Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

AI-IDS: Application of Deep Learning to Real-time Web Intrusion Detection

AECHAN KIM[1,2], MOHYUN PARK[2], AND DONG HOON LEE[1], (Member, IEEE)
[1]Graduate School of Information Security, Korea University, Seoul 02841, South Korea
[2]Financial Security Institute (FSI), Yongin 16881, South Korea

Corresponding author: Dong Hoon Lee (donghlee@korea.ac.kr)

ABSTRACT Deep Learning has been widely applied to problems in detecting various network attacks. However, no cases on network security have shown applications of various deep learning algorithms in real-time services beyond experimental conditions. Moreover, owing to the integration of high-performance computing, it is necessary to apply systems that can handle large-scale traffic. Given the rapid evolution of web-attacks, we implemented and applied our Artificial Intelligence-based Intrusion Detection System (AI-IDS). We propose an optimal convolutional neural network and long short-term memory network (CNN-LSTM) model, normalized UTF-8 character encoding for Spatial Feature Learning (SFL) to adequately extract the characteristics of real-time HTTP traffic without encryption, calculating entropy, and compression. We demonstrated its excellence through repeated experiments on two public datasets (CSIC-2010, CICIDS2017) and fixed real-time data. By training payloads that analyzed true or false positives with a labeling tool, AI-IDS distinguishes sophisticated attacks, such as unknown patterns, encoded or obfuscated attacks from benign traffic. It is a flexible and scalable system that is implemented based on Docker images, separating user-defined functions by independent images. It also helps to write and improve Snort rules for signature-based IDS based on newly identified patterns. As the model calculates the malicious probability by continuous training, it could accurately analyze unknown web-attacks.

INDEX TERMS Computer networks, intrusion detection, neural networks, large-scale systems, intelligent systems, Real time systems, security, CNN-LSTM

I. INTRODUCTION

As technology evolves, cyber-criminals are also improving their attack methods, tools, and techniques to exploit organizations. In particular, public web-services are common services that anyone can access, and many companies provide their services through open webpages. If a web-service fails or is compromised, it can cause a drop in corporate reputation or revenue. In general, security managers prevent intrusions from external attacks by registering all denied black-list policies for unused services in the firewall, but web-services in the Internet Demilitarized Zone (DMZ) cannot be blocked by firewalls because they are always open to public access. As such, identifying normal access and differentiating it from malicious attacks is an important task in cybersecurity. In reality, many security incidents originated with web-attacks such as information disclosure, service failures, and malware infections.

Hypertext transfer protocol (HTTP) [1] is an application-level protocol for distributed, collaborative, and hypertext information systems. Today's HTTP is evolving where the information is transferred from web pages, and it is also used for exchanges or sending system commands to various devices, such as command-lines, update scripts, and mobile apps. Web-attacks often exploit vulnerabilities in applications in open web services rather than perform a host-level system penetration. The attacker attempts to attack by sending exploitational code using a vulnerability in a specific domain or path file of the webserver. The webserver or device that is injected with the code can subsequently be compromised by the attacker [2].

Traditionally, intrusion detection is a major research field in network security, as it is important to identify unusual access to secure internal networks. An Intrusion Detection System (IDS) is used to identify intrusions, attacks, or violations of security policies in a network or host system promptly [3]. An IDS system that inspects a packet of networks to detect attacks is called Network Intrusion Detection System (NIDS). An NIDS is collected using network equipment via mirroring by network devices, such as switches, routers, and network terminal access points (TAP), which is a surveillance device for monitoring network infringements and policy violations [4]. Many organizations operate NIDS with firewalls and an application firewall (L7) to protect webservers that are on the same network and system. An NIDS runs mostly signature-based detection by Snort IDS rules. The analyst writes a user-defined pattern into the rules to detect an attack. When there is a malicious payload on the network traffic, the rule triggers security events, including detection time, source/destination IP (metadata), and some raw packets (payloads). String or pattern match is reliable and generates very few false alarms but does not identify unknown or irregular pattern attacks. Recent sophisticated cyber-attacks use irregular patterns such as encoding and obfuscation to bypass security systems. To solve these problems, we applied AI-IDS to detect variant attacks that cannot be identified by legacy signature-based NIDS.

A. LIST OF CONTRIBUTIONS

The main contributions of this paper are summarized as follows:

1) APPLYING DEEP LEARNING TO REAL-WORLD NETWORKS

We have successfully applied AI-IDS to big-data scale traffic. The AI-IDS is a flexible and scalable system that is implemented based on Docker images, and separates user-defined functions by independent images.

2) PROPOSE A FAST AND EFFECTIVE PREPROCESSING METHOD

We implemented fast and effective spatial feature learning through normalized UTF-8 character encoding, even if we do not apply computationally intensive algorithms, such as entropy, compression, and encryption. For example, the entropy of a string requires probability calculation, followed by multiplication and logarithm. Instead, our proposed method can preprocess strings with a single operation.

3) PROPOSE OPTIMIZED CNN-LSTM MODEL FOR BIG DATA

We demonstrated the process of model design in detail via performance evaluation between CNN-LSTM, LSTM-CNN, and DNN models based on fixed real-time data from HTTP request packets. Hyper-parameters were determined in each model through repeat experiments. An optimized neural network model was validated through experiments on public datasets (CSIC-2010, CICIDS2017) and fixed real-time data.

4) PROVE OF EFFICACY AND APPLICATION

We proved that AI-IDS could detect unknown attacks, such as obfuscated or encoded malicious payloads; it can write improved existing Snort rules and new rules for newly identified patterns.

B. CONDITIONS AND ASSUMPTIONS

This study uses the following conditions and assumptions:

1) AI-IDS: AN OPEN SOURCE SOFTWARE

AI-IDS software contains the following license and notice below: Licensed under the *MIT License*. You can access the source-code directly on Github in our repositories [5].

2) PARALLEL OPERATIONS: IDS, TAS

An IDS and a Traffic Analysis System (TAS) operate independently and do not affect each other. We used a signature-based NIDS for intrusion detection and a Splunk StreamApp-based TAS for collecting real-time traffic. A TAS is equal to a packet monitoring system.

3) APPLICATION-LEVEL PACKETS INSPECTION

We focused on the HTTP commonly used in web services that request headers and payload data. We did not address low-risk attacks from protocols below the application layer, such as user datagram protocol (UDP).

The remainder of this paper is organized as follows. Section II presents related works, limitations of meta-datasets, and the motivation for this study. Section III introduces the security operations for deep learning. Section IV shows our spatial feature learning algorithms for big-data, optimal CNN-LSTM model, and AI-IDS infrastructure. Section V shows the experimental results. Section VI introduces the efficacy and applications. The last Section VII shows the conclusion.

II. BACKGROUND

This section describes related studies on deep learning-based IDS, and the limitations of meta-datasets and the motivation for this study are also described.

A. RELATED WORKS

Recent studies on intrusion detection using various deep learning (DL) techniques have been published since 2017. In Table I, related studies focusing on intrusion detection using DL algorithms based on models, features, datasets, and performance measures are given. Liu et al. [6] showed that when compared with other IDS classifiers, intrusion detection models based on a convolutional neural network (CNN) have the highest detection rate and precision. The feasibility of applying a CNN in highly intruded detection has been proven. The authors argue that the performance of CNN-based techniques is better than that of other machine learning classification techniques. Wang et al. [7] designed an IDS using a CNN to automatically train and look for traffic characteristics, effectively reducing the false alarm rate (FAR). This study shows that deep learning techniques can be used to

TABLE I
RELATED WORKS ON INTRUSION DETECTION

Year	Authors	Focus Area	Deep Model	Features	Dataset	Performance
2017	Liu at al. [6]	Attack classification	CNN	41 features	KDD Cup 1999	DR=97.66%
2017	Wang et al. [7]	Feature learning for intrusion detection	CNN	16 Features based on netflow	DARPA 1998, ISCX2012	DR=99.89%, FAR=0.02%
2017	Yin et al. [8]	Binary and multi-class intrusion detection	RNN	41 features mapped into 122	NSL-KDD	ACC=83.28%
2018	Shone et al. [9]	Attack classification using unsupervised feature learning	Non-Symmetric Deep Auto-Encoders	41 features	KDD Cup 1999, NSL-KDD	ACC=97.85%
2018	Wu et al. [10]	Attack classification	CNN, RNN	41 features	NSL-KDD	ACC=79.48%
2018	Naseer et al. [11]	DL approaches for anomaly-based IDS	DNN, CNN, RNN	41 features	NSL-KDD	DR=85-89%
2018	Ding et al. [12]	multi-class intrusion detection	CNN	41 features	NSL-KDD	ACC=80.13%
2019	Otoum et al. [13]	Intrusion detection on wireless sensor networks (WSNs)	Restricted Boltzmann Machine (RBM)	41 features	KDD Cup 1999	DR=99.91%
2019	Chouhan et al. [14]	Intrusion detection based on unsupervised training	CNN	41 features	NSL-KDD	ACC=89.41%
2019	Vinayakumar et al. [15]	Intrusion detection on ML/DL approaches	DNN	41/41/49/23 features	KDD Cup 1999 NSL-KDD, UNSW-NB15, WSN-DS,	KDD Cup 1999, NSL-KDD: ACC=95-99%, UNSW-NB15, WSN-DS: ACC=65-75%
2019	Chiba et al. [16]	Intrusion detection on cloud environment	DNN	80/41/14 features	CICIDS2017, NSL-KDD version 2015, CIDDS-001	ACC=99-99.9%
2019	Zhang et al. [17]	Identify SQL injection attacks in network traffic	Deep Belief Network (DBN)	6 features	Private datasets	ACC=92-96%
2019	Faker et al. [18]	Attack classification on ML/DL approaches	DNN, Random Forest (RF), Gradient Boosting Tree (GBT)	80/49 features	CICIS2017, UNSW NB15	ACC=91-98%
2019	Aloqaily et al. [19]	Intrusion detection for connected vehicles	Deep Belief Network (DBN), Decision Tree (DT)	41 features mapped into 122	NSL-KDD, NS-3 Featured Traffic	ACC=99.43% DR=99.92%
2020	Kim et al. (This study)	Intrusion detection for real-time web-service	CNN-LSTM	256 features (all extended ASCII characters)	Real-time web traffic data, CSIC-2010, CICIS2017	ACC=98.07% DR=99.22%

extract and learn the characteristics of network traffic in detail. Yin et al. [8] proposed an RNN-IDS and compared it with ANN, random forest (RF), SVM, and other machine learning methods. An RNN-IDS is suitable for modeling a classification model with high accuracy, and its performance is superior to that of traditional machine learning classification methods in both binary and multiclass classification.

Shone et al. [9] showed a non-symmetric deep auto-encoder (NDAE) for unsupervised feature learning. This study improves the classification performance of KDD99 and NSL-KDD99 by comparing an auto-encoder with a non-symmetric deep auto-encoder (NDAE). Wu et al. [10] devised CNN and RNN for attack detection; however, their model differs from the model used in this study because it performed separate experiments on the CNN and RNN model. Naseer et al. [11] investigated the suitability of deep learning approaches for anomaly-based intrusion detection systems. Ding et al. [12] compared the performance of models using multi-class classification with the performance of traditional machine learning methods.

Otoum et al. [13] devised DL for an IDS available on wireless sensor networks (WSNs), and also compared the Boltzmann machine-based clustered IDS (RBC-IDS) and adaptive machine learning-based IDS: the adaptively supervised and clustered hybrid IDS (ASCH-IDS). Chouhan et al. [14] proposed a Channel Boosted and Residual learning-based deep Convolutional Neural Network (CBR-CNN) architecture for the detection of network intrusions. This study used Stacked Auto-encoders (SAE) and unsupervised training, and the performance of the proposed CBR-CNN technique is evaluated with an NSL-KDD dataset. Vinayakumar et al. [15] developed an IDS to detect and classify unforeseen and unpredictable cyberattacks by DNN. The performance was tested with the DNN model and compared to the results of the NSL-KDD, UNSW-NB15, Kyoto, WSN-DS, and CICIDS2017 datasets. Chiba et al. [16] proposed a DNN model in a cloud environment based anomaly network IDS using recent datasets, such as CICIDS2017, NSL-KDD version 2015, and CIDDS-001, using a hybrid optimization framework (IGASAA) based on the Improved Genetic

Algorithm (IGA) and Simulated Annealing Algorithm (SAA). Zhang et al. [17], used a deep belief network (DBN) model to identify SQL injection attacks in network traffic. Faker et al. [18] experimented with improving the performance of intrusion detection systems on CICIS2017 and UNSW NB15 datasets using a DNN and two ensemble techniques, RF and gradient boosting tree (GBT). Previous research has suggested new ideas or algorithms for improving deep learning algorithms. Aloqaily et al. [19] introduced an automated secure continuous cloud service availability framework for smart connected vehicles that enables an intrusion detection mechanism against security attacks.

However, most previous deep learning-based studies have difficulty applying intrusion detection in real-world environments because the models were usually pre-processed into metadata formats in an experimental environment. Few studies have proven how to apply them in real-time in the real world.

B. LIMITATIONS OF META-DATASETS

Previous studies [8, 12, 15, 16] mainly focused on extracting or analyzing features from metadata rather than paying attention to exploited raw packets. Owing to network traffic changing with trends, the accuracy rates of real-world without continuous re-training is significantly reduced even if a system is 99.9% accurate in an experimental environment. The following is a description of the KDD99, NSL-KDD, and PU-IDS datasets that have been widely used in previous works.

1) KDD CUP 1999 DATASETS

KDD Cup 1999 Dataset [20] is the most widespread dataset for intrusion detection based on the DARPA dataset. The dataset contains TCP high-level attributes, such as the connection window, but no IP addresses. KDD99 involves more than 20 different types of attacks and comes along with redundant records in the test-set [21].

2) NSL-KDD DATASETS

NSL-KDD [22] NSL-KDD is a dataset that has been enhanced from KDD99, removing much of the duplicated data from KDD99 and creating a more advanced sub-dataset. The dataset consists of separate and predefined training data and test data for intrusion detection. NSL-KDD uses the same attributes as KDD 99 and belongs to the four attack categories: R2L, Prob, U2R, and DOS. belongs to the other category [23].

3) PU-IDS DATASETS

PU-IDS [24] is a derivative of the NSL-KDD data set, and its author has developed a generator that extracts the statistics of the input data set and then creates a new data set. A traffic generator has the same attributes and format as the NSL-KDD data set.

While previous studies mainly used KDD99 or KDD, and NSL-KDD, they are not suitable as datasets for real-time detection. These datasets deal with metadata and therefore make it difficult to identify invalid attacks in a practical environment, because metadata are not attack-attempts. Moreover, most public datasets contain redundant information and an unbalanced number of categories. For instance, Ring et al. [21] compared the characteristics of intrusion detection datasets used in previous works. This study shows that various previously published datasets model repetitive and inefficient attacks, such as DOS, UDP Flooding, and brute force, which are different to recent web-attacks trends. In fact, types of attacks and trends in the data are constantly changing; therefore, it is necessary to develop a general-purpose model that is not biased toward current trends.

Another problem with most published datasets is that they are often over-fitted due to duplicated or flow-based metadata, and the performance of the model is significantly upgraded in experimental conditions. If the model is applied in practical services, it will face a serious problem with false-positive alarms. Likewise, the work in Sabhnani et al. [25] has shown that when using the KDD99 dataset, it is not possible to successfully train pattern classifications or machine learning algorithms for misuse detection.

Nevertheless, most previous studies measured the model performance on deep learning or machine learning techniques in experiments using KDD99 datasets. Yin et al. [8] also used the KDDTest+- dataset to compare performance with the RNN model, and a recent work in Vinayakumar et al. [15] experimented using the DNN model through public data such as KDD99, NSL-KDD, and UNSW-NB15 using machine learning techniques such as LR, NB, RF, and DT. Gu et al. [26] demonstrated that validated training data is an essential determinant for successful research that can greatly enhance the detection capability. Moreover, Moustafa et al. [27] compared the characteristics of various public datasets and suggested that datasets that are not based on reality can lead to misguided research.

C. MOTIVATION

One of the challenges faced by security operations is an inefficient operation due to false-positive alarm events. It wastes IDS resources and reduces the performance for effective deep learning; therefore, the issue of false-positives should be addressed properly to detect threats in big-data infrastructure. Misuse detection that broadly applied in SOC uses predefined signatures for filtering and to detect attacks. It relies on human inputs by constantly updating the signature database. This method is accurate in finding known attacks but is completely ineffective for unknown attacks. In most cases in real-world environments, misuse detection generates a high false-positive rate similar to anomaly-based IDS. In the study of Mishra et al. [28], performance optimization was needed during the detection process to deal with false-positive issues. However, most previous works do not adequately address the false-positives issue in the real-world due to performance evaluations with limited datasets in experimental environments. To mitigate the false-positive problem, high-quality training data is a basic determinant for improving DL model performance.

The most common issues with existing solutions based on learning models include

--First, the learning models produce a high false-positive rate with a wide range of attacks.

--Second, the learning models are not adaptive to the real-world, as meta-datasets like KDD Cup 99 were mainly used to evaluate the performance of the learning model.

--Third, previous studies were unable to foresee today's huge network traffic; therefore, scalable solutions are required to maintain a high performance with a rapidly increasing high-speed network size.

--Finally, no cases have been published on DL applications for the detection of unknown attacks on real-world computer networks. These challenges form the primary motivation for the application of deep learning-based NIDS.

III. SECURITY OPERATIONS FOR DEEP LEARNING

This section introduces the security operations for deep learning applications and the data design for practical training.

A. OVERVIEW OF SECURITY OPERATIONS

We detected and analyzed intrusion attempts into financial networks to protect electronic financial incidents. The SOC also plays the role of an Information Sharing and Analysis Center (KF-ISAC). Fig. 1 shows that the SOC collected real-time network traffic, and detected malicious network traffic by directly installing an NIDS, a TAS, a TAP, and a virtual private network (VPN) on the Internet DMZ area of many financial companies in South Korea. The SOC operated continuously for 24 hours a day, 7 days a week, and 365 days a year. About 20 people work in shifts and generate daily analytical information for training. The IDS and TAS data were transferred to the SOC via VPN from financial institutions, and the SOC collected approximately 1 billion real-time HTTP traffic per day (Sep. 2019).

An NIDS is a signature-based misuse detection system based on Snort rules, and a TAS is a system that collects network traffic in a user-defined function. A TAS is a type of system that collects network traffic and enables users to analyze traffic by collecting various protocols, including HTTP, SMTP, and SSH. It allows analysts to analyze anomalies by collecting various network protocols from the network layer to the application layer. We use the StreamApp [29] as a Splunk software for traffic collection and an analysis system. For the effective detection and analysis of cyber-attacks, we recommend running NIDS and TAS in parallel. If security events are alerted on an NIDS, a TAS could inspect the same malicious payloads on the network.

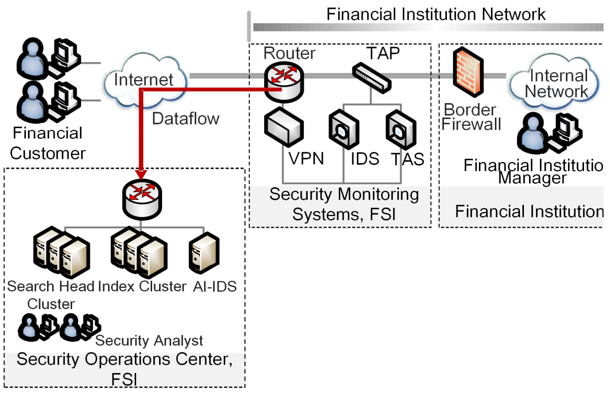


FIGURE 1. AI-IDS applied Security Operations Center (SOC)

TABLE II
ATTRIBUTES OF ANALYSIS INFORMATION

Feature	Description	Data Type
Detect. Time	Security Event Alert Time	Meta-data (non-training data)
Detect. Site	IDS, TAS Sensor Name	
Direction	Dataflow Directivity	
Src IP	Source IP Address	
Src Port	Source Port Number	
Dest IP	Destination IP Address	
Dest Port	Destination Port Number	
Signature Name	Security Event Name	
Raw Packet	Captured Strings Content	Payload data (training data)
Analysis Result	True (malicious) / False (benign)	Label data (training data)

An NIDS and a TAS inspected a variety of protocols, such as SSDP, DNS, SMTP, POP3, HTTP, and SSL, from the network layer to the application layer on the network. As the UDP-based protocol does not establish 3-ways handshaking, it is difficult to attribute it to an IP address and can easily be forged. Thus, we did not analyze invalid UDP-type or denial of service (DOS) attacks to maintain stable performance. SSL protocol was excluded from our scope because it is not possible for an analyst to review the malicious payload.

In managed security monitoring operations, security managers process security events in the order of Detection, Analysis, and Prevention. “Detection” means to collect security alerts generated by user-defined Snort rules on NIDS or TAS, which include detection time, source IP, destination IP, port information and signature messages in Table II. “Analysis” refers to classifying events into true or false positives by reviewing detection information. “Prevention” is to register malicious IP addresses to blacklists, which are then blocked from accessing service websites. Prevention is applied to very obvious attack patterns, and it is recommended to block access from certain attacks only after being verified by an analyst or system. The proposed AI-IDS is used as a supplement system with legacy signature-based NIDSs for network layer security.

B. DATA DESIGN FOR PRACTICAL TRAINING

The proposed AI-IDS trains the labeled analysis information based on HTTP data in-bounding from the managed services instead of metadata sets in a constrained environment. We detect about 200,000 attacks on about 1 billion HTTP per day on legacy signature-based NIDS, and we perform about 10,000 automatic and manual analyses. During general security operations, malicious detection information is triggered by NIDS when an attack packet occurs in the network communication. Daily training-data on the production environment is labeled in real-time by security analysts using labeling tools.

The analysis information labeled is shared with AI-IDS and used as training data for prediction in neural networks. We implemented deep learning models in real-time HTTP traffic – "Password guessing and Authentication bypass (AUT)," "SQL Injection (SQL)," and "Application vulnerability attack (APP)." For UDP-type attacks, such as "information gathering" or "denial of service," it is difficult to identify the attacker's IP address when compared to TCP, because the session is not connected perfectly, and contains meaningless repeated data; therefore, we excluded it from the deep learning model. Besides, HTTP traffic related to malware infection events are often detected when the malware connects to the C2 server after infection. Unlike general intrusion events of which traffic are sent from an external IP to an internal IP, malware events' traffic is usually in the opposite direction.

The security event shown in Table II consists of the detection time, detection site, direction, source IP, source port, destination IP, destination port, signature name, raw packet (pcap file), and flag. "Detection Time" is the time the signature generated the event, and "Detection Site" is the location identifier where IDS and TAS were installed. "Direction" shows the direction of attacks based on assets between the source IP and the destination IP. "Source IP/port (src_ip, src_port)" is the IP/Port address that requests a connection from the client to the server, and "Destination IP/port (dest_ip, dest_port)" is the IP/Port address from the server to the client. Most of the above metadata are managed as Critical IP or Threat Intelligence by security administrators.

The number of HTTP requests collected per day was approximately 1 billion, of which about ten thousand were analyzed information about attack events detected in HTTP. Assuming a normal to abnormal ratio of 5:5, the amount of malicious analysis information is 65 MB for the last year, but benign HTTP traffic is 6 GB per day. To equalize the data rate for training in the deep learning model, the 65 MB HTTP payload, which was analyzed during one year, was multiplied 100 times by concatenation and shuffle, and the ratio of the analysis information and normal traffic was adjusted to be equal to 6 GB per day. Malicious events identified by analysts were used as data for re-training. The training data was approximately 6 GB per day, and the analysis information from the duration of 1 year was changed sequentially like a sliding window.

IV. DESIGN AND IMPLEMENTATION

This section introduces a fast and effective spatial feature learning based on normalized UTF-8 character encoding, detailed AI-IDS architecture, and the structure of a neuron network model for large-scale web traffic.

A. SPATIAL FEATURE LEARNING BASED ON NORMALIZED UTF-8 CHARACTER ENCODING

Feature extraction is one of the most important tasks in designing an efficient learning algorithm. Mamun et al. [30] devised a combined preprocessing technique using attributes of information theory such as entropy, encoding, and compression. Theoretically, the entropy of encrypted or irregular data is high as there are many uncertainties in the data stream. The entropy value indicates the degree of uncertainty of the information, but it is difficult to extract the feature by matching the unique characters of the given data 1:1. For example, entropy can express the uncertainty of information as a number in the range of 0 to 1, but a collision problem would be calculated with the same entropy even if different data were given. For this reason, it is difficult to extract unique features of a given string, as it is. Thus, we use UTF-8 encoding that normalizes the deep learning model to recognize the data with its own characteristics. It is simple and fast, because it does not include unnecessary entropy calculations, compression, encryption, or anything else. Assuming that all data preprocessing for billions of HTTP within 1,000 bytes per day is executed, the UTF-8 encoding method can achieve fast data preprocessing at only about $1 \times 2^8 \times 1{,}000$ billion. The biggest advantage of UTF-8 is that it cannot be confused with a single encoding method, so there is no possibility of wrong encoding in other ways, such as for the national language encoding method such as UTF-16, EUC-KR (Korean), GB2312 (Simplified Chinese). As both browsers and web servers are now developed assuming UTF-8, it is a very efficient way to preprocess HTTP traffic.

UTF-8 encoding in Algorithms 1-2 converts up to 256 characters into floats, which can be encoded into numbers, including special characters that include Simplified Chinese in the packet, such as WebDAV attacks. When preprocessing a string of 7 bits or less, it is difficult to preprocess various characters in a real environment. We used the normalized UTF-8 encoding and the module developed on "parse" and shown in Figs. 2 and 3. The input variable was replaced with a value corresponding to a unique string in the range of 0 to 255 (256 features), and the input string was converted into a float value between -1.000 and 1.000 given that $y_s = -(y_s - 128)/128$. The output variables y_s for a transformed set of input data, for one training-data size $s \in [0, 2, 3, \dots, 999]$.

Example of preprocessing: http://target.com/ manager/html/

Normalized UTF-8 Encoding

original characters:	m	a	n	a	g	e	r	/	h	t	m	l	/
decimal values:	109	97	110	97	103	101	114	47	104	116	109	108	47
normalized values:	0.148	0.242	0.141	0.242	0.195	0.211	0.109	0.633	0.188	0.094	0.148	0.156	0.633

Real-time Preprocessing for a fast way
$Y(s_1)$ = - (109-128) / 128 = 0.148, $Y(s_2)$ = - (97-128) / 128 = 0.242, $Y(s_2)$ = - (110-128) / 128 = 0.141, …
$Y(s_n)$ = - (Ys-128) / 128

Entropy-based Encoding

original characters:	m	a	n	a	g	e	r	/	h	t	m	l	/
Probablitiy	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077

2-Phase Entropy calculation
Phase 1. Probability calculation
0.154 -> "/", 0.154 -> "a", 0.077 -> "e", 0.077 -> "g", 0.077 -> "h", 0.077 -> "l", 0.154 -> "m", 0.077 -> "n", 0.077 -> "r", 0.077 -> "t"

Phase 2. Multiplication and Logarithm
$H(X_1) = -(0.154\log_2 0.154) = -0.415$
$H(X) = -[(0.154\log_2 0.154)+(0.154\log_2 0.154)+(0.077\log_2 0.077)+ \ldots +(0.077\log_2 0.077)] = 3.2389$

FIGURE 2. Comparison of Normalized UTF-8 encoding and Entropy-based encoding

Algorithm 1 UTF-8 Character Encoding

```
Input: content_string (web traffic)
Output: *.npy (preprocessed file)
1: FUNCTION save_data(content_string_list, npy_filename)
2: numpy_array <- numpy.empty()
3: FOR content_string in content_string_list
4:    byte_array <- []
5:    FOR character in content_string
6:        byte_array.append(character.encode('utf-8')
7:    ENDFOR
8:    int8_array <- []
9:    FOR byte in byte_array
10:        int8_array.append(byte.toint8())
11:   ENDFOR
    //
    //float_array <- []
    //FOR int_8 in int8_array
    //    float_array.append(int_8 - 128.0 / -128.0)
    //ENDFOR
    //
    //content_array <- numpy.array(float_array)
    //numpy.append(content_array)
    //
    // for data size issues,
    // the actual array is saved from int8_array
    // and the float is calculated just before training
12:   content_array <- numpy.array(int8_array)
13:   numpy.append(content_array)
14: ENDFOR
15: numpy.save(npy_filename, numpy_array)
16: ENDFUNCTION
```

Algorithm 2 Spatial Feature Learning (train/test)

```
Input: *.h5(model), *.npy (preprocessed file)
Output: performance metrics
1: FUNCTION train_model(model, train_file_list, x_dim):
2:    npy_list <- list(load(filename) for filename in train_file_list
3:    data <- concatenate(npy_list)

4:    train_size <- data.shape[0]
5:    x_train <- array(data[:, :-1])
6:    x_train <- (x_train - 128.0) / -128.0
7:    x_train <- x_train.reshape(train_size, x_dim, 1)
8:    y_train <- data[:, [-1]].reshape(train_size, 1)
9:    y_prediction <- model.predict(x_train, batch_size=4096)
10:   y_merged <- (y_prediction.round()*2 + y_train).flatten()
11:   value, counts <- unique(y_merged, return_counts=True)
12:   value_str <- list(map(lambda x: str(int(x)), value))
13:   metrics <- dict(zip(value_str, counts))


14:   loss <- binary_crossentropy(y_train, y_prediction))
15:   metrics['Loss'] <- average(loss)
16:   RETURN metrics
17: ENDFUNCTION
```

Fig. 2 shows a preprocessing example for "http://target.com/manager/html/". When comparing preprocessing methods with our proposed UTF-8 encoding and entropy-based encoding, our proposed method is a normalized calculation expression. The entropy of a string requires probability calculation, followed by multiplication and logarithm. Entropy-based preprocessing involves two steps of calculating the probability of each string and then calculating the log. Instead, our proposed method can preprocess strings with a single operation and have no data transformation or substitution in the progress. Normalized UTF-8 encoding generates input values so that the deep learning model can train immediately.

Previous [31] research designed a CNN that can be trained as a corpus to process natural language between sentences and words. However, it functions closer to image processing than natural language processing because cybersecurity corpora have a different attribution compared with natural language. A corpus in the field of cybersecurity is difficult to create because it needs to understand string classifications and

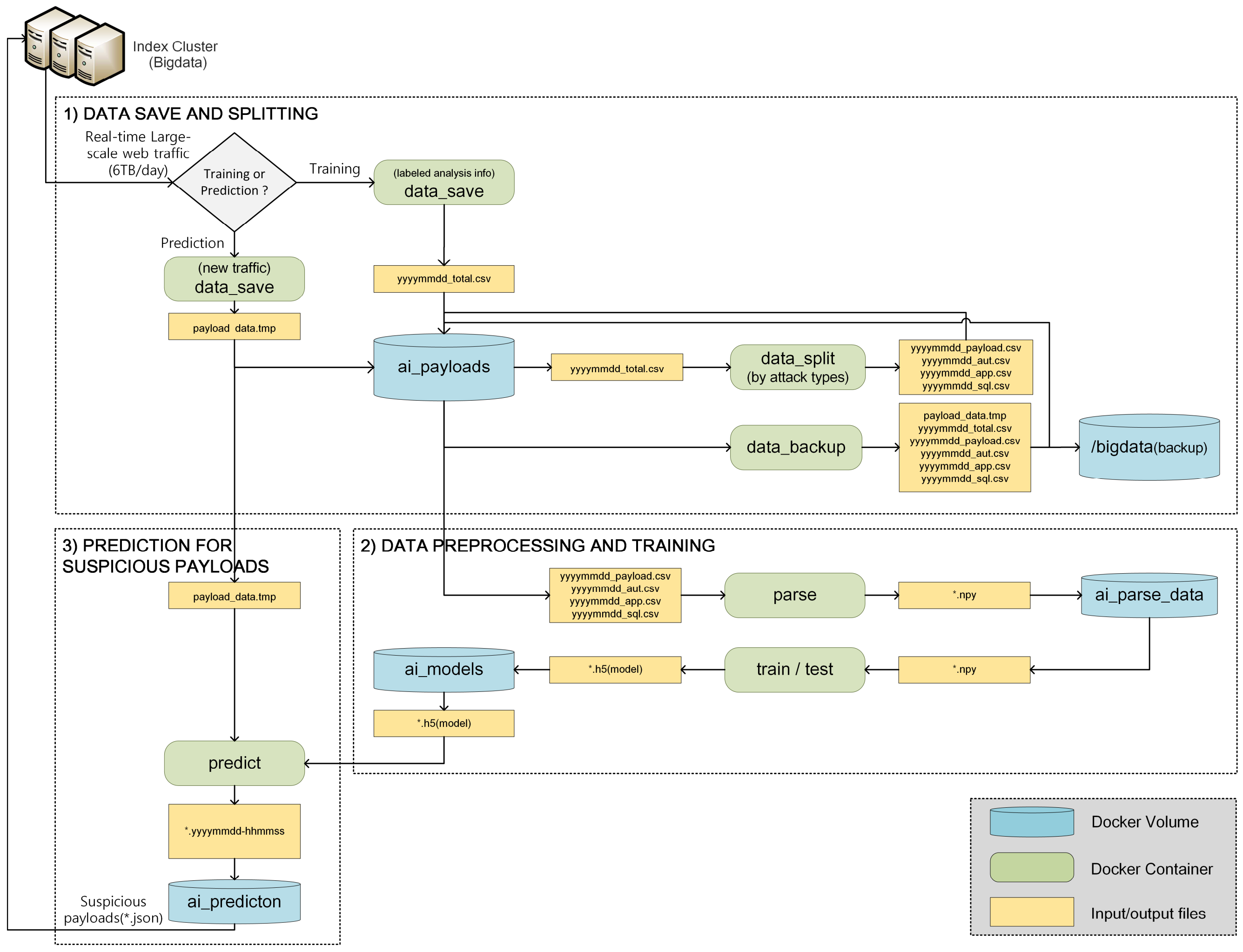

Index Cluster (Bigdata)
1) DATA SAVE AND SPLITTING
Real-time Large-scale web traffic (6TB/day)
Training or Prediction ?
Training
Prediction
(labeled analysis info) data_save
(new traffic) data_save
yyyymmdd_total.csv
payload_data.tmp
ai_payloads
yyyymmdd_total.csv
data_split (by attack types)
yyyymmdd_payload.csv yyyymmdd_aut.csv yyyymmdd_app.csv yyyymmdd_sql.csv
data_backup
payload_data.tmp yyyymmdd_total.csv yyyymmdd_payload.csv yyyymmdd_aut.csv yyyymmdd_app.csv yyyymmdd_sql.csv
/bigdata(backup)
3) PREDICTION FOR SUSPICIOUS PAYLOADS
2) DATA PREPROCESSING AND TRAINING
payload_data.tmp
yyyymmdd_payload.csv yyyymmdd_aut.csv yyyymmdd_app.csv yyyymmdd_sql.csv
parse
*.npy
ai_parse_data
ai_models
*.h5(model)
train / test
*.npy
*.h5(model)
predict
*.yyyymmdd-hhmmss
ai_predicton
Suspicious payloads(*.json)
Docker Volume
Docker Container
Input/output files

FIGURE 3. **AI-IDS Architecture**

attributes, for example, "get, post, head, put, php, cgi, admin, wget, 'POST /manager/html', 'User-Agent: Mozilla/5.0'". In our initial model, we were trying to train the security corpus into the CNN filter and LSTM layers. However, as there is currently not enough research on cybersecurity corpora, we have implemented deep learning on all strings of the HTTP data. If a cybersecurity's corpus was created, deep learning model performance is expected to be improved.

B. AI-IDS ARCHITECTURE

Fig. 3 shows an enlarged representation of the AI-IDS and Index Cluster, as shown in Fig. 1. Individual modules are configured as Docker image/containers that output files after the Docker process. No Docker container affects another and they all run independently. However, Docker volumes are shared as same data in a series of processes, from pre-processing (parsing), training and testing, to prediction. Our AI-IDS process is as follows: (i) data save and splitting - collecting web traffic and splitting training data for each model (ii) data preprocessing and training by labeled analysis information (iii) prediction for suspicious payloads on new web-traffic.

The following is a detailed process description of the AI-IDS, as shown in Fig. 3:

1) DATA SAVE AND SPLITTING

Index Cluster collects true or false positive analysis information and normal traffic for training data and then stores it in "(labeled analysis info) data_save" (name of docker image). "data_save" saves legacy NIDS payload data along with its analysis results, and also previously labeled data by AI-IDS. Simultaneously, "(new traffic) data_save" stores real-time HTTP traffic for prediction in "ai_payloads" for the previous 3h to -10 min. "data_split" saves data in "ai_payloads" by splitting the data according to the intrusion attack types (AUT, SQL, APP) to generate training data for each deep learning model. Each process module has one or more input and output files. The real-time web traffic is transferred into application-level strings for AI-IDS, and the "data_backup" module backs up raw-data which has been collected more than 24 hours in the past.

2) DATA PREPROCESSING AND TRAINING

Output files for parsing is shared into the "ai_payloads" volume. Each spitted raw-data (in the form of .csv files) is classified by its attack type. Then it is preprocessed by "parse", and the preprocessed data (npy files) is saved into a docker volume named "ai_parse_data". The raw web-traffic strings are transformed into trainable float data for deep learning through our UTF-8 encoding with zero-padding. As a result of searching the optimum performance using the "train/test" module, the h5 filetypes in "ai_model" stores the model's best hyper-parameters and states achieved by deep learning, for classifying malicious and benign traffic.

3) PREDICTION FOR SUSPICIOUS PAYLOADS

To predict suspicious payloads, "predict" inspects the real-time data (payload_data.tmp) stored in "ai_payloads" using the h5 model trained by the "train/test" module. "predict" also stores output as JSON files, including metadata, suspicious payloads, prediction for the malicious-ness probability. The predicted data in "ai_prediction" volume are potential suspicious events identified by each model and are stored periodically (saved 8 times a day) until they are finally reviewed or labeled by a security analyst. The output files are accumulated into labeling tools in fig. 7.

Table III shows the contents of a sample JSON file generated by "predict" and stored in "ai_predction". The file type is stored in the data frame in the following order: "_time", "payload_id", "model_name", "prediction", "src_ip", "src_port", "dest_ip", "dest_port" and "src_content", and "src_content" means payload that "POST /manager/html". "payload_id" is the prediction event id, whose value can be identified and is generated by "hexdigest (sha1(_time@src_ip: src_port->dest_ip: dest_port))".

The infrastructure of the deployed center system consists of Splunk Architecture and our AI-IDS. Splunk Architecture consists of a Search Head Cluster with multiple search headers and an Index Cluster with dozens of Indexes. One of the search heads was built as an independent and dedicated system to interface with AI-IDS. The AI-IDS was developed in the Docker 18.09.5, Python 3.6.7, Tensorflow 1.13.1, Keras 2.2.4 and Splunk SPL 7.2.3. The test-bed system is HP DL380G9: 2.1 GHz 2P/8C(16C) CPU, 416 GB RAM, Tesla P100 16 GB×1EA GPU, 960G×2(RAID-1) SSD, 6 TB×2(RAID-1) HDD and 10Gbps LAN. The operating server consisted of an HP DL390G10: 2.4 GHz 2P/20C(40C) CPU, 1 TB RAM, Tesla V100 32 GB RAM×2EA GPU, 960G×4(RAID-5) SSD, 10 TB×4(RAID-5) HDD and 10 Gbps LAN.

As shown in Fig. 1, the sensor systems were located in several financial institutions, and IDS alert events and TAS traffic were collected and transmitted to the SOC via VPN from financial institutions. The experiment was conducted in a test-bed system, which was deployed to the operating system only when the performance and function verification were completed.

TABLE III
SAMPLES OF PREDICTION OUTPUT

Contents of JSON file
["_time":"2019-10-17T11:12:35.015+09:00", "payload_id":"9f701440043030c6471e7d9d73219d0c", "model_name":"inv-aut-prepad-model", "prediction":0.9550831914,"src_ip":"1.1.*.93","src_port":61750, "dest_ip":"203.234.*.27","dest_port":80, "src_content":"POST /manager/html"]
["_time":"2019-10-17T11:12:35.015+09:00", "payload_id":"9f701440043030c6471e7d9d73219d0c", "model_name":"inv-aut-prepad-model", "prediction":0.9550831914, "src_ip":"1.1.*.93","src_port":61750,"dest_ip":"203.234.*.27", "dest_port":80, "src_content":"POST /%20and%20select%..."]

TABLE IV
SUMMARY OF PROPOSED CNN-LSTM MODEL

Model	Layer	Output Shape*	# of params
CNN	Conv1d_1	(None, 1000,12)	60
	Max_pooling1d_1	(None, 200, 12)	0
	Batch_normalization_1	(None, 200, 12)	48
	Conv1d_2	(None, 200, 60)	2940
	Max_pooling1d_2	(None, 40, 60)	0
	Batch_normalization_2	(None, 40, 60)	240
RNN	LSTM_1	(None, 40, 16)	4928
	Bidirectional_1	(None, 32)	4224
DNN	Dense_1	(None, 12)	396
	Dropout_1	(None, 12)	0
	Leaky_relu1	(None, 12)	0
	Batch_normalization_3	(None, 12)	48
	Dense_2	(None, 12)	156
	Dropout_2	(None, 12)	0
	Leaky_relu2	(None, 12)	0
	Batch_normalization_4	(None, 12)	48
	…	…	…
	Dense_8	(None, 12)	156
	Dropout_8	(None, 12)	0
	Leaky_relu8	(None, 12)	0
	Batch_normalization_10	(None, 12)	48
	Dense_9	(None, 1)	13

Total params: 14,325
Trainable params: 13,989
Non-trainable params: 336
* (None, X, Y): None means this dimension is variable.

C. OPTIMIZED CNN-LSTM MODEL

Table IV and Fig. 4 show the CNN-LSTM structure, which illustrates the hyper-parameters. One UTF-8 encoded HTTP data, including the variable-length HTTP header and payloads, which is the initial input value of the proposed neuron network model, is made into a fixed-length input

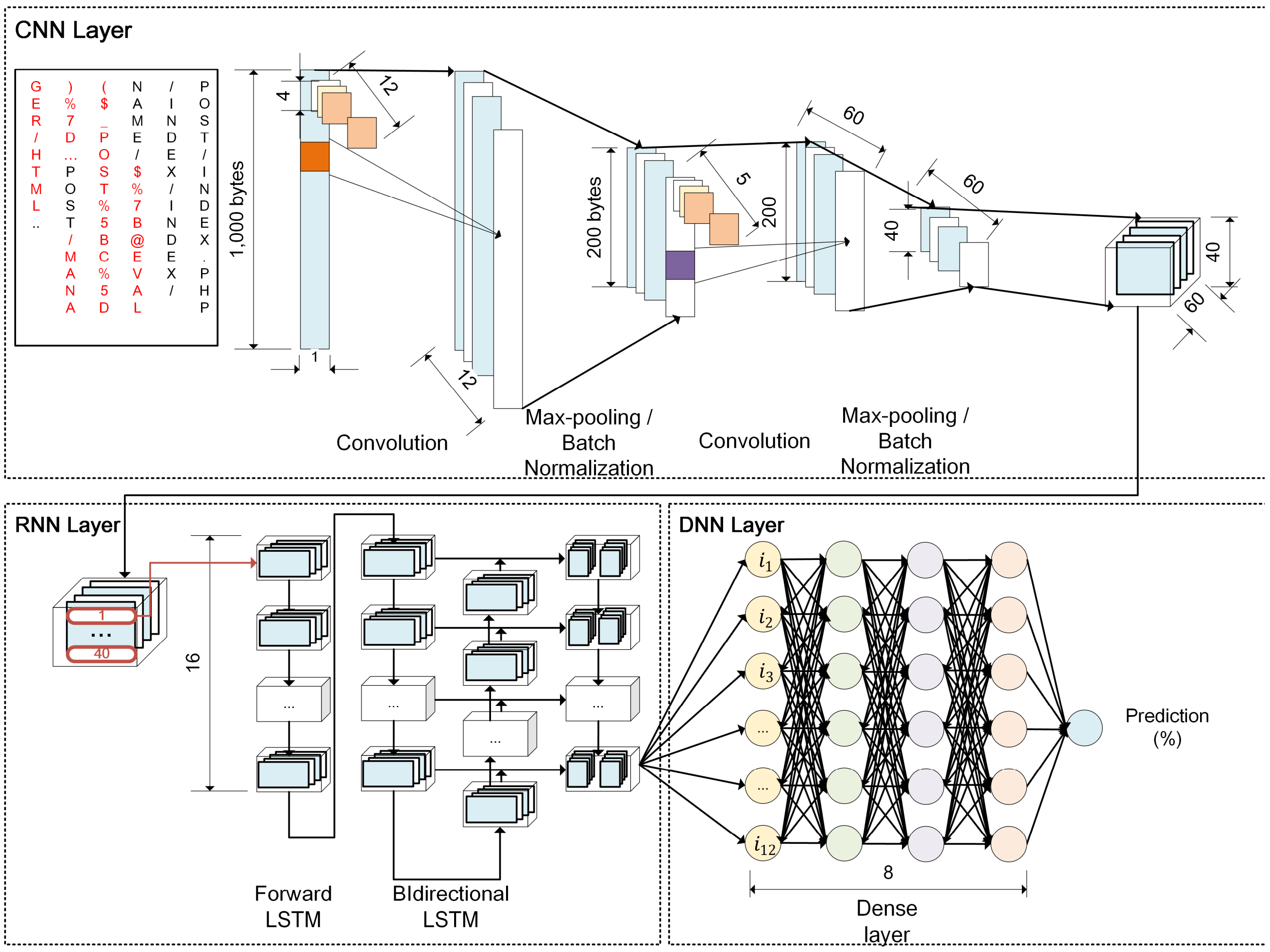


FIGURE 4. **Structure of Optimized Convolutional Recurrent Neural Networks**

value of 1,000 bytes (1 dimension × 1,000 bytes). Strings corresponding to the header and body of the HTTP request from the 0-th byte to the n-th byte are aligned, and the rest of the data is zero-padded. AI-IDS preprocessing continuously collects data for 3 hours in 1 cycle. AI-IDS operates 8 times of learning, and predicts every 3 hours for real-time traffic, which allows for real-time monitoring for 24 hours. In the training phase, the labels indicating “malicious,” “benign,” and “unknown” are recorded at the end of 1,000 bytes of an HTTP request, and the model calculates a malicious probability when all neuronal network operations are completed. The initial input-data at the CNN layer generates 1×1,000×12 composited data through an operation with 1×4×12 filters. After 1/5 max-pooling, 1×200×12 pieces of data are stored in the memory in normalized form. In the second convolutional layer, 1×200×60 data are generated through the composite product of a 1×4×5 filter, and then 1×40×60 data are generated as a result of 1/5 max-pooling and normalization. Data output from the CNN layer is used as an input to the RNN layer, and data processed into cells of 1×40×60 are sequentially input to Forward LSTM and Bidirectional LSTM. The first LSTM cell is calculated in the forward direction with 16 cells, the second LSTM cell is processed in a bidirectional flow, and the last 32nd LSTM Cells are transferred to the DNN layer by combining the accumulated forward and backward cells.

The output value of the calculated RNN is input into each of the 12 fully connected DNN layers. Until the DNN output layer, dropout was set to 0.1, and the LeakyReLU function was applied. Sigmoid activation function was used at the DNN output layer and the model was trained for prediction on malicious payloads using the Adam optimizer along with binary-crossentropy (BCE) as the loss function. The probability is calculated in the output layer which includes the JSON output-file shown in Table III and the output files are shared with Index Cluster, as shown in Figs. 1 and 3.

The analyst reviews the probability calculated by AI-IDS and examines the payload to determine whether an attack warning is valid or not. During the training phase, AI-IDS uses labeled analysis information from an analyst: (i) attack alert events detected by IDS and (ii) valid attack events that the analyst has confirmed. As the AI-IDS aims to detect new threats in the predict phase, the security events detected by legacy signature-based IDS are considered duplicate data. It calculates malicious probability for new and real-time payloads and outputs prediction results.

The composition and depth of each layer of CNN, RNN, and DNN derives the optimal parameters for the model through repeated experiments in the training phase. The structure and parameters of the neuron network are slightly different when iterative experiments are performed on various datasets to select optimal performance parameters. The proposed model is devised with an intuitive design based on the theoretical basis of a previous study, and we proved the model validity through repeat experiments. In the next section, we present the detailed experimental results to demonstrate the validity and performance of our proposed model.

V. EXPERIMENTS

This section demonstrates performance measurements, experimental design, and results: comparing the performance of the CNN-LSTM, LSTM-CNN, and DNN models and the experimental results of the KF-ISAC, CSIC-2010 HTTP, and CICIDS 2017 datasets.

We have defined the following experimental statements for the deep learning application:

- **Selection of experimental data:** CSIC-2010, CICIDS 2017 HTTP dataset, real-time HTTP data
- **Design of optimal model structure using deep learning:** CNN-LSTM model, LSTM-CNN model
- **Determination of hyper-parameters:** This is required for individual neural networks, such as CNN, RNN, DNN: conv_depth, conv_filter, and lstm_units, dense_units
- **Model validation:** experiments on two public datasets (CSIC-2010, CICIDS2017) and fixed real-time data

We experimented to select the optimal model by comparing CNN-LSTM with LSTM-CNN based on real-time HTTP traffic on a fixed date. In the second experiment, we validated the model through experiments using two public datasets (CSIC-2010, CICIDS 2017 HTTP dataset) and real-time data on the optimal model. Recently, various models have been introduced that optimize performance by combining CNN, RNN (LSTM), and DNN. Liu et al. [32] and Wu et al. [10] devised CNN and RNN for intrusion detection, but it was different from the model of this study because it performed experiments each separated model in CNN and RNN. In this paper, a DNN was selected as the last layer to output a single result; we chose a model that can best characterize the data among a CNN-LSTM or LSTM-CNN.

A. Performance Measurement

We used a confusion matrix to evaluate the performance of the deep learning model. A confusion matrix is a popular indicator of the performance of classification models. The matrix in Table V shows us the number of correctly and incorrectly classified results, compared to the actual outcomes in the test data. One of the advantages of a confusion matrix as an evaluation tool is that it allows for a more detailed analysis. The matrix is n by n, where n is the number of classes. The simplest classifiers, called binary classifiers, have only two classes: positive/negative. The performance of a binary classifier is summarized in a confusion matrix that cross-tabulates predicted and observed examples into four categories [8,33].

TABLE V
A Confusion Matrix

	Predicted as Positive	**Predicted as Negative**
Labeled as Positive	True Positive (TP)	False Negative (FN)
Labeled as Negative	False Positive (FP)	True Negative (TN)

TABLE VI
Rules for Performance Measurement

Rule	**Formula**
Accuracy (ACC)	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall (Sensitivity, Detection Rate)	$\frac{TP}{TP+FN}$
Specificity	$\frac{TN}{TN+FN}$
F-Score	$\frac{2\times(Recall\times Precision)}{(Recall+Precision)}$

In our deep learning model, Precision and F-Score are more important performance indicators than others. Moreover, the purpose of AI-IDS is to obtain a higher accuracy with a lower false-positive rate.

We describe the four indicators that make up the confusion matrix in Table V, as follows:

- **True Positive (TP):** the number of cases correctly predicted and labeled as positive.
- **False Positive (FP):** the number of cases incorrectly predicted and labeled as positive.
- **True Negative (TN):** the number of cases correctly predicted and labeled as negative.
- **False Negative (FN):** the number of cases incorrectly predicted and labeled as positive.

We use the following notation in Table VI for the model evaluation:

TABLE VII
Experimental Datasets

Datasets	Name of Files	Description	Normal flows	Attack flows	Total	Data
Real-time HTTP data (KF-ISAC)	20191027_INV_APP_prepad_0.npy 20191027_INV_SQL_prepad_0.npy 20191027_INV_AUT_prepad_0.npy (train data)	Normal or Web attacks data, Labeled data (Benign, Web attacks),	3,373,190	3,227,965	6,601,155	Labeled HTTP Header, Payload
	20191027_INV-prepad._0.npy (test data)	Real-time HTTP data: Normal or Web attacks per 3h	674,638	645,593	1,320,231	HTTP Header, Payload
CSIC-2010	NormalTrafficTrain.txt	Normal HTTP data 1	36,000	-	36,000	HTTP Header, Payload
	NormalTrafficTest.txt	Normal HTTP data 2	36,000	-	36,000	
	AnomalousTrafficTest.txt	HTTP Web attacks data	-	25,065	25,065	
CICIDS 2017	Thursday-WorkingHours-MorningWebAttacks.pcap, Thursday-WorkingHours-MorningWebAttacks.pcap_ISCX.csv	Normal or Web attacks data Labeled data (Benign, Web attacks), Meta-data	27,129	2,180	29,309*	*extracting only HTTP Labeled Metadata

- **Accuracy (ACC):** the proportion of the number of correctly predicted cases to the labeled total of records.
- **Precision:** the proportion of the number of correctly predicted cases as positive to the number of predicted cases as positive, high precision relates to a low false-positive rate.
- **Recall (Sensitivity, Detection Rate):** the proportion of the number of correctly predicted as positive to the number of cases labeled as positive.
- **Specificity:** the proportion of the number of correctly predicted as negative to the number of cases predicted as negative.
- **F-Score:** the weighted average of Precision and Recall; this score considers both false positives and false negatives.

B. Experimental Design

We describe the details of the experimental datasets in the following paragraphs and in Table VII.

1) REAL-TIME HTTP DATASETS (KF-ISAC)

KF-ISAC HTTP data is real-time HTTP stream data during fixed dates from a TAS. The proposed model trains a mix of benign HTTP data and labeled malicious payloads that have been analyzed over the past year. It evaluates performance by separating training and test data at an 8:2 ratio. The label in the training data is located at the end of the preprocessed data.

2) CSIC-2010 HTTP DATASETS

CSIC-2010 HTTP data [34] was provided by Aberystwyth University. The contributors collected HTTP packets to detect web attacks. The dataset contains 36,000 normal requests and more than 25,000 anomalous requests. The data consists of normal HTTP data for training and normal/abnormal data for testing. We generated 1,941,300 records by augmenting 20 times from the original 77,652 records and split the set in a ratio of train 8: test 2, except for 6 error records during data import.

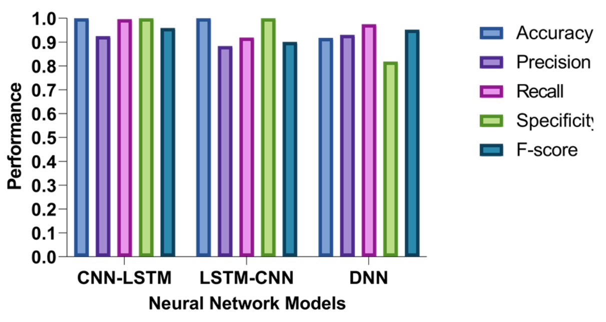
3) CICIDS2017 HTTP DATASETS

The CICIDS2017 datasets [35] generated in 2017 by the Canadian Institute of Cybersecurity overcome these issues. The CICIDS2017 benchmark dataset contains the abstract behavior of 25 users based on HTTP, HTTPS, FTP, SSH, and email protocols. We use only HTTP datasets, including web attacks and generated 586,180 records by augmenting 20 times from the original 29,309 records. The dataset consists of the entire abnormal/normal pcap file, the unlabeled HTTP attack, and the metadata, including label data.

C. Experimental Results

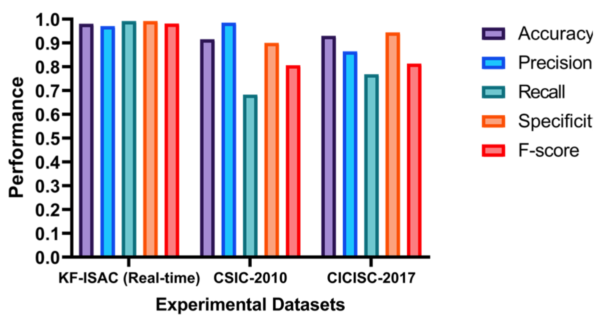
1) MODEL SELECTION

We implemented CNN-LSTM and LSTM-CNN structures for an optimal deep learning model selection and then performed 10 iterations using real-time HTTP data shown in Table VII. The training data of KF-ISAC consisted of approximately 6.6 million records extracted and proposed on a specific date, and each of the normal/attack classes was composed of approximately 3.3 million records. The test datasets were set to a ratio of 8:2. The results of the experiment shown in Fig. 5 are the average values of the results of 50 epochs. The overall model performance of CNN-LSTM is better than LSTM-CNN, in areas such as accuracy, precision, and F-Score. In particular, there are many differences in Precision, and F-Score because of the True/False Positive Rate. The model performance starting from the highest to the lowest is CNN-LSTM, LSTM-CNN, and DNN. CNN-LSTM reduces the dimension by max-pooling at the initial step, but LSTM-CNN takes more time to train because the dimension and parameters are increased through LSTM Cell. The DNN is relatively fast but it has low rates for the scores of Accuracy and Specificity.



Model	Accuracy	Precision	Recall	Specificity	F-score
CNN-LSTM	0.9999	0.9249	0.9961	1.0000	0.9591
LSTM-CNN	0.9997	0.8832	0.9186	0.9999	0.9005
DNN	0.9169	0.9300	0.9748	0.8179	0.9519

FIGURE 5. Performance Comparison of NN Models



Dataset	Accuracy	Precision	Recall	Specificity	F-score
KF-ISAC (Real-time)	0.9807	0.9706	0.9922	0.9917	0.9813
CSIC-2010	0.9154	0.9854	0.6826	0.9002	0.8065
CICISC-2017	0.9300	0.8647	0.7683	0.9440	0.8136

FIGURE 6. Experimental Results on Public Datasets

2) DETERMINATION OF HYPER-PARAMETERS

We chose the best-performing deep learning model according to the experimental results. The CNN-LSTM model needs to determine the optimal hyper-parameters for stable operations. We considered a high precision such that the time needed to train or to validate events by true/false-positive rates in a practical environment is minimized. The experiment used real-time HTTP (KF-ISAC) data shown in Table VII.

The CNN layer determines the conv_depth, conv_filter, conv_kernel_width, and conv_pool variables. In detail, one variable has to be selected from conv_depth $\in [2]$, conv_filter $\in [2, 4, 8, 12]$, conv_kernel_width $\in [4]$ and conv_pool $\in [3, 4, 5]$. The RNN layer determines the lstm_units and lstm_depth variables. In detail, one of the following values has to be selected from lstm_unit$\in [16]$ and lstm_depth $\in [1,2,4,8]$. The DNN layer determines dense_depth, dense_units, dense_dropout, and dense_relu_alpha. In detail, one value of dense_depth $\in [1, 2, 4, 8]$, dense_units$\in [4, 8, 12, 16]$ and dense_dropout $\in [0.1, 0.5]$ is selected. The experiment was conducted 270 times, with one or more of the five indicators converging to zero or one, and then moving on to the next parameters.

Aiming for the high F-score and the high precision, which means minimum with false-positive values, the hyper-parameters of an optimal model are shown as follows: 2 for convolution depth, 12 for convolution filter, 4 for convolution kernel size, 5 for max-pooling size, 16 for LSTM Cell, 2 for LSTM depth (1 forward LSTM, 1 Bidirectional LSTM), 12 dense units, 8 for dense depth, and 0.1 for dense dropout.

3) MODEL VALIDATION

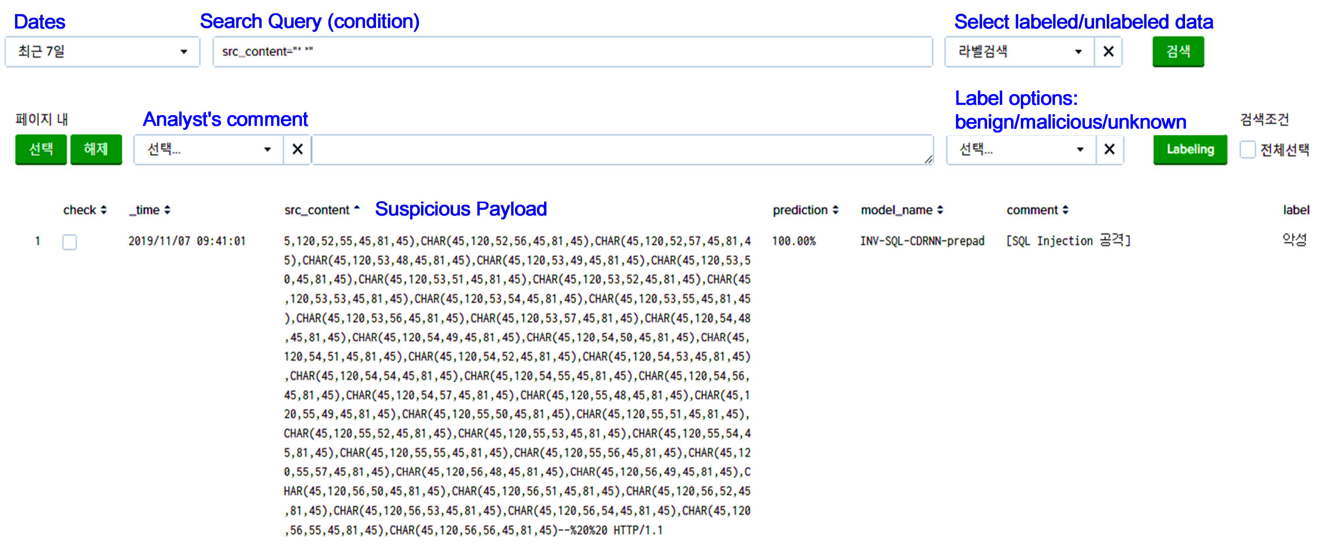
To validate the performance of deep learning models, we used real-time data and public HTTP datasets (CSIC-2010, CICIDS 2017 HTTP datasets), and experimented with 50 epochs on the previously selected model. The experimental results of real-time data showed that the proposed CNN-LSTM model can be used for general HTTP data with a high performance. The AI-IDS is a deep learning-based model with no pre-feature extraction and therefore all strings can be processed.

For all experiments for each dataset, the model parameters were modified to obtain the results above and to optimize the performance on different datasets. Considering that our model has 14,000 trainable parameters, the CSIC-2010 and CICIDS-2017 are relatively small, which leads to overfitting and low performance. Experimental results shown in Fig. 6 showed a high accuracy of 91–93% for each dataset in CSIC-2010 and CICISC-2017. The precision was in the range of 86–98%, and the F-Score was in the range of 80–82%, which is lower than the experimental results of the previous real-time data. Experimental results showed that the performance of the model is affected by the number of samples and the diversity of the training data. It was difficult to cross-validate our model with two published datasets owing to small samples. If we had a large amount of non-repeated HTTP data, the experimental performance would improve and would return more reliable results. Considering the above results, our model is more suitable for a large amount of data, and we demonstrated the excellence of our model by training with various datasets of more than 6 million HTTP traffic data.

VI. EFFICACY AND APPLICATION

This section describes cases of how AI-IDS detects variant attacks that bypass detection on legacy signature-based NIDS, and how Snort rules can be rewritten or improved.

The AI-IDS in Fig. 7 performs “predict” based on the completed h5-model shown in Fig. 3, and it predicts real-time data by inspecting the attack as a prediction output. When the prediction value is 100%, the NIDS knows the payload is malicious, but the results of analysts are not perfectly reliable because an initial AI-IDS result may contain an analysis error. Thus, an analyst needs to confirm the final step until a stable



	check	_time	src_content	prediction	model_name	comment	label
1	☐	2019/11/07 09:41:01	5,120,52,55,45,81,45),CHAR(45,120,52,56,45,81,45),CHAR(45,120,52,57,45,81,45),CHAR(45,120,53,48,45,81,45),CHAR(45,120,53,49,45,81,45),CHAR(45,120,53,50,45,81,45),CHAR(45,120,53,51,45,81,45),CHAR(45,120,53,52,45,81,45),CHAR(45,120,53,53,45,81,45),CHAR(45,120,53,54,45,81,45),CHAR(45,120,53,55,45,81,45),CHAR(45,120,53,56,45,81,45),CHAR(45,120,53,57,45,81,45),CHAR(45,120,54,48,45,81,45),CHAR(45,120,54,49,45,81,45),CHAR(45,120,54,50,45,81,45),CHAR(45,120,54,51,45,81,45),CHAR(45,120,54,52,45,81,45),CHAR(45,120,54,53,45,81,45),CHAR(45,120,54,54,45,81,45),CHAR(45,120,54,55,45,81,45),CHAR(45,120,54,56,45,81,45),CHAR(45,120,54,57,45,81,45),CHAR(45,120,55,48,45,81,45),CHAR(45,120,55,49,45,81,45),CHAR(45,120,55,50,45,81,45),CHAR(45,120,55,51,45,81,45),CHAR(45,120,55,52,45,81,45),CHAR(45,120,55,53,45,81,45),CHAR(45,120,55,54,45,81,45),CHAR(45,120,55,55,45,81,45),CHAR(45,120,55,56,45,81,45),CHAR(45,120,55,57,45,81,45),CHAR(45,120,56,48,45,81,45),CHAR(45,120,56,49,45,81,45),CHAR(45,120,56,50,45,81,45),CHAR(45,120,56,51,45,81,45),CHAR(45,120,56,52,45,81,45),CHAR(45,120,56,53,45,81,45),CHAR(45,120,56,54,45,81,45),CHAR(45,120,56,55,45,81,45),CHAR(45,120,56,56,45,81,45)--%20%20 HTTP/1.1	100.00%	INV-SQL-CDRNN-prepad	[SQL Injection 공격]	악성

FIGURE 7. Labeling Tools for deep learning on AI-IDS

level has been reached. We classified the suspicious payloads as a prediction value within a range of 50–100%, and an average of 100-500 events occurred every 3h. We assumed that AI-IDS is classified as normal or malignant, and less than 50% of the predicted values are labeled as "benign," and 50–100% are classified as "malicious" payloads. In Fig. 7, the analyst labels suspicious payloads on the program as "benign," "malicious," or "unknown" using a conditional search. The labeled data is used for daily retraining. The label program shows the prediction value (%) generated by the optimal CNN-LSTM model, and the analyst can use it as a reference for identifying the actual malicious payloads. Some of the analysis information, such as src_ip/port and dest_ip/port can be used to register a blocking policy in the firewall.

The effects of applying AI-IDS are summarized as follows:

--First, it can detect variant bypass attacks that are not detected on legacy Signature-based NIDS. For all AI-IDS predictions, security events on the legacy NIDS are automatically excluded, such that no duplicate events can occur.

--Second, it is possible to write or modify Snort rules for new patterns. If legacy NIDS have existing rules but cannot detect attacks, then this had to be caused by Snort grammatical errors or missing patterns in the rules. However, it can also be a detection failure due to low performance or functional failure.

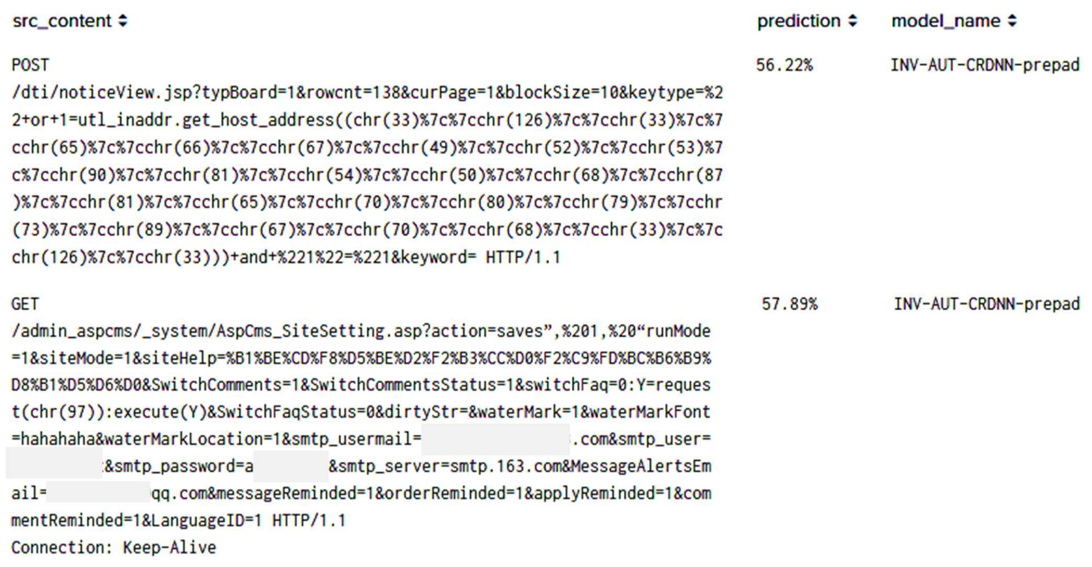
A. Detection of Obfuscated Variants

Table VIII shows an example of a variant attack detection. A common intrusion pattern is a scan of an admin page or file upload page, usually accessed by an attacker via a known open source path. Suppose that there is an admin page such as "http://target.com/admin/index.php" and a rule that detects "/admin/index.php" in legacy NIDS. The AI-IDS examines payloads coming from the trained CNN-LSTM model in real-time to detect abnormal URI accesses that detect variant attacks on "index.php" parameters and subpaths. It also detects similar and different new variant attacks for all attack types and patterns, as well as the examples shown in Table VIII. In the case of SQL Injection, the detection accuracy of variant patterns is close to 100%.

TABLE VIII
Detecting variant patterns on AI-IDS

An original malicious pattern (NIDS)	Variant patterns (AI-IDS)
/admin/index.php	/admin/index.php?login /admin/index.php?page=unexisting../../../../../../../windows/win.ini%00 /admin/lib/gradient/gradient.php?tam=..%2f..%2f..%2f..%2fboot.ini%00 /admin/js/kindeditor/php/file_manager_json.php /admin/kindeditor/php/file_manager_json.php /admin/login.php/"onmouseover=alert("xss-test")> /admin/mysql2/index.php /admin/lost-pass.php

The AI-IDS can also detect unknown variant patterns or obfuscated attacks, as shown in Fig. 8. An attacker can use URL Encoding or base64 to bypass arbitrary payloads in the security system to attack web servers effectively. An attacker uses the Char( ) function to insert code into noticeView.jsp to attempt to acquire system information. In other cases, the attacker attempts to send spare-phishing mails, attempting to communicate with an external SMTP server by injecting irregular or encoded code to AspCms_SiteSetting.asp (AspCMS). Recent malicious HTTP payloads contain irregular patterns that are difficult to detect as simple strings. A commercial NIDS detects most known attacks or patterns but does not detect strings that do not have a registered pattern. By contrast, the AI-IDS can detect variant and obfuscated attacks that cannot be detected with legacy NIDS.



```
src_content ⬍                                                                  prediction ⬍   model_name ⬍

POST                                                                           56.22%         INV-AUT-CRDNN-prepad
/dti/noticeView.jsp?typBoard=1&rowcnt=138&curPage=1&blockSize=10&keytype=%2
2+or+1=utl_inaddr.get_host_address((chr(33)%7c%7cchr(126)%7c%7cchr(33)%7c%7
cchr(65)%7c%7cchr(66)%7c%7cchr(67)%7c%7cchr(49)%7c%7cchr(52)%7c%7cchr(53)%7
c%7cchr(90)%7c%7cchr(81)%7c%7cchr(54)%7c%7cchr(50)%7c%7cchr(68)%7c%7cchr(87
)%7c%7cchr(81)%7c%7cchr(65)%7c%7cchr(70)%7c%7cchr(80)%7c%7cchr(79)%7c%7cchr
(73)%7c%7cchr(89)%7c%7cchr(67)%7c%7cchr(70)%7c%7cchr(68)%7c%7cchr(33)%7c%7c
chr(126)%7c%7cchr(33)))+and+%221%22=%221&keyword= HTTP/1.1

GET                                                                            57.89%         INV-AUT-CRDNN-prepad
/admin_aspcms/_system/AspCms_SiteSetting.asp?action=saves”,%201,%20“runMode
=1&siteMode=1&siteHelp=%B1%BE%CD%F8%D5%BE%D2%F2%B3%CC%D0%F2%C9%FD%BC%B6%B9%
D8%B1%D5%D6%D0&SwitchComments=1&SwitchCommentsStatus=1&switchFaq=0:Y=reques
t(chr(97)):execute(Y)&SwitchFaqStatus=0&dirtyStr=&waterMark=1&waterMarkFont
=hahahaha&waterMarkLocation=1&smtp_usermail=[redacted].com&smtp_user=
[redacted]&smtp_password=a[redacted]&smtp_server=smtp.163.com&MessageAlertsEm
ail=[redacted]qq.com&messageReminded=1&orderReminded=1&applyReminded=1&com
mentReminded=1&LanguageID=1 HTTP/1.1
Connection: Keep-Alive
```

FIGURE 8. Detecting encoded and obfuscated payloads

B. Improvement of signature-based Snort rules

The second effect is to improve the signature-based Snort Rule. The AI-IDS does not generate detected events in duplicate on legacy signature-based NIDS, analysts can identify new patterns for threats by analyzing suspicious payloads: (i) A new rule can be written for a new pattern (ii) If a rule exists, but cannot be detected, the detection rule can be improved by correcting an error in the rule's options: an offset, depth, distance or within and so on. We write or improve on average about five new detection rules per month manually. If an event occurs in the AI-IDS when the rules are normally applied, we have to suspect a detection failure on signature-based NIDS. Table VII shows that AI-IDS detects a vulnerability attack (CVE-2018-9174) of DedeCMS. When NIDS rules for related attacks are not registered in the currently operating NIDS, new rules can be registered based on the detection of AI-IDS.

AI-IDS detects attacks that are not detected in the existing NIDS in step 1. As AI-IDS double-checks with existing NIDS, basically all events detected by AI-IDS are not detected by NIDS. The analyst examines the existing rules in step 2 to review why the existing NIDS did not detect payloads. It is usually found that attackers used several methods to randomly change the encoding or attacked strings to bypass NIDS. In addition, if the signature is individually over-customized, there are few types of attacks that cannot be detected, even if there is only a slight change in the attack pattern. Therefore, step 3 modified existing signatures to rewrite detection rules that typically detect PHP webshell code attacks. However, if a general-purpose detection rule is written without regard to the environment, appropriate optimization tasks are required as the number of detections increases.

Step 1. AI-IDS detection for suspicious payloads

Suspicious payloads:
GET
/uploads/dede/sys_verifies.php**?action=getfiles&refiles%5B0%5D=123&refiles%5B1%5D=%5C%22;eval($_POST%5Bysy%5D);die()**;// HTTP/1.1
Connection: Keep-Alive

Decoded patterns:
php?action=getfiles&refiles[0]=123&refiles[1]=\%22;eval($_POST[ysy]);die();//

Step 2. Analysis of related existing rules
(Why not detectable?)

(Rule 1)alert tcp any any <> any 80
(msg:"PHP_Webshell_Backdoor ";
flow:established; **content:"($_POST[cmd])";**
nocase; fast_ pattern:only; metadata:service http;)

(Rule 2)alert tcp any any <> any any (msg:"MAL-INF-Malware-C1(Caidao).";flow:established,to_server;
content:"GET";offset:0;depth:4;
content:"$_GET[z0]))";content:"&z0=";
content:"base64_decode";content:"eval";
fast_pattern:only; metadata:service http;)

Step 3. Writing a new Snort rule or improving an existing Snort rule (general use in case)

(New Rule) alert tcp any any -> any any
(flow:established,to_server;
content:"$_GET"; nocase; fast_pattern;
content:"eval"; nocase; **distance:-20;**
within:15; pcre:"/^(GET| POST)/M";
metadata:service http; msg:"INV-SHL-PHP_Webshell _GET_Attempt";)

VII. CONCLUSION

We proposed an optimal CNN-LSTM model based on SFL and successfully applied payload-level deep learning techniques in a high-performance computing environment. The AI-IDS distinguishes between normal and abnormal traffic on HTTP traffic that could not be detected in legacy signature-based NIDS because AI-IDS can formalize unknown patterns, help write or improve signature-based rules for new vulnerabilities, variants, and bypass attacks.

Network meta-data, without its payload is usually difficult to identify whether it is malicious or not. Thus, we review the HTTP header and body of web attacks in detail. We also used real-time web traffic for deep learning, but initially, we learned that AI-IDS needed to be re-validated for predicted suspicious events due to false positives alarms. The AI-IDS performs continuous optimization by re-training analysis information that is labeled “benign,” “malicious,” and “unknown.” Thus, it should be used as an assistant system until it reaches a high-quality level. If the quality goes beyond the ability of humans by continually learning, it could be executed as an automated analysis. Ultimately, the goal of AI-IDS is to outperform human analysis quality and to help analysts handle large quantities of unknown security events.

Previous works have mainly considered accuracy (ACC) in terms of performance measures, but scalability and precision are also important indicators for applying deep learning in the real-world. In practical security services, re-validation for predicted events is a required task because of the low tolerance for analysis errors.

REFERENCES

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol–HTTP/1.1," 1997.

[2] D. Atienza, Á. Herrero, and E. Corchado, "Neural analysis of http traffic for web attack detection," in *Computational Intelligence in Security for Information Systems Conference*, 2015: Springer, pp. 201-212.

[3] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE network*, vol. 8, no. 3, pp. 26-41, 1994.

[4] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 686-728, 2019.

[5] "FSI AI-IDS software for Splunk," 2020. [Github repository]. Available: https://github.com/ackim-fsi/AI-IDS

[6] Y. Liu, S. Liu, and X. Zhao, "Intrusion detection algorithm based on convolutional neural network," *DEStech Transactions on Engineering and Technology Research*, 2017.

[7] W. Wang *et al.*, "HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792-1806, 2017.

[8] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access*, vol. 5, pp. 21954-21961, 2017.

[9] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41-50, 2018.

[10] K. Wu, Z. Chen, and W. Li, "A Novel Intrusion Detection Model for a Massive Network Using Convolutional Neural Networks," *IEEE Access*, vol. 6, pp. 50850-50859, 2018.

[11] S. Naseer *et al.*, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48231-48246, 2018.

[12] Y. Ding and Y. Zhai, "Intrusion Detection System for NSL-KDD Dataset Using Convolutional Neural Networks," in *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, 2018: ACM, pp. 81-85.

[13] S. Otoum, B. Kantarci, and H. T. Mouftah, "On the feasibility of deep learning in sensor network intrusion detection," *IEEE Networking Letters*, vol. 1, no. 2, pp. 68-71, 2019.

[14] N. Chouhan, A. Khan, and H.-u.-R. Khan, "Network anomaly detection using channel boosted and residual learning based deep convolutional neural network," *Applied Soft Computing*, vol. 83, 2019.

[15] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," *IEEE Access*, vol. 7, pp. 41525-41550, 2019.

[16] Z. Chiba, N. Abghour, K. Moussaid, A. El omri, and M. Rida, "Intelligent approach to build a Deep Neural Network based IDS for cloud environment using combination of machine learning algorithms," *Computers & Security*, vol. 86, pp. 291-317, 2019.

[17] H. Zhang, B. Zhao, H. Yuan, J. Zhao, X. Yan, and F. Li, "SQL Injection Detection Based on Deep Belief Network," in *Proceedings of the 3rd International Conference on Computer Science and Application Engineering*, 2019: ACM, p. 20.

[18] O. Faker and E. Dogdu, "Intrusion Detection Using Big Data and Deep Learning Techniques," in *Proceedings of the 2019 ACM Southeast Conference*, 2019: ACM, pp. 86-93.

[19] M. Aloqaily, S. Otoum, I. Al Ridhawi, and Y. Jararweh, "An intrusion detection system for connected vehicles in smart cities," *Ad Hoc Networks*, vol. 90, p. 101842, 2019.

[20] "KDD Cup 1999 Data," 1999. [Datasets]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. last accessed on: 17.11.19

[21] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, vol. 86, pp. 147-167, 2019.

[22] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009: IEEE, pp. 1-6.

[23] A. Shenfield, D. Day, and A. Ayesh, "Intelligent intrusion detection systems using artificial neural networks," *ICT Express*, vol. 4, no. 2, pp. 95-99, 2018.

[24] R. Singh, H. Kumar, and R. Singla, "A reference dataset for network traffic activity based intrusion detection system," *International Journal of Computers Communications & Control*, vol. 10, no. 3, pp. 390-402, 2015.

[25] M. Sabhnani and G. Serpen, "Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set," *Intelligent data analysis*, vol. 8, no. 4, pp. 403-415, 2004.

[26] J. Gu, L. H. Wang, H. W. Wang, and S. S. Wang, "A novel approach to intrusion detection using SVM ensemble with feature augmentation," *Computers & Security*, vol. 86, pp. 53-62, 2019.

[27] N. Moustafa, J. Hu, and J. Slay, "A holistic review of Network Anomaly Detection Systems: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 128, pp. 33-55, 2019.

[28] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 686-728, 2018.

[29] Splunk, "Splunk Stream (STM)," 2019. [Online]. Available: https://splunkbase.splunk.com/app/1809/. last accessed on: 20.03.20

[30] M. Mamun, R. Lu, and M. Gaudet, "Tell Them from Me: An Encrypted Application Profiler," in *International Conference on*

Network and System Security, Lecture Notes in Computer Science, vol. 11928, 2019: Springer, pp. 456-471.

[31] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820,* 2015.

[32] H. Liu, B. Lang, M. Liu, and H. Yan, "CNN and RNN based payload classification methods for attack detection," *Knowledge-Based Systems,* vol. 163, pp. 332-341, 2019.

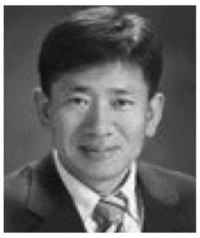
[33] Y. Xin *et al.*, "Machine learning and deep learning methods for cybersecurity," *IEEE Access,* vol. 6, pp. 35365-35381, 2018.

[34] C. T. Giménez, A. P. Villegas, and G. Á. Marañón, "HTTP data set CSIC 2010," *Information Security Institute of CSIC (Spanish Research National Council),* 2010. [Datasets]. Available: https://www.isi.csic.es/dataset/ last accessed on: 17.11.19

[35] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *ICISSP*, 2018, pp. 108-116. [Datasets]. Available: https://www.unb.ca/cic/datasets/ids-2017.html last accessed on: 17.11.19

AECHAN KIM is an assistant manager in Financial Security Institute (FSI), Yongin, South Korea. He received the B.S. degree in Industrial Engineering from Seoul National University of Science and Technology, Seoul, South Korea, in 2009, and the M.S. degree in financial information security from Korea University, Seoul, in 2014, where he is currently pursuing the Ph.D. degree in Graduate School of Information Security. His research interests include applied deep learning and intrusion detection on computer networks.

MOHYUN PARK is a manager in Financial Security Institute (FSI), Yongin, South Korea. He received the B.S. degree in Computer Science from Seoul National University, Seoul, South Korea, in 2013. His research interests include applied deep learning and intrusion detection on computer networks.



DONG HOON LEE (M'06) received the B.S. degree from the Department of Economics, Korea University, Seoul, in 1985, and the M.S. and Ph.D. degrees in computer science from The University of Oklahoma, Norman, in 1988 and 1992, respectively, where he is currently a Professor with the Graduate School of Information Security, Korea University, where he has been with the Faculty of Computer Science and Information Security, since 1993. His research interests include cryptographic protocols, applied cryptography, functional encryption, software protection, mobile security, vehicle security, and ubiquitous sensor network (USN) security.