IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 24, 2019, accepted June 24, 2019, date of publication June 28, 2019, date of current version July 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2925639

Detection and Recognition of Atomic Evasions Against Network Intrusion Detection/ Prevention Systems

JIA JINGPING[1], CHEN KEHUA[1], CHEN JIA[2], ZHOU DENGWEN[1], AND MA WEI[1]
[1]School of Control and Computer Engineering, North China Electric Power University, Beijing 102206, China
[2]China Communications Asset Management Company Ltd., Beijing 100013, China

Corresponding author: Chen Jia (snownight.chen@163.com)

This work was supported in part by the Fundamental Research Funds for the Central Universities through the Project "Research on the Detection of Network Intrusion Evasion Based on Big Data Technology" under Grant 2016MS33, and in part by the Beijing Natural Science Foundation under Grant 4162056.

ABSTRACT Network evasions can bypass network intrusion detection/prevention systems to deliver exploits, attacks, or malware to victims without being detected. This paper presents a novel method for the detection and recognition of atomic network evasions by the classification of a transmission control protocol (TCP) stream's packet behavior. The syntax for the conversion of TCP streams to codeword streams is proposed to facilitate the extraction of statistical features while preserving the evasion behavior attributes of original network flows. We developed a feature extraction method of employing the normalized term frequencies of codewords to characterize intra and inter packet attribute patterns hidden in actual TCP streams. A TCP stream is then transformed to a fixed length numeric feature vector. Supervised multi-class classifiers are built on the extracted feature vectors to differentiate different types of evasions from normal streams. The quantitative evaluations on an evasion dataset consisting of normal network flows and eight types of atomic evasion flows demonstrated that the proposed approach achieved an encouraging performance with an accuracy of 98.95%.

INDEX TERMS Network intrusion detection/prevention, network evasion, term frequency and inverse document frequency.

I. INTRODUCTION

Network Intrusion Detection/Prevention Systems (further NIDS/NIPS) are now widely used to improve the security of networks run by providers, enterprises and even home users. These systems analyze traffic on networks, detect any malicious activities and make alerts to system security operators (NIDS) or disrupt suspicious network connections (NIPS). Based on the detection approach, NIDS can be divided into two categories: the anomaly-detection NIDS and the misuse-detection NIDS. The former gets a notion of normal activity and flags deviation from that profile. The latter monitors activities with precise descriptions of known malicious behaviors. In terms of actual deployments, the misuse-detection NIDS products are found almost exclusively in use, commonly in the form of signature-based NIDS which scans network traffic and seeks for characteristic byte sequences [1].

As soon as they are deployed, NIDS/NIPS themselves become targets of attacks aiming to undermine their capabilities. These attacks continue to be one of the most serious threats in the domain of cyber security. The advent of network evasion techniques enables attackers to deliver exploit code to victims without being detected by a misuse-based NIDS/NIPS. Due to the robustness principle [2] in an internet protocol design, which means that an implementation of a protocol should be careful to send well-formed datagrams, but should accept any datagrams that it can interpret, there are various interpretations in different protocol implementations. Attackers can use these ambiguities to deliberately craft network traffic so that an NIDS/NIPS and endpoint systems process packets in different ways. If the processing of the packets generates different representations of the raw data in the NIDS/NIPS and in the end systems, an attack

The associate editor coordinating the review of this manuscript and approving it for publication was Dezhong Peng.

can reach the destination undetected. Such concealment techniques are collectively referred to as evasion techniques. First introduced by Ptacek and Newsham [3], evasion techniques have evolved from the exploiting tactics at the network and transport layers to transforming application layer messages, such as the HTTP IDS evasion [4] and polymorphic shellcode [5]. Vidal *et al.* [6] classified the most representative evasion techniques into five categories: Insertion and Evasion, Denial of Service, Malware Obfuscation, Link Layer, and Application Layer. A single evasion technique is called an atomic evasion [7]. Xiong *et al.* [8] summarized common atomic evasion techniques and at least 147 atomic evasions have been discovered [9]. Atomic evasions can be combined to further confuse analysis devices and the number of their unique combinations is 2^{147} which is a truly massive potential threat. Although not all of the combinations work, many of them do. Chammem [10] investigated non-exhaustive strategies to enhance the search performance of effective combinations.

Modern NIDS vendors publish extensive performance testing results regarding the line speed and breadth of attacks detected by their systems, but little information regarding their resilience to evasions. A recent effectiveness evaluation of 35 well-known evasions against nine commercial and one free state-of-the-art NIDS showed that even a single evasion technique from the 1990s with suitable parameters can successfully evade the detection of the best existing NIDS [7]. Cheng *et al.* [11] assessed the effectiveness of evasion techniques for FortiGate, Snort and ZyXEL. These three signature-based NIDS all operated with up-to-date firmware/code and rules. However, they were still bypassed by the IP fragmentation atomic evasion. Gorton discovered that some combinations of segmentation, overlapping and chaffing could make Snort fail to detect the attacks underneath [12]. A test performed in 2017 by Levomaki *et al.* [13] showed that TCP level atomic evasions were able to bypass commercial products of Gartner ''Intrusion Detection and Prevention'' and ''Enterprise Firewall'' 2017 magic quadrant leading vendors.

Faced with the constant threat of network evasions, we propose to address the detection and recognition problem of atomic evasions in the network and transport layers by means of the classification of a TCP stream's packet behavior.

The philosophy of our method is to extract features discriminative enough to differentiate evasions from normal behaviors, and train a classifier to identify the evasions' types. Specific evasion techniques modify traffic streams in their unique ways. This leads to value deviation of the related fields in a packet's header, or deviation of the affected fields' relative value between neighboring packets in the receiving order. These deviations also occur in normal streams for benign reasons. So it is not appropriate to use simple thresholds on the deviations to determine if evasions actually occurred. Instead we employ the normalized term frequencies of codewords learned from a training dataset to characterize the intra and inter packet attribute patterns hidden in actual evasion traffic streams, and use a classifier to differentiate different types of evasions from normal streams.

The first contribution of this paper is that, to the best of our knowledge, we are the first to introduce the TF-IDF (term frequency–inverse document frequency) model to the network evasion problem. As a simple and effective representation used in the Natural Language Processing (NLP) and information retrieval, the TF-IDF model has found enormous success in document classification. In our evasion detection problem, it offers an effective way of mapping intra and inter packet attributes from TCP streams of variable length to fixed length numeric vectors. The second contribution is the proposal of a method for converting network flows to codeword streams, which converts a difficult raw binary network stream classification problem to a text classification task. Consisting of finite codewords, codeword streams preserve the evasion behavior attributes of original TCP streams and facilitate the extraction of statistical features.

The rest of this paper is organized as follows. In the next section, we survey related work. Section III describes the proposed approach, details the generation of codeword streams and specifies the feature extraction method. In section IV we present our dataset setup and give the performance results. Section V concludes the paper.

II. RELATED WORK

Some previous studies on anti-evasion techniques have focused on the elimination of ambiguities between an NIDS and endpoint systems. One of the most representative approaches is the traffic normalizer [14] proposed by Handley and Paxson. It is an inline network element which patches packet streams to remove potential ambiguities. Because it consumes a large quantity of resources to store the state and previous packets of each connection when analyzing the consistency of connections, the traffic normalization's performance and robustness is often an issue when working with high speed networks [15]. A similar approach was suggested by Watson *et al.* [16], who introduced an intermediate system called the Protocol Scrubbing which converted traffic to well-formed TCP data. However, it may erode the transport semantics and disrupt the useful traffic [14]. Another way of disambiguating is the Active Mapping proposed by Paxson and Shankar [15]. It builds profiles of a network topology and hosts' TCP/IP policies which an NIDS can use to disambiguate the interpretation of network traffic on a per-host basis. This technique has been adopted by Snort [17] in its IP defragmentation preprocessor (frag3) and stream reassembly preprocessor (stream5) [18]. Active Mapping avoids the performance and semantic drawbacks of traffic normalization but it may fail to map hosts whose IP addresses are allocated via DHCP. Also the anomalous traffic it actively sends out may be rejected by firewalls or routers [19]. Frag3 utilizes target based fragmentation reassembly [20] and stream5 uses target based TCP stream reassembly [21]. These two preprocessors can detect and alert on IP fragmentation and TCP reassembly related anomalies. However, the alerts they raise

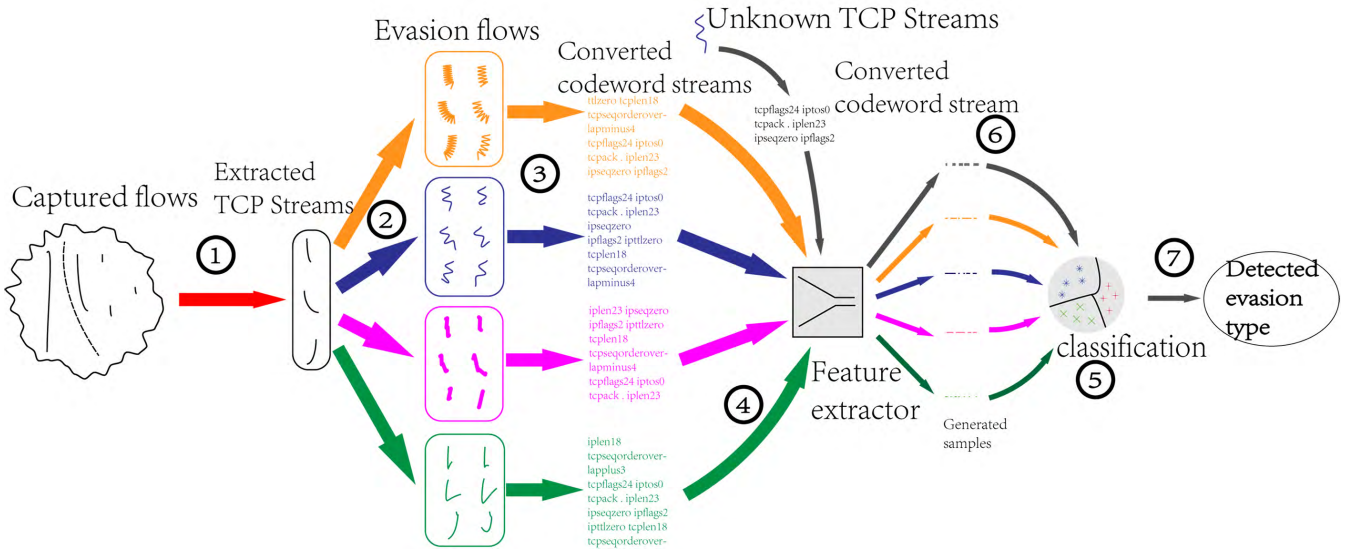


FIGURE 1. Overall procedure of the proposed approach. Our evasion detector first extracts normal TCP streams from a set of captured network flows(①), and applies atomic evasion techniques on these normal streams to get evasion streams(②). Then it converts the normal and evasion streams to codeword streams(③), extracts numeric features from these codeword streams(④), and trains a classifier to classify them(⑤). For an unknown TCP stream, its features are extracted from its converted codeword stream (⑥), and the trained classifier is used to identify the stream as a normal flow or one type of evasion.

are minimally accurate because alerts can also be generated by normal traffic. It is not easy to distinguish false positives from the alerts that accompany an actual attack [12]. So these alerts are mostly for debugging purpose [22]. While attackers can actively exploit ambiguities to evade an NIDS, ambiguities unfortunately also rise in traffic streams for benign reasons, requiring analysis for ascertaining whether the condition constitutes a threat.

While earlier studies had tried to eliminate the ambiguities to prevent evasions, recent machine learning based approaches to combating evasions devoted to increasing their classifiers' resilience to evasions. Zhang *et al.* [23] proposed an adversary-aware feature selection model that incorporated specific assumptions on an evaion's data manipulation strategy. Anindya and Kantarcioglu [24] developed a strategy to defend against evasions from a game theoretic viewpoint by searching for the optimal parameters of centroid-based clustering models. Katzir and Elovici [25] suggested building robust classifiers consisting of only resilient features to remain unchanged despite the occurrences of evasions. Homoliak *et al.* [26] improved an anomaly-detection NIDS's resilience to link layer evasions by augmenting the NIDS's training dataset with attack samples on which evasions were applied. The NIDS they improved was a non-payload-based intrusion detection classifier which distinguished attacks from legitimate traffic based on the features extracted only from the IP and TCP headers. Their experiments showed that evasions caused an exacerbation of the True Positive Rate(TPR) ranging from 7.8% to 66.8%, while the dataset augmentation increased the TPR by 4.21%-73.3%. However, all these proposed approaches are based on the anomaly-detection NIDS whose actual deployment is rarely seen due to its high false positive rate and low accuracy [27]. Du and Yang [28] developed probabilistic graphical models to analyse the impact of removal, insertion, and alteration evasion actions on performance of an NIDS. They proposed the Expected Classification Accuracy (ECA) to assess the impact of evasions in terms of class distribution overlap. Their study solved the problem of evasion evaluation but not the problem of evasion detection.

In this paper, we propose another way to prevent evasions. Instead of eliminating ambiguities or increasing an NIDS's resilience to evasions, we can detect and recognize evasion behaviors in network activities and may further inform other security devices to terminate the involved stream connections.

III. THE PROPOSED APPROACH

The overall approach is illustrated in FIGURE1. Our approach works in two stages: training and detection. In the training stage, we first acquire normal TCP streams by extracting TCP streams from captured network flow trace files. Evasion streams are then obtained by applying evasion techniques on the normal TCP streams. Next, codeword streams are generated from the normal and evasion streams(SectionIII-A). Finally, feature vectors are extracted from the codeword streams(SectionIII-B) and used to train a multi-class classifier. In the detection stage, an unknown TCP stream is converted to a codeword stream and a feature vector is extracted from the codeword stream. The trained classifier is employed to categorize the feature vector and determine the type of the TCP stream. Our approach focuses on the inspection of a stream's packet behavior, not its content. The proposed codeword stream generation method encodes a stream's packet behavior attributes in codewords, which

converts the difficult raw binary network stream classification problem to a textual codeword stream classification task. In addition, in feature extraction we exploit the TF-IDF method which characterizes the different attribute patterns hidden in different types of evasion and normal streams. This transforms the variable length codeword stream classification problem into a fixed length vector classification task, which can benefit from classical classifiers such as the linear SVM and the Randowm Forrest.

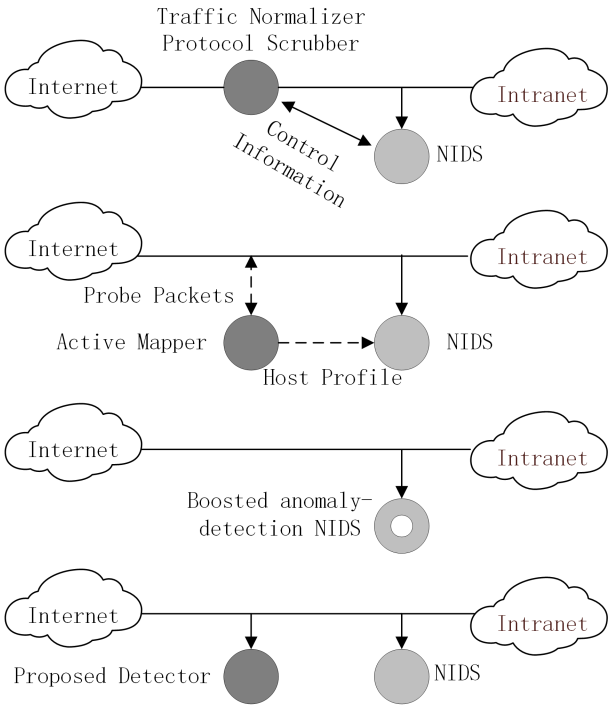


FIGURE 2. Illustration of the proposed method and previous approaches.

FIGURE2 illustrates the difference between the proposed approach and previous methods. Traffic normalizer and Protocol Scrubber, as shown in the first row, sit in the traffic path, patch up packets and eliminate potential ambiguities before the traffic is seen by the NIDS and hosts in the intranet it's protecting. Active Mapper, as shown in the second row, sits beside an NIDS, builds and provides the NIDS with hosts' profiles via the probe packets sent by the mapper. Armed with the host profiles, the NIDS can resolve ambiguities in traffic streams. The machine learning based approaches mentioned in Section II build an anomaly-detection NIDS whose classifier has better resilience to evasions. Our approach works like an NIDS. As shown in the last row of FIGURE2, it monitors the TCP streams in the traffic path and reports the atomic evasion events it finds. Its report can be valuable information for the NIDS administrator, and can be used to inform firewalls to terminate suspicious evading TCP connections.

A. CODEWORD STREAM GENERATION

Although there have been lots of datasets [29] available in the field of cyber security, none of them fits for the scenario of network evasion detection. The DARPA Intrusion Detection Data Sets [30], its famous derivation KDD Cup 1999 Data [31], and the improved version [32], for instance, are widely used for the IDS evaluation without evasion. The UNB ISCX 2012 and 2017 Intrusion Detection Evaluation Datasets [33] are for the same purpose. Public datasets from CAIDA [29] and Internet Traffic Archive [29] are mainly for the study of network dynamics, usage characteristics, and growth patterns. The MAWILab [34] trace repositories provide datasets for traffic anomaly detection. For the purpose of evasion detection we need to build a dataset consisting of samples of different evasion types as well as samples without evasions.

First, normal network flows were captured from a small intranet network whose bandwidth was 10Mbps. We ensured that no evasion tools were running on any hosts of the network. These network flows were captured at different time periods on a typical working day. The capturing was performed on a mirror port of the intranet's hub using tcpdump. The hub worked at the transfer rate of 10Mbps while the desktop performing the capturing had its gigabit network interface card connecting to the hub's mirror port. The desktop was equipped with an Intel I5 CPU and eight gigabytes primary memory to make sure that no packets would be dropped due to the lack of processing efficiency.

TCP streams were then extracted from the captured flows and saved in separate trace files in the tcpdump's pcap format. These TCP streams are taken as normal network streams. Evasion techniques were applied on these trace files to generate evasion network streams. Since we focused only on the detection of atomic evasions in this study, evasion techniques were applied individually and not in combination. The evasion techniques we applied were ip_chaff, ip_frag, ip_opt, ip_ttl, ip_tos, tcp_chaff, tcp_opt and tcp_seg. Based on fragroute [35] we developed a program which was able to apply an evasion technique to the trace file under operation if it decided that the evasion technique was applicable to that trace. According to the evasion technique specifications, this program read each packet from the normal network flow trace files, then dropped the packet or saved the modified copy or inserted new packets to the resulting trace files. Each resulting trace file was comprised of packets generated by a specific evasion technique from the originating normal stream, and it was labeled as the corresponding evasion class.

Streams of evasion attacks differ from normal streams in some quantitative attributes. In our study, we formulate these attributes into intra-packet ones and inter-packet ones. Intra-packet attributes such as the length of an IP packet can be extracted from a single packet. Inter-packet attributes such as the difference between the IP fragment offset values of two packets can be derived from more than one packet of the same stream. We use codewords to code these attributes and convert each TCP stream to a codeword stream.

TABLE1 lists and describes the conventions that are used in the conversion syntax of network traces to codeword streams. The complete conversion syntax of a TCP stream in a

TABLE 1. Syntax conventions.

Convention	Used for
<label>::=	The name for a block of syntax. This convention is used to group and label sections of lengthy syntax or a unit of syntax that can be used in more than one location within a stream. Each location in which the block of syntax can be used is indicated with the label enclosed in chevrons: <label>.
[...*n*]	Indicates the preceding item can be repeated n number of times. The occurrences are separated by blanks.
| (vertical bar)	Separates syntax items enclosed in brackets or braces. You can use only one of the items.
[] (brackets)	Optional syntax items.
{ } (braces)	Required syntax items.
italic	Parameter.
bold	Text that must be present exactly as shown.

TABLE 2. Conversion syntax.

Syntax name	Syntax defination
<codeword stream>::=	<frame codewords>[...n]
<frame codewords>::=	{<IP feature codewords>}[...n][<TCP feature codewords>][...n].
<IP feature codewords>::=	{<IP length>}[<IP fragment>]{<IP flag >}{<IP ttl>}[<IP route option>][<IP option>][<IP tos>]
<IP length>::=	**iplen**{*the length of the IP packet in 8 bytes*}
<IP fragment>::=	{**ipseqplus** | **ipseqzero** | **ipseqminus**}
<IP flag >::=	**ipflags**{*the decimal value of the IP packet header's Flags field*}
<IP ttl>::=	{**ipttlzero** |{**ipttlplus** | **ipttlminus**}{ *the absolute value of the difference between the current IP packet's TTL field and the previous IP packet's TTL field* }}
<IP route option>::=	**ipoptroute** {0|1}
<IP option>::=	**ipbadoption**
<IP tos>::=	**iptos**{*decimal value of the IP header's DSCP field, originally defined as the tos*}
<TCP feature codewords>::=	[<TCP checksum>]{<TCP length>}{<TCP segment>}{<TCP flag>}[<TCP timestamp>][<TCP sync>][<TCP seq_position>][<TCP mss>][<TCP wscale>][<TCP acknowledgement>]
<TCP checksum>::=	**tcpchecksum1**
<TCP length>::=	**tcplen** { *the length of the TCP segment in 8 bytes* }
<TCP segment>::=	{**tcpseqorderoverlap0**| **tcpseqorderoverlapplus** *code for TCP segment absence*| {**tcpseqorderoverlapminus** *absolute value of the encoded TCP segments overlapping*}}
<TCP flag>::=	**tcpflags**{ *the decimal value of the TCP header's Flags field* }
<TCP timestamp>::=	{ **tcptimestampplus** | **tcptimestampzero** | **tcptimestampminus** }
<TCP sync> ::=	**tcpsyn**
<TCP seq_position> ::=	{**notcpchaffseq** | **tcpchaffseq**}
<TCP mss> ::=	**tcpoptmss**
<TCP wscale> ::=	**tcpwscale**
<TCP acknowledgement> ::=	**tcpack**

TABLE 3. Attributes, codewords and their explanations.

Attributes	Codewords	Explanation
IP length	**iplen**	The length of he current IP packet in 8 bytes
IP fragment	**ipseqplus**	The current IP packet's fragment offset > the previous one's
	ipseqzero	The current IP packet's fragment offset == the previous one's
	ipseqminus	The current IP packet's fragment offset < the previous one's
IP flag	**ipflags**	The decimal value of the current IP packet header's Flags field
IP ttl	**ipttlplus**	The current IP packet's ttl value > the previous one's
	ipttlminus	The current IP packet's ttl value < the previous one's
	ipttlzero	The current IP packet's ttl value == the previous one's
IP route option	**ipoptroute0**	The current IP packet contains Strict Source and Record Route(SSRR) option.
	ipoptroute1	The current IP packet contains Loose Source and Record Route (LSRR) option.
IP option	**ipbadoption**	The value of the current IP packet's Options field is invalid.
IP tos	**iptos**	The decimal value of the IP header's DSCP field, originally defined as tos.
TCP checksum	**tcpchecksum1**	The current TCP segment's Checksum is incorrect.
TCP length	**tcplen**	The length of the TCP segment in 8 bytes.
TCP segment	**tcpseqorder-overlap0**	In terms of sequence number, current TCP segment starts after the previous one.
	tcpseqorder-overlapplus1	Neither the current IP packet nor the previous contains a TCP segment.
	tcpseqorder-overlapplus2	The previous IP packet contains a TCP segment but the current packet does not.
	tcpseqorder-overlapplus3	The current IP packet contains a TCP segment but the previous packet does not.
	tcpseqorder-overlapminus1	The current TCP segment ends before or on the previous one.
	tcpseqorder-overlapminus2	The current TCP segment starts before and ends before or on the previous one.
	tcpseqorder-overlapminus3	The current TCP segment starts before and ends after the previous one.
	tcpseqorder-overlapminus4	The current TCP segment starts after or on, and ends before or on the previous one.
	tcpseqorder-overlapminus5	The current TCP segment starts after or on, and ends after the previous one.
TCP flag	**tcpflags**	The decimal value of the TCP header's Flags field.
TCP timestamp	**tcptimestampplus**	The current TCP segment's timestamp > the previous one's.
	tcptimestamp-zero	The current TCP segment's timestamp == the previous one's
	tcptimestamp-minus	The current TCP segment's timestamp < the previous one's
TCP sync	**tcpsyn**	The current TCP segment has set its SYN flag
TCP seq position	**tcpchaffseq**	The current TCP segment is not within the receiver's receive window.
	notcpchaffseq	The current TCP segment is within the receiver's receive window.
TCP mss	**tcpoptmss**	The current TCP segment contains MSS option.
TCP wscale	**tcpwscale**	The current TCP segment contains window scale option.
TCP acknowledgement	**tcpack**	The current TCP segment has set its ACK flag.

trace file to a codeword stream is listed in TABLE2. TABLE3 explains the correspondence between the packet attributes and codewords.

Table4 lists the complete correspondence between codewords and the intra&inter-packet attributes they code. For instance, codeword **iplen** codes the length of the current IP

TABLE 4. Correspondence between codewords and intra&inter packet attributes.

Codeword	Intra attribute	Inter attribute
iplen	Y	
ipseqplus		Y
ipseqzero		Y
ipseqminus		Y
ipflags	Y	
ipttlplus		Y
ipttlminus		Y
ipttlzero		Y
ipoptroute0	Y	
ipoptroute1	Y	
ipbadoption	Y	
iptos	Y	
tcpchecksum1	Y	
tcplen	Y	
tcpseqorder-overlap0		Y
tcpseqorder-overlapplus1		Y
tcpseqorder-overlapplus2		Y
tcpseqorder-overlapplus3		Y
tcpseqorder-overlapminus1		Y
tcpseqorder-overlapminus2		Y
tcpseqorder-overlapminus3		Y
tcpseqorder-overlapminus4		Y
tcpseqorder-overlapminus5		Y
tcpflags	Y	
tcptimestampplus		Y
tcptimestamp-zero		Y
tcptimestamp-minus		Y
tcpsyn	Y	
tcpchaffseq	Y	
notcpchaffseq	Y	
tcpoptmss	Y	
tcpwscale	Y	
tcpack	Y	

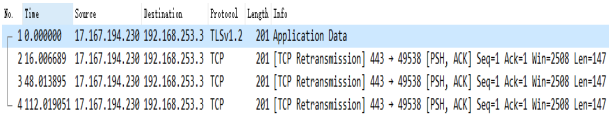


FIGURE 3. A TCP stream consisting of four packets.

packet which is one of the intra-packet attributes, so it's labeled 'Y' in the column "Intra attribute". **ipseqplus** is one of the inter-packet attributes, so it's labeled 'Y' in the column "Inter attribute".

The following is a codeword stream converted from an example TCP stream with four packets shown in FIGURE3.

iplen23 ipflags2 ipttlplus1 tcplen18 tcpseqorderoverlapplus3 tcpflags24 iptos0 tcpack. iplen23 ipseqzero ipflags2 ipttlzero tcplen18 tcpseqorderoverlapminus4 tcpflags24 iptos0 tcpack. iplen23 ipseqzero ipflags2 ipttlzero tcplen18 tcpseqorderoverlapminus4 tcpflags24 iptos0 tcpack. iplen23 ipseqzero ipflags2 ipttlzero tcplen18 tcpseqorderoverlapminus4 tcpflags24 iptos0 tcpack.

Converted codeword streams can be seen as text documents and can be further processed by NLP models. Other publications which employed NLP models for intrusion detection take a packet analyzer's (e.g. TShark) text output [36] or ready-made features [37], [38] from publicly available datasets (e.g. KDD CUP'99 and UNSW-NB15) as their source input text. They do not have a step for preparing the model's input. The proposed conversion syntax enables the transforming from raw binary network streams to text, and converts the network evasion behavior detection problem to a text classification task.

B. FEATURE EXTRACTION

Features are subsequently extracted from the codeword streams to generate samples in the form of numerical vectors. We borrowed the idea of TF-IDF (term frequency–inverse document frequency) in text mining. First all the unique codewords in all the streams are collected to get the codeword set. For each codeword we calculate its term frequency (tf) in each stream adjusted for the stream's length. Then we measure the information provided by each codeword with its inverse document frequency (idf) which is the logarithmically scaled inverse fraction of the streams that contain the codeword. The importance of each codeword for discriminating types of streams is taken as the multiplication of its tf and idf values, and is called tfidf. The detailed computation is as follows.

For codeword t_i, its term frequency $tf_{i,j}$ in stream s_j is

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

where $n_{i,j}$ is the number of occurrences of t_i within s_j and the denominator is the number of codewords in s_j. For codeword t_i, its inverse document frequency is

$$idf_i = log\frac{1+|S|}{1+|\{j : t_i \in s_j\}|} + 1 \tag{2}$$

where $|S|$ is the number of streams and $|\{j : t_i \in s_j\}|$is the number of the streams in which t_i occurs. For codeword t_i its importance in stream s_j is

$$tfidf_{i,j} = tf_{i,j} \times idf_i \tag{3}$$

If the number of unique codewords in all streams is denoted by $|U_C|$, for stream s_j we produce a vector $X^{(j)}$ of length $|U_C|$ whose ith element is

$$X_i^{(j)} = tfidf_{i,j} \tag{4}$$

And we normalize $X^{(j)}$ by

$$X^{(j)} = \frac{X^{(j)}}{\sqrt{X_0^{(j)2} + X_1^{(j)2} + \cdots + X_{|U_A|-1}^{(j)2}}} \tag{5}$$

The normalized $X^{(j)}$ is the feature vector of s_j. $X^{(j)}$ and the j_{th} stream's class label y_j constitute the j_{th} sample $\left(X^{(j)}, y_j\right)$ of the training set.

In Figure4, we demonstrate the ability of the extracted features to capture the intra&inter packet attribute difference between different types of streams. We randomly chose 300 normal TCP streams and 300 ip_frag streams, and drew a subset of their features comprised of only the codewords starting with "**iplen**" (e.g. **iplen**5, **iplen**6,etc). There are 187 different **iplen** codewords (from **iplen**5 to **iplen**188). So each

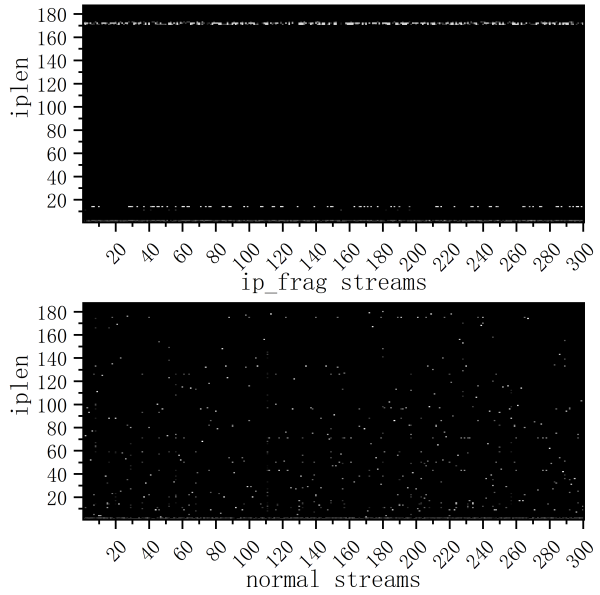


FIGURE 4. Images of iplen feature subset for 300 ip_frag and 300 normal streams showing different patterns.

image has 187 rows as shown in Figure4. The 300 streams' feature vectors form the columns of the images. It can be seen that the feature value distribution of ip_frag streams is different from that of normal streams. Supervised multi-class classifiers can be trained to differentiate the hidden patterns, and later be employed to classify unknown TCP streams.

In the cyber security domain the TF-IDF method has been employed to detect spam emails, phishing attempts in emails, exfiltration events in logs, malicious URL, and masquerade attacks, etc. Besides the TF-IDF there is the Recurrent AutoEncoder (RAE) [39] which also maps variable length text to fixed length vectors. The RAE is not considered in our study because it is intended for semantic understanding but not for classification. The TF-IDF has achieved notable success in document classification and that's why we chose it to extract features for evasion detection by codeword stream classification.

Table5 details the workflow of the proposed evasion detection algorithm. It consists of an offline training procedure and an online detection procedure. In the training procedure, the algorithm extracts normal TCP streams, generates evasion streams, computes codeword stream set, builds training dataset and trains the classifier. In the detection procedure, the algorithm computes the codeword stream of the TCP stream under test, generates the feature vector and determines its evasion type by the classifier's prediction of the generated feature vector.

If numeric features are extracted from each packet of a network flow, a TCP stream can be converted to a Multi-Dimensional Time Serie(MDT). And a multi-dimensional DTW [40] could be defined on these converted MDTs as a distance measure. Then the atomic network evasion detection would be a classification problem which could be solved by a minimal distance classifier. However, the number of packets in each TCP stream varies widely. Short TCP streams may consist of only four packets while long streams may contain several hundred packets. Due to the common constraints of DTW implementations (e.g. the Sakoe-Chiba Band constraint) [41], which limit the warping path within several cells from the diagonal of the warping matrix, the DTW may fail to find the ideal path that minimizes the warping cost when the numbers of two streams' packets are of significant difference. This may lead to a poor distance measure and affect the classification accuracy.

TABLE 5. Atomic network evasion detection and recognition algorithm.

Procedure train()

Input: The captured network traffics $\mathbb{T}$
Output: The codewords idf set $IDF = \{idf_i, i \in [0, |U_C|)\}$. The trained classifier H.

1: Extract TCP streams from $\mathbb{T}$ to get normal flows.
2: Apply atomic evasion techniques on the extracted TCP streams to get evasion flows. The normal and evasion flows constitute the TCP stream set $\mathbb{D}$.
3: Generate codeword stream set $\mathbb{S}$ from $\mathbb{D}$ according to the syntax in Table2. The element number of $\mathbb{S}$ is denoted by $|S|$.
4: Calculate each of the IDF's element idf_i according to (2), $i \in [0, |U_C|)$, $|U_C|$ is the number of unique codewords.
5: Compute $X^{(j)}$ for the jth codeword stream s_j in $\mathbb{S}$ according to (1)(3)(4)(5), $j \in [0, |S|)$, and build the training dataset $\{(X^{(j)}, y_j), j \in [0, |S|)\}$.
6: Train and test a multi-class classifier H with the training dataset built in step 5.

Procedure detect()

Input: The TCP stream under test d_u, the codewords idf set IDF. The trained classifier H.
Output: y_u – the evasion type of d_u.

1: Generate codeword stream s_u for d_u according to the syntax in Table2.
2: Calculate the feature vector $X^{(u)}$ of s_u according to (1)(3)(4)(5) using the input IDF.
3: Predict the class label y_u of $X^{(u)}$ by H.

IV. EXPERIMENTAL RESULTS

Although to our knowledge there are no published results on the atomic network evasion detection and recognition which we can compare our approach with, we give our dataset setup, detection and computation performance in this section in detail.

A. DATASET

We evaluated our model on a dataset consisting of 280,217 samples generated from normal TCP streams and 8 types of atomic network evasion streams. Every evasion sample was obtained by applying an evasion technique to the original TCP stream with specific option values. TABLE6 shows the different option values for the eight types of evasions in the dataset. For instance, twenty four samples of the ip_frag evasion may be generated from one normal stream by applying 24($3 \times 2 \times 4$) different value combinations of "order"

TABLE 6. Option values of different evasions in the dataset.

Evasion	Option values
ip_frag	order: random, none, reverse favor: new, old size: 8,16,24,32
tcp_seg	order: random, none, reverse favor: new, old size: 8,16,24,32
ip_opt	mode: lsrr, ssrr ptr: 4,8,12 ip_addr: 192.168.1.1, 192.168.2.2
tcp_opt	mode: mss, wscale size: 1,25,49,. . . ,241
tcp_chaff	cksum, null, paws, seq, syn, 1,2,3,4,5,6,7,8,9
ip_chaff	dup, opt, 1,2,3,4,5,6,7,8,9
ip_tos	1,25,49,. . . ,241
ip_ttl	+1, -1

TABLE 7. Sample numbers of different evasion types in the dataset.

Types of flows	Number of samples
normal	29922
tcp_chaff	26012
ip_frag	44736
tcp_seg	44244
ip_chaff	20691
ip_ttl	29922
ip_tos	20702
ip_opt	22584
tcp_opt	41404

(3 different values), "favor" (2 different values) and "size" (4 different values) options. And there may be 14 tcp_chaff samples derived from one normal stream by applying 14 different tcp_chaff options (cksum, null, paws, seq, syn, 1-9). For a detailed explanation of all these evasion options, please refer to the website of fragroute [35]. The numbers of samples for different evasion types in the dataset are listed in TABLE7. It is noted that not all value combinations of a particular evasion type's options work on all normal streams. Therefore the number of samples for each evasion type is not proportional to the number of the evasion options' value combinations.

The option value configurations in TABLE6 were chosen with consideration. Gorton [12] and Cheng [11] considered only the IP fragments and TCP segments of length less than or equal to 8 bytes in their test since larger sizes were unlikely for an evasion attempt to succeed. That is the reason why we chose the maximum size of ip_frag and tcp_seg to be 32. Some options' values such as the size in tcp_opt, ip_addr and ptr in ip_opt are insignificant because they produce the same codewords. Our previous experimental results indicated that those numeric option values in tcp_chaff, ip_chaff, ip_tos and ip_ttl didn't have significant impact on the recognition performance. They were chosen to cover the range of the options or to generate a balanced number of samples.

The total number of unique codewords in our experiment configuration is 1487, i.e. $|U_C| = 1487$. Each TCP stream is then represented as a sparse vector sample with 1487 elements.

B. DETECTOR PERFORMANCE

We trained a Support Vector Machine (SVM) and a Random Forest (RF) classifier respectively on our evasion stream dataset built in Section IV-A and employed cross-validation to evaluate the performance of evasion recognition. For the RF the size of the random subsets of features to consider when splitting a node was set to 38. And the number of trees was 200. The minimum number of samples in a leaf node was fixed to 50. Each training sample was weighted inversely proportional to its class frequencies in the dataset. For the SVM a linear kernel was used with the penalty parameter of the error term set to the reciprocal of class frequencies in the dataset. FIGURE5 shows the receiver operating characteristic (ROC) curves of our detector when using the SVM on one of the test datasets. Each curve indicates the performance of the detector for the normal network streams or one of the eight evasion types' streams. For a better view of the curves in the false positive rate (FP) range from 0 to 0.01, an enlarged

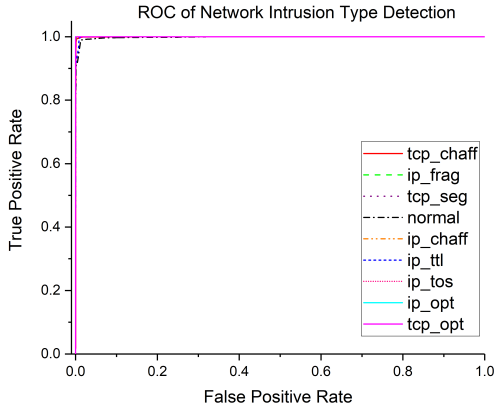


FIGURE 5. ROC of the proposed approach using SVM.

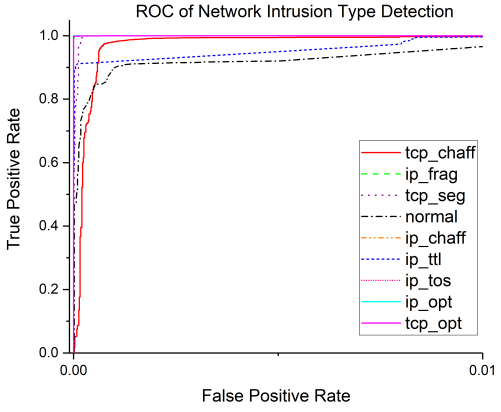


FIGURE 6. Partial enlarged view of ROC using SVM in the range from 0.0 to 0.01.

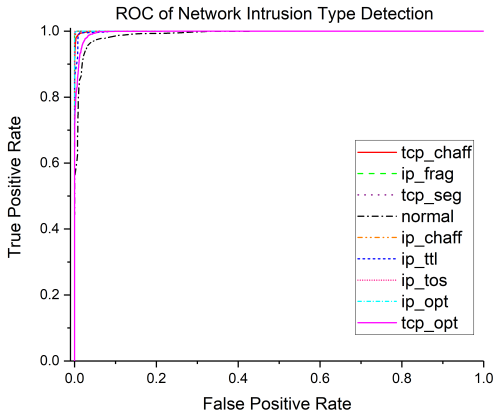


FIGURE 7. ROC of the proposed approach using RF.

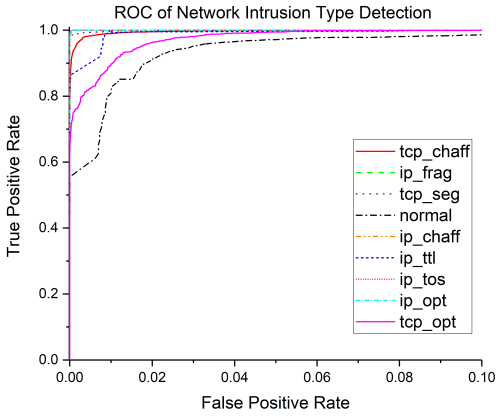


FIGURE 8. Partial enlarged view of ROC using RF in the range from 0.0 to 0.1.

partial view of FIGURE5 is shown in FIGURE6. It can be seen that when the FP is 0.01, the true positive rates (TP) of our model for all types are close to 1. FIGURE7 shows the ROC curves of our detector when using the RF on one of the test datasets. FIGURE8 is the enlarged partial view of FIGURE 7 with the FP ranging from 0 to 0.1. It is shown that when the FP is 0.1 the TP of our model for all types are close to 1. The performance measures of the SVM classier validated by cross validation are presented in TABLE8. The accuracies on the test sets via fivefold cross validation are [0.98888453, 0.98922314, 0.98968631, 0.98924021, 0.99027463] which is 0.9895(mean) +/−0.00053628(std). TABLE9 shows the performance measures of the RF classier validated by cross validation. The accuracies on the test sets via fivefold cross validation are [0.97182571, 0.9712012, 0.97111147, 0.97155755, 0.97230698] which is 0.9716(mean)+/−0.00048741(std). All cross validation experiments have been adjusted to employ stratified sampling during assembling of folds, which ensured equally balanced class distribution of each fold.

FIGURE9 and FIGURE10 show the confusion matrices of our approach using the two classifiers on one of the test sets in the five-fold cross validation. They both have a strong diagonal.

TABLE 8. Performance of the proposed approach using SVM.

	Precision	Recall	F1	Support
tcp_chaff	0.98	0.99	0.99	6497
iP_frag	1.00	1.00	1.00	11346
tcp_seg	1.00	1.00	1.00	11089
normal	0.99	0.91	0.95	7365
ip_chaff	1.00	1.00	1.00	5246
ip_ttl	0.93	0.99	0.96	7463
ip_tos	1.00	1.00	1.00	5187
ip_opt	1.00	1.00	1.00	5542
tcp_opt	1.00	1.00	1.00	10320
Avg	0.9889	0.9878	0.9889	

TABLE 9. Performance of the proposed approach using RF.

	Precision	Recall	F1	Support
tcp_chaff	0.97	0.97	0.97	6497
iP_frag	1.00	1.00	1.00	11346
tcp_seg	0.98	0.99	0.99	11089
normal	0.97	0.80	0.88	7365
ip_chaff	0.99	1.00	0.99	5246
ip_ttl	0.93	0.99	0.96	7463
ip_tos	0.99	1.00	0.99	5187
ip_opt	0.99	1.00	0.99	5542
tcp_opt	0.93	0.98	0.96	10320
Avg	0.9722	0.9700	0.9700	

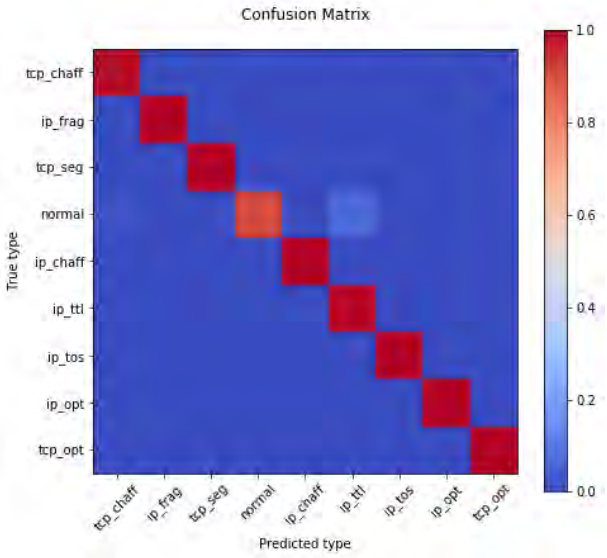


FIGURE 9. Confusion matrix of the proposed approach using SVM.

C. COMPUTATION PERFORMANCE

We do not compare our approach's resource consuming and computation performance with prior studies for several reasons. First of all, they work in different modes. As shown in Figure2 some prior techniques are inline (e.g. the traffic normalizer and Protocol Scrubbing) whereas our approach is not. Second, as shown in Table5 our approach consists of an offline training and an online detection procedures while Active Mapping does not have an offline stage. Finally, machine learning based approaches [23]–[26] gave only classification performance data, but no computation

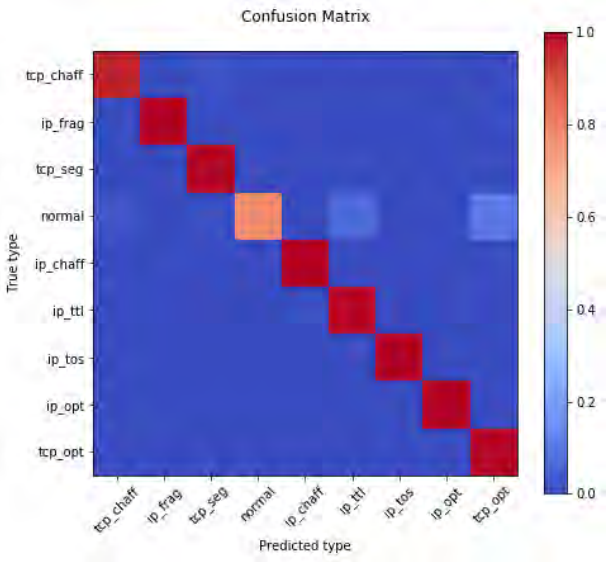


FIGURE 10. **Confusion matrix of the proposed approach using RF.**

performance data. We list available resource consuming and computation performance information of prior techniques here for completeness. The traffic normalizer was able to normalize a TCP traffic stream at 100,000 pkts/sec in a bidirectional 100Mb/sec environment using commodity PC hardware [14]. Active Mapping was reported to operate with a steady-state rate of about 5 seconds per host, producing additional traffic about 19 KB per host and memory footprint approximately 100 bytes per host [15]. Protocol Scrubbing was reported to achieve 322.3Mb/sec throughput while the crossover's throughput was 359.13Mb/sec under the same benchmark [16]. We evaluated our approach's computation performance in terms of the average number of streams processed per second by running the detection procedure in Table5 on the dataset consisting of 280,217 streams described in SectionIV-A for 10 times. Implemented in Python the detection procedure of our approach operated at a rate of 41617 streams per second when using the RF classifier, and 399 streams per second when using the linear SVM. The evaluation was carried out on a laptop with an Intel I5 CPU, 16GB memory and Windows 10 operating system.

V. CONCLUSION

Early studies on anti-evasion techniques focused on the elimination of ambiguities between an NIDS and endpoint hosts. Recent machine learning based approaches devoted to increasing their classifiers' resilience to evasions. And we proposed a method to detect and recognize network evasions by means of intra&inter packet behavior classification.

First we created a network evasion trace set by applying evasion techniques on normal TCP streams with different option value combinations. Then we built a network evasion dataset by converting TCP streams to codeword streams and extracting statistical features from the intra&inter packet attributes of the codeword streams. Based on this dataset we trained SVM and RF classifiers to differentiate different types of evasions from normal streams. Most notably, this is the first study to address the network evasion problem from the view of detection and recognition via machine learning techniques. Experiments showed that the proposed approach achieved an average recognition accuracy of 98.95%. Although this preliminary study has examined only the detection of some types of atomic network evasions, it indicates the possibility and potential advantages of machine learning techniques for evasion detection in application layers and even for the AET detection. And that will be our further research efforts.

REFERENCES

[1] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 305–316.
[2] J. Postel, "Dod standard transmission control protocol," Inf. Sci. Inst., Univ. Southern California, Marina del Rey, CA, USA, Tech. Rep. RFC: 761, 1980.
[3] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Secure Netw. Inc, Calgary, AB, Canada, Tech. Rep., 1998.
[4] D. Roelker, *1HTTP IDS Evasions Revisited*. Columbia, MD, USA: Sourcefire Inc., 2003.
[5] O. Nbou, "Detecting and modeling polymorphic shellcode," Ph.D. dissertation, Concordia Inst. Inf. Syst. Eng. (CIISE), Concordia Univ., Montreal, QC, Canada, 2010.
[6] J. M. Vidal, J. M. Castro, A. S. Orozco, and L. G. Villalba, "Evolutions of evasion techniques aigainst network intrusion detection systems," in *Proc. 6th Int. Conf. Inf. Technol.*, May 2013, pp. 1–6.
[7] M. Särelä, T. Kyöstilä, T. Kiravuo, and J. Manner, "Evaluating intrusion prevention systems with evasions," *Int. J. Commun. Syst.*, vol. 30, no. 16, p. e3339, Nov. 2017.
[8] Q. Xiong, Y. Xu, B.-F. Zhang, and F. Wang, "Overview of the evasion resilience testing technology for network based intrusion protecting devices," in *Proc. IEEE 18th Int. Symp. High Assurance Syst. Eng. (HASE)*, Jan. 2017, pp. 146–152.
[9] K. Majewski, *Advanced Evasion Techniques for Dummies*. Hoboken, NJ, USA: Wiley, 2014.
[10] M. Chammem, M. Hamdi, and T.-H. Kim, "Extending advanced evasion techniques using combinatorial search," in *Proc. 7th Int. Conf. Secur. Technol.*, Dec. 2014, pp. 41–46.
[11] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, "Evasion techniques: Sneaking through your intrusion detection/prevention systems," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 4, pp. 1011–1020, 4th Quart., 2012.
[12] A. Gorton and T. Champion, *Combining evasion techniques to avoid network intrusion detection systems*. North Chelmsford, MA, USA: Skaion Inc., 2004.
[13] A. Levomaki, O.-P. Niemi, and C. Jalio, "Automatic discovery of evasion vulnerabilities using targeted protocol fuzzing," in *Proc. Briefing, Black Hat Eur.*, 2017, pp. 1–4.
[14] C. Kreibich, M. Handley, and V. Paxson, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," in *Proc. USENIX Secur. Symp.*, 2001, pp. 1–18.
[15] U. Shankar and V. Paxson, "Active mapping: Resisting NIDS evasion without altering traffic," in *Proc. Symp. Secur. Privacy*, May 2003, pp. 44–61.
[16] D. Watson, M. Smart, G. R. Malan, and F. Jahanian, "Protocol scrubbing: Network security through transparent flow modification," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 261–273, Apr. 2004.
[17] M. Roesch, "Snort: Lightweight intrusion detection for networks," *Lisa*, vol. 99, no. 1, pp. 229–238, Nov. 1999.
[18] S. Pastrana, A. Orfila, and A. Ribagorda, "A functional framework to evade network IDS," in *Proc. 44th Hawaii Int. Conf. Syst. Sci.*, Jan. 2011, pp. 1–10.
[19] M. Vutukuru, H. Balakrishnan, and V. Paxson, "Efficient and robust TCP stream normalization," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2008, pp. 96–110.
[20] J. Novak, "Target-based fragmentation reassembly," Sourcefire, Columbia, MD, USA, Tech. Rep., 2005.
[21] J. Novak and S. Sturges, "Target-based TCP stream reassembly," Sourcefire, Columbia, MD, USA, vol. 3, Aug. 2007, pp. 1–23.

[22] V. Bukac, "IDS system evasion techniques," M.S. thesis, Masarykova Univ., Brno, Czech Republic, 2010.
[23] F. Zhang, P. P. K. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 766–777, Mar. 2016.
[24] I. C. Anindya and M. Kantarcioglu, "Adversarial anomaly detection using centroid-based clustering," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, Jul. 2018, pp. 1–8.
[25] Z. Katzir and Y. Elovici, "Quantifying the resilience of machine learning classifiers used for cyber security," *Expert Syst. Appl.*, vol. 92, pp. 419–429, Feb. 2018.
[26] I. Homoliak, M. Teknos, M. Ochoa, D. Breitenbacher, S. Hosseini, and P. Hanacek, "Improving network intrusion detection classifiers by non-payload-based exploit-independent obfuscations: An adversarial approach," 2018, *arXiv:1805.02684*. [Online]. Available: https://arxiv.org/abs/1805.02684
[27] Y. Hamid, M. Sugumaran, and V. R. Balasaraswathi, "IDS using machine learning-current state of art and future directions," *Brit. J. Appl. Sci. Technol.*, vol. 15, no. 3, pp. 1–22, 2016.
[28] H. Du and S. J. Yang, "Probabilistic modeling and inference for obfuscated cyber attack sequences," 2018, *arXiv:1809.01562*. [Online]. Available: https://arxiv.org/abs/1809.01562
[29] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," 2019. *arXiv:1903.02460*. [Online]. Available: https://arxiv.org/abs/1903.02460
[30] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Comput. Netw.*, vol. 34, no. 4, pp. 579–595, 2000.
[31] S. Hettich, and S. D. Bay. (1999). *The UCI KDD Archive*. [Online]. Available: http://kdd.ics.uci.edu
[32] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Jul. 2009, pp. 1–6.
[33] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, 2012.
[34] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proc. 6th Int. Conf. (CoNEXT)*, Nov. 2010, p. 8.
[35] M. Holstein. (2002). *How does Fragroute Evade NIDS Detection?* Accessed: Oct. 10, 2015. [Online]. Available: https://www.sans.org/security-resources/idfaq/fragroute.php
[36] M. Mimura and H. Tanaka, "Reading network packets as a natural language for intrusion detection," in *Proc. Int. Conf. Inf. Secur. Cryptol.* Cham, Switzerland: Springer, 2017, pp. 339–350.
[37] R. S. M. Carrasco and M.-A. Sicilia, "Unsupervised intrusion detection through skip-gram models of network behavior," *Comput. Secur.*, vol. 78, pp. 187–197, Sep. 2018.
[38] X. Zhuo, J. Zhang, and S. W. Son, "Network intrusion detection using word embeddings," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 4686–4695.
[39] T. Wong and Z. Luo, "Recurrent auto-encoder model for multidimensional time series representation," in *Proc. ICLR*, Feb. 2018, pp. 1–13.
[40] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, and E. Keogh, "Generalizing dtw to the multi-dimensional case requires an adaptive approach," *Data Mining Knowl. Discovery*, vol. 31, no. 1, pp. 1–31, 2017.
[41] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping," *ACM Trans. Knowl. Discovery Data (TKDD)*, vol. 7, no. 3, p. 10, Sep. 2013.

JIA JINGPING received the M.S. and Ph.D. degrees in computer science from Northwestern Polytechnical University, Xi'an, China, in 2004 and 2006, respectively.

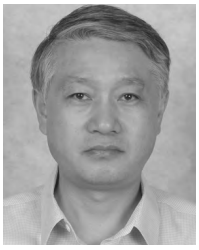
From 2007 to 2009, he was a Postdoctoral Fellow with Peking University. He started working at the School of Control and Computer Engineering, North China Electric Power University, in 2009. His research interests include machine learning, network security, and computer vision.

CHEN KEHUA received the B.S. and M.S. degrees in computer science from North China Electric Power University, Beijing, China, in 2016 and 2019, respectively.

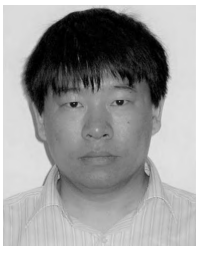
His research interests include machine learning and network security.

CHEN JIA received the B.S. degree from Sun Yat-sen University, Guangzhou, China, in 2006, and the M.S. degree from Peking University, Beijing, China, in 2010.

After 5 years work in Xinhua News Agency of New Media GIS, he is currently with China Communications Asset Management Company Ltd., as a Senior Engineer and an Information System Project Manager. His research interests include enterprise process and information management, GIS, remote sensing, intelligent transportation, traffic environment and safety, traffic big data mining, and traffic economy.



ZHOU DENGWEN is currently a Professor with the School of Control and Computer Engineering, North China Electric Power University, China. His research interests include image processing, computer vision, and machine learning.



MA WEI received the B.S. degree in electronic engineering from Xidian University, Xi'an, China, in 1994, and the M.S. degree in computer science from North China Electric Power University, Beijing, China, in 1997.

He is currently with the School of Control and Computer Engineering, North China Electric Power University. His research interest includes machine learning.