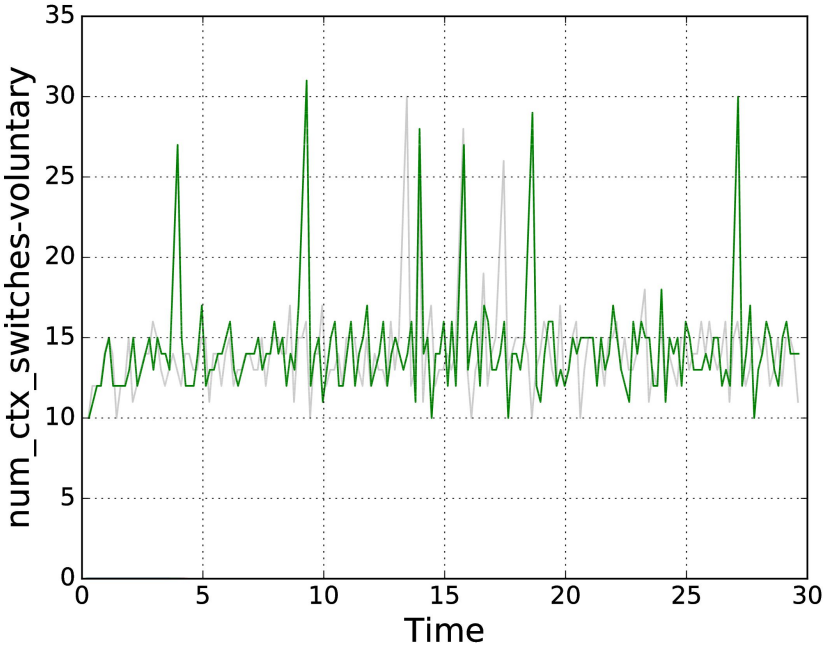

*World Leading Research with Real World Impact!*

# Key Intuition

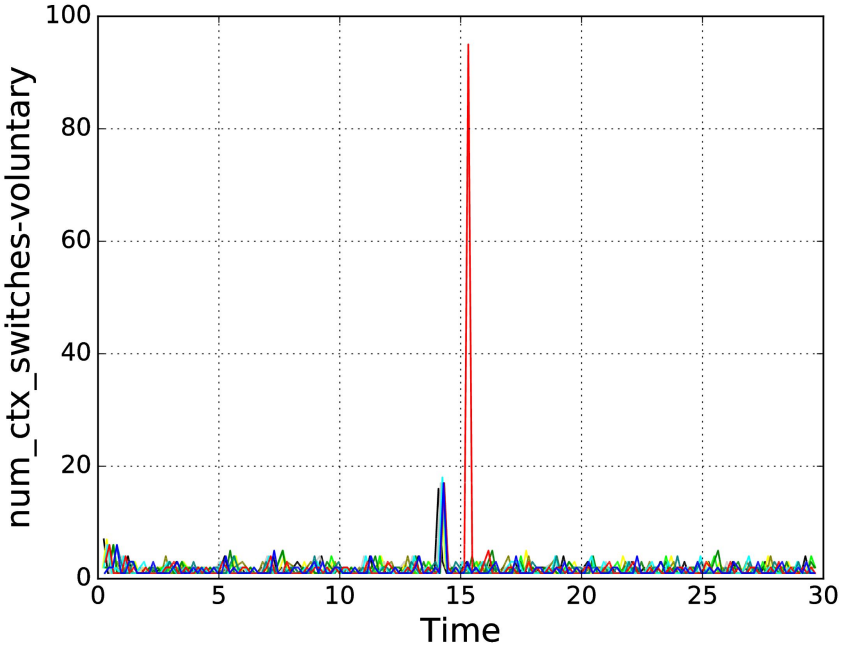


What do we gain from having multiple VMs in an auto-scaling scenario?

**"Correlation between VMs"**



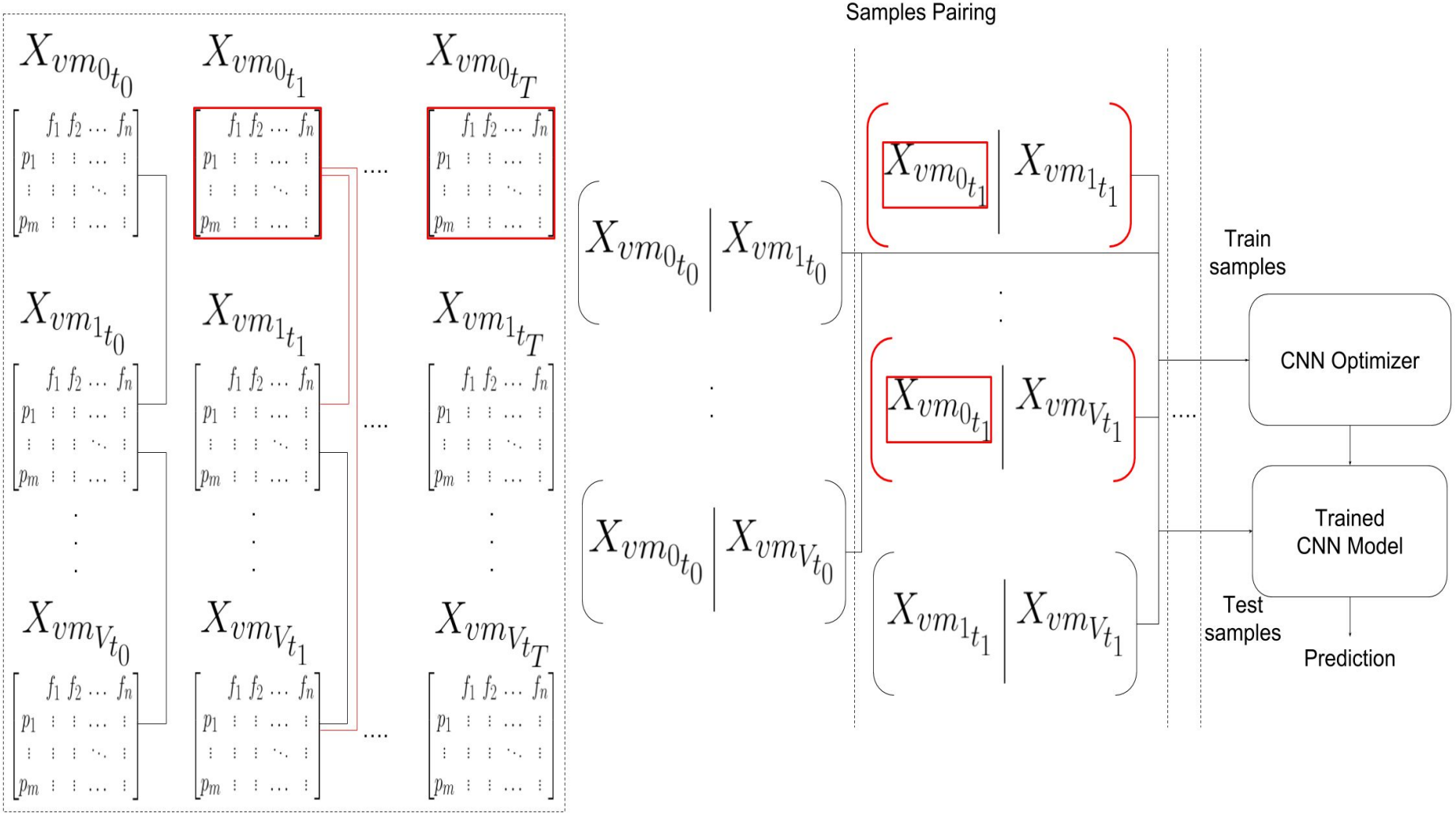

Number of used voluntary context switches over 30 minutes for two different runs of the same unique process




Number of used voluntary context switches over 30 minutes for one run of 10 VMs in an auto-scaling scenario.

# Multiple VMs Paired Samples (MVPS)







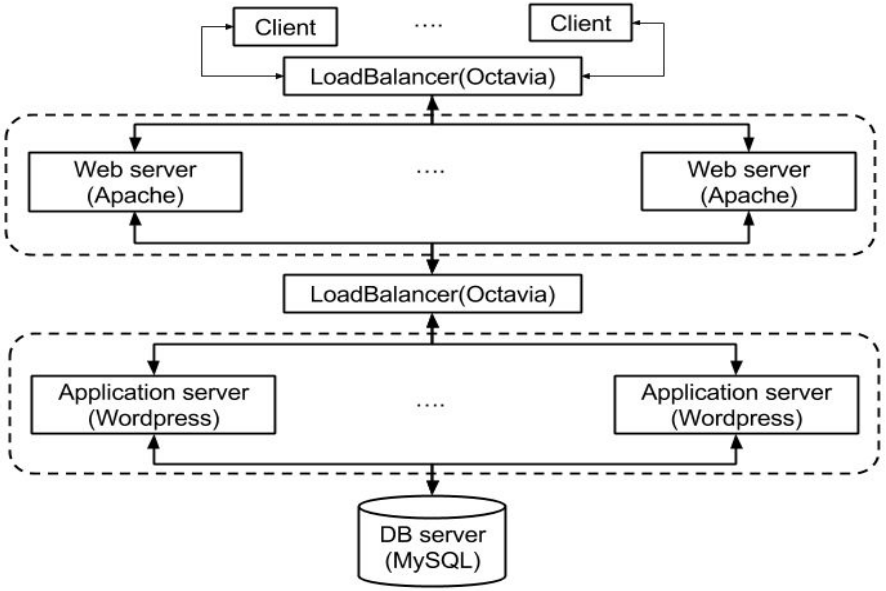

*World Leading Research with Real World Impact!*

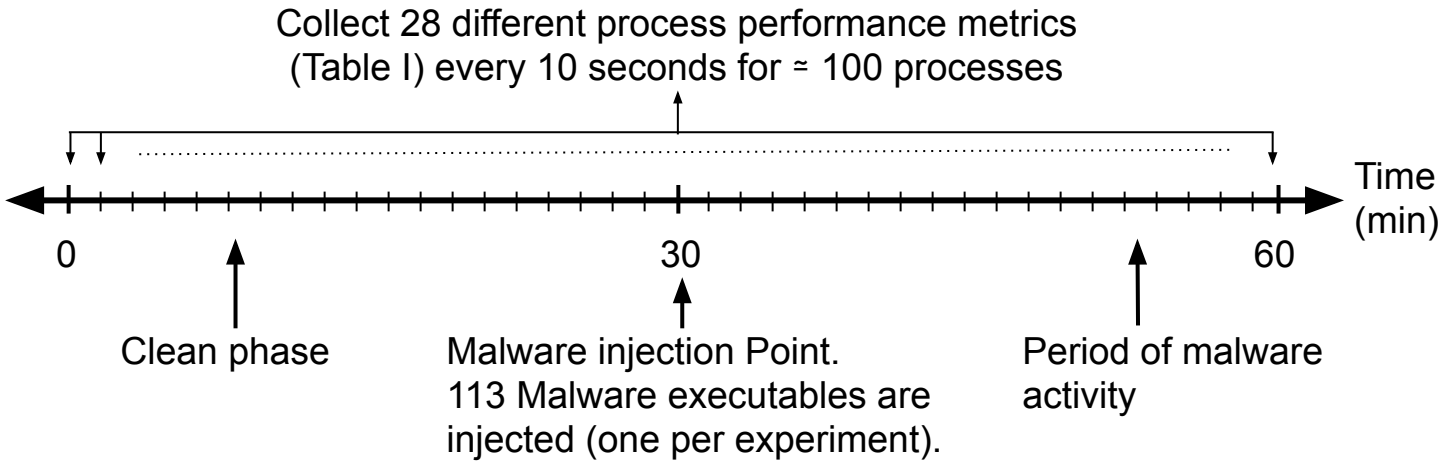
# Experimental Setup and Results

# 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:

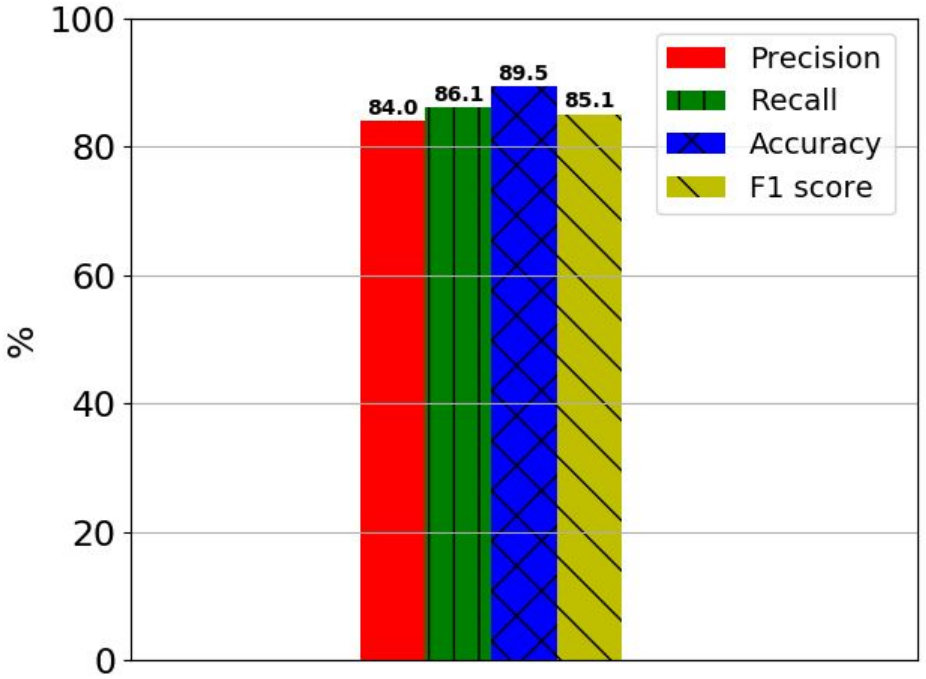


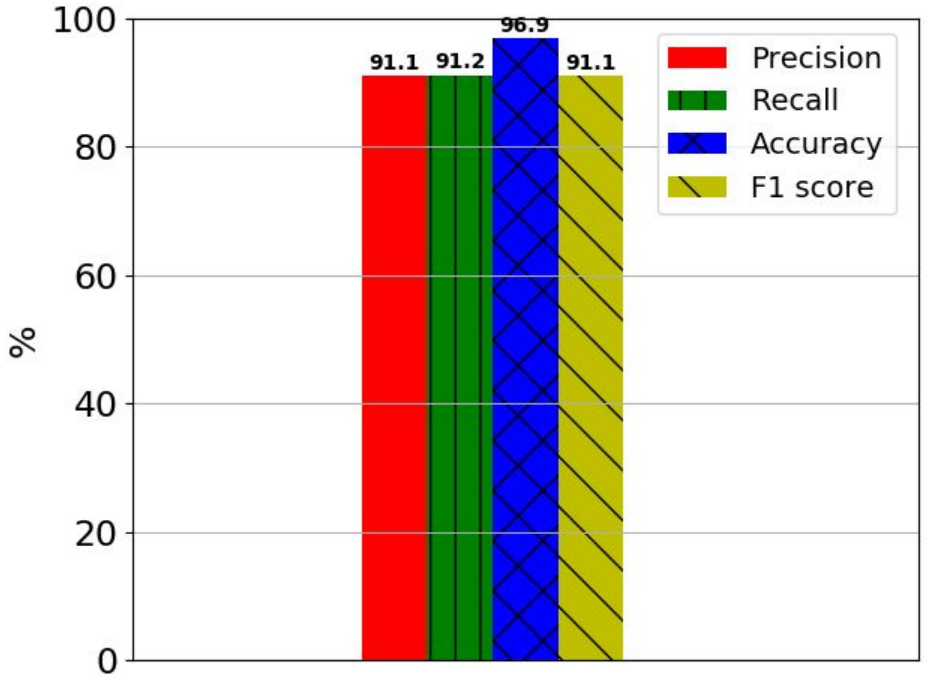
# Results



## MVSS




## MVPS



# Conclusion & Future Work



The goal of this paper was to provide a develop cloud-specific online malware detection method by leveraging cloud characteristics (i.e., auto-scaling).

1. We developed an effective approach for detecting malware using process-level features for low-level malware in an auto-scaling scenario.
2. We introduced a novel pairing samples approach for capturing correlations between VMs.

Future Work:

- Applying and testing multiple architectures (e.g., hadoop systems or containers)
- Investigating and leveraging more cloud characteristics for security.
- Develop techniques to handle the situation when multiple VMs are infected simultaneously by an attacker.

# Questions/Comments