A Big Data Analytical Approach to Cloud Intrusion Detection

Halim Görkem Gülmez, Emrah Tuncel, and Pelin Angin(✉)

Department of Computer Engineering, Middle East Technical University, Ankara, Turkey
{gorkem.gulmez,emrah.tuncel}@metu.edu.tr, pangin@ceng.metu.edu.tr

Abstract. Advances in cloud computing in the past decade have made it a feasible option for the high performance computing and mass storage needs of many enterprises due to the low startup and management costs. Due to this prevalent use, cloud systems have become hot targets for attackers aiming to disrupt reliable operation of large enterprise systems. The variety of attacks launched on cloud systems, including zero-day attacks that these systems are not prepared for, call for a unified approach for real-time detection and mitigation to provide increased reliability. In this work, we propose a big data analytical approach to cloud intrusion detection, which aims to detect deviations from the normal behavior of cloud systems in near real-time and introduce measures to ensure reliable operation of the system by learning from the consequences of attack conditions. Initial experiments with recurrent neural network-based learning on a large network attack dataset demonstrate that the approach is promising to detect intrusions on cloud systems.

Keywords: Intrusion detection · Cloud · Recurrent neural networks
Big data analytics

1 Introduction

Advances in the networking and virtualization technologies in the past decade have made cloud platforms a popular choice for the data storage and computing needs of many enterprises. Many companies have moved their infrastructures to major cloud platforms for improved reliability. As cloud services have gotten popular and common with decreased costs, they have also become hot targets for attacks. The rise of the Internet of Things (IoT) paradigm, which is closely connected to cloud computing, has increased the attack surface for cloud systems through increased connections and vulnerabilities attackers can exploit. The vulnerability of cloud platforms to attacks calls for security measures to prevent those attacks or detect them and take action when an attack happens. Although some of the existing security methods can be applied to cloud systems, new methods designed truly for the cloud systems would be more preferable.

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 377–388, 2018.
https://doi.org/10.1007/978-3-319-94295-7_26

Cloud systems are prone to zero-day attacks that are malicious activities not observed previously. Even though security mechanisms are updated quickly, even seconds are important to provide high reliability in such systems. Hence, a real-time security system which is not signature-based is more suitable for the cloud. The system should be able to detect both old and new attack methods as fast as possible. As the system will be under new attacks everyday, it is important that the system will easily adapt to those attacks. Unsupervised learning methods will enable the system to handle new attacks and learn from them.

Not unlike cloud computing, big data has become a popular concept with the technological advances in the past decade. Today, many different frameworks and tools exist to handle different types of big data [18]. The ability to store and process data in large volumes and velocity provides a significant means for analyzing cloud data in real time to detect anomalies that could signal presence of attacks on the cloud system. In this work, we propose a big data analytics approach for intrusion detection in cloud systems, based on recurrent neural networks that are capable of incorporating the temporal behavior of the system into the anomaly detection task. Since we want our system to respond in near real-time, we use streaming data from cloud platforms. An initial prototype of the proposed system using Apache Spark and the TensorFlow machine learning framework for recurrent neural networks has been developed and promising results have been achieved with experiments on a large network attack dataset.

2 Related Work

Network intrusion detection has been a well studied topic with many different approaches proposed along the years. While some of these solutions can also be applied in a cloud environment, because of the different characteristics of the cloud environments and comparatively larger network traffic volumes, several new approaches are suggested to solve the problem of intrusion detection in the cloud. In a cloud environment, intrusion detection monitors can be deployed at various locations. With the usage of virtualization techniques, the intrusion detection monitors can be deployed in a guest VM or in virtual machine monitors [12,17]. Beyond host or network device monitoring, distributed collaborative monitoring approaches are also utilized to catch system-wide attacks [4]. In these types of solutions, infrastructure problems need to be solved, since the system must support massive amounts of data gathered from different monitors and these data must be processed quickly to detect possible attacks as soon as possible.

Recent works have proposed using big data processing approaches to solve the problem of intrusion detection in cloud environments [5]. To detect a possible attack using intrusion detection systems (IDS), basically two techniques can be used [13]: In misuse detection, the IDS knows about previous attack patterns and tries to catch an attack by comparing the collected data with previous patterns. In anomaly detection, the IDS does not know about any previous attacks and tries to find anomalies in the network data, which could be possible signs of attacks. In recent years, machine learning approaches have been used successfully for both of these techniques [9,10,16].

Recent years show many promising results of applying deep learning methods to machine learning problems and intrusion detection is not an exception for this case. In [8,20], the authors propose applying recurrent neural networks to intrusion detection systems and got very promising results. These works only show that RNN could be used while detecting anomalies in related data and they do not propose a complete end-to-end intrusion detection system. Our approach differs from these previous approaches in that it attempts to build a self-healing cloud system through deep learning with recurrent neural networks, which integrates time dependencies between observations (data points) in the system into the learning process to provide a more accurate representation of the attack progression and normal system processes.

3 Proposed Approach

In this work, we propose a cloud intrusion detection system that works with real-time cloud data analytics to detect possible attacks and develop a resilience mechanism through deep learning with recurrent neural networks.

The solution involves the collection of system metrics from cloud platforms and near real-time processing of those metrics using big data analytics to discover anomalies. Metric collection is done by metric collection agents deployed in related parties like guest VMs. These data include network packets and other related metrics like VM usage metrics, HTTP server performance etc. After collection, these metrics are sent to a stream to be processed by a stream processing engine. The stream processing engine gathers the metrics inside the stream, considering their timestamps and processes these data by feeding them to a recurrent neural network trained previously. If the network finds an anomaly in the data, it labels it and triggers an alarm to inform the system administrators. The details of these steps are listed below.

3.1 Metric Collection

Popular cloud system providers such as AWS[1] share the statistics and state of their cloud systems through an API. These statistics contain utilization of CPU, RAM, disks, number of packets/bytes received/transmitted, and many other details about the current state of the system. In this work, we utilize guest VM agents for metric collection, since this approach does not depend on the cloud infrastructure and is more flexible than virtual machine monitor solutions. At the metric collection phase, the agents collect the required metrics from the guest VM like network flow, basic system usage metrics such as CPU utilization, disk read/write metrics etc. and usage metrics of applications that can affect the system performance. The metric collection agent has two components, the

[1] https://aws.amazon.com/.

producer and the consumer. The producer component gathers the system and application metrics from the VM using different interfaces. To achieve this, the producer must have a pluggable architecture that written plug-ins can gather the metrics from, knowing how to get them. The responsibility of the consumer side is to gather metric records from the producer and pass them onto the processing phase.

3.2 Metric Processing

Due to the large volume and velocity of the data collected from cloud systems, big data processing frameworks are needed to analyze the data. Big data can be processed as batches or as streams. Deciding which type of processing is needed is up to the task. If we handle the data as batches, we need to wait for some amount of time to create batches from the given data. After the data become batch, the processing starts. This contradicts with our purpose of near real-time detection in this work, as we need to act in real time in order to prevent or stop attacks before they can harm the cloud system. Stream processing on the other hand involves handling the data in memory as they arrive.

In this work, we utilize Apache Spark [2] to process the stream data collected from cloud systems. Spark has advantages like fault-tolerance, in-memory computation, being faster than similar frameworks, having a wider community, multiple language support etc. The data that streams from our cloud systems are handled by Spark and served to our algorithm in order to detect possible attacks. Multiple networks can be watched by using this framework.

To support stream processing, many tools are available to specifically handle the requirements of this process. Tools like Apache Kafka [1] and Amazon Kinesis [3] streams provide great support for handling stream data in a scalable way. In this solution, we use Apache Kafka to collect the metrics from the guest VM agents and pipe them to the stream processing engine.

3.3 RNN-Based Learning for Anomaly Detection

Signature-based intrusion detection systems rely on detailed information about previously observed attacks. These approaches fail in the case of cloud systems, which are open to attacks that might be novel. On the other hand, unsupervised learning methods enable us to prevent or at least detect changes in the system parameters, i.e. the normal behavior of the system. By this way, the system will be able to detect anomalies and will try to prevent if there is an attack going on. In the mean time, alarms will be created in the system so that if the security system cannot stop the attack, it will warn the user/owner of the cloud system. This is actually the main difference from a signature-based intrusion detection system. If this type of system does not have any information about an attack, it will most likely be missed. On the other hand, for a system with an unsupervised learning algorithm, even a minor anomaly might cause the system to detect if something is wrong. When run on isolated data points/cloud activity logs, unsupervised algorithms may not achieve very high accuracy due to noise in the data. For instance, observation of a sudden spike in CPU utilization might signal a possible attack even if it was caused by a legitimate process and does not persist for a long period, not causing any degradation in the performance of the system. Precisely for this reason, we need to be able to model the time-based behavior of the system by considering the data points collectively as a time series rather than isolated incidents.

Recent advances in deep neural networks have made it an effective tool for many supervised and unsupervised learning tasks, achieving higher accuracy than competing approaches. Recurrent neural networks (RNN) are machine learning models consisting of nodes that are connected to each other. These nodes can memorize and pass information in a sequence, though they process the data one by one. Therefore they can handle inputs and outputs that are dependent on each other. RNNs have been successful in various tasks such as image captioning, speech synthesis, time series prediction, video analysis, controlling a robot, translating natural language and music generation [11].

Normally, there is only one single network layer in a node of a classic RNN. In conventional neural networks, it is not defined how the network will remember events of the past to use the information about them in the future. Recurrent neural networks aim to solve this issue by using the architecture depicted in Fig. 1:

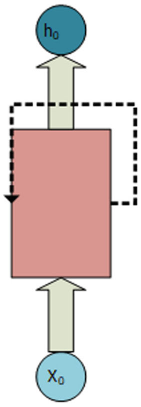

Fig. 1. An RNN loop

As shown in the diagram, the network gets an input x, processes it, and outputs an output h. The outcome of the process is used in the next step. To make it clear, the loop is demonstrated in an open form in Fig. 2.

The equation below shows the network mathematically:

$$h_t = \theta(Wx_t + Uh_{t-1})$$

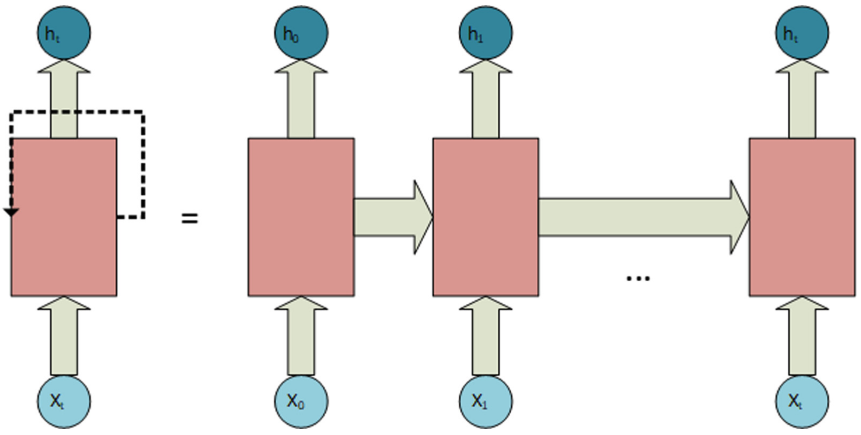

Fig. 2. An unrolled RNN loop

Here W stands for the weight matrix, which is multiplied by the input of the current time. The result is added to the multiplication of the output (hidden state) of the previous time step and its own hidden state and the hidden state matrix (transition matrix) U. These weight matrices are used to define how much of the information both from the current input and past output will be used to determine the current output. If they generate an error, it will be used to update the weights to minimize error. The resulting sum is condensed by the hyperbolic tangent function θ [6].

Some example uses of this standard RNN architecture include predicting the next character in a series of letters, picking the next note after a sequence of notes of a song, deciding where to go when controlling the motion of a robot etc. In our case, we use RNN in order to predict an intrusion, but we use LSTM-RNN because of the reasons explained below.

LSTM stands for Long Short Term Memory. Without it, gradients that are computed in training might get closer to zero (in case of multiplying values between zero and one) or overflow (in case of multiplying large values). In other words, as the time sequences grow, RNN might not connect older inputs to the outputs. LSTM adds additional gates to the architecture to control the cell state. By this modification, training over long sequences is not a problem anymore.

In an LSTM-RNN there are four layers, which interact with each other. First of all, the input is received and copies itself into four. The first one goes into a sigmoid layer. This layer decides whether the output of the previous layer is needed and should be used or it should be thrown away. Then another sigmoid layer decides which values are going to be updated. A tanh layer generates possible values, which might be included in the state. These two layers get combined to update the state. Finally another sigmoid layer picks what we are going to output from our cell state (Fig. 3).

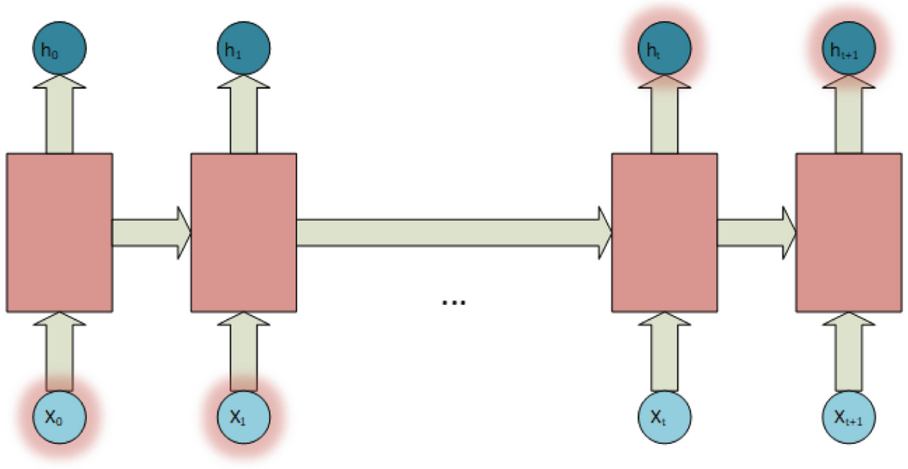

Fig. 3. Disconnected dependencies

In the proposed model, we utilize the LSTM recurrent neural networks (RNN) [7] algorithm to detect deviations from the normal behavior of the cloud system under monitoring. Note that because of the nature of the algorithm, it first needs to learn the normal state of the system. By processing the normal state, the system will detect anomalies when metric values that deviate significantly from the normal behavior of the system are observed. In RNNs, inputs are not independent, every time sequence uses information from the previous ones. This feature perfectly suits our task, as we cannot directly specify if there is an anomaly without analyzing the system's state for the time being (Fig. 4).

The algorithm receives parameters of the system from Spark and uses those parameters as a time series input. The parameters indicate the state of the system's properties for that time series. The algorithm then serves these parameters to its prediction function. The prediction function tries to find out if there is an anomaly in the system. For example, if there is an unrealistic peak in the CPU utilization and number of disk operations and incoming network packets, this might indicate that the system is under a denial of service attack. From this point, the system can create an alarm to warn system administrators or initiate a security action (Fig. 5).

We have used LSTM-RNN in Tensorflow. LSTM is actually handled by Tensorflow itself, but we needed to convert some of the fields in the data as we could not pass them directly to the algorithm. For example, fields like ip addresses, protocol types, service types etc. were converted to data types that LSTM-RNN accepts, as strings are not accepted. There was only one output for our experiment, which is the actual result: whether there was an attack (1) or not (0). How LSTM-RNN works in general is described below step by step [15]:

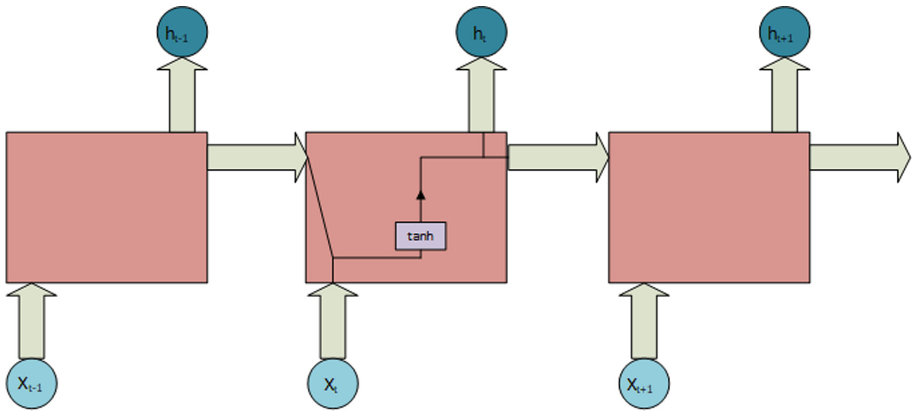

Fig. 4. Single layered structure of standard RNN

1. The first layer gets the current input and output of the past time series, then decides if the previous information is needed now. Actually this layer can be called the *forget* layer. h stands for the output of the past, x stands for the current input, W is the weight of this layer, and b is the bias.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

2. Then we move onto the input layer. This layer is another sigmoid layer, which decides the values that are going to be updated.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

3. A hyperbolic tangent layer creates candidate values, which might be included in the cell state. Cell state is a straight line in our network that flows for entire network. LSTM changes information on this state across the road with the help of the gates.

$$cdt_t = \tanh(W_{cdt}.[h_{t-1}, x_t] + b_{cdt})$$

4. Results of all previous steps are combined in order to create an update to the cell state.

$$c_t = f_t * c_{t-1} + i_t * cdt_t$$

5. Finally, the output is decided. Naturally, the cell state is used in deciding. Another sigmoid layer takes part, and its output is multiplied the by cell state (state will go into tanh first).

$$h_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) * \tanh(c_t)$$

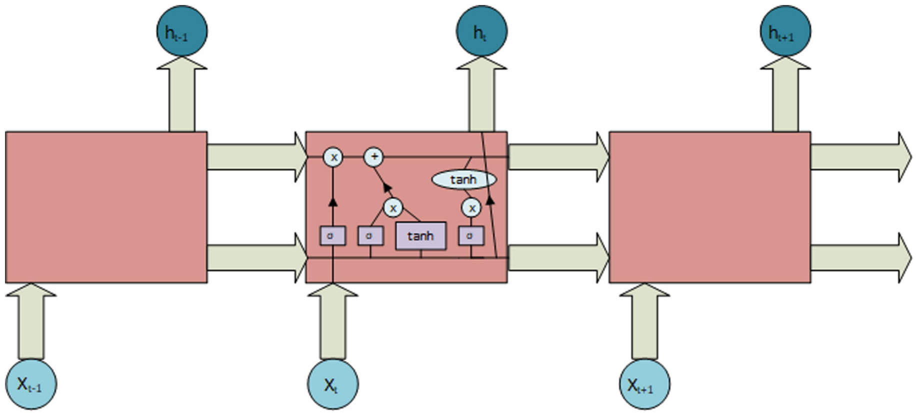

Fig. 5. Four layered structure of LSTM-RNN

4 Evaluation

We have developed an initial prototype of the proposed cloud IDS using Apache Spark collecting data from machine instances on AWS. Figure 6 demonstrates the high-level operation of the developed prototype.

To evaluate the accuracy of the RNN component in the proposed cloud intrusion detection system, we have used the "UNSW-NB15" dataset of UNSW ADFA [14] for experiments. In this dataset, there are attack and non-attack records of network traffic. The total number of records in the dataset is approximately two and a half millions, making it quite comprehensive. The test and training sets as partitions of the original dataset are also provided. The records have 49 fields in total, a subset of which is listed in Table 1 below, including information like source IP address, source port, protocol, number of packets on requests and responses, bytes of those packets, number of losses, attack type if there is one etc.

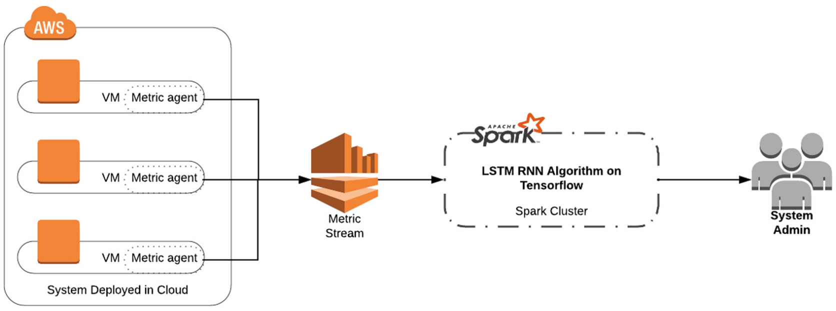

Fig. 6. Big data processing system operation

We have simulated the data so that the system behaves like the data are streaming. At first, a training set is provided for LSTM-RNN, which runs on TensorFlow [19], which is an open-source machine learning framework. With the help of the model that is created on the framework, the system decides whether a record seems like an attack or not. We ran the model using 50000 records for training, and 650000 records for testing. The error rate was below 7%, which is a promising accuracy for intrusion detection.

Table 1. UNSW-NB15 parameters

No.	Name	Type	Description
1	srcip	Nominal	Source IP address
2	sport	Integer	Source port number
3	dstip	Nominal	Destination IP address
4	dsport	Integer	Destination port number
5	proto	Nominal	Transaction protocol
6	state	Nominal	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)
7	dur	Float	Record total duration
8	sbytes	Integer	Source to destination transaction bytes
9	dbytes	Integer	Destination to source transaction bytes
10	sttl	Integer	Source to destination time to live value
11	dttl	Integer	Destination to source time to live value
12	sloss	Integer	Source packets retransmitted or dropped
13	dloss	Integer	Destination packets retransmitted or dropped
14	service	Nominal	http, ftp, smtp, ssh, dns, ftp-data, irc and (-) if not much used service
15	Sload	Float	Source bits per second
16	Dload	Float	Destination bits per second
17	Spkts	Integer	Source to destination packet count
18	Dpkts	Integer	Destination to source packet count
19	swin	Integer	Source TCP window advertisement value
20	dwin	Integer	Destination TCP window advertisement value
21	stcpb	Integer	Source TCP base sequence number
22	dtcpb	Integer	Destination TCP base sequence number
...	...	...	...
48	attack cat	Nominal	The name of each attack category. In this data set, nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
49	Label	Binary	0 for normal and 1 for attack records

5 Conclusion

In this work, we proposed an intrusion detection system for cloud computing that leverages big data analytics with recurrent neural networks. We have implemented an initial prototype of the proposed architecture with Apache Spark and evaluated the performance of recurrent neural networks on a network attack dataset, which provided promising attack detection accuracy results. We have used LSTM-RNN algorithm in Tensorflow to detect attacks in the network. LSTM-RNN has been picked as we were interested with using time series data. Results of this work looked promising about integrating big data analytics to intrusion detection in cloud. Our future work will focus on the integration of a deep reinforcement learning based model, which will provide automated self-healing of cloud systems by learning from the consequences of attacks on the system, thereby avoiding states that are vulnerable to attacks. We will also experiment with real data streams gathered using cloud APIs and quantify the delays of big data processing as well as the mean time to detect attacks and recover in major cloud platforms. Extensive experimentation with various attack types on cloud systems will be conducted to evaluate and optimize the big data analytics-based intrusion detection framework.

Acknowledgement. This work was supported by Middle East Technical University under grant number: YOP-312-2018-2819.

References

1. Apache: Apache kafka, a distributed streaming platform. https://kafka.apache.org/. Accessed Apr 2018
2. Apache: Apache spark, lightning-fast unified analytics engine. https://spark.apache.org/. Accessed Apr 2018
3. AWS: Amazon kinesis. https://aws.amazon.com/kinesis/. Accessed Apr 2018
4. Bharadwaja, S., Sun, W., Niamat, M., Shen, F.: Collabra: a xen hypervisor based collaborative intrusion detection system. In: 2011 Eighth International Conference on Information Technology: New Generations, pp. 695–700, April 2011. https://doi.org/10.1109/ITNG.2011.123
5. Casas, P., Soro, F., Vanerio, J., Settanni, G., D'Alconzo, A.: Network security and anomaly detection with big-DAMA, a big data analytics framework. In: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), pp. 1–7, September 2017. https://doi.org/10.1109/CloudNet.2017.8071525
6. DL4J: A beginner's guide to recurrent networks and LSTMs. https://deeplearning4j.org/lstm.html. Accessed Apr 2018
7. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
8. Kim, J., Kim, H.: Applying recurrent neural network to intrusion detection with Hessian free optimization. In: Kim, H., Choi, D. (eds.) WISA 2015. LNCS, vol. 9503, pp. 357–369. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31875-2_30

9. Kumar, R.S.S., Wicker, A., Swann, M.: Practical machine learning for cloud intrusion detection: challenges and the way forward. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec 2017, pp. 81–90. ACM, New York (2017). https://doi.org/10.1145/3128572.3140445
10. Li, Z., Sun, W., Wang, L.: A neural network based distributed intrusion detection system on cloud platform. In: 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, vol. 01, pp. 75–79, October 2012. https://doi.org/10.1109/CCIS.2012.6664371
11. Lipton, Z., Elkan, C., Berkowitz, J.: A critical review of recurrent neural networks for sequence learning. Accessed Apr 2015
12. Maiero, C., Miculan, M.: Unobservable intrusion detection based on call traces in paravirtualized systems. In: Proceedings of the International Conference on Security and Cryptography, pp. 300–306, July 2011
13. Mishra, P., Pilli, E.S., Varadharajan, V., Tupakula, U.: Intrusion detection techniques in cloud environment: a survey. J. Netw. Comput. Appl. **77**, 18–47 (2017). https://doi.org/10.1016/j.jnca.2016.10.015. http://www.sciencedirect.com/science/article/pii/S1084804516302417
14. Moustafa, N., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: Proceedings of the Military Communications and Information Systems Conference, pp. 1–6 (2015)
15. Olah, C.: Understanding LSTM networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed Apr 2018
16. Pandeeswari, N., Kumar, G.: Anomaly detection system in cloud environment using fuzzy clustering based ANN. Mob. Netw. Appl. **21**(3), 494–505 (2016). https://doi.org/10.1007/s11036-015-0644-x
17. Sharif, M.I., Lee, W., Cui, W., Lanzi, A.: Secure in-VM monitoring using hardware virtualization. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 477–487. ACM, New York (2009). http://doi.acm.org/10.1145/1653662.1653720
18. Sinanc, D., Demirezen, U., Sagiroglu, S.: Evaluations of big data processing. Serv. Trans. Big Data **3**, 44–54 (2016). https://doi.org/10.29268/stbd.2016.3.1.4
19. TensorFlow: an open source machine learning framework for everyone. https://www.tensorflow.org/. Accessed Apr 2018
20. Yin, C., Zhu, Y., Fei, J., He, X.: A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access **5**, 21954–21961 (2017). https://doi.org/10.1109/ACCESS.2017.2762418