RATAFIA: Ransomware Analysis using Time And Frequency Informed Autoencoders

Manaar Alam*, Sarani Bhattacharya*, Swastika Dutta*, Sayan Sinha*,
Debdeep Mukhopadhyay*, and Anupam Chattopadhyay†
*Department of Computer Science and Engineering, IIT Kharagpur, India
{alam.manaar, swastika, sayan.sinha}@iitkgp.ac.in, {sarani.bhattacharya, debdeep}@cse.iitkgp.ac.in
†School of Computer Science and Engineering, NTU, Singapore
anupam@ntu.edu.sg

***Abstract*—Ransomware can produce direct and controllable economic loss making it one of the most prominent threats in cybersecurity. According to the latest statistics, more than half of the malwares reported in Q1 of 2017 are ransomwares, and there is a potential threat of novice cybercriminals accessing ransomware-as-a-service. The concept of public-key based data kidnapping and subsequent extortion was first introduced in 1996. Since then, variants of ransomware emerged with different cryptosystems and larger key sizes; however, the underlying techniques remained the same. There are several works in the literature which propose a generic framework to detect these ransomwares; though, most of them target ransomwares having specific classes of the encryption algorithm. In addition to it, most of these methods either require Operating System (OS) kernel modification or have high detection latency. In this work, we present a generalized two-step unsupervised detection framework: *RATAFIA* which uses a Deep Neural Network architecture and Fast Fourier Transformation to develop a highly accurate, fast and reliable solution to ransomware detection using minimal tracepoints. The proposed method does not require any OS kernel modification making it adaptable to most of the modern-day system. We also introduce a special detection module for successful identification of benign disk encryption processes having similar characteristics like malicious ransomware programs but having a different intention. We provide a comprehensive study to evaluate the performance of *RATAFIA* in the presence of standard benchmark programs, disk encryption and regular high computational processes in the light of software security.**

***Index Terms*—Ransomware, Hardware Performance Counters, Fast Fourier Transformation, Long-Short-Term-Memory**

I. Introduction

Ransomwares are one of the prime security threat in traditional cyber-physical entities. The number of medium to large-scale enterprise falling prey to ransom payment and extortion of their private databases have increased manifold since its introduction in [1]. These malicious executables infect victim machines and demand a ransom amount after encrypting files and documents of the device. In May 2017, WannaCry ransomware affected approximately 400,000 computers across 150 countries [2]. Moreover, there were 4.3 times new ransomware variants in Q1 2017 than in Q1 2016 [3]. Identification, blocking of these ransomwares at the earliest along with recovering contents of already encrypted files is already an open challenge.

Kharraz *et al.* [4] studied a range of ransomwares and identified different ransomware families. It is suggested that despite advancing encryption systems, the prominent ransomwares leave a trait in the access of I/O and file-systems. Accordingly, Kharraz *et al.* [5] proposed a technique of correlating high file system activities with the intrusion of ransomwares, which, however, is susceptible to false positives and also can be defeated with a slow encryption process. Moreover, the technique requires modification in the Operating System kernel, which may not be practical in many real-life scenarios. Ransomware detection using honeypot type fake folders is discussed in [6]; however, any ransomware can bypass the detection if it does not invade the honeypot area in the system. Scaife *et al.* [7] presented an approach to detect ransomwares by analyzing the changes due to encryption of user data, but fails to distinguish whether a malicious ransomware process or a benign user process makes the change. Ransomware detection has also been studied in the context of formal methods. Mercaldo *et al.* [8] proposed a detection approach in Android platform by constructing formal models of ransomware bytecodes considering temporal logic properties of different ransomware families. However, the paper does not provide any insight into the efficiency of the proposed model considering disk encryption programs, which are benign user programs having similar behavior as ransomwares. In a recent work, Kiraz *et al.* [9] presented a method, where large integer multiplication blocks are identified within an execution to deal with crypto-ransomwares. Since public-key cryptosystems rely on large integer multiplications, it can detect the threat at an early stage. Similar approaches for detection of symmetric-key cryptographic primitives via data flow graph isomorphism [10] or by identifying characteristics of a cipher in a binary code [11] are also presented. In this paper, neither we target a specific family of ransomwares nor the properties corresponding to a particular cipher implementation used by a ransomware. Instead, we develop a generic anomaly-based approach relying on the hardware activities of an unknown program obtained from Hardware Performance Counter (HPC) statistics.

HPCs were first introduced by Malone *et al.* [12] for checking static and dynamic integrity of programs to detect any malicious modifications to them. There have been several works [13]–[15] which utilize HPCs for detecting the presence of malwares. Demme *et al.* [13] used HPCs to build a malware detector in hardware. Detecting malware which modifies kernel control flow has been targeted in [14], [15]. These works use HPCs to monitor a target system for identifying the

vulnerabilities. However, detection of ransomwares through the HPCs, to the best of our knowledge, has not been attempted so far. Though the underlying technique is similar [13], ransomware detection requires far more accuracy and faster response time to limit the damage.

A major difficulty in any ransomware detection approach is to differentiate between a benign disk encryption process and a ransomware executable, both of which serve the same objective, but the latter being a malicious one. Most of the popular disk encryption programs require administrative privilege and use similar algorithms [16] in their encryption operation. In this paper, we utilize this fact and try to address a harder problem to differentiate between a disk encryption program and a ransomware not only by checking the privilege of a program but also by observing its behavior in terms of HPCs. We also present a basic idea to recover the files encrypted by a ransomware before its detection by utilizing the Linux file locking mechanism using `mlock()` system call.

Motivation and Contribution: Our primary motivation behind the work and the major contributions are listed below:

1) We propose a comprehensive ransomware detection method using the HPCs, which have been shown to be very efficient in detecting different categories of malwares. **However, existing methods for ransomware detection, to the best of our knowledge, do not use HPCs in their techniques**. The use of HPCs helps us to obtain valuable information with finer granularity and thereby aids us to devise a method to detect ransomwares with minimal detection latency.
2) The main objective of the proposed method: RATAFIA is to learn the behavior of a system under observation with the statistics obtained from a cluster of HPC events. **In this work, we propose an unsupervised learning technique with the help of an Long-Short-Term-Memory (LSTM)-based Autoencoder which, to the best of our knowledge, has not been attempted so far for detecting ransomware families.** The advantage of unsupervised technique is that the learning process does not require a labeled dataset, which is often difficult to obtain considering the occurrences of several newer unknown varieties of ransomwares, and the detection of unknown ransomware is a far greater challenge than known ones. The Autoencoder is trained with abundant data on normal system behavior of the target system, which can be obtained without any difficulty.
3) The general operation of ransomwares involves accessing, opening, encrypting and closing files one after another in a very high frequency, and this fast repetitive sequence of file access is certainly impossible with human intervention. Hence, ransomwares are computationally intensive programs, and to distinguish them from other benign computationally intensive programs we utilize the repetitive property of ransomwares in our framework. Thus, we propose to use another LSTM-based Autoencoder trained on the values obtained by applying Fast Fourier Transformation (FFT) on the time-domain HPC data for normal observation. **The proposed method in this paper, to the best of our knowledge, is the first one to use FFT for the detection of ransomware programs and differentiating it with other computationally intensive processes to reduce the false positive rates.**
4) Most of the popular ransomware detection methods [5]–[8] do not evaluate their detection framework in the presence of *Disk Encryption* programs, which are benign in nature having similar behavior as ransomwares. **In this paper, we also evaluate the performance of RATAFIA in the presence of popular disk encryption programs and devise a correlation-based approach to differentiate between these two processes**.

RATAFIA is a lightweight method, which does not require any hardware or kernel level modification, thereby making it practical to use in almost every environment.

II. Background on LSTM-based Autoencoder

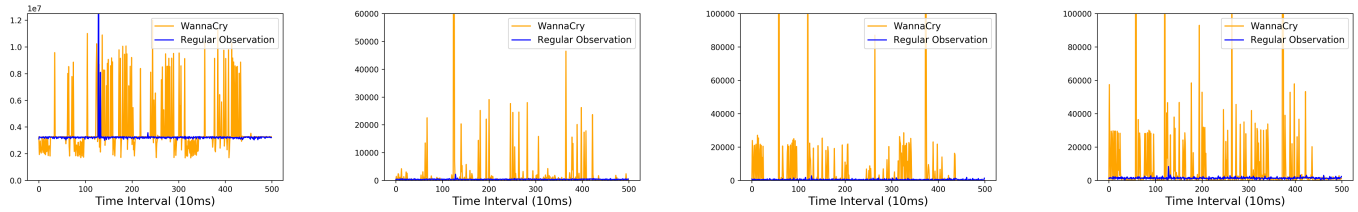
In this section, we present a brief overview on the working of an LSTM-based Autoencoder which is essential for understanding the proposed methodology. LSTM networks are deep recurrent neural network models which have been recently used for many sequence learning tasks [17], [18]. An LSTM-based Autoencoder model consists of two modules: *Encoder* and *Decoder*. The Encoder is an LSTM network which is used to map an input sequence to a vector representation of fixed dimensionality, and the Decoder is another LSTM network which uses this fixed vector to reproduce the target sequence. Thus, the primary goal of an Autoencoder is to induce a representation for a set of sequence data by learning an approximate identity function.

Let us consider a time-series sequence data $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$ of length N, where each point $x_i \in \mathcal{R}^m$ is an m-dimensional vector obtained at time-instance t_i. We consider a scenario where we have a very large time-series data of length L. A set of multiple short time-series data can be obtained by taking a window of length N and sliding over the larger time-series, where $N \ll L$. Let us denote the set of such short time-series as $\mathcal{S}_T$, which we call the *training data*. The Encoder and the Decoder is jointly trained for each $\mathcal{X} \in \mathcal{S}_T$ to obtain a fixed length vector representation $\mathcal{F}$, where $\mathcal{F} \in \mathcal{R}^n$ is an n-dimensional vector which represents the common characteristic existing in the training data.

Since, the Autoencoder learns an *approximate* identity function, it will incur some errors while reproducing the target sequence for a given input. Let, for the input sequence $\mathcal{X}$, the reconstructed sequence is $\mathcal{X}' = \{x'_1, x'_2 \ldots x'_N\}$. The error while generating $\mathcal{X}'$ from $\mathcal{X}$ is termed as *Reconstruction Error*. The reconstruction errors for all the sequences in $\mathcal{S}_T$:

$$\mathcal{L} = \sum_{\mathcal{X} \in \mathcal{S}_T} \|\mathcal{X} - \mathcal{X}'\|^2 = \sum_{\mathcal{X} \in \mathcal{S}_T} \sum_{i=1}^{N} \|x_i - x'_i\|^2$$

The learning goal of the Autoencoder is to minimize reconstruction errors for all the input samples. In the next section, we discuss the benefit provided by Autoencoder to detect the presence of an anomaly using the data observed through HPCs.



(a) **# Branch Instructions** (b) **# Branch Misses** (c) **# Cache Misses** (d) **# Cache References**

Fig. 1. Variation of different Performance Counter Events: **a)** `branches`, **b)** `branch-misses`, **c)** `cache-misses`, and **d)** `cache-references` with respect to the regular observation (blue) and also in the presence of Wannacry Ransomware (orange).

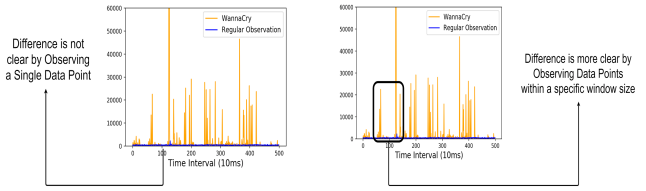
III. Detecting Anomalous System Behavior

In this section, we first analyze the *normal behavior*[1] of a system by monitoring appropriately selected HPC events in parallel. We then present a notion of anomalous activity in the system and demonstrate a detailed method to detect those anomalies using an *Autoencoder*.

A. Observing the System Behavior using HPCs: The HPCs are a set of special purpose registers built into modern processors to dynamically observe the hardware related activities in a system. HPCs can be monitored dynamically using the `perf` tool, available in Linux 2.6.31+ kernels. One interesting property of this tool is that a user can observe the HPCs associated with a system with some time interval, thereby giving the benefit of observing the system behavior continuously in a succession of time. The command to monitor a particular HPC event for a specific executable in such a way is as follows:

```
perf stat -e <event> -I <time_interval> <executable>
```

The range of HPC events those can be monitored using the perf tool is more than 1000. However, in most Linux based systems, the perf tool is limited to observing a maximum of 6 to 8 events in parallel depending on the processor type. Moreover, some of the events are not even supported by all the processors. Our objective in this work is to detect the presence of ransomwares, which mainly contain an encryption program, typically involving both symmetric and asymmetric key encryptions. Micro-architectures have been widely used as a source of side-channel information to thwart both symmetric [19], [20] and asymmetric encryptions [21]. Secret information leakages through cache and branch predictor units of a processor are main contributors behind the success of these attacks. Ransomware programs also use encryption algorithms in their operation; hence, we selected events related to cache and branch prediction and are supported in most of the processors to utilize the well-established information leakages for our proposed method. The HPC events selected for our study are `instruction`, `cache-references`, `cache-misses`, `branches`, and `branch-misses`. The events are self-explanatory by their names. Generally, the symmetric encryption affects the cache based events while the asymmetric encryptions affect the instruction and branching events. We have measured the effects of ransomware programs on other HPC events, and we found out that the events mentioned above are affected most, which supports our hypothesis.



(a) (b)

Fig. 2. Importance of windowed observation over single data-point for decision making considering the event `branch-misses`. A single data point is unable to detect the presence of WannaCry for a particular time-instance, whereas, a window of observation with a specific length, helps to differentiate between a WannaCry (orange) and Regular Observation (blue).

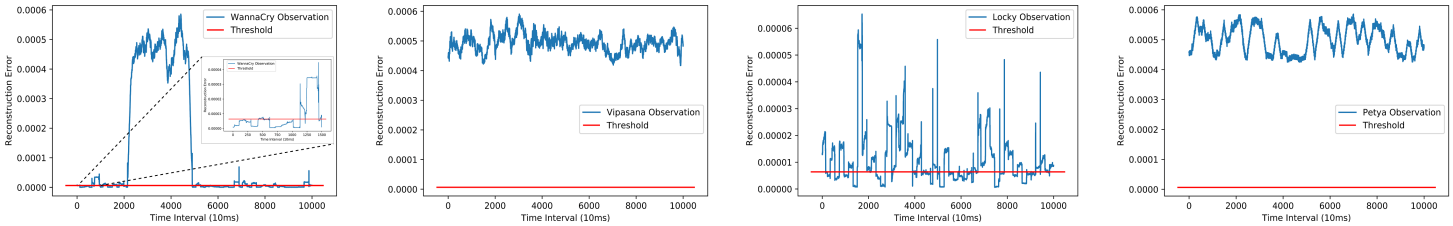
In order to construct the prototype of normal system behavior, we designed a *watchdog program* and collected the `perf stat` values with `10ms` time interval[2] for that executable. We collected these values at different points of time in the target system and created a *dataset of regular observations*. The effects of all other processes including the ransomwares running in the system will have an impact on the HPC values. We articulate that any behavior which is not close to this dataset is unusual activity, but may not be a malicious one.

We show the effect of a Ransomware (for example, WannaCry) on the HPC events in Fig. 1. Blue lines in the figure represent the effect of normal system programs on the watchdog for different HPC events, whereas the orange lines show the effect of ransomware. We can observe that the behavior of ransomware with respect to the HPC event statistics are significantly different from the normal observation.

An important point to be observed is that for some particular time instances, the behavior of WannaCry does not change much from normal system behavior. For example, around time interval of 100, as shown in Fig. 2(a), the effect of WannaCry on the event branch misses is same as normal system behavior. However, if we consider a window of a specific length, as shown in Fig. 2(b), the behavior of WannaCry is more distinguishable from normal observation. So, instead of considering individual points for decision making, we select a window of observations considering each of the five events collectively.

[1]We term regular execution pattern of an uncompromised system as *normal behavior*, which captures all the daily operations of benign executables. We do not consider execution patterns of computationally intensive programs within the normal behavior, as ransomwares, which are also computationally intensive programs, may be missed by the detection framework by doing so.

[2]The minimum interval of time after which `perf` is allowed to sample a data point is 10ms. We have selected minimum time-interval to sample data as fast as permitted, to enhance the mechanism regarding detection time.



(a) **WannaCry** (b) **Vipasana** (c) **Locky** (d) **Petya**

Fig. 3. The sequence of Reconstruction Errors for different Ransomwares (blue) in the Autoencoder. Reconstruction errors for **WannaCry** and **Locky** are lower than the threshold $\mathcal{R}_t$ (red) for some time-instances; however, the errors are higher for most of the times, which is sufficient to detect them as anomalies.

Thus, we transform the problem into anomaly detection in multivariate time-series data. In the next subsection, we use the LSTM-based Autoencoder, as discussed in Section II, to detect the anomalies with respect to the regular observation.

B. Detecting Anomaly Using an Autoencoder: LSTM-based Autoencoder has recently become popular to detect anomalies with multi-sensor inputs [22]. In this paper, to present a generalized ransomware detection strategy, we avoid modeling behavior of ransomwares as there can be a potential new one whose behavior is unknown. Instead, we model normal system behavior, as we can get a large number of such instances. Another advantage of detecting anomalies by modeling normal behavior is that we do not need labeled dataset since any activity with unusual behavior crossing an empirically calculated threshold can be detected as an anomaly. Thus, we propose an unsupervised approach to detect these anomalies.

We have already seen in the previous subsection that HPC values observed over the watchdog are considered as time-series data. The LSTM-based Autoencoder, as discussed in Section II, is trained to reconstruct instances of normal time-series with the target time-series being the input itself. *Intuition behind the anomaly detection is that the Autoencoder is only shown normal instances during the training phase and learned to reconstruct them*. However, when an anomalous sequence is given as an input to the Autoencoder, it will not be able to reconstruct it accurately, and hence would lead to higher reconstruction errors in comparison to the normal sequences.

The training dataset is constructed from the observed data for normal system behavior by taking a window of 100 tracepoints (i.e., a window of tracepoints collected over 1 second, since each interval data is collected after 10ms. The window size of 100 is chosen empirically). We shift the window by one time-interval (i.e., 10ms) repeatedly to consider consecutive 100 sample point for learning. In order to quantify the threshold for detecting anomalous activities, we calculate reconstruction error distribution ($\mathcal{R}$) for some unknown samples of normal behavior, which are not used in the training phase. Since the unknown samples belong to normal behavior, according to the 3σ rule of thumb, all the error values in $\mathcal{R}$ should lie within three standard deviations of the mean. Hence, we set the threshold ($\mathcal{R}_t$) for reconstruction error as $\mathcal{R}_t = \mu_{\mathcal{R}} + 3 * \sigma_{\mathcal{R}}$, where $\mu_{\mathcal{R}}$ and $\sigma_{\mathcal{R}}$ are mean and standard deviation of distribution $\mathcal{R}$. In our experimental setup $\mathcal{R}_t$ came out to be 5.38×10^{-6}. Thus, for an unknown sequence, if the reconstruction error is greater than $\mathcal{R}_t$, we state that the sequence belongs to an anomalous observation. In the next subsection, we present experimental validations of the claim considering different ransomwares.

C. Anomalous Behaviors of Ransomwares: In our study, we considered four ransomware programs: namely *WannaCry*, *Vipasana*, *Locky*, and *Petya* to show the impact of selecting threshold $\mathcal{R}_t$ in detecting them as anomalies. Fig. 3 shows the sequence of reconstruction errors for these ransomwares. Blue lines indicate reconstruction errors of each window, whereas red lines signify threshold $\mathcal{R}_t$ as calculated before. First point on each error plot represents the reconstruction error of the first window of 100 time-interval (equivalently 1 second). The successive points come after each interval of 10ms as we slide by one time-interval for calculating next reconstruction error.

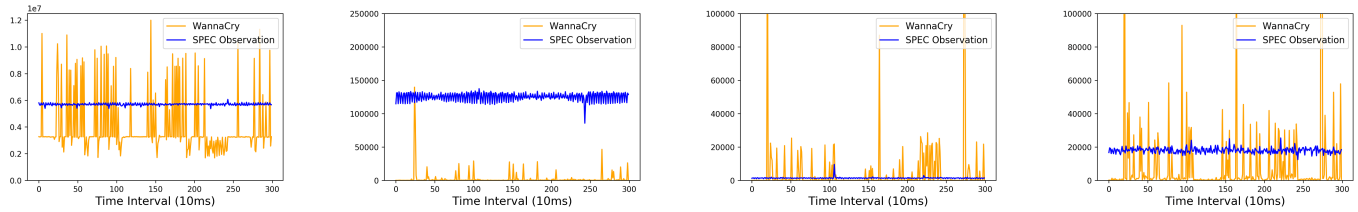
We can observe in Fig. 3(a)[3], the execution of WannaCry starts behaving like a regular program (since the reconstruction errors lie well below the threshold), but the error shoots over the threshold at 432^{nd} observation. Thus, the WannaCry is detected as anomaly $(1000+431*10) = 5310$ ms after the start of execution. Whereas, from Fig. 3(b), Fig. 3(c), and Fig. 3(d), we can observe that the ransomwares Vipasana, Locky, and Petya are detected as an anomaly at the first window itself, i.e., 1 second after the start of execution. In all these cases there is an extra overhead of time due to the processing time associated with the Autoencoder, which we discuss in Section VIII.

IV. Is Reconstruction Error Good as Decider?

In the previous section, we suggested that a threshold as high as $\mathcal{R}_t$ can be used to decide whether a process behavior significantly deviates from the normal system behavior. Next, we explain why a single decision step is not enough to claim that the anomaly observed is from a malicious process.

False Positives due to Heavy Computation Processes: In order to test the robustness of our scheme, we incorporate an analysis in the presence of computationally expensive SPEC benchmarks. We consider the Gshare predictor implementation as provided in [23] and observe the HPC statistics exactly like our previous setting. Fig. 4 presents the variation of different HPC events in the presence of both SPEC benchmarks and WannaCry ransomware. We can observe that the execution behaviors for both the programs are significantly different from normal observations. Thus, the sequences of time-series data

[3]The embedded image in the box is the zoomed version of the same image for the first 1500 window data.



(a) **# Branch Instructions** (b) **# Branch Misses** (c) **# Cache Misses** (d) **# Cache References**

Fig. 4. Comparison of the effects on different HPC events in the presence of Wannacry Ransomware (orange) and also the SPEC Benchmark Programs (blue).

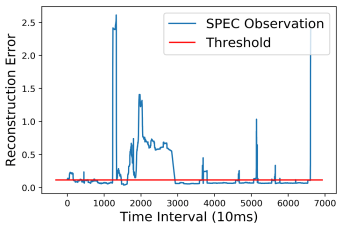


Fig. 5. The sequence of Reconstruction Errors for the SPEC Benchmark (blue) in the Autoencoder. As the errors are higher than the threshold (red) for most of the times, it is detected as an anomaly producing a false positive.

for SPEC programs may also create significant reconstruction errors. Fig. 5 clearly shows that the sequence of reconstruction errors in the presence of SPEC benchmark programs is above the predetermined threshold for most of the times. It substantially raises an alarm to RATAFIA that the benchmark program is a potentially malicious program which deviates to an extent from the normal system behavior. But surely, in this case, it is a false alarm, since the benchmark is composed of server and multimedia programs and can be considered as the representative of high computational processes which may deviate highly from the normal running processes in a system.

Next, we perform a transformation of the values obtained through HPC statistics from time domain to frequency domain to differentiate actual malicious processes from false positives.

Eliminate False Positives using FFT: In the second phase of detection using RATAFIA, we transform the values from time domain to frequency domain using FFT, which is the most efficient way to implement Discrete Fourier Transformation. The primary reason to convert the analysis from the time domain to frequency domain is to understand the repetitive pattern within the values. The ransomware executable runs encryption repeatedly on multiple files; thus it repeats the same set of operations of opening, encrypting and closing followed by deleting a file for multiple files one after another.

We have applied FFT on time domain values for different HPC events as mentioned in Section III-A, to obtain frequency domain values. Fig. 6 presents FFT plots for the normal system behavior in blue lines, the SPEC Observations in green lines and WannaCry in orange lines for different HPC events, which typically indicates that the amplitudes for each frequency bins are constantly higher for the ransomware in contrary to SPEC benchmark. Fig. 6 also shows that for most HPC events (apart from the cache misses), the FFT plot of the SPEC benchmark overlaps exactly with the FFT of the normal system behavior. Also, it is quite clear from Fig. 6(a), Fig. 6(b), and Fig. 6(d) that the amplitudes of almost all the frequency bins are higher for WannaCry than the SPEC observation, which is eminent as the WannaCry program repeatedly encrypts multiple files.

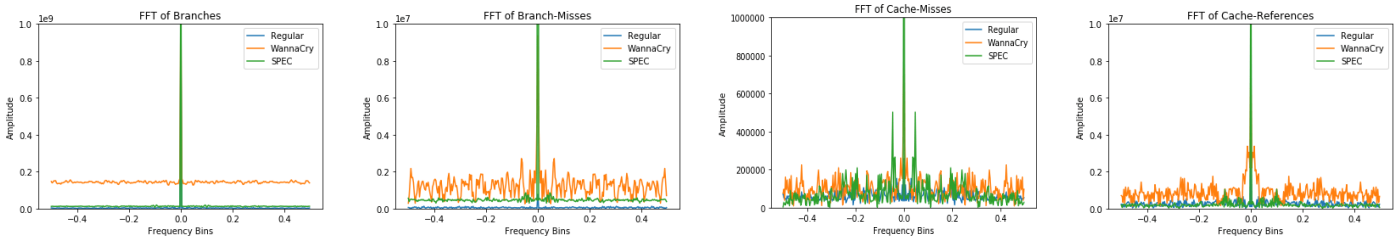
The variations of amplitudes for different frequency bins can again be considered as a multivariate time-series data, and an LSTM-based Autoencoder can be used to detect the anomalies, as discussed in Section III-B. The amplitudes for SPEC benchmark are very close to that of regular observations for most of the HPC events. Thus, modeling the FFT data for regular sequences using an Autoencoder will result in reconstruction errors close to the threshold (say $\mathcal{R}'_t$) for SPEC benchmarks, and the error will be much higher in case of ransomwares because of the higher frequency amplitude values due to repeated encryptions. We modeled another Autoencoder following the procedure mentioned before with the FFT transformed data and calculated the threshold $\mathcal{R}'_t$ to be 0.002829 for our experimental setup. Fig. 7 presents the sequence of reconstruction errors for both SPEC and different ransomwares in the second Autoencoder. We can easily verify that the reconstruction errors of SPEC programs always lie below the threshold, whereas the errors of all the ransomwares always remain higher to the threshold[4]. Thus, the use of Autoencoder on FFT data helps to remove false positives (i.e., heavy computation programs) from the detection framework.

Need of Both the Autoencoders: One interesting point that may arise in this framework is that what is the requirement of both the autoencoders, when it is evident from the fact that the second Autoencoder is sufficient enough to discard ransomwares. The second Autoencoder takes FFT converted values of window data, and the FFT conversion requires some computational effort for the conversion. Hence, to reduce the computational complexity of the framework, we apply a first level filter in terms of the first Autoencoder to remove the less computational heavy programs in the first stage itself and apply FFT on the sequence data detected as an anomaly in the first stage before sending it to the second Autoencoder.

V. Experiments on Standard Linux Benchmark

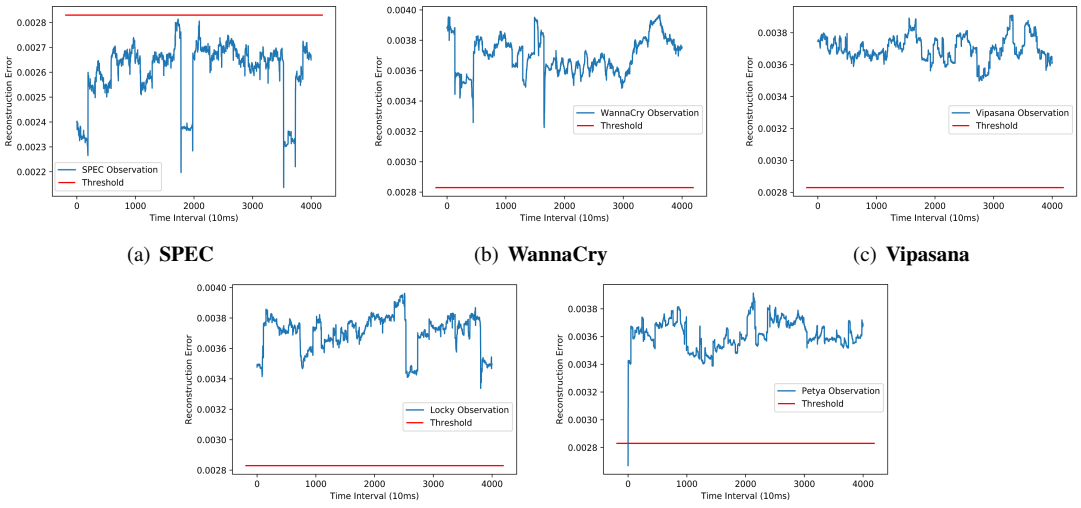
In this section, we consider a standard Linux benchmark CHStone to analyze the efficiency of both the Autoencoders. CHStone is a Linux benchmark suite which represents various application domains such as arithmetic, media processing, and security. Hence, it would be intriguing to evaluate the performance of RATAFIA in the presence of this benchmark. The

[4]The reconstruction error for Petya Ransomware on the first window is lower than the threshold, but for subsequent windows, errors are always higher.



(a) **# Branch Instructions** (b) **# Branch Misses** (c) **# Cache Misses** (d) **# Cache References**

Fig. 6. Variation of Amplitudes in frequency domain in comparison to regular observation (blue) for different HPC events in the presence of SPEC Benchmark (green) and Wannacry Ransomware (orange) after the Fast Fourier Transformation.

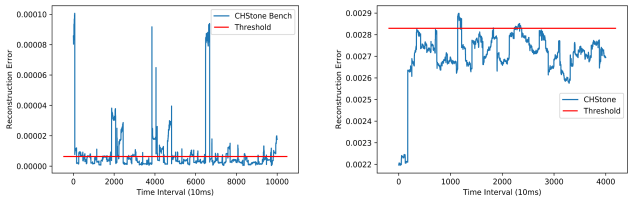


(a) **SPEC** (b) **WannaCry** (c) **Vipasana**

(d) **Locky** (e) **Petya**

Fig. 7. Sequence of Reconstruction Errors (blue) for **a)** SPEC Benchmark and different **b)-e)** Ransomwares in the second Autoencoder. The errors are constantly higher than the threshold $\mathcal{R}'_t$ (red) for all the ransomwares, and thus easily detected as anomalies.

sequence of reconstruction errors for both the Autoencoders for the execution of CHStone programs is shown in Fig. 8. We can observe from Fig. 8(a) that in most cases the error value in the first Autoencoder is lower than the threshold $\mathcal{R}_t$. However, in some of the cases, the error is higher (i.e., it is detected as an anomaly), but the reconstruction error, as shown in Fig. 8(b), in the second Autoencoder is always lower than the threshold $\mathcal{R}'_t$ (except at some specific time interval). We can hypothesize that, if the reconstruction error in *both the Autoencoders* are constantly higher for some specific time we conclude that behavior as anomaly instead of considering a single spike[5]. We also experimented with two other benchmarks such as Unix-Bench and LMBench, and presented the results in Fig. 9, which shows that the results are similar in nature.



(a) **First Autoencoder** (b) **Second Autoencoder**

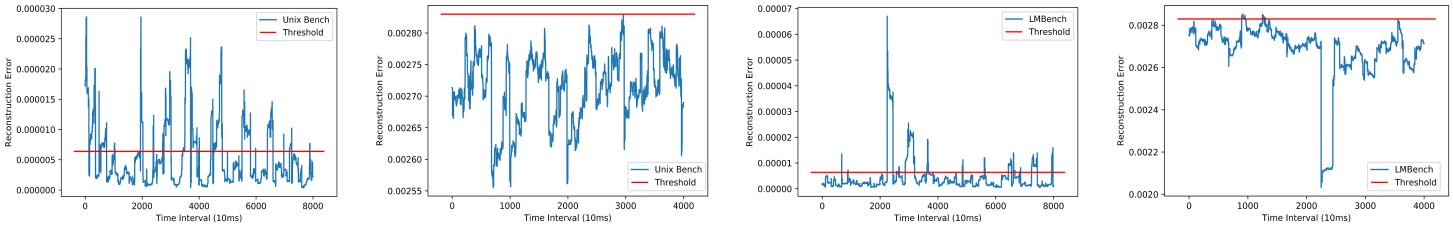
Fig. 8. The sequence of Reconstruction Errors (blue) for the CHStone benchmark programs in both the Autoencoders.

VI. Identifying Disk Encryption Programs

The Disk Encryption processes are very similar in operation to the malicious ransomware processes. Both of these processes access files frequently and encrypt them one after another, though the intentions of the processes are entirely different. While designing RATAFIA, one of the most significant challenges is to differentiate the disk encryption processes from malicious ones. Also, there are some ransomwares having disk encryption program as their encryption engine, which typically puts the security engineers in a very delicate state. In this section, we discuss how the disk encryption programs differ from the ransomware processes which does not use disk encryption as their intermediate software routine. The detection module proposed in this section helps us to differentiate disk encryption modules from the general set of ransomware programs. Later we use this detection module to demonstrate a reasonable solution to this problem of ransomware detection.

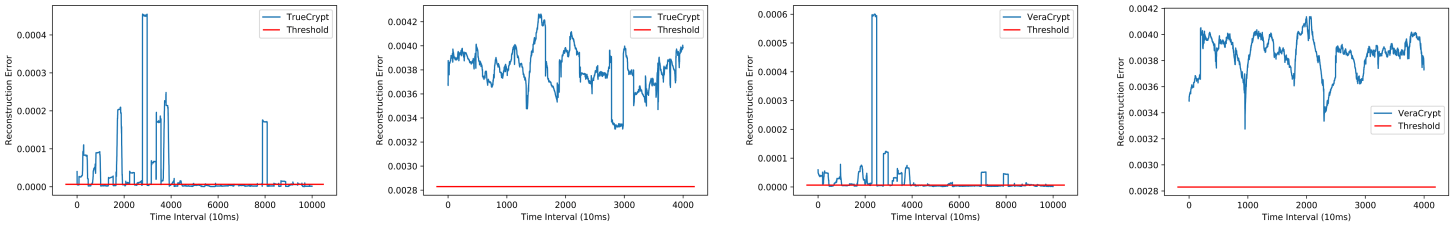
In order to manifest the problem, we consider two popular disk encryption processes in our study, namely *TrueCrypt*, and *VeraCrypt*. The behavior of both the processes in both the Autoencoders are shown in Fig. 10. We can see from the figure that, both the disk encryption processes are detected as ransomwares by RATAFIA, as the reconstruction errors in both

[5]The spikes in reconstruction error plots occur when the execution behavior of a program is significantly dissimilar from the benign execution template.



(a) (b) (c) (d)

Fig. 9. Sequence of Reconstruction Errors (blue) for Benchmark Programs in both the Autoencoders. **(a)** Unix-Bench in First Autoencoder, **(b)** Unix-Bench in Second Autoencoder, **(c)** LMBench in First Autoencoder, **(d)** LMBench in Second Autoencoder.



(a) (b) (c) (d)

Fig. 10. Sequence of Reconstruction Errors (blue) for Disk Encryption Programs in both the Autoencoders. **(a)** TrueCrypt in First Autoencoder, **(b)** TrueCrypt in Second Autoencoder, **(c)** VeraCrypt in First Autoencoder, **(d)** VeraCrypt in Second Autoencoder.

the Autoencoders are constantly higher than the thresholds for a significant interval of time. *One naïve solution is to check the privilege of the current process under suspicion*. Since disk encryption processes can only be run by an administrator with highest privilege, checking the privilege of the running application can be a quick check to determine whether the target process is malicious or not.

In this paper, however, we also delved into a harder problem of differentiating these two sets of processes by looking at the nature of HPC values. All the popular disk encryption processes use AES-XTS [16] mode of encryption for their operations. We utilize this characteristic to template the operation of a disk encryption, and in the online phase, we check for whether the suspicious program is a disk encryption or not. In order to find similarity with the stored template, we calculate a cumulative correlation of it with the suspicious process. If the correlation is high for a successive interval of time, we conclude that the process is a disk encryption process.

The watchdog program generates continuous windows of multivariate data with significantly different magnitudes for various processes. Instead of using complex multivariate correlation, we use the univariate reconstruction error from first Autoencoder for a simpler *Pearson's Correlation* to make the computation less expensive and with fewer storage requirements. We store a template of reconstruction errors from first Autoencoder for a disk encryption process instead of multivariate window data and correlate it with the reconstruction errors from first Autoencoder of the unknown process.

In order to demonstrate the approach we use reconstruction errors of *TrueCrypt* as our template, and present the cumulative correlation values with *VeraCrypt* and previously mentioned ransomwares in Fig. 11. We can easily observe that the correlation values of VeraCrypt are high for a successive interval of time, whereas, correlation values for all the ransomwares

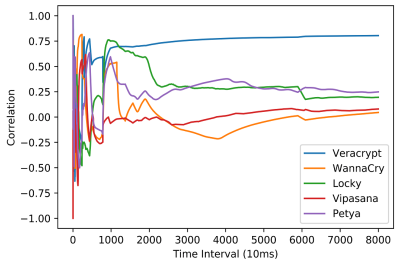


Fig. 11. Cumulative Correlation of Veracrypt and different Ransomware programs with another disk encryption process TrueCrypt.

converges to a very low value. Hence, we conclude that it is feasible for RATAFIA to differentiate the behavior of disk encryption programs from ransomwares with the hypothesis that most of the popular disk encryption programs use the same mode of encryptions in their operations.

Comprehensive Detection and Temporary Suspension of Disk Encryption Processes: The discussion in the previous section shows that a particular mode of encryption can be differentiated with high confidence if the HPC events are monitored in an efficient manner. This identification specifically means that all disk encryption algorithms running AES in XTS mode can be differentiated from the general genre of malicious ransomware programs which have no disk encryption subroutine in them. Though as mentioned earlier, their exists some ransomware like MAMBA, which uses disk encryption modules in its subroutine to maliciously encrypt files [24]. Our detection module, as described previously, can successfully identify that whether a disk encryption module is running in the background; however, it turns out that the disk encryption could also be a part of ransomware operation. In this paper, we propose a solution to this problem by temporary suspending the suspected disk encryption program, which raises an alarm and waits for a confirmation from the user whether

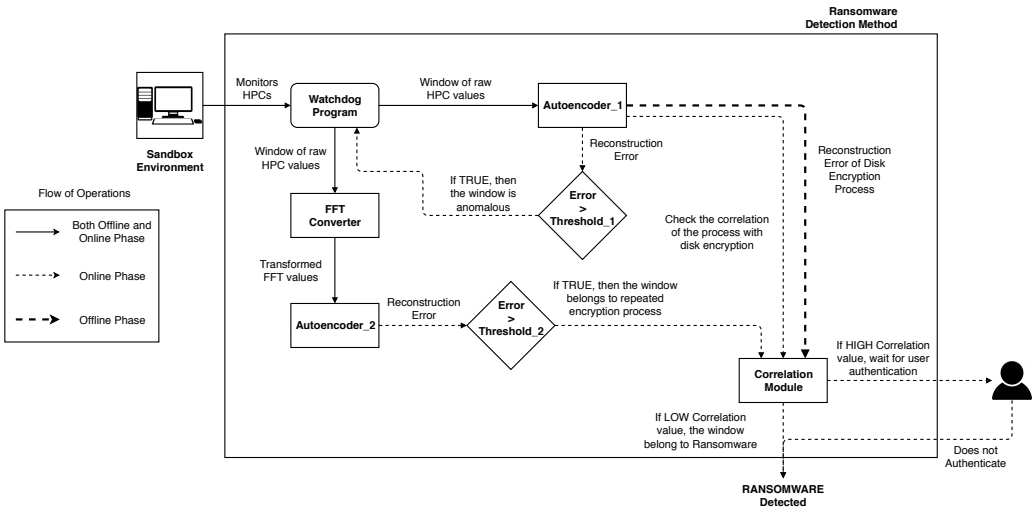


Fig. 12. **Detection Methodology of RATAFIA**: In the offline phase, two Autoencoders are trained based on the normal observation, and the correlation module is prepared using the template of popular disk encryption process with AES-XTS mode operation. In the online phase, an unknown program is considered as a Ransomware if and only if *all the following three cases* are satisfied. **a)** It is detected as an anomaly in Autoencoder_1, **b)** It is detected as an anomaly in Autoencoder_2, **c)** It has low correlation value or it has high correlation value but user does not confirm it as authentic disk encryption process.

the suspected program is actually launched by the user. The confirmation will automatically resume the disk encryption module intended to run from the user's end but prevents the unintended ones which gets launched by the ransomwares.

VII. Architecture of RATAFIA

In this section, we present a detailed architecture of the proposed methodology. Basic diagram of the system is shown in Fig. 12. The architecture contains five modules: *Watchdog Program*, *Autoencoder_1*, *FFT Converter*, *Autoencoder_2*, and *Correlation Module*. The methodology works in two phases, namely *Offline Phase* and *Online Phase*. The functioning of each module in both the phases are described below:

Offline Phase: In the offline phase, the detection methodology is trained with the normal behavior of the target system, such that any unusual activity of a ransomware is detected correctly in the real-time scenario. The functioning of each of the modules in this phase is described below.

1) *Watchdog Program*: Monitors HPCs of the target environment continuously and forwards a window of data to Autoencoder_1 and FFT Converter in parallel.
2) *Autoencoder_1*: Trains an Autoencoder with the dataset forwarded by watchdog program and also forwards the reconstruction error corresponding to a disk encryption process to the correlation module directly[6].
3) *FFT Converter*: Computes Fast Fourier Transformation of each window forwarded by the watchdog program and passes the results to the Autoencoder_2.
4) *Autoencoder_2*: Collects data passed by FFT Converter and trains another Autoencoder based on the dataset.
5) *Correlation Module*: Stores reconstruction errors related to a disk encryption for analysis in the *Online Phase*.

Online Phase: In the online phase, the detection module is deployed in the target system for real-time monitoring to detect ransomwares. The functioning of each module for an unknown process in this phase is discussed below.

1) *Watchdog Program*: Continuously monitors the system and forwards a window data to Autoencoder_1. Watchdog program does not forward data to FFT converter for monitoring the system with lower computational cost.
2) *Autoencoder_1*: Calculates reconstruction error of the data received from the watchdog program. If the error is higher than the predefined threshold $\mathcal{R}_t$, it sends a signal to watchdog program to transmit the same window to FFT Converter. Otherwise, the process is allowed to execute in the system. This module also forwards the data to correlation module directly, irrespective of it being lower or more than the threshold value.
3) *FFT Converter*: Converts the data received from the watchdog program into the frequency domain, and forwards the transformed data to the Autoencoder_2, but with a condition imposed by the Autoencoder_1 module.
4) *Autoencoder_2*: Calculates the reconstruction error of the received FFT data, and if the error is higher than the predefined threshold, $\mathcal{R}'_t$, it sends a signal to the correlation module to check for its correlation with the template of disk encryption process. Otherwise, the process is considered as simply a high computational process and is allowed to execute in the system.
5) *Correlation Module*: Calculates cumulative correlation of the unknown process with known disk encryption process. If the correlation is low for a considerable duration of time, the process is considered as a Ransomware and is terminated from the system, else it is forwarded to user for confirmation as a legitimate disk encryption process.

[6]The behavior of disk encryption is not included in the training as this may produce false negatives for ransomwares since both are repeated encryption process. However, the inclusion of other encryption programs in the training dataset will not create any difficulties, as long as they are not repeated file encryption programs. The reconstruction errors due to the disk encryption program are calculated after the training is completed.

TABLE I
MODEL ARCHITECTURE FOR AUTOENCODERS

Layer Number	Layer Type	Input Shape	Output Shape
Autoencoder_1			
1	LSTM	(None, 100, 5)	(None, 100, 32)
2	LSTM	(None, 100, 32)	(None, 100, 5)
Autoencoder_2			
1	LSTM	(None, 100, 5)	(None, 100, 64)
2	LSTM	(None, 100, 64)	(None, 100, 5)

VIII. EVALUATING THE PERFORMANCE OF RATAFIA

We performed all the experiments in a desktop system having `Intel Core i5 7500` processor with 2.6GHz clock frequency running Linux 4.10.0-38-generic kernel. We used popular open source Python-based neural network library `Keras` for the implementation of both the Autoencoders. The architecture used to model the Autoencoders are mentioned in Table I. The thresholds, as mentioned previously, came out to be 5.38×10^{-6} and 0.002829 respectively from these distributions following 3σ rule of thumb.

The FFT converter usually takes 0.0003 milliseconds to convert a sequence within a window into the frequency domain. The model building times for Autoencoder_1 and Autoencoder_2 are on average 10 and 14 minutes respectively. Testing time to calculate whether a single window is an anomaly or not is 1.321 milliseconds for Autoencoder_1 and 1.699 milliseconds for Autoencoder_2 respectively. As shown in the architecture of RATAFIA in Fig. 12, the testing of a regular observation only passes through the Autoencoder_1, thereby taking only 1.699 milliseconds, and an anomalous observation passes through all the three modules: Autoencoder_1, FFT Converter, and Autoencoder_2, thus taking $1.321 + 0.0003 + 1.699 = 3.0203$ milliseconds to be detected. The time to correlate two reconstruction errors from Autoencoder_1 and the stored error trace is on an average 0.0001 milliseconds, which will be calculated only for either disk encryption or ransomwares. In both the cases of regular and anomalous observation, detection time is less than sampling interval, which is 10 msec. Hence, *the detection is performed seamlessly, without the need for any storage buffer*, as a new window of data will be created after 10 msec.

Without loss of generality, we first present the calculation of detection time for most recent WannaCry ransomware by RATAFIA. As shown in Fig. 3(a), the WannaCry is detected as an anomaly at the 432^{nd} window and instantly detected as repeated encryption process at the same time because it's reconstruction error is always higher than the threshold of Autoencoder_2. Hence, total time taken to detect WannaCry as a repeated encryption process is equal to (Time taken to generate the first window) + 431 * (time interval for each sample) + (Autoencoder_1 testing time) + (Time for single FFT Conversion) + (Autoencoder_2 testing time) $= 1000 + 431 * 10 + 1.321 + 0.0003 + 1.699$ millisecond $= 5313.0203$ milliseconds. Thus, WannaCry is detected by RATAFIA as a repeated encryption process in approximately 5.313 seconds. We can check privilege of the program and terminate it instantly as ransomware does not have any administrative privilege. However, if we assume that any ransomware gets temporary access to the administrative privilege, its execution can still be prevented by the correlation module, though with a small increase in detection latency. For example, the correlation module discriminates WannaCry from the disk encryption process starting at 1002^{nd} window, as shown in Fig. 11. So, for confirming it as a ransomware program, by correlation, it takes almost extra 5 seconds of execution time. Since at this stage we check for a potential ransomware, we always pause the execution of the suspicious process before sending it to the correlation module and resume it after the verification from the user. Similarly, the detection time for the other ransomwares Vipasana, Locky, and Petya are 1.003, 3.123, and 1.203 seconds respectively. We would like to mention that none of the recent state-of-the-art ransomware prevention methods, as discussed in Section I, considered *detection latency* as a parameter of performance, which is indeed crucial in the context of ransomware detection. Hence, we are unable to provide any comparison related to the detection latency. As a sample run with the WannaCry, out of 10000 files of approximately 21 bytes each, when RATAFIA stops its execution, 68 files are encrypted. It may be noted that the size of a typical file is much larger than 21 bytes, and hence, a lesser number of files will be encrypted in the contemporary systems before RATAFIA prevents the execution of WannaCry. We provide a comparative study with state-of-the-art techniques in Table II considering the efficiency to deal with false positives in terms of computationally intensive programs and disk encryptions.

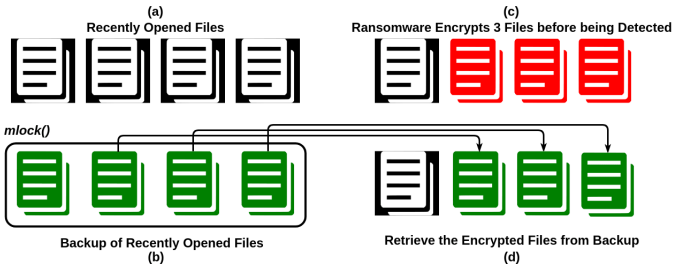


Fig. 13. Notion of File Recovery using Linux `mlock()`. **(a)** Let there are 4 files which are opened within a specific time quantum. **(b)** Backup these files with Linux `mlock()` command (marked with green color). **(c)** Let ransomware encrypts 3 files before being detected by RATAFIA (marked with red color). **(d)** We can easily retrieve the encrypted files from the backup.

IX. A BASIC IDEA OF FILE RECOVERY

RATAFIA is thus capable of detecting the presence of ransomwares very quickly. Depending on the detection latency, ransomware can encrypt few files (say n). We conclude with a suggested approach for data retrieval. A practical solution would be to take backups of n-recently opened files, and after the lapse of time quantum required to encrypt these files, we delete the copies if RATAFIA raises no ransomware alarm, which also minimizes storage requirement for the backup files. In order to further ensure that the backup files are not encrypted we perform locking operation using Linux `mlock()`. A basic idea of this approach is presented in Fig. 13, which

TABLE II
PERFORMANCE COMPARISON OF RATAFIA WITH STATE-OF-THE-ART METHODS

	Kharraz *et al.* [5]	Mercaldo *et al.* [8]	C. Moore [6]	Scaife *et al.* [7]	**RATAFIA**
Computationally Intensive Programs	Did not consider evaluation on computationally intensive benchmark programs explicitly				**Can differentiate from ransomwares**
Disk Encryption Programs	Not considered false +ve evaluation	Not considered false +ve evaluation	Not considered false +ve evaluation	Needs kernel modification for the detection	**Can detect as false positives without any kernel modification**

shows the operation of Linux `mlock()` command in order to recover the files which were encrypted by the ransomware.

X. CONCLUSION AND FUTURE WORKS

In this paper, we provided a detailed understanding of the effect of ransomwares on normal system behaviors. We take the aid of Deep Neural Network to detect the presence of ransomwares by a two-step framework using LSTM-based Autoencoders. The entire detection procedure does not need any template of the malicious process from beforehand. Instead, it thrives on an anomaly detection procedure to detect infectious ransomwares in as less as 5 seconds with almost zero false positives, using frequency analysis. The proposed detection method will work on any platform having HPCs. However, the tunable hyper-parameters (like thresholds, window size etc.) will be different for different systems. The determination of values for these parameters is a one-time process, which will be accomplished during the training of autoencoders.

We also explored the opportunity of applying side channel techniques to recover the secret key used to encrypt the files from the performance counter statistics. We found for ransomwares like WannaCry; each file is encrypted using AES-128 CBC (Cipher Block Chaining) with a randomly generated distinct key. These keys are in turn encrypted using an infection specific RSA public key and stored in the memory. It would be indeed a challenging exercise to recover the AES key by targeting the AES CBC operation. However, we leave that as a future scope of work.

RATAFIA uses a template of the normal system behavior in terms of HPC values to train the autoencoders. The advantage of using HPCs is that they are difficult to tamper. In particular, while one may increase some HPC values by a program, it is difficult to reduce the HPC values without explicitly targeting the HPC registers. There is a provision that the HPC values can be forcefully reset to zero using `PERF_EVENT_IOC_RESET` ioctl() system call. However, in order to do that one requires administrative privilege. Hence, even if an adversary has the knowledge of the template used to train the autoencoders, it does not have the privilege to change the execution footprint of its own program and bypass the detection scheme. It is expected that a *black box attack* on the DNN used in RATAFIA would be more challenging given the fact that the HPC values are robust against modifications or poisoning. Thus it would be difficult for the attacker to make small changes in the input (HPC values) to fool the target model. However, it would be an interesting future direction of research.

REFERENCES

[1] A. Young *et al.*, "Cryptovirology: Extortion-based security threats and countermeasures," in *IEEE Symposium on Security and Privacy*, 1996, pp. 129–140.

[2] J. Crowe, "Wannacry ransomware statistics: The numbers behind the outbreak," May 2017. [Online]. Available: https://blog.barkly.com/wannacry-ransomware-statistics-2017

[3] ——, "Must-know ransomware statistics 2017," Jun 2017. [Online]. Available: https://blog.barkly.com/ransomware-statistics-2017

[4] A. Kharraz *et al.*, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2015, pp. 3–24.

[5] ——, "Unveil: A large-scale, automated approach to detecting ransomware." in *USENIX Security Symposium*, 2016, pp. 757–772.

[6] C. Moore, "Detecting ransomware with honeypot techniques," in *IEEE Cybersecurity and Cyberforensics Conference*, 2016, pp. 77–81.

[7] N. Scaife *et al.*, "Cryptolock (and drop it): stopping ransomware attacks on user data," in *IEEE International Conference on Distributed Computing Systems*, 2016, pp. 303–312.

[8] F. Mercaldo *et al.*, "Ransomware steals your phone. formal methods rescue it," in *Springer International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, 2016, pp. 212–221.

[9] M. S. Kiraz *et al.*, "Detecting large integer arithmetic for defense against crypto ransomware," Cryptology ePrint Archive. http://eprint.iacr.org/2017/558, Tech. Rep., 2017.

[10] P. Lestringant *et al.*, "Automated identification of cryptographic primitives in binary code with data flow graph isomorphism," in *ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 203–214.

[11] F. Gröbert *et al.*, "Automated identification of cryptographic primitives in binary programs," in *Springer International Workshop on Recent Advances in Intrusion Detection*, 2011, pp. 41–60.

[12] C. Malone *et al.*, "Are hardware performance counters a cost effective way for integrity checking of programs," in *ACM workshop on Scalable trusted computing*, 2011, pp. 71–76.

[13] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, 2013, pp. 559–570.

[14] X. Wang *et al.*, "Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters," in *ACM/EDAC/IEEE Design Automation Conference*, 2013, pp. 1–7.

[15] ——, "Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 485–498, 2016.

[16] M. V. Ball *et al.*, "The xts-aes disk encryption algorithm and the security of ciphertext stealing," *Cryptologia*, vol. 36, no. 1, pp. 70–79, 2012.

[17] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[18] I. Sutskever *et al.*, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[19] D. J. Bernstein, "Cache-timing attacks on aes," 2005.

[20] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 605–622.

[21] O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert, "Predicting secret keys via branch prediction," in *Cryptographers' Track at the RSA Conference*. Springer, 2007, pp. 225–242.

[22] P. Malhotra *et al.*, "Long short term memory networks for anomaly detection in time series," in *Proceedings*. Presses universitaires de Louvain, 2015.

[23] "How to use the championship branch prediction evaluation framework," Feb 2011. [Online]. Available: https://www.jilp.org/jwac-2/cbp3_framework_instructions.html

[24] A. Ivanov *et al.*, "The return of mamba ransomware," Aug 2017. [Online]. Available: https://securelist.com/the-return-of-mamba-ransomware/79403/