FINAL PROJECT Nazif ÇELİK 090200712

2211.10062

INTRUSION DETECTION IN INTERNET OF THINGS USING CONVOLUTIONAL NEURAL NETWORKS

**INTRODUCTION**

Internet of Things (IoT) interconnects billions of devices and keeps growing. Within the IoT, all kinds of objects communicate without human intervention. By reading data from sensors, or controlling actuators remotely, IoT enables services ranging from home automation to optimisation of complex workflows in the industry. Despite the great potential of IoT and IIoT, economically successful devices often lack a robust security implementation. This is especially alarming in industrial applications. A compromised IIoT system may lead to economic loss, physical damage to equipment or goods, and can even cause bodily harm to people. Along protective measures, cyber security involves the recognition of potential threats. An Intrusion Detection System (IDS) is a software component that monitors and analyses various activities and measures of the system to detect attacks. The complexity and quantity of attacks push for more efficient IDSs. Although the traditional machine learning provides quick processing, its design is slowed down by the manual feature engineering for each new threat. In contrast, the deep learning brings an end-to-end approach combining feature selection and classification. Its automation speeds up the defence response against the fast-evolving cyberattacks. Our approach in intrusion detection is a novel use of classical CNNs on sensor readings with a time-based encoding leveraging on the visual patterns of missing values. Akin to slit-scan photography, stacked sensor data represent an evolution of a state in time. Our method thus enables a CNN to directly capture temporal patterns.

**MAIN BODY**

Over the past 5 years, the interest in securing IoT has seen an increasing trend indicated by a growing number of intrusion detection surveys that either include IoT or also focus specifically on IoT security. The increased vulnerability of IoT devices stem from their limited resources (memory, power supply, processing power, bandwidth). Others include real-time attack detection; improvement of validation methods; availability of training datasets and their efficient use, zero-day attacks, integration with other technologies (e.g. blockchain); and dealing with evasion techniques. Common IoT attacks, such as password attacks via Recurrent Generative Adversary Networks (GANs), expose the lack of secure device configuration and use of default passwords. Jamming and Denial of service (DoS) attacks prevent correct functioning of devices or disrupt the stream of information critical for some processes. DoS attack floods the victim with requests and can slow down, induce crash, or shutdown the target. proposes an entropy-based solution in Software Defined Networking (SDN). In a Man-in-the-Middle attack, the malicious agent covertly intercepts and alters the communication. Introduces an attack on MQTT exchange using an adversarial message generation. Simple web servers are often a user interface for IoT devices, or the dashboard for their management. Therefore, cross-site scripting (XSS) on web resources that do not properly sanitise their inputs, triggering of a broken session management, or having deficiencies in their application logic are of concern. characteristics of the network and devices in order to exploit their weaknesses. For instance, makes use of Shodan, an online database, to find potentially vulnerable devices. Zero-day attacks are characterised by their novelty and can be of any of previously mentioned types. They present a serious threat as the attack vector had not been described before their deployment. To deal with them, proposes a context-graph model to support zero-day attack detection, response and re-establishment of trust. A detection using DL which uses auto-encoders on CICIDS2017 network capture dataset. From another perspective, an anomalous power consumption of IoT devices can reveal an attack on the network. Deep Learning (DL) is becoming an unavoidable part of the attack detection toolset. Compared to shallow neural networks, deep neural networks are able to detect higher-level concepts. Competitiveness of deep learning improved with an increased Graphical Processing Units (GPUs) availability and parallel computing of layers that make the complex computations faster. Their approach reshapes the features of a single record into a square matrix that is then used as the input to a CNN, while in our approach the features of a single record form only one row of the input matrix. Aimed at IoT botnet detection a system performs a continuous analysis based on several ML methods, which include Elliptic Envelope, Isolation Forest, Local Outlier Factor, and One-class Support Vector Machine. They evaluated the impact of their solution in terms of memory and CPU load. Our approach adapts timeseries directly in the encoding as a dimension in the input tensor for the CNNs. CNNs were incorporated in network traffic intrusion detection as a part of an ensemble method applied to Ethernet Consist Network. However, the temporal aspect of the data was addressed by another component based on LSTM. Similarly, uses CNNs uniquely on a set of features reshaped in a two-dimensional array. In our approach, CNNs are applied on “slit-scan” images to capture the temporal links.

**EXPERIMENTS**

Our aim is to use sensor readings to detect attacks in an IoT system. The process starts from reading the annotated multiple sensor sources and prepares the data for deep learning. Deep learning models are then used to predict whether there is an attack and if so, determine its type. In 2019, TON IoT dataset was created based on a testbed environment at the Cyber Range and IoT Labs at the University of New South Wales (UNSW) Canberra, Australia. The experimental setting comprises a combination of physical and simulated connected devices. The dataset name “TON IoT” refers to its contents and provenance: Telemetry data, Operating systems’ data, and Network data from the IoT/IIoT testbed. The testbed was organised into three layers connected with virtualised switches and routers: Cloud layer (virtualised), Fog layer (virtualised), and Edge layer (physical). Cloud layer contains a Message Queuing Telemetry Transport (MQTT) broker. Fog layer contains the monitored vulnerable hosts and attacker machines. It includes a Node-RED server that simulates 6 out of 7 sensor sources. Edge layer contains the physical devices, from which an ESP8266 weather station acts as the 7th sensor data source. Other devices in the edge layer are two smartphones and a smart TV and the NSX VMware server emulating virtualised layers. Aiming at capturing the specifics of IoT and IIoT systems, we focus on the IoT telemetry part of this dataset. TON IoT telemetry dataset provides the sensor readings as registered by a Node-RED server, delivered through the MQTT protocol via the MQTT broker in the Cloud layer. This part of data is provided in the form of comma-separated values (CSV) files, one per sensor. The selected TON IoT dataset is highly heterogeneous. We provide a closer look at its content for a better understanding of its characteristics. The data is of these categories: IoT telemetry data (i.e. IoT/IIoT sensor readings, on which we focus in this paper), network data, Linux host data, and Windows host data. Additionally, there is a specification of ground truth for the security events. The dataset also contains basic statistics about the number of records in each category. For clarity, we will refer to them by aliases. TON IoT’s “Processed IoT dataset” will be referred to as “PR dataset”, TON IoT’s “Train Test IoT dataset” will be called “TT dataset”. “Train Test dataset”, despite its name, actually does not contain a dataset split into Train and Test datasets. Instead it is a single dataset that its authors mention to be used for training and testing in 80 : 20 proportion. The Raw dataset contains the source data in different formats (e.g. log, JSON) depending on sensor type. Its conversion into CSV format produces the PR dataset where each of 7 sensors has the data in a separate CSV file. The TT dataset, composed of 7 CSV files as well, is a subset of the records from the PR dataset. The telemetry data come from 7 sensors. Six sensors were simulated in Node-RED framework. The seventh sensor, ESP8266 weather station, was physically connected to the testbed infrastructure. Both datasets (PR and TT) offer 17 features from 7 sensors along with a date and time indication. Every data recording was annotated with a binary label field (0 means normal operation, 1 indicates an attack). In addition, the type field specifies the event type: if label = 0, normal, otherwise it indicates backdoor, ddos, injection, password, ransomware, scanning, or xss. In the TT dataset, there is a total of 401,119 readings from all sensors, which were recorded having 85,664 unique timestamps. In an ideal data processing case, all sensors would provide a reading at the same time, and timestamps would be unique within each sensor’s readings. In reality, neither condition is satisfied. After removing duplicate entries with respect to date-time, i.e. a single data point represents a given moment, segments marked as “A” and “B” would reduce to a few points. The most visible contraction of the data is that the first 150,000 records, marked in the figure as “A” (almost a third of the TT dataset) spread only across less than 2000 different timestamps representing less than 8 minutes. The removal of redundant timestamps affects the balance of different classes in the dataset. As the most of the duplicate timestamps are in the normal class, its proportion drops from 61 % to only 25 %. The backdoor class becomes the dominant class with almost 26 %. As the models tend to be biased towards the majority class, a bias towards the backdoor class is to be expected when using the datasets with this distribution. The data processing method proposed in this experiment converts the time-based one-dimensional data from the PR dataset into tensors that are accepted by CNNs. consists of the following steps: combination, aggregation, segmentation, partitioning, imputation, tensor construction, before serving the data to CNNs for training and testing. Specifically, the combination puts various sensor readings into a single dataset. The aggregation deals with the issue of multiple readings from a sensor recorded with the same timestamp. The segmentation divides the whole dataset into a set of chunks to preserve temporal contiguity of sensor readings. These chunks are then randomly dispatched into either training set or test set by the partitioning operation. They are used to compare the performance of various proposed strategies. An imputation strategy defines how missing values, that appear during the combination step, are interpreted. The tensor construction selects sequences of records, applying the imputation strategy on them to form a three-channel imagelike tensor as the input to the CNNs.

1) Combination: The combination step takes seven sensors’ source CSV files as inputs and outputs a fusion of the records in a single object that contains all features in columns. concatenation and group-bytimestamp. Concatenation, places different sensor readings on separate rows, leaving other sensor’s values unassigned. Group-by-timestamp approach combines sensor readings having the same timestamp within one row. In case of multiple rows having the same timestamp within one source (i.e. sensor), each row has an order number assigned, which can be used to join several readings without aggregating the result. As a result, the dataset can contain multiple readings for one timestamp. The representation is more dense (less unassigned measurement values) and takes into account the temporal co-occurrence of different values as they are combined into one data point. An issue with redundant timestamps is that there is no other ordering indication other than order of appearance in the original dataset (CSV file).

2) Aggregation: The aggregation takes a table-like object and contracts rows with the same timestamp into one, effectively reducing the number of records, outputting the same table-like object with an additional counter column with the number of represented rows. The data exploration showed that the datasets contain multiple records for the same sensor within the same second. The aggregation step has a twofold purpose: enabling a time-based learning and making the detection system more robust to measurement flooding. The learning based on past states is supported in later steps by selecting a fixed number of records for the machine learning. In this way, the machine can take into account the previous states of the system. We explore the performance of two approaches: event-based (not aggregated where all sensor readings are provided sequentially), and time-based (timeaggregated where the records represent a fixed time frame). All records with the same timestamp are replaced by a single record. A counter column is added to indicate how many measurements each output record represents. This method implemented the strategy of keeping the first reading of each sensor with the given timestamp. The counter column has not been used in training or testing of the models.

3) Segmentation: This operation accepts a series of records and outputs a series divided in blocks of the given size. Our method intends to leverage on the evolution of the sensor readings in time and therefore, temporal continuity must be conserved as much as possible. Furthermore, we need to make sure that training and test datasets are absolutely disjoint. As the TON IoT dataset does not provide separate training and test sets, we segment the full dataset into chunks of a parametrisable size. We argue that the more subsequent readings are kept together, the better we can capture continuous evolution of the system and the patterns of an attack.

4) Partitioning: Taking the blocks of rows as input, this step outputs two datasets composed of records from these blocks. Partitioning decides whether a segment of the aggregated dataset records will be part of the training or the testing dataset. A random seed is stored to be able to reproduce the uniform distribution given the percentage driving the partitioning. A verification is performed to ensure that each class is well represented in both partitions. The resulting datasets are TT500 and PR1000.

5) Imputation: The imputation step accepts a set of records and outputs the same records modified in a way that there are no missing values. To construct a picture of the system’s status from multiple sources of sensor readings, we face their misalignment. Sensors have different reporting frequencies and periods of activity. In order to keep a maximal extent of the detection, we are interested in all recordings having at least one sensor emitting. When a sensor is not emitting any data (while others are emitting), it appears as having “missing values”. The processing of the missing values is worth of a particular attention. Imputing them without careful analysis may worsen the overall result as indicated by our results of DataWig imputation. IoT devices, due to their embedded character, are prone to being affected by network saturation. Therefore, missing values or even duplicated readings can provide valuable information for network status inference. The strategies we implemented to deal with missing values constant imputation, fill-forward, median or mode value (according to the data type), miss, and “DataWig” that uses the eponymous toolbox. It trains a deep learning model to choose the best fit for each column based on other columns, our strategy implements option datawig.SimpleImputer.complete(). Constant imputation strategy replaces missing fields with a pre-defined arbitrary value (e.g. −100 or 0). Fill-forward strategy propagates the last available value. In order to fill all values, backward fill was used as well. Median and mode imputation is a common practice, too. An experimental approach, dubbed miss, has been introduced by ignoring all values completely and to perform machine learning only on the mask of missing values, that is a binary array indicating 1 where a value is missing and 0 where a value was found. Finally, DataWig’s SimpleImputer that analysed data and filled the missing values in an autonomous manner, by applying deep learning. As three channels are available as inputs for the chosen CNNs, mixed strategies were designed where different channels used different strategies. The first five rows contain pure strategies where all three channels receive the same inputs, equivalent to a greyscale picture. The remaining rows describe composed strategies to explore the cases when channels’ strategies differ.

6) Tensor Construction: The transformation of multivariate time-series vectors into a tensor data was performed in a straightforward manner, where the columns in the tensor are the readings from different sensors and the rows represent points in time. As the target CNN architectures (i.e. ResNet-50 and EfficientNet-B0) take a tensor of 224 × 224 × 3 channels, the input tensors were padded with zeros to reach the required shape (column count). In addition, each channel of the tensor refers to a particular imputation strategy. The step parameter is a positive integer. The sample contains 224 records from our input. The following sample contains the same data but shifted by step rows, adding step more recent rows, thus forming a sliding window on our data. Lower values of step produce more data samples with lower variation. However, based on our early experiments, the value of 20 reduces the training time without negatively affecting the outcome. As a sample contains several annotated records, we need to determine the label for the sample. We adopted the strategy of the latest label appearing in the set of records used for the sample. The rationale is that the previous records are the past readings and only the last one is the current one, thus we chose its label to decide the ground truth. This approach is applicable on TT and PR datasets as the labels are grouped in time and do not interfere (with the exception of scanning attack that is interlaced with password attack for the Motion Light sensor).

**DISCUSSIONS**

Two particular cases of wrong detections could be noticed: attack switched to a different type within the current window, a single-point different attack amidst of a uniform type. In order to explain them, we look at the dataset and the sampling. The implementation of the sliding window does not skip discontinuities. When the attack type changes, the most of the data indicates previous attack and it is understandable that the previous type is detected. Similarly, a single-point attack detection is an outlier within a continuous block of the same-type attack. It is challenging or impossible to detect these with convolution networks, as a single row does not change the overall appearance of the image. A change in temporal representation could address this issue. For instance, more recent rows could be given more space, older rows less space. Newer ones could be duplicated to be noticed and older rows aggregated to provide a more stable history background. Although we were able to obtain an important improvement of TPR and FPR (from 88 % to 92.43 % and from 12 % to 0.66 % respectively) compared to the baseline, we would like to point out limitations of this method. TON IoT datasets lack repetitions of the same attack, the observations contain significant time gaps, the time without observations is not labelled. For example, all events and only events on 27-Apr-19 between 04:57:36 and 05:40:53 are labelled as XSS. In addition, no transition from normal state to any instance of the attack appears in the data. Normal observations are separated from attack scenarios by an observation gap more than 20 days long. As the recordings of the data during attacks are concentrated within a relatively short period of time (hours), we have no data to confirm that the same attack on the next day could be detected. In other words, as there is little variability, the model might as well learn ephemeral patterns that depend more on the time of day than on what actually happens in the network. Therefore, we cannot rule out that the system performs well on this dataset only because the dataset is flawed in a way that produces specific patterns for each segment of time. Multiple scenarios of the same type of attack could immensely improve the confidence in the tested methods. A reliable comparison to the baseline is challenging as the TON IoT dataset does not provide separate training and test sets. A future extension of our experimentation could explore simulated removal, addition or replacement of sensors. The approach is potentially scalable to 224 features as only 17 slots out of 224 were used in the current experimentation, for higher levels of EfficientNet and ResNet, more features can be used. These observations may contribute to future dataset evaluation methods. Amidst calls for more real-time IoT datasets, a framework for their evaluation could benefit potential data providers during their testbed design and data collection considerations.

**CONCLUSION**

In the context of cybersecurity challenges in IoT and IIoT, this project explores novel possibilities of deep learning applications on sensor readings to determine the intrusion state of the system. We deployed two CNN architectures on the same data to observe how their performance differs. The deployed architectures are ResNet-50 and EfficientNet-B0. We presented an approach to represent sensor readings for use with CNNs that takes into account past sensor readings as the context for the current prediction. The time axis corresponds to the vertical axis in the encoding of the image to be processed by CNNs. We also showed how the presence or absence of values in the dataset can influence the intrusion detection of the monitored system. Overall results in binary classification show a significant improvement compared to the baseline. Leveraging on patterns of missing values in time, false alarm rate fell from 12 % in the best baseline to 0.3 % or 0.7 % with ResNet or EfficientNet respectively, while the recall (true positive rate) raised from 88 % to 92 % or 94 % respectively.