Hindawi
Security and Communication Networks
Volume 2020, Article ID 8826038, 15 pages
https://doi.org/10.1155/2020/8826038

WILEY | Hindawi

Research Article

BLATTA: Early Exploit Detection on Network Traffic with Recurrent Neural Networks

Baskoro A. Pratomo,[1,2] **Pete Burnap**,[1] **and George Theodorakopoulos**[1]

[1]*School of Computer Science and Informatics, Cardiff University, Cardiff, UK*
[2]*Informatics Department, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia*

Correspondence should be addressed to Baskoro A. Pratomo; me@baskoroadi.web.id

Received 8 April 2020; Revised 25 June 2020; Accepted 9 July 2020; Published 4 August 2020

Academic Editor: Muhammad Faisal Amjad

Copyright © 2020 Baskoro A. Pratomo et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Detecting exploits is crucial since the effect of undetected ones can be devastating. Identifying their presence on the network allows us to respond and block their malicious payload before they cause damage to the system. Inspecting the payload of network traffic may offer better performance in detecting exploits as they tend to hide their presence and behave similarly to legitimate traffic. Previous works on deep packet inspection for detecting malicious traffic regularly read the full length of application layer messages. As the length varies, longer messages will take more time to analyse, during which time the attack creates a disruptive impact on the system. Hence, we propose a novel early exploit detection mechanism that scans network traffic, reading only 35.21% of application layer messages to predict malicious traffic while retaining a 97.57% detection rate and a 1.93% false positive rate. Our recurrent neural network- (RNN-) based model is the first work to our knowledge that provides early prediction of malicious application layer messages, thus detecting a potential attack earlier than other state-of-the-art approaches and enabling a form of early warning system.

1. Introduction

Exploits are attacks on systems that take advantage of the existence of bugs and vulnerabilities. They infiltrate the system by giving the system an input which triggers malicious behaviour. As time passes, the number of bugs and vulnerabilities increases, along with the number of exploits. In the first quarter of 2019, there were 400,000 new exploits [1], while more than 16 million exploits have been released in total. Exploits exist in most operating systems (OSs); hence, detecting exploits early is crucial to minimise potential damage.

By exploiting a vulnerability, attackers can, for example, gain access to remote systems, send a remote exploit, or escalate their privilege on a system. Exploits-DB [2] is a website that archives exploits, both remote and local ones. The number of existing remote exploits on the website is almost double that of the local ones, which suggests remote exploits are more prevalent. No physical access to the system is required to execute a remote exploit; thus, the attack can be launched from anywhere in the world.

Remote exploits normally carry a piece of code as a payload, which will be executed once a vulnerability has been successfully exploited. An exploit is analogous to a tool for breaking into a house, and its payload is something the burglar would do once they are inside the house. Without this payload, exploits would be merely a tool to demonstrate that an application is vulnerable. Exploit payloads may take many forms; they could be written in machine code, a server-side scripting language (e.g., PHP, Python, and Ruby), or OS specific commands (e.g., Bash).

One way to detect exploits is to scan network traffic for their presence. In doing this, the exploit can be detected before it arrives at the vulnerable system. If this is achieved, earlier action can be taken to minimise or nullify the damage. There is also no need to run the exploit in a clone server or virtual machine (VM) to be analysed—as it is usually the case in host-based detection approaches, making this approach more time efficient to block and provide rapid response to attacks. Therefore, detecting exploits in network traffic is a promising way to prevent remote exploits from infecting protected systems.

Detecting exploits on the wire has challenges: firstly, processing the vast amount of data without decreasing network throughput below acceptable levels; quality of service is still a priority. Secondly, there are various ways to encode exploit payloads [3], by modifying the exploit payload to make it appear different, yet still achieve the same goal. This technique makes it easy to evade any rule-based detection. Lastly, encrypted traffic is also a challenge; attackers may transmit the exploit with an encrypted protocol, e.g., HTTPS.

There are many ways to detect exploits in network traffic. Rule-based detection systems work by matching signatures of known attacks to the network traffic. Anything that matches the rule is deemed malicious. The most prevalent open-source intrusion detection system, Snort [4], has a rule that marks any traffic which contains byte 0×90 as shellcode-related traffic. This rule is based on the knowledge that most x86-based shellcodes are preceded by a sequence of *no operation* (NOP) instructions in which the bytes normally contain this value. However, this rule can easily be evaded by employing other NOP instructions, such as the "0×41 0×49" sequence. Apart from that, rule-based detection systems are susceptible to zero-day attacks for which no detection rule exists. Such systems are unable to detect these attacks until the rule database is updated with the new attack signature.

Machine learning (ML) algorithms are capable of classifying objects and artefacts based on features exhibited in data and handle various modalities of input. ML has been successfully applied in many domains with a high success rate, such as image classification, natural language processing, speech recognition, and even intrusion detection system. There has been much research on implementing machine learning to address network intrusion detection [5]. Researchers typically provide training examples of malicious and legitimate traffic to the ML algorithm that can then be used to determine whether new (unseen) traffic is malicious. However, there are three key limitations with existing research: firstly, most of the research uses old datasets, e.g., either KDD99 or DARPA99 [6]. The traffic in those datasets may not represent recent network traces since network protocols have evolved during these years, as have the attacks. Secondly, many of the previous works focus solely on the header information of network packets and process packets individually [6]. Yet, it is known that exploits may exhibit similar statistical attributes to legitimate traffic at a header-level and use evasion techniques such as packet fragmentation to hide their existence [3]. Therefore, we argue that network payload features may capture exploits better, and this area is still actively expanding as shown by the number of research mentioned in Table 1. This argument brings us to the third limitation: existing methods that use payload features, i.e., byte frequencies or *n*-grams, usually involve reading the payload of whole application layer messages (see Section 2). The issue is that these messages can be lengthy and spread over multiple network packets. Reading the whole messages before making a decision may lead to a delay in detecting the attack and gives the exploit time to execute before an alert is raised.

We, therefore, propose Blatta, an early exploit detection system which reads application layer messages and predicts whether these messages are likely to be malicious by reading only the first few bytes. This is the first work to our knowledge that provides early prediction of malicious application layer messages, thus detecting a potential attack earlier than other state-of-the-art approaches and enabling a form of early warning system. Blatta utilises a recurrent neural network- (RNN-) based model and *n*-grams to make the prediction. An RNN is a type of artificial neural networks that takes a sequence-based vector as input—in this case, a sequence of *n*-grams from an application layer message—and considers the temporal behaviour of the sequence so that they are not treated independently. There has been a limited amount of research on payload-based intrusion detection which used an RNN or its further development (i.e., long short-term memory [24, 25] and gated recurrent unit [20]), but earlier research did not make predictive decisions early as they only used 1-grams of full-length application layer messages as features, which lacks contextual information—a key benefit of higher-order *n*-grams. Other work that does consider higher-order sequences of *n*-grams (e.g. [12, 29]) is also yet to develop methods that provide early-stage prediction, preferring methods that require the full payload as input to a classifier.

To evaluate the proposed system, we generated an exploit traffic dataset by running exploits in Metasploit [30] with various exploit payloads and encoders. An exploit payload is a piece of code that will be executed once the exploit has successfully infiltrated the system. An encoder is used to change how the exploit payload appears while keeping its functionality. Our dataset contains traffic from 5,687 working exploits. Apart from that, we also used a more recent intrusion detection dataset, UNSW-NB15 [31], thus enabling our method to be compared with previous works.

To summarise, the key contributions of this paper are as follows:

(i) Proposed an early prediction of exploit traffic, Blatta, using a novel approach of using an RNN and high-order *n*-grams to make predictive decisions while still considering temporal behaviour of a sequence of *n*-grams. Blatta is the first, to the best of our knowledge, who introduces early exploit detection. Blatta detects exploit payloads as they enter the protected network and is able to predict the exploit by reading 35.21% of application layer messages on average. Blatta thus enables actions to be taken to block the attack earlier and therefore reduce the harm to the system.

(ii) Generated a dataset of exploit traffic with various exploit payloads and encoders, resulting in 5,687 unique connections. The exploit traffic in the dataset was ensured to contain actual exploit code, making the dataset closer to reality.

The rest of this paper is structured as follows: Section 2 gives a summary of previous works in this area. Section 3

Table 1: Related works in exploit detection. Unlike previous works, Blatta does not have to read until the end of application layer messages to detect exploit traffic.

Paper	Features	Detection method	Dataset(s)	Learning type	Protocol(s)	Early prediction
PAYL [7]	Relative frequency count of each 1-gram	Based on statistical model and Mahalanobis distance	D, SG	U	HTTP, SMTP, SSH	No
RePIDS [8]	Mahalanobis distance map which is originated from relative frequency count of each 1-gram, filtered by PCA.	Based on statistical model and Mahalanobis distance	D, M	U	HTTP	No
McPAD [9]	2ν-grams	Multi one-class SVM classifier	D, M	U	HTTP	No
HMMPayl [10]	Byte sequences of the L7 payload.	Ensemble of HMMs	D, M, DI	U	HTTP	No
Oza et al. [11]	Relative frequency count of each 1-gram.	Based on statistical model	D, M, SG	U	HTTP	No
OCPAD [12]	High-order n-grams ($n > 1$).	Based on the occurrence probability of an n-grams in a packet	M, SG	U	HTTP	No
Bartos et al. [13]	Information from HTTP request headers and the lengths	SVM	SG	S	HTTP	No
Zhang et al. [14]	Packet header information and HTTP and DNS messages	Naïve Bayes, Bayesian network, SVM	D, SG	S	DNS, HTTP	No
Decanter [15]	HTTP messages	Clustering	SG	U	HTTP	No
Golait and Hubbali [16]	Byte sequence of the L7 payload	Probabilistic counting deterministic timed automata	SG	U	SIP	No
Duessel et al. [17]	Contextual n-grams of the L7 payload	One-class SVM	SG	U	HTTP, RPC	No
Min et al. [18]	Words of the L7 payload	CNN and random forest	I	S	HTTP	No
Jin et al. [19]	2ν-grams	Multi one-class SVM classifier	M	U	HTTP	No
Hao et al. [20]	Byte sequence of the L7 payload	Variant gated recurrent unit	I	S	HTTP	No
Schneider and Bottinger [21]	Byte sequence of the L7 payload	Stacked autoencoder	O	U	Modbus	No
Hamed et al. [22]	n-grams of base64-encoded payload	SVM	I	S	All protocols in the datasets	No
Pratomo et al. [23]	Byte frequency of application layer messages	Outlier detection with deep autoencoder	SW	U	HTTP, SMTP	No
Qin et al. [24]	Byte sequence of the L7 payload	Using a recurrent neural network	O	S	HTTP	No
Liu et al. [25]	Byte sequence of the L7 payload	Using a recurrent neural network with embedded vectors	D, O	S	HTTP	No
Zhao and Ahn [26]	Disassembled instructions of bytes in network traffic	Employing Markov chain-based model and SVM	SG	S	Not mentioned	No
Shabtai et al. [27]	n-grams of a file and n-grams of opcodes in a file, then calculated TF/IDF of those n-grams	Various ML algorithm, e.g., random forest, decision tree, Naïve Bayes, and few others	SG	S	File classification	No
SigFree [28]	Disassembled instructions of bytes in application layer payload	Analyses of instruction sequences to determine if they are code	SG	Non-ML	HTTP	No
Proposed approach	High-order n-grams of application layer messages	Uses of recurrent neural network to early predict exploit traffic	SW, SG	S	HTTP, FTP	Yes

D = DARPA99; M = McPAD attacks dataset [9]; I = ISCX 2012; SG = self-generated; DI = DIEE; SW = UNSW-NB15; O = others; U = unsupervised; S = supervised; non-ML = non-machine learning approach.

explains the datasets we used to test our proposed method. How Blatta works is explained in Section 4. Then, Section 5 explains our extensive experimentation with Blatta. We also discuss possible evasion techniques to our approach in Section 6. Finally, the paper concludes in Section 7.

2. Related Works

The earliest solution to detecting exploit activities used pattern matching and regular expression [4]. Rules are defined by system administrators and are then applied to the

network traffic to identify a match. When a matching pattern is found, the system raises an alert and possibly shows which part of the traffic matches the rule. The disadvantage of this approach is that the rules must be kept up to date. Any variation to the exploit payload which is done by using encoders may defeat such detection.

ML approaches are capable of recognising previously seen instances of objects, and such methods aim to generalise to unseen objects. ML is able to learn patterns or behaviour from features present in training data and classify which class an unseen object belongs to. However, to work, suitable hand-crafted features must be engineered for the ML algorithm to make an accurate prediction. Some previous works defined their feature set by extracting information from application layer message. In [14] and [15], the authors generated their features from HTTP request URI and HTTP headers, i.e., host, constant header fields, size, user-agent, and language. The authors then clustered the legitimate traffic based on those features. Golait and Hubballi [16] tracked DNS and HTTP traffic and calculated pairwise features of two events to see their similarity. These features were obtained from the transport and application layer. One of their features is the semantical similarity between two HTTP requests.

Features derived from a specific protocol may capture specific behaviour of legitimate traffic. However, this feature extraction method has a drawback. A different set of features is needed for every application layer protocol that might be used in the network. Some research borrows the feature extraction method from natural language processing problems to have protocol-agnostic features, using *n*-grams.

One of the first leading research in payload analysis is PAYL [7]. It extracts 1-grams from all bytes of the payload as a representation of the network traffic and is trained over a set of those 1 grams. PAYL [7] measures the distance between the new incoming traffic with the model with a simplified Mahalanobis distance. Similar to PAYL, Oza et al. [11] extracts *n*-grams of HTTP traffic with various *n* values. They compared three different statistical models to detect anomalies/attacks. HMMPayl [10] is another work based on PAYL which uses Hidden Markov Models to detect anomalies. OCPAD [12] stores the *n*-grams of bytes in a probability tree and uses one-class Naïve Bayes classifier to detect malicious traffic. Hao et al. [20] proposed an autoencoder model to detect anomalous low-rate attacks on the network, and Schneider and Bottinger [21] used stacked autoencoders to detect attacks on industrial control systems. Hamed et al. [22] developed probabilistic counting deterministic timed automata which inspects byte values of application layer messages to identify attacks on VOIP applications. Pratomo et al. [23] extract words from the application layer message and detect web-based attacks with a combination of convolutional neural network (CNN) and random forest. A common approach that is found on all of the aforementioned works is they read all bytes in each packet or application layer message and do not decide until all bytes have been read, at which point it is possible the exploit has already infected the system and targeted the vulnerability.

Other research studies argue that exploit traffic is most likely to contain shellcode, a short sequence of machine code. Therefore, to get a better representation of exploit traffic, they performed static analysis on the network traffic. Qin et al. [24] disassemble byte sequences to opcode sequences and calculate probabilities of opcode transition to detect shellcode presence. Liu et al. [25] utilise *n*-grams that comprise machine instructions instead of bytes. SigFree [28] detects buffer overflow attempts on Internet services by constructing an instruction flow graph for each request and analysing the graph to determine if the request contains machine instructions. Any network traffic that contains valid machine instructions is deemed malicious. However, none of these consider that an exploit may also contain server-side scripting language or OS commands instead of shellcode.

In this paper, we proposed Blatta, an exploit attack detection system that (i) provides early detection of exploits in network traffic rather than waiting until the whole payload has been delivered, enabling proactive blocking of attacks, and (ii) is capable of detecting payloads that include malicious server-side scripting languages, machine instructions, and OS commands, enhancing the previous state-of-the-art approach that only focuses on shellcode. The summary of the proposed method and previous works is also shown in Table 1.

Blatta utilises a recurrent neural network- (RNN-) based model which takes a sequence of *n*-grams from network payload as the input. There has been limited research on developing RNNs to analyse payload data [20, 24, 25]. All of these works feed individual bytes (1 grams) from the payload as the features to their RNN model. However, 1 grams do not carry information about context of the payload string as a sequence of activities. Therefore, we model the payload as high-order *n*-grams where $n > 1$ as to capture more contextual information about the network payload. We directly compare 1-grams to higher-order *n*-grams in our experiments. Moreover, while related works such as [19], [17], and OCPAD [12] previously used high-order *n*-grams, they did not utilise a model capable of learning sequences of activities and thus not capable of making early-stage predictions within a sequence. Our novel RNN-based model will consider the long-term dependency of the sequence of *n*-grams.

An RNN-based model normally takes a sequence as input, processes each item sequentially, and outputs the decision after it has finished processing the last item of the sequence. The earlier works which used the RNN has this behaviour [24, 25]. While for Blatta to be able to early predict the exploit traffic, it takes the intermediate output of the RNN-based model, not waiting for full-length message to be processed. Our experiments show that this approach has little effect to accuracy and enables us to make earlier network attack predictions while retaining high accuracy and a low false positive rate.

3. Datasets and Threat Model

Several datasets have been used to evaluate network-based intrusion detection systems. DARPA released the KDD99 Cup, IDSEVAL 98, and IDSEVAL 99 datasets [32]. They

have been widely used over time despite some criticism that it does not model the attacks correctly [33]. The Lawrence Berkeley National Laboratory released their anonymised captured traffic [34] (LBNL05). More recently, Shiravi et al. released the ISCX12 dataset [35] in which they collected seven-day worth of traffic and provided the label for each connection in XML format. Moustafa and Slay published the UNSW-NB15 dataset [31] which had been generated by using the IXIA PerfectStorm tool for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviours. This dataset provides PCAP files along with the preprocessed traffic obtained by Bro-IDS [36] and Argus [37].

Moustafa and Slay [31] captured the network traffic in two days, on 22 January and 17 February 2015. For brevity, we refer to those two parts of UNSW-NB15 as UNSW-JAN and UNSW-FEB, respectively. Both days contain malicious and benign traffic. Therefore, there has to be an effort to separate them if we would like to use the raw information from the PCAP files, not the preprocessed information written in the CSV files. The advantage of this dataset over ISCX12 is that UNSW-NB15 contains information on the type of attacks and thus we are able to select which kind of attacks are needed, i.e., exploits and worms. However, after analysing this dataset in depth, we observed that the exploit traffic in the dataset is often barely distinguishable from the normal traffic as some of them do not contain any exploit payload. Our explanation for this is that exploit attempts may not have been successful, thus they did not get to the point where the actual exploit payload was transmitted and recorded in PCAP files. Therefore, we opted to generate our exploit traffic dataset, the BlattaSploit dataset.

3.1. BlattaSploit Dataset. To develop an exploit traffic dataset, we set up two virtual machines that acted as vulnerable servers to be attacked. The first server was installed with Metasploitable 2 [38], a vulnerable OS designed to be infiltrated by Metasploit [30]. The second one was installed with Debian 5, vulnerable services, and WordPress 4.1.18. Both servers were set up in the victim subnet while the attacker machine was placed in a different subnet. These subnets were connected with a router. The router was used to capture the traffic as depicted in Figure 1. Although the network topology is less complex than what we would have in the real-world, this setup still generates representative data as the payloads are intact regardless of the number of hops the packets go through.

To launch exploits on the vulnerable servers, we utilised Metasploit, an exploitation tool which is normally used for penetration testing [30]. Metasploit has a collection of working exploits and is updated regularly with new exploits. There are around 9,000 exploits for Linux- and Unix-based applications, and to make the dataset more diverse, we also employed different exploit payloads and encoders. An exploit payload is a piece of code which is intended to be executed on the remote machine, and an encoder is a technique to modify the appearance of particular exploit code to avoid signature-based detection. Each exploit in Metasploit has its own set of compatible exploit payloads and encoders. In this paper, we used all possible combinations of exploit payloads and encoders.

We then ran the exploits against the vulnerable servers and captured the traffic using tcpdump. Traffic generated by each exploit was stored in an individual PCAP file. By doing so, we know which specific type of exploit the packets belonged to. Timestamps are normally used to mark which packets belong to a class, but this information would not be reliable since packets may not come in order, and if there is more than one source sending traffic at a time, their packet may get mixed up with other sources.

When analysing the PCAP files, we found out that not all exploits had run successfully. Some of them failed to send anything to the targeted servers, and some others did not send any malicious traffic, e.g., sending login request only and sending requests for nonexistent files. This supported our earlier thoughts on why the UNSW-NB15 dataset was lacking exploit payload traffic. Therefore, we removed capture files that had no traffic or too little information to be distinguished from normal traffic. In the end, we produced 5,687 PCAP files which also represent the number of distinct sets of exploit, exploit payloads, and encoders. Since we are interested in application layer messages, all PCAP files were preprocessed with tcpflow [39] to obtain the application layer message for each TCP connection. The summary of this dataset is shown in Table 2.

The next step for this exploit traffic dataset was to annotate the traffic. All samples can be considered malicious; however, we decided to make the dataset more detailed by adding the type of exploit payload contained inside the traffic and the location of the exploit payload. There are eight types of exploit payload in this dataset. They are written in JavaScript, PHP, Perl, Python, Ruby, Shell script (e.g., Bash and ZSH), SQL, and byte code/opcode for shellcode-based exploits.

There are some cases where an exploit contains an exploit payload "wrapped" in another scripting language. For example, a Python script to do reverse shell connection which uses the Bash *echo* command at the beginning. For these cases, the type of the exploit payload is the one with the longest byte sequence. In this case, the type of the particular connection is Python.

It is also important to note that whilst the vulnerable servers in our setup use a 7-year-old operating system, the payload carried by the exploit was the identical payload to a more recent exploit would have used. For example, both CVE-2019-9670 (disclosed in 2019) and CVE-2012-1495 (disclosed in 2012) can use generic/shell_bind_tcp as a payload. The traffic generated by both exploits will still be similar. Therefore, we argue that our dataset still represent the current attacks. Moreover, only successful exploit attacks are kept in the dataset, making the recorded traffic more realistic.

3.2. Threat Model. A threat model is a list of potential things that may affect protected systems. Having one means we can identify which part is the focus of our proposed approach;

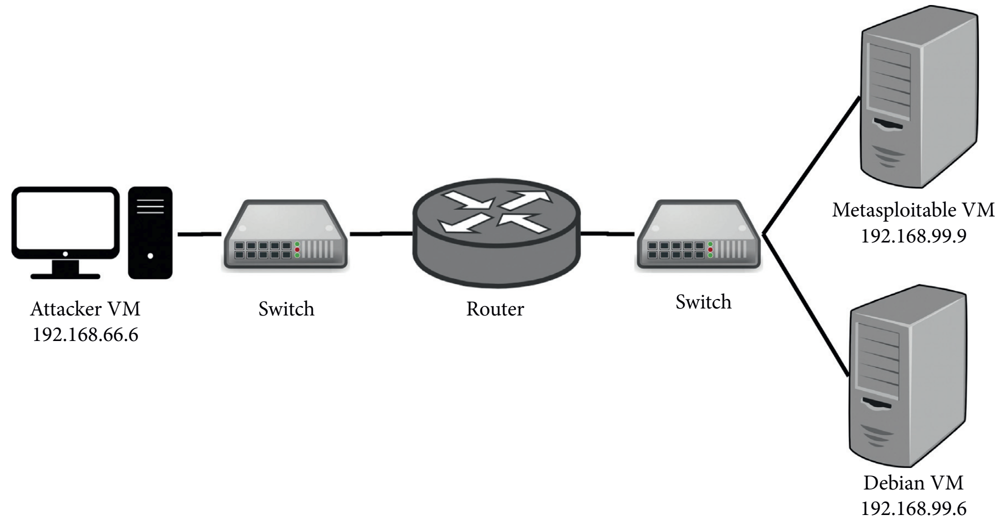

Figure 1: Network topology for generating exploit traffic. Attacker VM running Metasploit and target VMs are placed in different network connected by a router. This router is used to capture all traffic from these virtual machines.

Table 2: A summary of exploits captured in the BlattaSploit dataset.

Number of TCP connections	5687
Protocols	HTTP (3857), FTP (6), SMTP (74), POP3 (93)
Payload types	JavaScript, shellcode, Perl, PHP, Python, Ruby, Bash, SQL

Note. The numbers next to the protocols are the number of connections in the application layer protocols.

thus, in this case, we can potentially understand better what to look for in order to detect the malicious traffic and what the limitations are.

The proposed method focuses on detecting remote exploits by reading application layer messages from the unencrypted network traffic, although the detection method of Blatta can be incorporated with application layer firewalls, i.e., web application firewalls. Therefore, we can still detect the exploit attempt before it reaches the protected application. In general, the type of attacks we consider here are as follows:

(1) Remote exploits to servers that send malicious scripting languages (e.g., PHP, Ruby, Python, or SQL), shellcode, or Bash scripts to maintain control to the server or gained access to it remotely. For example, the apache_continuum_cmd_exec exploit with reverse shell payload will force the targeted server to open a connection to the attacking computer and provide shell access to the attacker. By focusing on the connections directed to servers, we can safely assume JavaScript code in the application layer message could also be malicious since normally JavaScript code is sent from the server to the client, not the other way around.

(2) Exploit attacks that utilise one of the text-based protocols over TCP, i.e., HTTP and FTP. Text-based protocols tend to be more well-structured; therefore, we can apply natural language processing based approach. The case would be similar to document classification.

(3) Other attacks that may utilise remote exploits are also considered, i.e., worms. Worms in the UNSW-NB15 dataset contain exploit code used to propagate themselves.

4. Methodology

Extracting features from application layer messages is the first step toward an early prediction method. We could use variable values in the message (e.g., HTTP header values, FTP commands, and SMTP header values), but it would require us to extract a different set of features from each application layer protocol. It is preferable to have a generic feature set which applies to various application layer protocols. Therefore, we proposed a method by using n-grams to model the application layer message as they carry more information than a sequence of bytes. For instance, it would be difficult for a model to determine what a sequence of bytes "G," "E," "T," space, "/," and so on means as those individual bytes may appear anywhere in a different order. However, if the model has 3-grams of the sequence (i.e., "GET," "ET," "T," and so on), the model would learn something more meaningful such as that the string could be an HTTP message. The advantage of using high-order n-grams where $n > 1$ is also shown by Anagram [29], method in [40], and OCPAD [12]. However, these works did not consider the temporal behaviour of a sequence of n-grams as their model was not capable of doing that. Therefore, Blatta utilised a recurrent neural network (RNN) to analyse the sequence of n-grams which were obtained from an application layer message.

An RNN takes a sequence of inputs and processes them sequentially in several time steps, enabling the model to learn the temporal behaviour of those inputs. In this case, the input to each time step is an n-gram, unlike earlier works which also utilised an RNN model but took a byte value as the input to each RNN time step [24, 25]. Moreover, these works took the output from the last time step to make decision, while our novel approach produces classification outputs at intermediate intervals as the RNN model is already confident about the decision. We argue that this approach will enable the proposed system to predict whether a connection is malicious without reading the full length of application layer messages, therefore providing an early warning method.

In general, as shown in Figure 2, the training and detection process of Blatta are as follows.

4.1. Training Stage. n-grams are extracted from application layer messages. l most common n-grams are stored in a dictionary. This dictionary is used to encode an n-gram to an integer. The encoded n-grams are then fed into an RNN model, training the model to classify whether the traffic is legitimate or malicious.

4.2. Detection Stage. For each new incoming TCP connection directed to the server, we reconstruct the application layer message, obtain a full-length or partial bytes of them, and determine if the sequence belongs to malicious traffic.

4.3. Data Preprocessing. In well-structured documents such as text-based application layer protocols, byte sequences can be a distinguishing feature that makes each message in their respective class differ from each other. Blatta takes the byte sequence of application layer messages from network traffic. The application layer messages need to be reconstructed as the message may be split into multiple TCP segments and transmitted in an arbitrary order. We utilise tcpflow [39] to read PCAP files, reconstruct TCP segments, and obtain the application layer messages.

We then represent the byte sequence as a collection of n-grams taken with various stride values. An n-gram is a consecutive sequence of n items from a given sample; in this case, an n-gram is a consecutive series of bytes obtained from the application layer message. Stride is how many steps the sliding window takes when collecting n-grams. Figure 3 shows examples of various n-grams obtained with a different value of n and stride.

We define the input to the classifier to be a set of integer encoded n-grams. Let $X = \{x_1, x_2, x_3, \ldots, x_k\}$ be the integer encoded n-grams collected from an application layer message as the input to the RNN model. We denote k as the number of n-grams taken from each application layer message. Each n-gram is categorical data. It means a value of 1 is not necessarily smaller than a value of 50. They are simply different. Encoding n-grams with one-hot encoding is not a viable solution as the resulting vector would be sparse and hard to model. Therefore, Blatta transforms the sequence of n-grams with embedding technique. Embedding is essentially a lookup table that maps an item to a dense vector with a fixed size *embedded*_dim. Using pretrained embedding vectors, e.g., GloVe [41], is common in natural language processing problems, but these pretrained embedding vectors were generated from a corpus of words. While our approach works with byte-level n-grams. Therefore, it is not possible to use the pretrained embedding vectors. Instead, we initialise the embedding vectors with random values which will be updated by backpropagation during the training so that n-grams which usually appear together will have vectors that are close to each other.

It is worth noting that the number of n-grams collected raises exponentially as the n increases. If we considered all possible n-gram values, the model would overfit. Therefore, we limit the number of embedding vectors by building a dictionary of most common n-grams in the training set. We define the dictionary size as l in which it contains l unique n-grams and a placeholder for other n-grams that do not exist in the dictionary. Thus, the embedding vectors have $l + 1$ entries. However, we would like the size of each embedded vector to be less than $l + 1$. Let ε be the size of an embedded vector (*embedded*_dim). If x_t represents an n-gram, the embedding layer transforms X to $\widehat{X}$. We denote $\widehat{X} = \{\widehat{x}_1, \ldots, \widehat{x}_k\}$ where each $\widehat{x}$ is a vector with the size of ε. The embedded vectors $\widehat{X}$ are then passed to the recurrent layer.

4.4. Training RNN-Based Classifier. Since the input to the classifier is sequential data, we opted to use a method that takes into account the sequential relationship of elements in the input vectors. Such methods capture the behaviour of a sequence better than processing those elements individually [42]. A recurrent neural network is an architecture of neural networks in which each layer takes time series data, processes them in several time steps, and utilises the output of the previous time step in the current step calculation. We refer to these layers as recurrent layers. Each recurrent layer consists of recurrent units.

The vanilla RNN has a vanishing gradient problem, a situation where the recurrent model cannot be further trained because the value to update the weight is too small; thus, there would be no point of training the model further. Therefore, long short-term memory (LSTM) [43] and gated recurrent unit (GRU) [44] are employed to avoid this situation. Both LSTM and GRU have cells/units that are connected recurrently to each other, replacing the usual recurrent units which existed in earlier RNNs. What makes their cells different is the existence of gates. These gates decide whether to pass certain information coming from the previous cell (i.e., input gate and forget gate in LSTM unit and update gate in GRU) or going to the next unit (i.e., output gate in LSTM). Since LSTM has more gates than GRU, it requires more calculations, thus computationally more expensive. Yet, it is not conclusive whether one is better than the other [44]; thus, we use both types and compare the results. For brevity, we will refer to both types as recurrent layers and their cells as recurrent units.

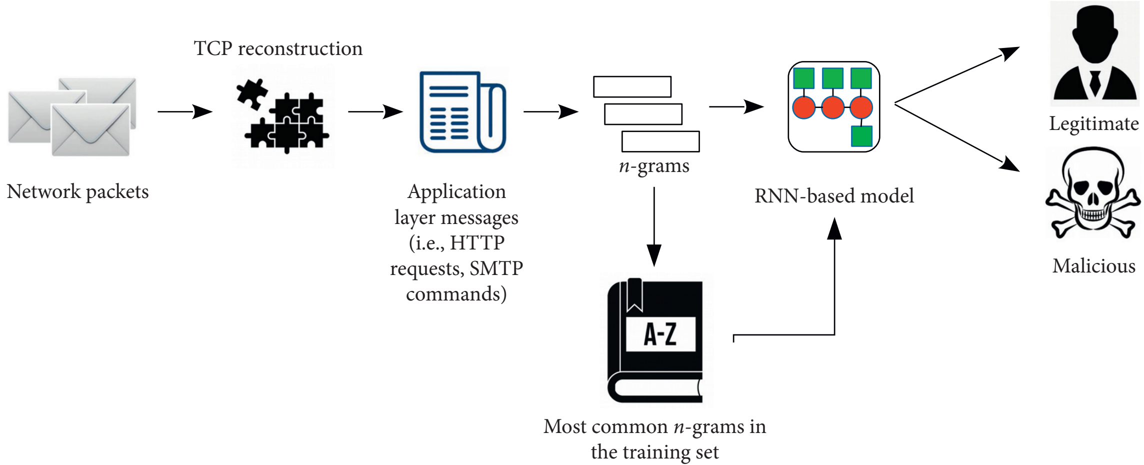

Figure 2: Architecture overview of the proposed method. Application layer messages are extracted from captured traffic using tcpflow [39]. n-grams are obtained from those messages. They will then be used to build a dictionary of most common n-grams and train the RNN-based model (i.e., LSTM and GRU). The trained model outputs a prediction whether the traffic is malicious or benign.

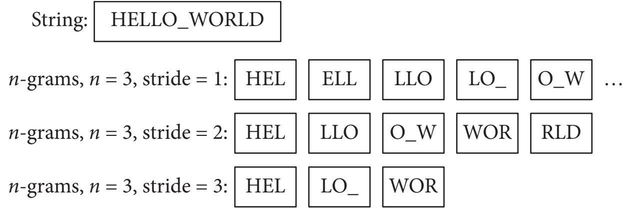

Figure 3: An example of n-grams of bytes taken with various stride values.

The recurrent layer takes a vector $\hat{x}_t$ for each time step t. In each time step, the recurrent unit outputs hidden state h_t with a dimension of $|h_t|$. The hidden state is then passed to the next recurrent unit. The last hidden state h_k becomes the output of the recurrent layer which will be passed onto the next layer.

Once we obtain h_k, the output of the recurrent layer, the next step is to map the vector to benign or malicious class. Mapping h_k to those classes requires us to use linear transformation and softmax activation unit.

A linear transformation transforms h_k into a new vector L using (1), where W is the trained weight and b is the trained bias. After that, we transform L to obtain $Y = \{y_i \mid 0 \le i < 2\}$, the log probabilities of the input file belonging to the classes with LogSoftmax, as described in (2). The output of LogSoftmax is the index of an element that has the largest probability in Y. All these forward steps are depicted in Figure 4:

$$L = W * h_k + b,$$
$$L = \{l_i \mid 0 \le i < 2\}, \tag{1}$$

$$Y = \underset{0 \le i < 2}{\arg\max} \left(\log \frac{\exp(l_i)}{\sum_{i=0}^{2} \exp(l_i)} \right). \tag{2}$$

In the training stage, after feeding a batch of training data to the model and obtaining the output, the next step is to evaluate the accuracy of our model. To measure our model's performance during the training stage, we need to calculate a loss value which represents how far our model's output is from the ground truth. Since this approach is a binary classification problem, we use *negative log likelihood* [45] as the loss function. Then, the losses are backpropagated to update weights, biases, and the embedding vectors.

4.5. Detecting Exploits. The process of detecting exploits is essentially similar to the training process. Application layer messages are extracted. n-grams are acquired from these messages and encoded using the dictionary that was built during the training process. The encoded n-grams are then fed into the RNN model that will output probabilities of these messages being malicious. When the probability of an application layer message being malicious is higher than 0.5, the message is deemed malicious and an alert is raised.

The main difference in this process to the training stage is the time when Blatta stops processing inputs and makes the decision. Blatta takes the intermediate output of the RNN model, hence requiring fewer inputs and disregarding the needs to wait for the full-length message to process. We will show in our experiment that the decision taken by using intermediate output and reading fewer bytes is close to reading the full-length message, giving the proposed approach an ability of early prediction.

5. Experiments and Results

In this section, we evaluate Blatta and present evidence of its effectiveness in predicting exploit in network traffic. Blatta is implemented with Python 3.5.2 and PyTorch 0.2.0 library. All experiments were run on a PC with Core i7 @ 3.40 GHz, 16 GB of RAM, NVIDIA GeForce GT 730, NVIDIA CUDA 9.0, and CUDNN 7.

The best practice for evaluating a machine learning approach is to have separate training and testing set. As the name implies, training set is used to train the model and the

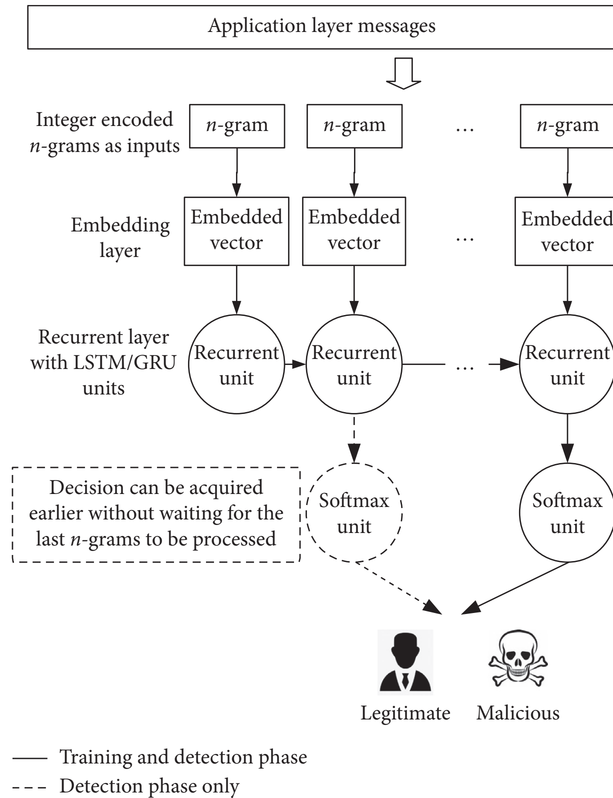

Figure 4: A detailed view of the classifier. n-grams are extracted from the input application layer message, which are then used to train an RNN model to classify whether the connection is malicious or benign.

testing set is for evaluating the model's performance. We split BlattaSploit dataset in a 60 : 40 ratio for training and testing set as malicious samples. The division was carefully taken to include diverse type of exploit payloads. As samples of benign traffic to train the model, we obtained the same number of HTTP and FTP connections as the malicious samples from UNSW-JAN. Having a balanced set of both classes is important in a supervised learning.

We measure our model's performance by using samples of malicious and benign traffic. Malicious samples are obtained from 40% of BlattaSploit dataset, exploit, and worm samples in UNSW-FEB set (10,855 samples). As for the benign samples, we took the same number (10,855) of benign HTTP and FTP connections in UNSW-FEB. We used the UNSW dataset to compare our proposed approach performance with previous works.

In summary, the details of training and testing sets used in the experiments are shown in Table 3.

We evaluated the classifier model by counting the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN). They are then used to calculate detection rate (DR) and false positive rate (FPR) which are metrics to measure our proposed system's performance.

DR measures the ability of the proposed system to correctly detect malicious traffic. The value of DR should be as close as possible to 100%. It would show how well our system is able to detect exploit traffic. The formula to calculate DR is shown in (3). We denote TP as the number of detected malicious connections and FN as the number of undetected malicious connections in the testing set:

$$\mathrm{DR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{3}$$

FPR is the percentage of benign samples that are classified as malicious. We would like to have this metric to be as low as possible. High FPR means many false alarms are raised, rendering the system to be less trustworthy and useless. We calculate this metric using (4). We denote FP as the number of false positives detected and N as the number of benign samples:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{N}. \tag{4}$$

5.1. Data Analysis. Before discussing about the results, it is preferable to analyse the data first to make sure that the results are valid and the conclusion taken is on point. Blatta aims to detect exploit traffic by reading the first few bytes of the application layer message. Therefore, it is important to know how many bytes are there in the application layer messages in our dataset. Hence, we can be sure that Blatta reads a number of bytes fewer than the full length of the application layer message.

Table 4 shows the average length of application layer messages in our testing set. The benign samples have an average message length of 593.25, lower than any other sets. Therefore, deciding after reading fewer bytes than at least that number implies our proposed method can predict malicious traffic earlier, thus providing improvement over previous works.

5.2. Exploring Parameters. Blatta includes parameters that must be selected in advance. Intuitively, these parameters affect the model performance, so we analysed the effect on model's performance and selected the best model to be compared later to previous works. The parameters we analysed are recurrent layer types, n, stride, dictionary size, the embedding vector dimension, and the recurrent layer type. When analysing each of these parameters, we use different values for the parameter and set the others to their default values (i.e., $n = 5$, stride = 1, dictionary size = 2000, $embedding_\mathrm{dim} = 32$, recurrent layer = LSTM, and number of hidden layers = 1). These default values were selected based on the preliminary experiment, which had given the best result. Apart from the modifiable parameters, we set the optimiser to stochastic gradient descent with learning rate 0.1 as using other optimisers did not necessarily increase or decrease the performance in our preliminary experiment. The number of epochs is fixed to five as the loss value did not decrease further, adding more training iteration did not give significant improvement.

As can be seen in Table 5, we experimented with variable lengths of input to see how much the difference when the number of bytes read is reduced. It is noted that some

Table 3: Numbers of benign and malicious samples used in the experiments.

Set	Obtained from	Num. of samples	Class
Training set	BlattaSploit	3406	Malicious
	UNSW-JAN	3406	Benign
Testing set	BlattaSploit	2276	Malicious
	UNSW-FEB	10855	Benign
	UNSW-FEB	10855	Malicious

Table 4: Average message length of application layer messages in the testing set.

Set	Average message length
BlattaSploit	2318.93
UNSW benign samples	593.25
UNSW exploit samples	1437.36
UNSW worm samples	848

messages are shorter than the limit. In this case, all bytes of the messages are indeed taken.

In general, reading the full length of application layer messages mostly gives more than 99% detection rate with 2.51% false positive rate. This performance stays still with a minor variation when the length of input messages is reduced down to 500 bytes. When the length is limited to 400, the false positive rate spikes up for some configurations. Our hypothesis is that this is due to benign samples have relatively short length. Therefore, we will pay more attention to the results of reading 500 bytes or fewer and analyse each parameter individually.

n is the number of bytes (n-grams) taken in each time step. As shown in Table 5, we experimented with 1, 3, 5, 7, and 9-gram. For brevity, we omitted $n = 2, 4, 6, 8$ because the result difference is not significant. As predicted earlier, 1-gram was least effective, and the detection rates were around 50%. As for the high-order n-grams, the detection rates are not much different but the false positive rates are. 5-gram and 7-gram provide better false positive rates (2.51%) even when Blatta reads the first 400 bytes. 7-gram gives lower false positive rate (8.92%) when reading first 300 bytes yet it is still too high for real-life situation. Having a higher n means more information is considered in a time step, this may lead to not only a better performance but also overfitting.

As the default value of n is five, we experimented with stride of one to five. Thus, it can be observed how the model would react depending on how much the n-grams overlapped. It is apparent that nonoverlapping n-grams provide lower false positives with around 1–3% decrease in detection rates. A value of two and three for the stride performs the worst, and they missed quite a few malicious traffic.

The dictionary size plays an important role in this experiment. Having too many n-grams leads to overfitting as the model would have to memorise too many of them that may barely occur. We found that a dictionary size of 2000 has the highest detection rate and lowest false positive rate. Reducing the size to 1000 has made the detection rate to drop for about 50%, even when the model read the full-length messages.

Without embedding layer, Blatta would have used a one-hot vector as big as the dictionary size for the input in a time step. Therefore, the embedding dimension has the same effect as dictionary size. Having it too big leads to overfitting and too little could mean too much information is lost due to the compression. Our experiments with a dimension of 16, 32, or 64 give similar detection rates and differ less than 2%. An embedding dimension of 64 can have the least false positive when reading 300 bytes.

Changing recurrent layer does not seem to have much difference. LSTM has a minor improvement over GRU. We argue that preferring one after the other would not make a big improvement other than training speed. Adding more hidden layers does not improve the performance. On the other hand, it has a negative impact on the detection speed, as shown in Table 6.

After analysing this set of experiments, we ran another experiment with a configuration based on the best performing parameters previously explained. The configuration is $n = 5$, stride $= 5$, dictionary size $= 2000$, embedding dimension $= 64$, and a LSTM layer. The model then has a detection rate of 97.57% with 1.93% false positives by reading only the first 400 bytes. This result shows that Blatta maintains high accuracy while only reading 35.21% the length of application layer messages in the dataset. This optimal set of parameters is then used in further analysis.

5.3. Detection Speed. In the IDS area, detection speed is another metric worth looked into, apart from accuracy-related metrics. Time is of the essence in detection, and the earlier we detect malicious traffic, the faster we could react. However, detection speed is affected by many factors, such as the hardware or other applications/services running at the same time as the experiment. Therefore, in this section, we analyse the difference of execution time between reading the full and partial payload.

We first calculated the execution time of each record in the testing set, then divided the number of bytes processed by the execution time to obtain the detection speed in kilobytes/seconds (KBps). Eventually, the detection speed of all records was averaged. The result is shown in Table 6.

As shown in Table 6, reducing the processed bytes to 700, about half the size of an IP packet, increased the detection speed by approximately two times (from an average of 8.366 kbps to 16.486 kbps). Evidently, the trend keeps rising as the number of bytes reduced. If we take the number of bytes limit from the previous section, which is 400 bytes, Blatta can provide about three times increment in detection speed or 22.17 kbps on average. We are aware that this number seems small compared to the transmission speed of a link in the network which can reach 1 Gbps/128 MBps. However, we argued that there are other factors which limit our proposed method from performing faster, such as the hardware used in the experiment and the programming language used to implement the approach. Given the approach runs in a better environment, the detection speed will increase as well.

Table 5: Experiment results of using various parameters combination and various lengths of input to the model

Parameter		No. of bytes							No. of bytes						
		All	**700**	**600**	**500**	**400**	**300**	**200**	**All**	**700**	**600**	**500**	**400**	**300**	**200**
n	**1**	47.22	48.69	49.86	50.65	51.77	54.99	65.32	1.18	1.19	1.21	1.21	71.31	78.7	89.43
	3	99.87	99.51	99.77	99.1	99.59	98.93	91.07	2.51	2.51	2.51	2.51	72.61	10.29	20.51
	5	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.6**	**11.08**
	7	99.86	99.47	99.59	99.37	99.19	98.53	97.08	2.51	2.51	2.51	2.51	2.51	8.92	80.92
	9	99.81	99.59	99.62	99.57	99.23	98.16	88.93	2.51	2.51	2.51	2.51	72.6	74.16	90.6
Stride	**1**	99.87	99.55	99.78	99.57	99.29	98.91	88.75	2.51	2.51	2.51	2.51	2.51	72.6	11.08
	2	73.39	74.11	74.01	74.45	74.69	74.62	77.82	1.81	1.81	1.81	1.81	71.92	72.46	19.86
	3	82.51	82.54	83.07	83.12	83.25	83.5	85.75	1.5	1.49	1.5	1.51	71.62	75.47	89.63
	4	99.6	99.19	99.26	99.28	98.61	98.55	98.37	1.93	1.93	1.93	1.93	1.93	74.09	10.5
	5	**99.73**	**98.95**	**98.88**	**98.65**	**98**	**95.77**	**88.29**	**1.93**	**1.92**	**1.93**	**1.93**	**1.93**	**54.16**	**90.02**
Dictionary size	**1000**	47.78	49.5	50.36	50.79	51.8	54.83	54.68	1.21	1.21	1.22	1.22	71.33	79.47	89.42
	2000	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.6**	**11.08**
	5000	99.87	99.37	99.75	99.79	99.62	99.69	99.66	2.51	2.51	2.51	2.51	72.61	10.03	90.61
	10000	99.86	99.44	99.74	99.55	99.44	98.55	98.33	2.51	2.51	2.51	2.51	72.61	79.06	90.15
	20000	99.84	99.81	99.69	99.24	99.21	99.43	98.91	2.51	2.51	2.51	2.51	72.61	80.46	89.64
Embedding dimension	**16**	99.89	99.65	99.7	99.67	99.22	99.09	98.81	2.51	2.51	2.51	2.51	2.51	76.77	80.94
	32	99.87	99.55	99.78	99.57	99.29	98.91	88.75	2.51	2.51	2.51	2.51	2.51	72.6	11.08
	64	**99.87**	**99.2**	**99.41**	**99.09**	**98.61**	**96.76**	**85.52**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**4.51**	**89.85**
	128	99.84	99.33	99.6	99.35	98.99	97.69	86.78	2.51	2.51	2.51	2.51	72.6	4.27	10.88
	256	99.88	99.76	99.8	99.22	99.38	98.64	90.34	2.51	2.51	2.51	2.51	72.6	80.79	90.6
Recurrent layer	**LSTM**	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.6**	**11.08**
	GRU	99.88	99.35	99.48	99.35	99.06	97.94	86.22	2.51	2.51	2.51	2.51	2.51	78.95	8.48
No. of layers	**1**	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.6**	**11.08**
	2	99.86	99.46	99.46	99.38	99.2	99.72	88.65	2.51	2.51	2.51	2.51	72.59	78.78	20.29
	3	99.84	99.38	99.68	99.1	99.18	98.16	87.35	2.51	2.51	2.51	2.51	2.51	74.94	10.83
		Detection rate							False positive rate						

Bold values show the parameter value for each set of experiment which gives the highest detection rate and lowest false positive rate.

Table 6: The effect of reducing the number of bytes to the detection speed.

No. of bytes	No. of LSTM layers		
	1	2	3
All	8.366 ± 0.238327	5.514 ± 0.004801	3.698 ± 0.011428
700	16.486 ± 0.022857	10.704 ± 0.022001	7.35 ± 0.044694
600	18.16 ± 0.020556	11.97 ± 0.024792	8.21 ± 0.049584
500	20.432 ± 0.02352	13.65 ± 0.036668	9.376 ± 0.061855
400	22.17 ± 0.032205	14.94 ± 0.037701	10.302 ± 0.065417
300	24.076 ± 0.022857	16.368 ± 0.036352	11.318 ± 0.083477
200	26.272 ± 0.030616	18.138 ± 0.020927	12.688 ± 0.063024

The values are average (mean) detection speed in kbps with 95% confidence interval, calculated from multiple experiments. The detection speed increased significantly (about three times faster than reading the whole message), allowing early prediction of malicious traffic.

5.4. Comparison with Previous Works. We compare Blatta results with other related previous works. PAYL [7], OCPAD [12], Decanter [15], and the autoencoder model [23] were chosen due to their code availability, and both can be tested against the UNSW-NB15 dataset. PAYL and OCPAD read an IP packet at a time, while Decanter and [23] reconstruct TCP segments and process the whole application layer message. None of them provides early detection, but to show that Blatta also offers improvements in detecting malicious traffic, we compare the detection and false positive rates of those works with Blatta.

We evaluated all methods with exploit and worm data in UNWS-NB15 as those match our threat model and the dataset is already publicly available. Thus, the result would be comparable. The results are shown in Table 7.

In general, Blatta has the highest detection rate—albeit it also comes with the cost of increasing false positives. Although the false positives might make this approach unacceptable in real life, Blatta is still a significant step towards a direction of early prediction, a problem that has not been explored by similar previous works. This early prediction approach enables system administrators to react faster, thus reducing the harm to protected systems.

In the previous experiments, as shown in Section 5.2, Blatta needs to read 400 bytes (on average 35.21% of application layer message size) to achieve 97.57% detection

Table 7: Comparison to previous works using the UNSW-NB15 dataset as the testing set.

Method	Detection rate (%)		FPR (%)
	Exploits in UNSW-NB15	Worms in UNSW-NB15	
Blatta	99.04	100	1.93
PAYL	87.12	26.49	0.05
OCPAD	10.53	4.11	0
Decanter	67.93	90.14	0.03
Autoencoder	47.51	81.12	0.99

rate. It reads fewer bytes than PAYL, OCPAD, Decanter, and [23] while keeping the accuracy high.

5.5. Visualisation. To investigate how Blatta has performed the detection, we took samples of both benign and malicious traffic and observed the input and output. We were particularly interested in the relation of *n*-grams that are not stored in the dictionary to the decision (unknown *n*-grams). Those *n*-grams either did not exist in the training set or were not common enough to be included in the dictionary.

On Figure 5, we visualise three samples of traffic taken from different sets, BlattaSploit and UNSW-NB15 datasets. The first part of each sample shows *n*-grams that did not exist in the dictionary. The yellow highlighted parts show those *n*-grams. The second part shows the output of the recurrent layer for each time step. The darker the red highlight, the closer the probability of the traffic being malicious to one in that time step.

As shown in Figure 5 (detection samples), malicious samples tend to have more unknown *n*-grams. It is evident that the existence of these unknown *n*-grams increases the probability of the traffic being malicious. As an example, the first five bytes of the five samples have around 0.5 probability of being malicious. And then the probability went up closer to one when an unknown *n*-gram is detected.

Similar behaviour also exists in the benign sample. The probability is quite low because there are many known *n*-grams. Despite the existence of unknown *n*-grams in the benign sample, the end result shows that the traffic is benign. Furthermore, most of the time, the probability of the traffic being malicious is also below 0.5.

6. Evasion Techniques and Adversarial Attacks

Our proposed approach is not a silver bullet to tackle exploit attacks. There are evasion techniques which could be employed by adversaries to evade the detection. These techniques open possibilities for future work. Therefore, this section talks about such evasion techniques and discuss why they have not been covered by our current method.

Since our proposed method works by analysing application layer messages, it is safe to disregard evasion techniques on transport or network layer level, e.g., IP fragmentation, TCP delayed sending, and TCP segment fragmentation. They should be handled by the underlying tool that reconstructs TCP session. Furthermore, those evasion techniques can also be avoided by Snort preprocessor.

Two possible evasion techniques are compression and/or encryption. Both compression and encryption change the bytes' value from the original and make the malicious code harder to detect by Blatta or any previous work [7, 12, 23] on payload-based detection. Metasploit has a collection of evasion techniques which include compression. The compression evasion technique only works on HTTP and utilises gzip. This technique only compresses HTTP responses, not HTTP requests. While all HTTP-based exploits in UNSW-NB15 and BlattaSploit have their exploit code in the request, thus no data are available to analyse the performance if the adversary uses compression. However, gzip compressed data could still be detected because they always start with the magic number 1f 8b and the decompression can be done in a streaming manner in which Blatta can do so. There is also no need to decompress the whole data since Blatta works well with partial input.

Encryption is possibly the biggest obstacle in payload-based NIDS: none of the previous works in our literature (see Table 1) have addressed this challenge. There are other studies which deal with payload analysis in encrypted traffic [46, 47]. However, these studies focus on classifying which application generates the network traffic instead of detecting exploit attacks; thus, they are not directly relevant to our research.

On its own, Blatta is not able to detect exploits hiding in encrypted traffic. However, Blatta's model can be exported and incorporated with application layer firewalls such as ShadowDaemon [48]. ShadowDaemon is commonly installed on a web server and intercepts HTTP requests before being processed by a web server software. It detects attacks based on its signature database. Since it is extensible and reads the same data as Blatta (i.e., application layer messages), it is possible to use Blatta's RNN-based model to extend the capability of ShadowDaemon beyond rule-based detection. More importantly, this approach would enable Blatta to deal with encrypted traffic, making it applicable in real-life situations.

Attackers could place the exploit code closer to the end of the application layer message. Hoping that in doing so, the attack would not be detected as Blatta reads merely the first few bytes. However, exploits with this kind of evasion technique would still be detected since this evasion technique needs a padding to place the exploit code at the end of the message. The padding itself will be detected as a sign of malicious attempts as it is most likely to be a byte sequence which rarely exist in the benign traffic.

Probabilities:
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

An exploit sample in BlattaSploit dataset

Unknown n-grams:
POST /setFileUpload HTTP/1.1 Host: 192.168.99.6 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Authorization: Basic YWRtaW46 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryeyfnwL Content-Length: 797 ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="ReplySuccessPage" replyuf.htm ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="ReplyErrorPage" replyuf.htm ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="Filename" /tmp/.hoqG2a ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="UploadFile"; filename="9MpXYBKzOQ" Content-Type: application/octet-stream ¨ELF◁◁◁◁√◁T@44 ◁◁@@ÐL◁☎◂õÿ..$'pÅ◁·ÿ.♪$ÿÿ◂.·.--√('X`◁! Ëë▷--⊤(üÿ1÷ûÿ $'h ◁ÿÿ¯!üÿ(÷!¸ï◁&←►◁+ðï◁üÿ#¯ÿÿ¦!ùÿÀ⊤!È-▷3◂◁$TJ◁ñZå▯Ð؟ã¡-¤#©ãrà¡▯C@©◂¤á¡_a© Uç¹ÿ[ä▯ ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="ConfigUploadFile" Upload File ------WebKitFormBoundaryeyfn

RNN output:
POST /setFileUpload HTTP/1.1 Host: 192.168.99.6 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Authorization: Basic YWRtaW46 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryeyfnwL Content-Length: 797 ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="ReplySuccessPage" replyuf.htm ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="ReplyErrorPage" replyuf.htm ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="Filename" /tmp/.hoqG2a ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="UploadFile"; filename="9MpXYBKzOQ" Content-Type: application/octet-stream ¨ELF◁◁◁◁√◁T@44 ◁◁@@ÐL◁☎◂õÿ..$'pÅ◁·ÿ.♪$ÿÿ◂.·.--√('X`◁! Ëë▷--⊤(üÿ1÷ûÿ $'h ◁ÿÿ¯!üÿ(÷!¸ï◁&←►◁+ðï◁üÿ#¯ÿÿ¦!ùÿÀ⊤!È-▷3◂◁$TJ◁ñZå▯Ð؟ã¡-¤#©ãrà¡▯C@©◂¤á¡_a© Uç¹ÿ[ä▯ ------WebKitFormBoundaryeyfnwL Content-Disposition: form-data; name="ConfigUploadFile" Upload File ------WebKitFormBoundaryeyfn

A benign sample

Unknown n-grams:
GET / HTTP/1.1 Host: server-95ab7e08.example.int Connection: Keep-Alive If-None-Match: 9510b9a66bd8fe6c41b661eed1706bc2 User-Agent: Client Agent Accept: */* Accept-Language: en-us Accept-Encoding: gzip,deflate UA-CPU: x86 GET /notthere.html HTTP/1.1 Host: server-95ab7e08.example.int Connection: close If-None-Match: 89e4e52c44a9f642acefd7905135d0c8 User-Agent: Client Agent Accept: */* Accept-Language: en-us Accept-Encoding: gzip,deflate UA-CPU: x86

RNN output:
GET / HTTP/1.1 Host: server-95ab7e08.example.int Connection: Keep-Alive If-None-Match: 9510b9a66bd8fe6c41b661eed1706bc2 User-Agent: Client Agent Accept: */* Accept-Language: en-us Accept-Encoding: gzip,deflate UA-CPU: x86 GET /notthere.html HTTP/1.1 Host: server-95ab7e08.example.int Connection: close If-None-Match: 89e4e52c44a9f642acefd7905135d0c8 User-Agent: Client Agent Accept: */* Accept-Language: en-us Accept-Encoding: gzip,deflate UA-CPU: x86

An exploit sample in UNSW-NB15 dataset

Unknown n-grams:
GET http://149.171.126.18:8080/requests/status.xml?
command=in_play&input=smb://KyRXAQOg.jgr/wUUnlulfuL/QWblfhTZXMZKwYBiSCKnTWMIXTNyjxfndXHZGTKrKKLdhqfInMYRINqmRUvzFRxOebClslYZyqqRdSedANwe
Mozilla/5.0 (Android; Tablet; rv:13.0.1) Gecko/13.0.1 Firefox/13.0.1 Accept: */* Connection: keep-alive

RNN output:
GET http://149.171.126.18:8080/requests/status.xml?
command=in_play&input=smb://KyRXAQOg.jgr/wUUnlulfuL/QWblfhTZXMZKwYBiSCKnTWMIXTNyjxfndXHZGTKrKKLdhqfInMYRINqmRUvzFRxOebClslYZyqqRdSedANwe
Mozilla/5.0 (Android; Tablet; rv:13.0.1) Gecko/13.0.1 Firefox/13.0.1 Accept: */* Connection: keep-alive

Figure 5: Visualisation of unknown n-grams in the application layer messages and outputs of the recurrent layer for each time step. It shows how the proposed system observes and analyses the traffic. Yellow blocks show unknown n-grams. Red blocks show the probability of the traffic being malicious when reading an n-gram at that point.

7. Conclusion and Future Work

This paper presents Blatta, an early prediction system for exploit attacks which can detect exploit traffic by reading only 400 bytes from the application layer message. First, Blatta builds a dictionary containing most common n-grams in the training set. Then, it is trained over benign and malicious sequences of n-grams to classify exploit traffic. Lastly, it continuously reads a small chunk of an application layer message and predicts whether the message will be a malicious one.

Decreasing the number of bytes taken from application layer messages only has a minor effect on Blatta's detection rate. Therefore, it does not need to read the whole application layer message like previously related works to detect exploit traffic, creating a steep change in the ability of system administrators to detect attacks early and to block them before the exploit damages the system. Extensive evaluation of the new exploit traffic dataset has clearly demonstrated the effectiveness of Blatta.

For future study, we would like to train a model that can recognise malicious behaviour based on messages exchanged between clients and a server since in this paper we only consider messages from clients to a server, but not the other way around. Detecting attacks on encrypted traffic while retaining the early prediction capability could be a future research direction. It also remains a question whether the approach in mobile traffic classification [46, 47] would be applicable to the field of exploit detection.

Data Availability

The compressed file containing the dataset is available on https://bit.ly/blattasploit.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Indonesia Endowment Fund for Education (Lembaga Pengelola Dana Pendidikan/LPDP) under the grant number PRJ-968/LPDP.3/2016 of the LPDP.

References

[1] McAfee, "McAfee labs threat report—august 2019," Report, McAfee, Santa Clara, CA, USA, 2019.
[2] E. DB, "Offensive security's exploit database archive," EDB, Bedford, MA, USA, 2010.
[3] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, "Evasion techniques: sneaking through your intrusion detection/prevention systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1011–1020, 2012.
[4] M. Roesch, "Snort: lightweight intrusion detection for networks," in *Proceedings of LISA'99: 13th Systems Administration Conference*, vol. 99, pp. 229–238, Seattle, Washington, USA, November 1999.
[5] R. Sommer and V. Paxson, "Outside the closed world: on using machine learning for network intrusion detection," in *Proceedings of the 2010 IEEE Symposium on Security And Privacy*, pp. 305–316, Berkeley/Oakland, CA, USA, May 2010.
[6] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: a review," *Computers & Security*, vol. 30, no. 6-7, pp. 353–375, 2011.
[7] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," *Lecture Notes in Computer Science, Recent Advances in Intrusion Detection*, in *Proceedings of the 7th International Symposium, Raid 2004*, pp. 203–222, Gothenburg, Sweden, September 2004.
[8] A. Jamdagni, Z. Tan, X. He, P. Nanda, and R. P. Liu, "RePIDS: a multi tier real-time payload-based intrusion detection system," *Computer Networks*, vol. 57, no. 3, pp. 811–824, 2013.
[9] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "McPAD: a multiple classifier system for accurate payload-based anomaly detection," *Computer Networks*, vol. 53, no. 6, pp. 864–881, 2009.
[10] D. Ariu, R. Tronci, and G. Giacinto, "HMMPayl: an intrusion detection system based on hidden markov models," *Computers & Security*, vol. 30, no. 4, pp. 221–241, 2011.
[11] A. Oza, K. Ross, R. M. Low, and M. Stamp, "HTTP attack detection using *n*-gram analysis," *Computers & Security*, vol. 45, pp. 242–254, 2014.
[12] M. Swarnkar and N. Hubballi, "OCPAD: one class naive bayes classifier for payload based anomaly detection," *Expert Systems with Applications*, vol. 64, pp. 330–339, 2016.
[13] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *Proceedings of the 25th {USENIX} Security Symposium*, pp. 807–822, Austin, TX, USA, August 2016.
[14] H. Zhang, D. Yao, N. Ramakrishnan, and Z. Zhang, "Causality reasoning about network events for detecting stealthy malware activities," *Computers & Security*, vol. 58, pp. 180–198, 2016.
[15] R. Bortolameotti, T. van Ede, M. Caselli et al., "DECANTeR: DEteCtion of anomalous outbound HTTP traffic by passive application fingerprinting," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, pp. 373–386, Orlando, FL, USA, December 2017.
[16] D. Golait and N. Hubballi, "Detecting anomalous behavior in voip systems: a discrete event system modeling," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 3, pp. 730–745, 2016.
[17] P. Duessel, C. Gehl, U. Flegel, S. Dietrich, and M. Meier, "Detecting zero-day attacks using context-aware anomaly detection at the application-layer," *International Journal of Information Security*, vol. 16, no. 5, pp. 475–490, 2017.
[18] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen, "TR-IDS: anomaly-based intrusion detection through text-convolutional neural network and random forest," *Security and Communication Networks*, vol. 2018, Article ID 4943509, 9 pages, 2018.
[19] X. Jin, B. Cui, D. Li, Z. Cheng, and C. Yin, "An improved payload-based anomaly detector for web applications," *Journal of Network and Computer Applications*, vol. 106, pp. 111–116, 2018.
[20] Y. Hao, Y. Sheng, and J. Wang, "Variant gated recurrent units with encoders to preprocess packets for payload-aware intrusion detection," *IEEE Access*, vol. 7, pp. 49985–49998, 2019.
[21] P. Schneider and K. Bottinger, "High-performance unsupervised anomaly detection for cyber-physical system networks," in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy—CPS-SPC'18*, pp. 1–12, Barcelona, Spain, January 2018.
[22] T. Hamed, R. Dara, and S. C. Kremer, "Network intrusion detection system based on recursive feature addition and bigram technique," *Computers & Security*, vol. 73, pp. 137–155, 2018.
[23] B. A. Pratomo, P. Burnap, and G. Theodorakopoulos, "Unsupervised approach for detecting low rate attacks on network traffic with autoencoder," in *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pp. 1–8, Glasgow, UK, June 2018.
[24] Z.-Q. Qin, X.-K. Ma, and Y.-J. Wang, "Attentional payload anomaly detector for web applications," in *Proceedings of the International Conference on Neural Information Processing*, pp. 588–599, Siem Reap, Cambodia, December 2018.
[25] 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.
[26] Z. Zhao and G.-J. Ahn, "Using instruction sequence abstraction for shellcode detection and attribution," in *Proceedings of the Conference on Communications and Network Security (CNS)*, pp. 323–331, National Harbor, MD, USA, October 2013.
[27] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, no. 1, p. 1, 2012.
[28] X. Wang, C.-C. Pan, P. Liu, and S. Zhu, "SigFree: a signature-free buffer overflow attack blocker," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 1, pp. 65–79, 2010.
[29] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: a content anomaly detector resistant to mimicry attack," *Lecture Notes in Computer Science*, in *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, pp. 226–248, Hamburg, Germany, September 2006.
[30] R. 7, "Metasploit penetration testing framework," 2003, https://www.metasploit.com/.
[31] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proceedings of the Military Communications and Information Systems Conference (MiLCIS)*, pp. 1–6, Canberra, ACT, Australia, November 2015.

[32] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, 2000.
[33] S. T. Brugger and J. Chow, "An assessment of the DARPA IDS evaluation dataset using snort," vol. 1, no. 2007, Tech. Report,. CSE-2007-1, p. 22, UCDAVIS Department of Computer Science, Davis, CA, USA, 2007.
[34] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "A first look at modern enterprise traffic," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, p. 2, Berkeley, CA, USA, October2005.
[35] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.
[36] I Bro, 2008, http://www.bro-ids.org.
[37] Argus, 2008, https://qosient.com/argus/.
[38] R. 7, "Metasploitable," 2012, https://information.rapid7.com/download-metasploitable-2017.html.
[39] S. L. Garfinkel and M. Shick, "Passive TCP reconstruction and forensic analysis with tcpflow," Technical Reports, Naval Postgraduate School, Monterey, CA, USA, 2013.
[40] K. Rieck and P. Laskov, "Language models for detection of unknown attacks in network traffic," *Journal in Computer Virology*, vol. 2, no. 4, pp. 243–256, 2007.
[41] J. Pennington, R. Socher, and C. D. Manning, "GloVe: global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014.
[42] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT press, Cambridge, MA, USA, 2016.
[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[44] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, https://arxiv.org/abs/1412.3555.
[45] PyTorch, "Negative log likelihood," 2016.
[46] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: a novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.
[47] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "MIMETIC: mobile encrypted traffic classification using multimodal deep learning," *Computer Networks*, vol. 165, Article ID 106944, 2019.
[48] H. Buchwald, "Shadow daemon: a collection of tools to detect, record, and block attacks on web applications," Shadow Daemon Introduction. 2015, https://shadowd.zecure.org/overview/introduction/.