Array 11 (2021) 100077

Contents lists available at ScienceDirect

Array

journal homepage: www.sciencedirect.com/journal/array

A survey on the application of deep learning for code injection detection

Stanislav Abaimov [a,*], Giuseppe Bianchi [b]

[a] Tor Vergata University of Rome, Italy/University of Bristol, UK
[b] CNIT/Tor Vergata University of Rome, Italy

ARTICLE INFO

Keywords:
Machine learning
Deep learning
Network intrusion detection
Code injection
Preprocessing

ABSTRACT

Code injection is one of the top cyber security attack vectors in the modern world. To overcome the limitations of conventional signature-based detection techniques, and to complement them when appropriate, multiple machine learning approaches have been proposed. While analysing these approaches, the surveys focus predominantly on the general intrusion detection, which can be further applied to specific vulnerabilities. In addition, among the machine learning steps, data preprocessing, being highly critical in the data analysis process, appears to be the least researched in the context of Network Intrusion Detection, namely in code injection. The goal of this survey is to fill in the gap through analysing and classifying the existing machine learning techniques applied to the code injection attack detection, with special attention to Deep Learning. Our analysis reveals that the way the input data is preprocessed considerably impacts the performance and attack detection rate. The proposed full preprocessing cycle demonstrates how various machine-learning-based approaches for detection of code injection attacks take advantage of different input data preprocessing techniques. The most used machine learning methods and preprocessing stages have been also identified.

1. Introduction

Code injection is the most popular and most impactful attack, which is at the top of the OWASP vulnerabilities list. The detection of code injection attacks, traditionally carried out using signature/pattern-based recognition techniques, has been recently supplemented by the application of advanced machine learning approaches. The advantage of such techniques is that similar algorithms, e.g., Deep or Convolutional Neural Networks, may find application in a broad range of various threat detection scenarios. Indeed, big cyber security companies are investing significant funds into the research and deployment of machine learning algorithms for the cyberthreat detection purposes, including malware analysis, vulnerable code detection, and intrusion detection (i.e. vulnerability exploitation attempts). AI-enhanced security adoption is growing rapidly,[1] and a range of machine learning methods for intrusion detection has already accumulated over the years.

Within the existing amount of approaches and techniques, which of the machine learning methods are more suitable for the code injection attack detection, and is there any consistency in their application? Which is the best approach for the dataset composition for enhanced performance and/or accuracy?

The hypothesised assumption, which we will investigate in depth in this paper, is that deep learning is the most suitable for code injection attack detection. We also argue that the way in which the input data is pre-processed (cleaned, reduced, reshaped, encoded, etc.) may significantly influence and affect the performance and effectiveness of the machine learning techniques employed to detect code injection attacks. In general, this was of course expected, as some of the features present in the data input may turn out to be random in nature and corrupt the output results, or, similarly, some of the features may introduce biases. But, we believe, a merit of our survey, and a clear diversification of our work from companion surveys on related topics, is in summarising and classifying how different machine-learning-based code injection attack detection techniques take advantage of (or rely upon) different input data pre-processing techniques. The relevance of the research is high, as data preprocessing appears to be the least researched in the context of Network Intrusion Detection in general and in code injection in particular, and it very often involves more effort and time (over 50% of total effort) within the entire data analysis process [1,2].

About 150 academic publications related to machine learning for

[*] Corresponding author.
E-mail address: stanislav.abaimov@uniroma2.it (S. Abaimov).

[1] Reinventing Cybersecurity with Artificial Intelligence, The new frontier in digital security, Capgemini Research Institute, 2019, https://www.capgemini.com/research/reinventing-cybersecurity-with-artificial-intelligence/.

https://doi.org/10.1016/j.array.2021.100077
Received 27 February 2021; Received in revised form 9 June 2021; Accepted 4 July 2021
Available online 17 July 2021

cyber security (specifically, network intrusion detection) have been analysed, out of which only 20 covered preprocessing for code injection detection in sufficient details (see Section 4.4). Further revision revealed a wide variety of diverse preprocessing methods, that as well will be documented later on in the survey.

More in details, the machine learning processing is based on four key steps, each with its own unique challenges.

1. Data collection is typically the initial step, an acquisition of raw data, that highly depends on the objective or subjective selection criteria.
2. Preprocessing, a highly critical step in the data analysis process, shapes data for the neural network to be trained, forming a structured multi-dimensional dataset. Preprocessing may increase or reduce the accuracy of the applied method based on a multitude of factors, discussed later in the survey. For example, duplicates can make the neural model biased.
3. Choosing, applying and fine-tuning the machine learning algorithm (e.g., state vector machine, neural network, decision tree, etc.) for a selected task is another challenge, as none of those algorithms is universal. The configuration of a selected algorithm becomes the sub-task mainly based on a trial and error approach, to improve the accuracy of a selected method.
4. The last step in the machine learning process is to read the output data, evaluate and present it in a way that can be further used by other systems or visually represented for human understanding. E.g., improper selection of a testing dataset can affect the metrics of the output performance.

The **goal** of this survey is to contribute to the body of knowledge related to maximising the performance and effectiveness of the machine learning techniques employed in intrusion detection, specifically in code injection attack detection. The widely used deep learning has been selected for analysis, supplemented by techniques allowing for potentially minimal or otherwise computationally optimal steps for a selected task.

The survey **objectives** included revision of the methods proposed in the latest academic publications (articles and surveys), identification of common steps in their approaches, their analysis and classification. Specific attention was paid to the modular steps in data preparation starting from data acquisition to the initiation of the training process, as well as selection of the machine learning method.

The survey scope is limited to reviewing the non-encrypted data transfer and features, that do not involve manipulations with the encrypted packet payload, traffic headers, or both. The issue of encrypted traffic is separated from our research as it is related to other techniques (e.g., cryptanalysis). The post-processing and result visualisation methods are not reviewed, as they have already been widely covered in several works (e.g. Refs. [3–5]).

Given the above goals, objectives, and limitations, to present the work in a structured manner, the survey is organised as follows:

Section 1 "Introduction" provides an overview of the researched area, sets the goal and objectives.

Section 2 "Code injection attack and defence" explains the code injection attack specificity and conventional defence. It also specifies the terms and notions relevant to the topic of the survey.

Section 3 "Machine learning application for the code injection detection" lists and details the types of the selected machine learning approaches, with an extra focus on deep learning. It also highlights specific requirements for the data selection and addresses the methods of hyperparameter optimisation to maximise the performance or accuracy of the selected methods.

Section 4 "Critical role of preprocessing in the detection enhancement" reviews a wide variety of modern methods used for preprocessing, identifies the gaps and proposes a classification of the full preprocessing cycle to maximise the performance effectiveness.

Section 5 "Conclusion" closes the current survey.

References list the publications reviewed in this survey.

2. Background

According to OWASP, "code injection is the general term for attack types which consist of injecting the code that is then interpreted/executed by the application".[2] In this survey, the code injection attack is understood as an intentional or unintentional use of the unforeseen software functionality, caused by processing of valid or invalid malicious input data. Code injection as an attack method is used by a malicious actor to input malicious code and system commands into a vulnerable point of entry in software (e.g., through the input field in a web application or header in the request) and change the course of execution. An algorithmic definition of code injection, that can be used to conventional intrusion detection systems, has been updated and improved by Ref. [6].

2.1. Code injection attacks

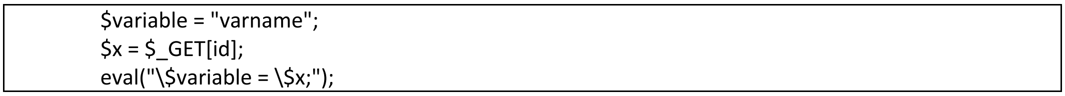
Code injection attack[3] relies on the input of the code that can be executed by a target program or an application. Code injection exploits improper handling and lack of data validation, which may include nonrestricted characters, data formats, or allocated memory space.

A malicious actor is constrained by the functionality of the language of the application. For example, if a malicious actor is able to inject an HTML code into a web application and it is stored in the page file, they are only limited to HTML and JavaScript injections, that in some cases can nonetheless be escalated to a remote command execution. Alternatively, a malicious code can be injected into the data about the file or into the command line through an application. Even though typically mentioned in the context of web applications, the code injection attack can target the application output or the memory used by the application.

The code injection attack can be split into two groups: Binary attack and Source Code attack [7]. Most well-known types of the Source Code injection vulnerabilities are the SQL injection, PHP injection, and JavaScript injection. The code can be injected through visible or hidden input fields, manually, automatically, or through uploaded or addressed files. If the code is injected into the system command line, this attack will be defined as the command injection. Binary Code injection can happen via shell code injection into the executable file input and cause stack or heap overflow.

2.1.1. Example: dynamic code evaluation vulnerability

When the PHP eval () function is used, it passes the untrusted data that an attacker can modify, allowing for the code injection to be possible. Take the code block:

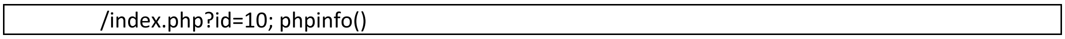


```
$variable = "varname";
$x = $_GET[id];
eval("\$variable = \$x;");
```

[2] https://owasp.org/www-community/attacks/Code_Injection.

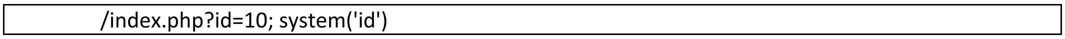
[3] https://owasp.org/www-community/attacks/Code_Injection.

In the presented code the input validation is not present; thus, the code above is vulnerable to a code injection attack.



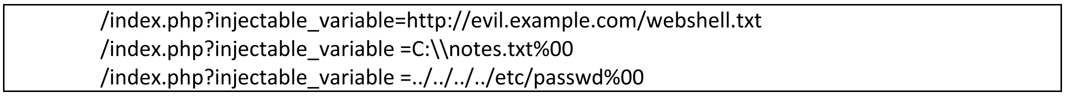
```
/index.php?id=10; phpinfo()
```

An attacker can then take a step further and execute system commands. With this condition, a code injection can also be used for command injection and further unauthorised access to the system (or a virtual container) hosting the web application:



```
/index.php?id=10; system('id')
```

An attacker can use URL input field to inject a local or remote file:

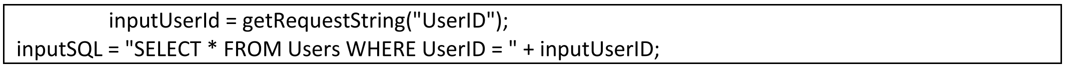


```
/index.php?injectable_variable=http://evil.example.com/webshell.txt
/index.php?injectable_variable =C:\\notes.txt%00
/index.php?injectable_variable =../../../../etc/passwd%00
```

In a similar way the injection can be performed on an already compromised system, by supplying a malicious file to a process, in a manner described in the Command injection example in Section 2.1.4.

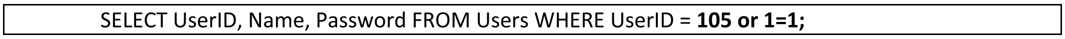
2.1.2. *Example: SQL injection*

SQL Injection is possible through the input fields or URL. Take the vulnerable code[4]:



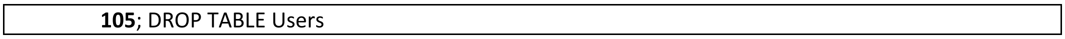
```
         inputUserId = getRequestString("UserID");
inputSQL = "SELECT * FROM Users WHERE UserID = " + inputUserID;
```

Should the attacker succeed in injecting the query with the value **105 or $1 = 1$;** then the already injected query *inputSQL* would look as follows:



```
SELECT UserID, Name, Password FROM Users WHERE UserID = 105 or 1=1;
```

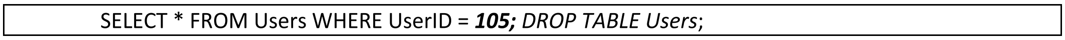
The output should return a page In the by replacing “or $1 = 1$” statement with “DROP TABLE Users”.



```
105; DROP TABLE Users
```

[4] https://www.w3schools.com/sql/sql_injection.asp.

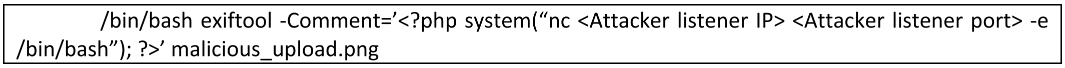
The result that is going to be executed will delete the table named "Users":



```
SELECT * FROM Users WHERE UserID = 105; DROP TABLE Users;
```

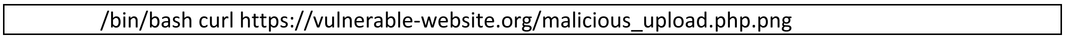
2.1.3. Example: exif code injection

Exchangeable Image File Format (exif) data contains information about the file: e.g., file source, creation and modification dates, GPS coordinates, camera model, time, compression type, etc. In the vast majority of cases, this data is present in every image or photo.



```
/bin/bash exiftool -Comment='<?php system("nc <Attacker listener IP> <Attacker listener port> -e
/bin/bash"); ?>' malicious_upload.png
```

The created file (malicious_upload.png) can be renamed as malicious_upload.php.png so the attackers can avoid basic defence mechanism. After the upload is complete, the file has to be addressed:

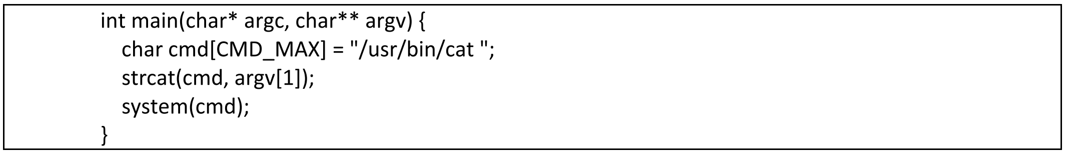


```
/bin/bash curl https://vulnerable-website.org/malicious_upload.php.png
```

The NetCat listener will receive the callback from the exploited webserver. The requirement for this to be exploited is the PHP interpreted on the target webserver. Without the PHP interpreter, the NetCat listener will still receive the reverse shell, but it will be to the attacker's own system.

2.1.4. Example: command injection

Take a simple main function, running as root, that executes a file using system command, by accepting a filename from an argument:



```
int main(char* argc, char** argv) {
   char cmd[CMD_MAX] = "/usr/bin/cat ";
   strcat(cmd, argv[1]);
   system(cmd);
}
```

This function can be used to display specific privileged data without the need for the user to interact with the system with the elevated privileges. As the entire program is executed with the highest privileges, *system()* executes with the root privileges. As a user or another program provides a legitimate filename, the call works as expected. If an attacker passes an argument *"filename; rm -rf/"* (including quotation marks, to ensure the entire string counts as a single argument). This way *system(),* after executing command *cat*, will execute *rm -rf/* with root privileges damage the system. Alternatively, an attacker can pass *"filename;/bin/bash"* and spawn a root shell.

2.2. Code injection defence

Code injection occurs when the application's output has an altered syntactic structure. Ray and Ligatti [6] argue that an algorithmic definition of the code injection is not complete, as there are two other cases that have to be addressed: code injection that does not alter the output syntactic structure, and non-code injection attack that alters the output syntactic structure. This complicates the detection of code injection attacks using conventional methods.

Malicious code is not different from the non-malicious code at the logical level, making it harder to detect using conventional non-machine learning detection methods. According to the interpreter, everything that goes into the input field is legitimate and can be processed with or without an error.

It is commonly believed that the code injection can be detected via signature recognition or via malicious access detection (e.g., canaries). As a countermeasure that does not require detection, sanitation is a technique of processing the input in such a way, that escape symbols cannot be injected into the code of the application. Automatic techniques, like input validation and input encoding, can be used for additional security; as well as output encoding can be used to prevent the system from disclosing sensitive information (e.g., system version, errors, successful injection output).

As a good practice, at the applications development stage the use of vulnerable functions should be avoided, or, if used, this should be according to the established secure development practices.[5]

The survey by Mitropoulos et al. [8] classifies 41 defence methods against SQL injection, XSS, and other web application attacks, the majority of which do not use machine learning.

SQL prevention techniques include prepared statements with Parameterized Queries, Escaping All User-Supplied Input, Hibernate Query Language Prepared Statement (Named Parameters), Whitelist Input Validation. As per [9] for the successful detection of an SQL injection vulnerability in a web application, a set of conditions should be met: 1) a path from an application input to a vulnerable function should exist, 2) parametric functions are used, 3) an attacker can access the result of a database query through feedback of the query propagation to

[5] Secure development and deployment guidance, UK National Cyber Security Centre, https://www.ncsc.gov.uk/collection/developers-collection.

Table 1
Code injection detection methods based on Machine Learning.

Year	Paper	Learning	Language
2005	*AMNESIA* [19]	*NDFA*	*SQL*
2007	*Swaddler* [35]	*libAnomaly*	*PHP*
2008	[24]	OC-SVM	PHP, SQL
2009	[30]	Clustering	SQL
2013	[20]	Bayesian	SQL
2017	HDLN [26]	Hybrid	JavaScript
2017	[25]	CNN, RNN	SQL
2017	AMODS (Y [11].	SVM	SQL, XSS
2018	DeepXSS [28]	LSTM	XSS
2018	WIRECAML [44]	DT, RF, LR, Naïve Bayse, TAN	SQL, XSS
2019	[33]	Autoencoder	SQL
2020	[74]	CNN, RNN, DT, RF	SQL
2021	[81]	MLP, CNN	SQL

the leakage functions.

The general syntax of URI in RFC 2616 [10] defines the unsafe characters that should not appear in benign queries. Queries that any of the unsafe characters are considered to be malicious and require filters to remove those queries, such as a filter, are proposed by Dong et al. (Y [11]. For example, the XSS prevention techniques typically include encoding of untrusted inputs with HTML, JavaScript, CSS, etc and sanitation of Markup. There are also anti-XSS headers available for additional security.

In general, the techniques for query languages, compiled languages, and network protocols, provided by OWASP,[6] include input validation, safe APIs, and context escape of user data. To visually illustrate the process of code injection prevention, Ray and Ligatti compare a basic algorithm in pseudocode with their own approach [6].

The machine learning intrusion detection, to be effective against the code injection, should consider its above-mentioned specificities and adjust relevant steps to maximise effectiveness.

3. Machine learning application for the code injection detection

The machine learning approaches can be presented as supervised (Regression and Classification), unsupervised (Clustering and Density estimation), semi-supervised and reinforcement learning.

Bishop defines [12] the supervised learning as applications in which the training data comprises examples of the input patterns of values paired with their corresponding output values. Goodfellow et al. define the unsupervised learning [13] as a process where the algorithm must learn "to make sense of the data without [the] guide". Semi-supervised learning contains the mixed characteristics of both, while the reinforcement learning is an intrinsically different type of machine learning [14]. In the reinforcement learning, the learner is not provided with any information about the actions to take, but instead must discover which actions yield the most reward by attempting them.

In the context of machine learning application to cyber security, the mindmap of anomaly detection methods for the web and HTTP attacks has been presented in publications (Y [15]. Out of those methods, deep learning is the one that is most commonly applied in detecting the code-related vulnerabilities. It is applied to such types of artificial neural networks as deep neural networks (DNN), deep belief networks (DBNN), recurrent neural networks (RNN) and convolutional neural networks (CNN) [16]. For example, Gu et al. [17] present a comprehensive survey on the advances in the CNN architecture and techniques. Meanwhile, the code injection chaining would require time distributed analysis, and will have to use RNN or even long short-term memory (LSTM).

The survey by Nagpal et al. [18] presents machine learning for the SQL injection detection. The authors review all the existing SQL injection attack types and their detection approaches, methods and tools, including the pre-deployment detection of vulnerable code in web applications. In 2019, Mitropoulos et al. [8] published a survey, reviewing methods of the web application attack detection, which included a class of hybrid approaches, some of which were based on machine learning (e.g., AMNESIA).

With relation to the tools, one of the first commonly presented publications on the SQL injection detection using deep learning was released in 2005. It outlined the methods to detect the SQL injection attacks using neural networks and the tool was named AMNESIA [19]. Cai et al. outlined the method of transformation of the Natural language into the SQL queries using DNN. Both of those publications also detailed preprocessing of the input data and queries. Cheon [20], who also researched the SQL injection attack detection, this time using Bayesian classifier, approached preprocessing though the dataset randomisation.

Uwagbole et al. [21] trained the support-vector machines (SVM) classifier to detect whether SQLIA is present in a web request. To train the model to a high performance, the authors present an algorithm with the dataset items input of the labelled class. It is also one of the few publications that provide insight into the real-life implementation of the SQL injection detection, using a proxy intercept web requests and analising them using the classifier.

In their survey Alwan and Younis [22] list and classify the approaches and methods of the SQLIA detection. The authors claim that none of the enumerated tools addresses the issues of the more recent types of SQLIA, e.g., fast flux SQLIA. In the advanced cases of non-typical injections, the data preprocessing can assist in successful detection.

Valeur et al. [23] proposed a method to identify queries that did not match multiple models of typical queries at runtime. As this was one of the early methods, it did not reach accuracy as high as modern approaches do, however the method has shown the potential of deep learning in malicious query detection.

In addition to the works, published in 2005 and 2007, and classified in Ref. [22] we add the methods from the publications that specifically define the machine-learning-based methods, used for the code injection detection. Table 1 presents this list.

The above list shows that there is a variety of machine learning methods applied to the code injection detection. The majority of the methods are supervised, with a half of them being deep learning methods, such as CNNs. The deep learning methods demonstrate themselves as more versatile, allowing to analyse 2D, 3D or 4D data using convolution (CNN), or LSTM, as they keep memory of the previous items from the dataset or network sessions. Detecting time-spread malicious behaviour can trigger intrusion prevention mechanisms and improve overall security of the application.

The machine learning approaches vary and the most widely used of them are reviewed below from the perspective of their use for the code analysis and intrusion detection.

3.1. Supervised learning

Supervised learning requires a dataset with marked samples that has to be collected and labelled for the optimal performance of the IDS. In the case of the code injection, the data set should contain both malicious (code injection queries) and non-malicious samples (benign queries).

Among the multiple types of models, there are Support Vector Machines, Deep Neural Networks, Decision Trees and Random Forest, Naïve Bayse, etc. In the reviewed publications, two of these models received a wider coverage.

Support vector machines (SVMs) are supervised learning models with the associated learning algorithms that analyse the data used for classification and regression analysis. For the code injection detection, SVMs were applied by Dussel et al. [24] and Dong et al. (Y [11].

Deep Neural Networks (DNN) is an artificial neural network that has layers between the input and output layers. For example, Cai et al. [25] use CNN and RNN, while Yan et al. [26] propose their own Hybrid Deep

[6] Injection Prevention Cheat Sheet, OWASP, https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html.

Neural Network (HDNN) using multiple hidden layers in order to achieve better accuracy of their system in a supervised setting. CODDLE [27] is tested using multiple types of DNN separately, while Fang et al. [28] use only LSTM.

More deep learning methods for intrusion detection have been reviewed in detail by Ferrag et al. [29].

3.2. Unsupervised learning

Unsupervised learning (also known as clustering) does not require a labelled data set and allows the system a higher level of independence in pattern recognition, leading to a higher level of autonomy. The main area of application of unsupervised learning in cyber security is anomaly detection through behavioural analysis (e.g., user and machine network activity).

Alternatively, unsupervised learning can be used for malformed code detection, which can be only insured with an adequate preprocessing. Thus, the learner can distinguish the difference between malicious and non-malicious queries or code strings. In other words, without sufficient preprocessing, the machine learning method will not be able to cluster patterns of symbols and detect any distinct difference between two (malicious and benign) queries. For example, Bockermann et all [30] applied clustering techniques to the SQL injection detection.

In the pursuit of the deeper understanding of DNN, the autoencoders were introduced. Autoencoder is a symmetric DNN trained to have a target value equal to the given input value [31]. Autoencoders consist of two DNNs, encoder and decoder, and can be easily applied to resolve the problem of data compression (e.g., in the image and video recognition, large data transfer, analysis of specific types of cyber attacks). Furthermore, autoencoders allow the researcher to input raw data, including software code and scripts. For instance, Pan et al. [32] presented a method of detection of attacks on the web applications based on the Robust Software Modelling Tool (RSMT), which analyses the call traces as runtime behaviour of web applications. RSMT uses an autoencoder for analysis and call graphs as data. In RSTM (F. [33], used by Pan et al. a smaller amount of the labelled data is used to calculate reconstruction error of the autoencoder and establish a numeric value (threshold) to distinguish between normal and abnormal behaviours.

Clustering can arguably be more efficient than the supervised learning for the detection of previously unknown attacks.

3.3. Reinforcement learning

As per [14], reinforcement learning is the learning of a mapping from the input situations (events) to the output actions so as to maximise a measurable reward or the reinforcement signal. In practice, it is a reward-based system, that works as a semi-supervised approach to learning. The machine-learning-based system learns from the environment and gets rewarded for correct predictions, and penalised for incorrect predictions.

To the extent of our knowledge, there have not yet been any applications of the reinforcement learning to the code injection detection, as its successful applications to intrusion detection have started only recently. In 2020, Lopez-Martin et al. [34] presented a survey on the application of reinforcement learning to the issues of network intrusion detection in general. They state that for the code injection detection, reinforcement has to be limited to only one numeric reinforcement value, i.e. successful detection.

3.4. Adversarial approach and datasets generation

Most of the reviewed publications confirm that the scarcity of the datasets remains one of the biggest issues for machine learning in cyber security, which has been also confirmed in the recent survey by Ferrag et al. [29]. Finding a suitable dataset for a specific type of cyber attacks is a complex task. With many datasets already developed and emerging sophisticated attacks, bigger and more comprehensive datasets are required on a daily basis. More often than not, the acquisition of code injection training samples requires a manual dataset composition. Data can originate from a variety of collection methods, which can be as follows:

- Manually collected from tutorials and public access (e.g, Cheat Sheets, GitHub, “hacker challenge” websites, etc.)
- Honeypot logs
- Payloads and recordings of the automatic tools (e.g., SQLmap, SQLninja, OWASP Xenotix XSS Exploit Framework, XSSer, Metasploit Framework, etc.)
- Publicly available data sets (e.g., CIC IDS, NSL-KDD, etc.)
- Sample generation methods, like Generative Adversarial Network (e.g., PyGenerator[7]).

A variety of automated tools can be used for data collection. For example, for he event collection Cova et al. [35] use modified Zend engine. Aceto et al. [36] collected data from a mobile service provider and from various applications on Android devices, which resulted in a binary dataset that was published by Yao et al. [37]. After collection or generation is complete, the raw data has to be preprocessed and then features must be extracted to transform any type of data into a numerical form that can be analysed during future stages.

To date, one of the most popular datasets in cyber security is KDDCUP′99[8]. It has been analysed by a large number of researchers for many types of cyber attacks and machine learning implementations. For example, using that dataset Li et al. (Y [38]. proposed a method based on autoencoders and deep belief network, with the final accuracy up to 92, 1%. Autoencoders were used for query processing in the presented hybrid malicious code detection scheme based on AutoEncoder and Deep Belief Networks. Preprocessing for KDD has been reviewed in detail by Molina-Coronado et al. [39].

In the context of code injection attacks, collection of benign and malicious queries can yield a limited number of samples, that is insufficient for the effective model training. Generation of new datasets from the existing data can resolve this challenge. A trend in the recent years shows the use of Generative Adversarial Networks (GAN), however, there are multiple ways to successfully generate new samples from the existing ones.

One of the issues with the generative approach (using GAN or any other algorithm) is related to the generation of the working samples. For example, in the image recognition setting the difference in images might be infeasible, while in the network intrusion setting, not all generated malicious samples can be successful attacks.

Adversarial inputs created by introducing permutations of patterns from the already existing datasets can easily subvert their predictions. The 2017 report by Biggio et al. [40] highlights common misconceptions related to the evaluation of machine learning methods and approaches for security applications.

3.4.1. Dataset generation

When the data is insufficient for the DNN adequate training, the researchers might seek to collect a bigger dataset, or to generate additional data from the already existing dataset by adding permutations in specific values.

Cheon et al. [20] develop their own way of generating samples based on the existing query templates, by randomising numbers, usernames, passwords, email addresses, etc. Edalat et al. [9] outline the dataset generation approach to multiply the number of malicious samples using concolic input generation [41].

[7] Isao Takaesu, PyGenerator, GitHub, 2017, https://github.com/13o-bbr-bbq/machine_learning_security/tree/master/Generator.

[8] https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Uwagbole et al. [21] explored the generation of a new dataset using the known attack patterns, such as SQL tokens and symbols present during the injection process.

Unlike in the SQL injection, generating the entire traffic units is a more sophisticated task. The definition of a specific traffic object determines how the raw traffic is segmented into multiple traffic units [42] and presented as different types of traffic. The publications describing the application of deep learning for the traffic analysis[9] [43–45] considered either flows or bidirectional flows as an object of classification, with the exception of Lotfollahi et al. [46], where the object of classification is the single packet.

In all the above cases, instances can occur that are longer or shorter than the considered fixed-length data inputs. In such cases, longer instances are truncated to the designed length of bytes or packets, whereas in the case of shorter instances, padding is applied in the majority of the surveyed works.

Generated code injection payloads can be used for the development of IDS, but can be also used to design advanced attack frameworks and fuzzers, that can bypass the most complex conventional filters and IDS.

Some datasets can be large in size, and universities and research centres provide the already pre-processed datasets and pre-trained models for further research (e.g., University of New Brunswick[10]).

3.4.2. Non-machine learning samples generation methods

There also exist the non-machine learning based methods that include fuzzing for the attack generation and testing of the experimental system. They can also be used to create a dataset, that is usually stored in text files or database files. One of the most popular tools in fuzzing SQL databases is SQLmap,[11] presented by Damele and Stampar. SQLmap is now a part of the majority of the existing security testing toolkits and operating systems. This non-machine learning toolkit can be used for the attack generation to test prototype systems against real life attacks. Cheon et al. [20] use SQLmap for the evaluation of their method. Edalat et al. [9], while also researching the SQL injection detection, instead of machine learning methods, use taint analysis for the evaluation of their ConsiDroid.

The automatic scripts, toolsets, and virtual networks can be used for the attack sample generation. Tools like TCPReplay can be used to emulate the traffic activity from an already existing **.pcap* file. Taken a step further, to generate a CICIDS2017 dataset,[12] the University of Brunswick used a Virtual Machine with Kali Linux as an attack station. The network traffic was recorded during several days of simulations on a virtual network. After that, the IP addresses were removed from the dataset (CSV file), anonymizing the data. The newer versions of 2018 and 2020 are also available.

3.4.3. Increasing complexity of samples

Using the adversarial approach, it is possible to create additional permutations and multiply the dataset, increasing the number of samples and improving the detective capabilities of the IDS. However, in real-life attacks, the malicious actors can use sophisticated techniques to avoid simple IDS. To be able to detect more sophisticated malicious queries, generative adversarial approach can be used to increase the complexity of code samples, both malicious and non-malicious.

Salgado [47] outlined the techniques to optimise and obfuscate SQLIA, providing insight on how to potentially generate the dataset of advanced SQLIA for training of neural models for IDS. To the extent of our knowledge, there are no public mentions of this method application, even though it has promising results for the generation of adversarial code injection samples.

3.4.4. Machine learning for attack simulation

Machine learning application for dataset generation vary from sample multiplexing using the existing smaller datasets to various attack simulations [40] and synthetic payloads [48].

Kreuk et al. [48] introduced an approach for generating adversarial samples for the discrete input sets, such as binaries. The functional malicious binaries are modified by introducing a small sequence of bytes to the binary source file. The modified files are then detected as benign by the IDSs, while preserving their malicious features. The approach was applied to an end-to-end CNN malware detector and presents a high evasion rate. Their research also showed that generated malicious payload can be placed in different positions of the same file and across different files.

Russell et al. [49] propose an adversarial learning approach without requiring paired labelled examples or source and target domains to be injections. They compare their method to the other approaches that require labelled pairs, and report almost similar performance.

As an example of a specific vulnerability, addressed by the Adversarial Machine Learning approach, we can mention the Return Oriented Programming (ROP). The ROP attack is an exploit technique (usually based on the buffer overflow vulnerability) that allows an attacker to execute code in the presence of security defences such as executable space protection and code signing. Li et al. (X [50]. present Ropnn, which uses the address space layout guided disassembly and DNNs to detect the ROP payloads in HTTP requests, PDF files, and images. The disassembler treats the input data as code pointers. The reported detection rate of such approach is 98.3% while using the adversarial dataset. Sun et al. (Y [51]. conducted a research in the injection attacks for the visual data using reinforcement learning. The research is not directly related to the code injection and cyber security, yet it provides insight into the possibilities and methods of injecting any data into the approach.

Adversarial approach can also create a dataset of attack recordings, that have not been publicly used yet. For example, a time of evasion attacks has been presented by Biggio and Roli [40,52], as well as a few common misconceptions, highlighting the vulnerability of machine learning detection and classification algorithms to new attacks.

3.5. Parameter fine-tuning

Parameter fine-tuning is an important step in machine learning. It can be defined as the adjustment of parameters of the model for optimal performance. A parameter or a hyperparameter is a value that impacts the learning process. Every machine learning method has a set of such (sometimes unique) parameters. For example, Artificial Neural Networks (ANN) have different layers of different types, number of neurons, activation functions for each layer, optimisers, loss function for the output layer, batch size, and number of epochs. Tuning hyperparameters aims to improve the speed of and overcome the limitations of small datasets [53].

Truncation: One of the methods is to truncate the output layer of the already trained ANN and replace it with another layer with same activation that is directly relevant to a selected problem.

Learning rate: Many methods have a variable parameter called “learning rate”, that can be increased to improve the speed of learning, or reduced to solve the issue of model overfitting in certain cases.

Freezing the weights: In addition, the weights of the initial few layers of a DNN can be fixed or frozen, as they represent the curves and edges, pre-trained for a particular task. After this is done, “frozen” weights do not change, and the network will only readjust the weights of the subsequent layers, relevant to a specific dataset.

[9] https://www.blackhat.com/docs/us-15/materials/us-15-Wang-The-Applications-Of-Deep-Learning-On-Traffic-Identification-wp.pdf.

[10] Intrusion Detection Evaluation Dataset (CICIDS2017), University of New Brunswick, https://www.unb.ca/cic/datasets/index.html.

[11] Bernardo Damele, A.G., Stampar, M.: Sqlmap: automatic SQL injection and database takeover tool, 2012, SQLmap, http://sqlmap.sourceforge.net/.

[12] Intrusion Detection Evaluation Dataset (CICIDS2017), University of New Brunswick, https://www.unb.ca/cic/datasets/index.html.

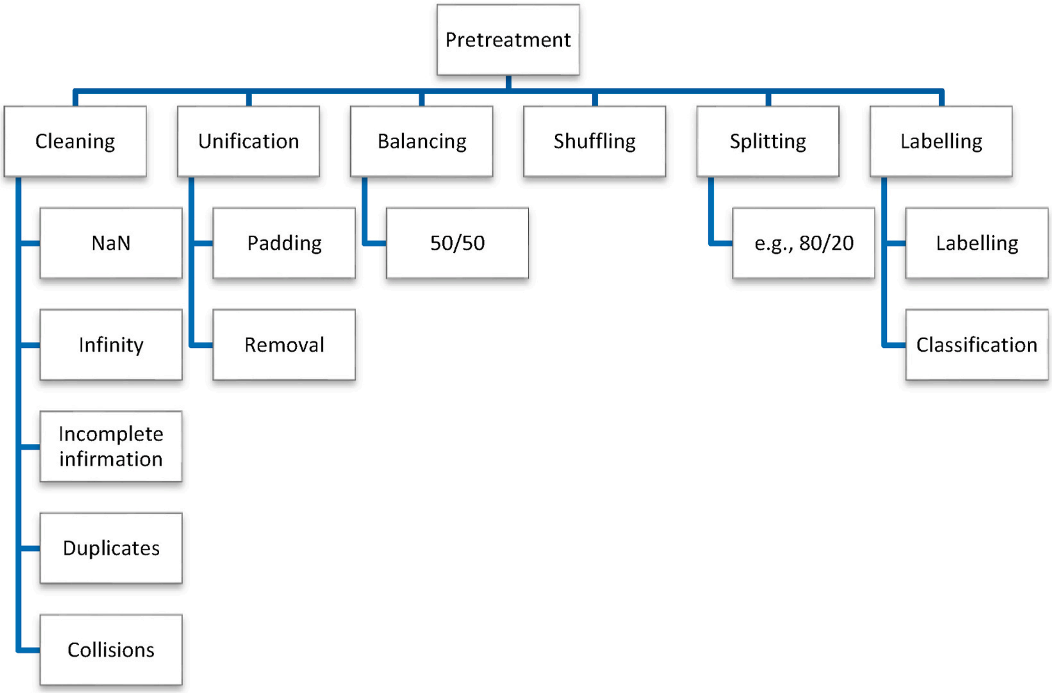


Fig. 1. Data pretreatment methods. Source: authors.

The tunability of machine learning hyperparameters has been reviewed and calculated by Probst et al. in Refs. [54,55], as well as in Ref. [56]. Hyperparameter fine-tuning can be done automatically using programming libraries and built-in functions, such as Keras Tuner.[13] Fine-tuning is also accessible via online services like Comet. ml,[14] OpenML,[15] and Weights&Biases.[16]

Literature review revealed that the process of machine learning has been widely covered in modern publications. Among the most broadly reviewed areas are the machine learning general approaches, algorithms, training, post-processing and result visualisation methods (e.g. Refs. [3,29,57–59]). However, the data preprocessing has been the least researched, especially in the area of the Network Intrusion Detection. It is at the same time one of the most critical and labour and time consuming steps (over 50% of total effort) within the entire data analysis process [1,2]. The next Sections will provide an overview of the preprocessing methods and their application, and will highlight its critical role in enhancing detection.

4. Critical role of preprocessing in the detection rate enhancement

Preprocessing is the process of data transformation and its conversion into another form, that aids the neural network with the learning process. It changes the initial data in such a way that it allows the machine learning algorithms to analyse the data. Preprocessing can include any process that involves data manipulation before the correlation process (using machine learning or any other algorithm), and task-specific, as the architecture of the neural networks can be tailored for detection (of any type), classification (detection and attack identification), or clustering. For example, it can include data cleaning and refinement (filtering, reducing, reshaping, encoding, etc), handling missing attributes, imbalanced dataset and elimination of noise or outliers.

Chitraa and Davamani [60] define preprocessing as a series of processing of web log file covering data cleaning, user identification, session identification, path completion and transaction identification. Tomar and Agarwal in their survey [3] define preprocessing as a step, that is used to enhance the reliability of the collected data. Some authors (e.g. Ref. [61]) never actually use the term “preprocessing”, instead they use the term “parcing” for that stage, in some cases completely replacing the entire process with automated libraries (e.g., ScaPy).

The revision of the selected academic publications revealed a wide variety of diverse preprocessing methods, that need to be summarized and classified for effective use in intrusion detection. The way in which the input data is pre-processed (cleaned, reduced, reshaped, encoded, etc.) significantly influences and affects the performance of the machine learning techniques employed to detect code injection attacks. For example, some of the features in the data set may be random in nature and corrupt the output results, or, similarly, may be more impactful and introduce biases.

For the purpose of this survey and based on the reviewed publications, we propose to subdivide preprocessing into the data pretreatment followed by the actual data treatment consisting of feature-based and encoding-based preprocessing as two different ways of data optimisation and transformation.

The following subsections will address in more details the Data Pretreatment, Feature-based and Encoding-based preprocessing and highlight their specificity in relation to the intrusion detection

Table 2
Classification of Tokenization in data pretreatment.

Classification	List
Start Label	<script>,<frame>,<img>,<body>,etc
End Label	</script>,</frame>,</body>,etc
Windows Event	onerror = , onload = , onblur = , oncut = , etc
Function Name	alert (, prompt (, String.fromCharCode (, etc
Script URL	javascript:, vbscript:, etc
Others	>), \#, etc

Source [28].

[13] Keras Tuner, https://keras-team.github.io/keras-tuner/.
[14] https://www.comet.ml/site/.
[15] https://www.openml.org/.
[16] https://wandb.ai/site.

effectiveness. This will be followed by summarising and classifying how different machine-learning-based code injection attack detection techniques take advantage of (or rely upon) different input data preprocessing techniques.

4.1. Pretreatement

Pretreatement may be defined as the process of preparing raw data for any further manipulations and correlations, including its cleaning and creating a database. This process follows the input data collection, that once selected for the dataset, has to be composed and prepared for the neural network to train on.

Following the literature review, we present the following classification of the pretreatment methods currently in use (See Fig. 1).

Pretreatement can include several or all of the following steps:

- Remove Null, NaN and Inf
- Remove incomplete information
- Remove duplicates, as they make neural model biased
- Remove inputs with different outputs
- Reshape (e.g., numpy array reshape)
- Balance (to make the neural network unbiased)
- Shuffle (many datasets are sorted, and before splitting they can be randomly shuffled for the experiment)
- Split on training and testing
- Split the training dataset further, if a neural network is analised for incremental learning, or multiple neural networks are trained for ensembles.
- Labelling may complete the pretreament process with the subsequent data classification and creation of the dataset.

It is worth noting that pretreatment does not involve any transformation of the data values. In specific cases some steps of pretreatment might be repeated after the feature-based and encoding-based preprocessing as well. For example, in the cases when the data cleaning (pretreatment), and its further encoding (preprocessing), produce duplicates, those have to be removed again (pretreatment).

Dong et al. (Y [11]. mention data normalisation, that may reduce the number of samples, by techniques, such as decoding the ASCII characters, transforming to lowercase, un-escaping, removing queries whose length is less than four symbols, etc. After such normalisation, the initially different queries may become identical, as the distinguishing items were removed. However, transformation like this might be irreversible. For example, Valeur et al. [23] replace certain values in the query white spaces. When queries with different values but similar structures are pretreated using this method, they become similar or even duplicates. With an approach like this it may be impossible to backtrack the event record and identify the origin or condition of the attack.

Deep learning is considered to be the most advanced technique and effective against the sophisticated attacks and constantly evolving attack vectors [29]. One of the earliest attempts to specifically detect code injection attacks (specifically SQL injections) using deep learning was AMNESIA, a method developed in 2005 by Halfond and Orso [19]. The entire preprocessing was similar to the one used by Gould et al. [62], except the use of NDFAs in AMNESIA instead of DFAs.

Cova et al. [35] use a modified Zend engine to perform a linear scan of the sequence of statements, that identifies the corresponding basic blocks in the SQL queries, and associates a unique ID with each of them.

In the method, suggested by Valeur et al. [23], the "event provider" forwards queries to the parser (pretreatment) that feeds it to the feature selector (preprocessing). The parser processes each incoming SQL query generating its high level-view. The parser outputs this representation as a sequence of tokens. Constants are the only elements of an SQL query that should contain the user supplied input. Thus, each token is meant to have a flag which indicates whether the token is a constant or not. To form a dataset, tokens representing database field names are augmented by a datatype attribute (e.g., varchar).

Abdulhammed et al. [63] apply a "preprocessing function" (pretreatment, according to the presented classfication) to the CICIDS2017 dataset by mapping the IP address. The mapped IP includes the Source IP Address (Src IP) as well as the Destination IP Address (Dst IP). As IP addresses cannot be processed by any classifier without pretreatment, they are converted either to a decimal format, or to assigned relevant ID numbers.

As an example of a successful data preprocessing, Dong et al. (Y [11]. use the logs (6.11 Gb) where during pretreatment, that included cleaning, normalisation, and filtering, the data effectively reduced in volume by 4 times while ensuring its high quality. Another example of a complete data preprocessing is the above-mentioned CICIDS 2017 dataset, which is reduced from 50 Gb of pcap files to less than 1 Gb csv file with all the features describing bidirectional traffic flow.

4.1.1. Extraction of data

The pretreatment begins with the data extraction from the collected data. Jayaprakash [64] presents a survey on the data pretreatment using web logs as a source of data. Their key steps are as follows:

- Data Cleaning
- User Identification
- Session Identification
- Path Completion
- Transaction Identification

Source data can be extracted from event history and system logs. For example, Jovanen et al. [65] acquired the log files containing textual data (strings) describing requests sent from the user to the server. N-gram analysis (See Section 4.3) is used for extracting meaningful features from the data, which means that in the pre-processing phase, textual logs are transformed into numerical matrices to facilitate the subsequent analysis phases. Dong et al. (Y [11]. examine the Web server logs to collect successful GET requests: requests with the return code that is equal to or greater than 200, and less than 300. Then, static requests (e.g.,.html,.wav,.txt,.jpg) are removed. Finally, the remaining successful GET requests are parsed to extract queries like $parameter1 = value1\&parameter2 = value2$. Dong et al. claimed to achieve 94.79% accuracy.

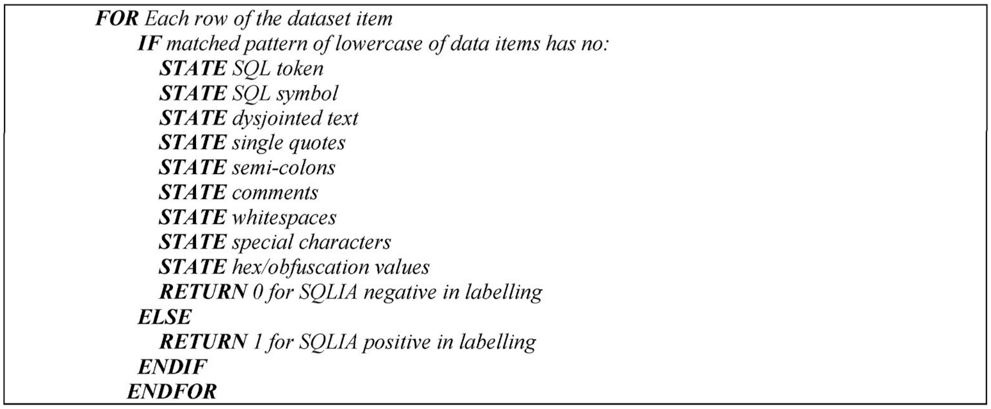
Dussel et al. [24] introduced a payload-based anomaly detection method through adding structural information from a protocol analyser, with the detection of SQL and PHP code injection attacks. The goal of that particular research was to analyse the network traffic based on the grammatical characteristics of an underlying protocol.

The event history (logs) is usually combined in a log format, which includes the system activity information. For examples, web servers collect information about the connections and data transferred (e.g, IP address, timestamp, the actual HTTP request, Apache server response code and user agent header field) [65]. These logs might contain several actual intrusions, especially inside the HTTP requests that are not static, i.e., they contain dynamic parameters that depend on the user input.

Meanwhile, Fang et al. [28] focus on generalisation and tokenization for pretreatment. Their DeepXSS uses a series of customizing regular expressions based on the features of the scripting language to tokenize the input data. The classification of the tokenization is presented in Table 2. DeepXSS relies on Word2vec,[17] a deep learning tool released by Google in 2013.

As per Uwagbole et al. [21], creating a dimension to accommodate the size of data by selecting the next hashing bits, that fit the dataset, can sometimes generate too much dimension and sparse data which are reduced by a filter-based featured selection, that leaves only top relevant vectors. The filter-based selection is used to achieve the reduced

[17] word2vec, Google, 2013, https://code.google.com/archive/p/word2vec/.



```
FOR Each row of the dataset item
    IF matched pattern of lowercase of data items has no:
        STATE SQL token
        STATE SQL symbol
        STATE dysjointed text
        STATE single quotes
        STATE semi-colons
        STATE comments
        STATE whitespaces
        STATE special characters
        STATE hex/obfuscation values
        RETURN 0 for SQLIA negative in labelling
    ELSE
        RETURN 1 for SQLIA positive in labelling
    ENDIF
ENDFOR
```

Fig. 2. Dataset feature labelling procedure. Source: [21].

computation complexity without affecting the prediction accuracy in the classification process. In their work, the Chi-squared scored function is used to rank the top 5000 hashing features in descending order to return the most appropriate labels to improve the SQLIA prediction accuracy.

Yan et al. [26] extract all JavaScript code written by developers (JavaScript libraries like JQuery are excluded) in the application to create their dataset.

It is worth mentioning beyond the scope of our research, that before the pretreatment stage the collected data has to go through a quality check and proofing. Ehrlinger et al. [66] identified 667 tools to evaluate the data quality and composed a review of ways to measure data quality, clean it, and monitor.

4.1.2. Cleaning

The removal of values that cannot be processed is a tedious process that can delay the research.

It is generally accepted, that removing duplicates reduces the size of dataset and reduces bias from the training process. If the dataset contains many duplicates, with a single training pass the DNN “sees” the same sample multiple times, each time readjusting the function.

Some programming and scripting languages have built-in or loadable functions for this step (e.g., R, Numpy in Python), however, out-of-the-box functions do not always work, and at the present stage the cleaning solutions have to be scripted manually.

The above-mentioned scripts aim to identify the values that match or do not match a specific regular expression, detect new lines to form database rows, and then remove duplicating data, if necessary. Values such as “NaN”, “Infinity”, or unmatching value type can be either completely removed during this step, or transformed into a numeric value during the encoding-based preprocessing stage.

Tomar et al. [3] present a notion of dirty data. They review the methods of handling the missing attributes in dirty data (filter-based method, imputation method, and embedded method), also the methods of handling noisy data.

Jayaprakash [64] extracts data from the web logs by using commas and quotation marks as separators, thus, removing them, for example, using spreadsheet software [64,67], and by this shaping data into a matrix.

In their experiment, Dong et al. (Y [11]. collect successful GET requests. They subsequently remove static requests (e.g.,.html,.wav,.txt,.jpg), and finally, the remaining successful GET requests are parsed to extract queries like *parameter1* = *value1&parameter2* = *value2.*

4.1.3. Unification

In this paper we present the data unification as the data transformation into a similar form. For example, if the mass media dataset contains a first name that is followed by a surname, it should be the case for every single cell in the dataset. Specific to code injection, values should be of specific type (e.g., some python libraries have issues with processing float32 and float64 together, even though both are the same numeric values).

In deep learning some neural networks (e.g., CNN) require rows with data to be the same length, thus, empty cells have to be filled with zeroes or other values, that represent an empty slot.

Fang et al. [28] pre-treat data by following specific steps: replace the various URLs in the input data with “*http://website*”; replace the numbers in the input data with “0”; replace the string as the function parameters with “*param_string*”. In addition, other special characters such as blank characters or control characters are removed. As per Dong et al. (Y. [11], data normalisation helps tighten the input space, including the decoding printable ASCII characters, un-escaping, transforming to lowercase and removing queries whose length is less than four items. After normalisation, initially different queries may appear identical, similar to the approach by Fang et al. [28]. The identical duplicates are then removed.

4.1.4. Balancing

Another important factor for a high-quality data is its balanced representation meeting the requirements of the set parameters. Let the dataset contain 80% of malicious samples and 20% benign samples (e.g. Ref. [21]). If the neural network predicts all the queries in the dataset as 1, the statistics will calculate the accuracy of the detection rate as 80%. To avoid biases in statistical calculations, and if the dataset allows it, it is advised to have the equal amount of malicious and benign samples. Though in practice, the script would simply compare the size of two subsets (benign and malicious) and remove (randomly or as per a specific logic) the excessive rows.

The issue of imbalanced datasets is discussed by Chawla et al. [68] who provide insight into solutions to over-sampling and under-sampling. They use multiple imbalanced datasets for their experiments.

Tomar et al. [3] suggest the sampling method and algorithm adjustment, to address the issues of imbalanced datasets.

Even though there are methods, like Synthetic Minority Over-Sampling Technique (SMOTE), for a guided methodological sampling, it is still advised to manually balance the dataset, before the evaluation in experiments sensitive to statistics.

Uwagbole et al. [21] balanced their dataset items and in their experiment these actions improve both the trained model recall and precision.

However, in the real-life systems the malicious and non-malicious traffic are never balanced, and developers of defence systems have to use machine learning methods for non-balanced dataset preprocessing.

4.1.5. Sampling and splitting on training and testing sets

In the method proposed by Uwagbole et al. [21], the text

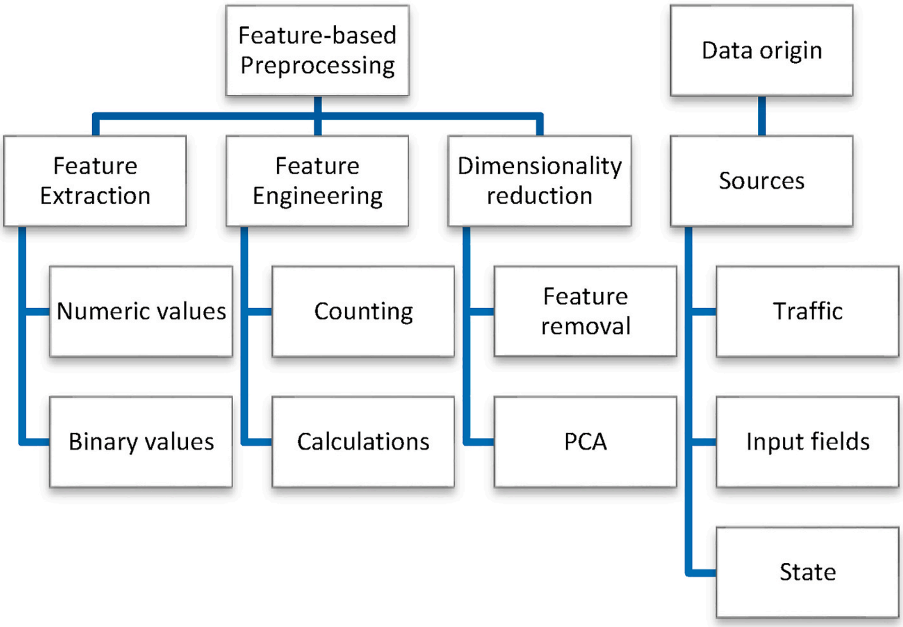


Fig. 3. Fearure-based preprocessing and sources of data. Source: authors.

preprocessing involves a regular expression pattern matching. The imbalanced data set was corrected with the Synthetic Minority Over-Sampling Technique (SMOTE) by Chawla et al. [68] to have a dataset split equally as malicious and benign. These actions have proven to increase the trained model recall and precision.

Most commonly used dataset splitting proportions are 80/20 (e.g. Ref. [68]) or 70/30 (e.g. Ref. [63]), which can be implemented through scripts, spreadsheet software or otherwise using functions from the programming libraries (e.g., SciKit).

4.1.6. *Labelling and classification*

For the supervised learning the data has to be labelled as well as marked. That is more a requirement, than an improvement, thus it is a part of pretreatment. However, if it is possible to classify an attack in a broader way then just a binary classification (benign or malicious), that might either increase or reduce the overall precision, on the case-specific basis.

The basic way to mark the dataset for code injection detection is “0″ or "-1″ for the benign traffic or “1″ for the attack, based on the activation function (e.g., Sigmoid or Tanh). For the code injection classification, the types have to be encoded in a sequence of ones and zeroes (e.g., [1 0] for SQL injection and [0 1] for XSS).

Once the data is shaped and is already in a database or a text file, it can be transformed into a format that can be readable by the neural networks.

Uwagbole et al. [21] label the dataset using the algorithm presented in Fig. 2.

Another example is the work of Gao et al. [69], who use the NSL-KDD dataset and mark the normal traffic as 0, DoS as 1, Probe as 2, etc.

4.1.7. *Negative impact of pretreatment*

There are a few warning messages with regards to the data pretreatment that have been identified through the literature review. During the pretreatment step, the data may be prepared in ways that may contain programming biases, resulting in a poor performance of the system. For example, during balancing the intuitive solution would be to remove the random values from the dataset. However, removal of sequences at random may lead to the removal of patterns with a high impact on the learning process, and leave only more complex cases, which require more training to be identified.

Shuffling the dataset in an uneven manner would result in a biased statistics, similar to the bias in an unbalanced dataset training. Furthermore, splitting dataset 50/50 as compared to 80/20 in an unbalanced way may increase that bias. For example, if the training dataset contains only non-malicious samples, the model will not be able to detect malicious samples during the testing phase.

Padding incomplete information, instead of removing, it is a common practice, yet it may result in a poor performance, as the neural networks may ignore certain features entirely.

After the entire data preparation process is complete, the dataset may contain additional unintended duplicates, as a result of feature extraction and encoding. Thus, it is advised to perform additional checks for duplicates at the end.

4.2. *Feature-based preprocessing*

In this survey, we propose to single out a feature-based preprocessing as a separate step to form a dataset. Its function is to select, extract, or generate features. Selecting the right features for a particular task can improve the performance by reducing the amount of noise and randomness, as well as minimising complexity and generally reducing the amount of data required for effective analysis. The raw data may already have them but, more often, needs features to be calculated. For example, the number of packets in a single session is a feature that has to be counted. These values can be further used to generate an average payload size in a session as another feature. To be able to detect an attack, with a sufficient number of features, there is no need for the analysis of the payload data.

“A Comprehensive Survey on Data Preprocessing Methods in Web Usage Mining” outlines the ways of data processing from web logs [64]. This approach is further enhanced in a survey by Ramirez-Gallego et al. [2], that also outlines dimensionality and instance reduction techniques for the data mining.

4.2.1. *Use of a feature-based preprocessing*

Feature-based preprocessing is used for refinement and enhancement of data though the following:

- Removing excessive data and minimising the volume of data - to emphasize only relevant values, and disregards irrelevant ones completely (e.g., turn pcaps into lists of sessions).
- Replacing specific data - (e.g., to reduce or increase the number of variations of the same value)
- Enhancing random data with additional knowledge - to add additional values, with information about the other values, thus increasing the overall knowledge of the data.
- Highlight specific features to study their correlation.

As per our proposed classification, the feature-based preprocessing stage can be grouped into three substages (Fig. 3):

- Feature extraction (mandatory)
- Feature engineering (optional)
- Dimensionality reduction (optional)

The feature-based prerocessing and feature extraction create a training dataset in the exact way the research is aiming for. For example (Fig. 3), the network traffic files (i.e., *.cap, *.pcap, *.pcapng, etc.) can be stripped of IP addresses and payloads, and merged in sessions, creating a table that depicts the traffic flow in the network. Data from the input fields can have similar features as the natural language (e.g., words, punctuation, etc.), while the machine state can have a pattern of configuration values as features.

In the code injection detection, the feature-based preprocessing extracts data from request payloads, request headers, and web application input fields, and then analyses it, classifies (if needed), and encodes the query for the neural model predictions. The approach is generally similar to the use of the text data for machine learning, meaning the code symbols are analysed as a sequence.

The feature-based preprocessing module usually applies one or several conversion methods (e.g., Encoding, Reshaping, Hashing, Reducing Dimensionality via mathematical analysis), transforms text data into a sequence of numeric values (either binary or decimal), and forwards it to the DNN.

To detect the code injection attack, a neural network has to have samples of both malicious and non-malicious code, whether it is for supervised or unsupervised learning. In supervised learning it will also need to be labelled, in order to correlate a sequence of features with the desired output, while in unsupervised learning, the system will correlate the existing features and potentially detect anomalies on its own.

The raw input dataset can contain the following:

- Values of variables and input queries
- Single line of code
- Blocks of code (line by line with the memory of previous code)
- Full execution analysis (state)

Dong et al. (Y [11]. claim, that the key steps of query processing are data cleaning, data normalisation, and character filtering. Data cleaning and data normalisation in our classification are a part of pretreatment, while “character filtering” would be classified as feature extraction.

4.2.2. Feature selection

Feature selection, also known as feature extraction, is a technique which is used to derive values of information from an initial set of the measured pre-treated data. Those features are meant to facilitate the subsequent analysis and learning. Further dimensionality reduction techniques may include the Principle Component Analysis (PCA), K-means clustering and other clustering techniques, etc.

The preprocessing approach by Valeur et al. [23] replaces variables with the “empty space” placeholders, generating a “skeleton query” and creating a set of query “profiles”.

The feature extraction process by Dussel et al. [24] maps the application layer messages, such as the HTTP requests, into a feature space in which similarity between messages can be represented with a numeric value. The method uses the data structures such as suffix trees or hash tables.

Bockermann et al. [30] proposed an approach of using clustering for

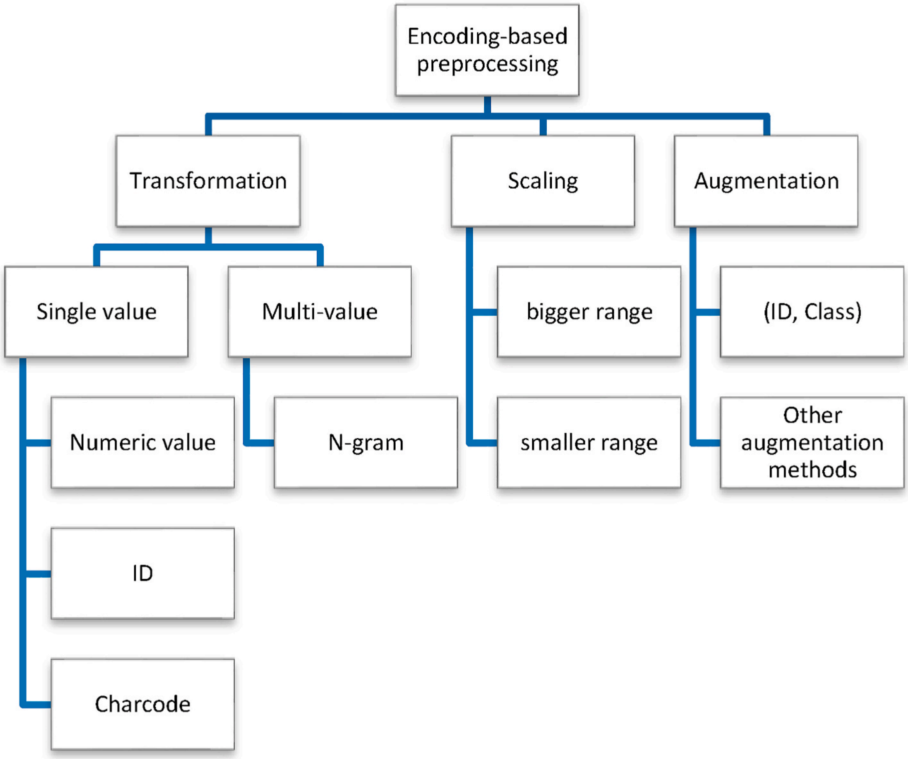


Fig. 4. Encoding-based preprocessing characteristics. Source: authors.

modelling the SQL statements to parse the tree structure of SQL queries as features, e.g., for correlating SQL queries with applications and distinguishing malicious and non-malicious queries.

Tomar et al. [3] use feature selection for dimensionality reduction using two methods: feature ranking and feature subset selection. They propose feature selection using four different approaches - filter method, wrapper method, embedded method, and hybrid method, while also presenting comparative characteristics of these four methods.

Features can be hashed, which allows to translate the dataset text items into a binary vector matrix suitable for training a model in Machine Learning. The hashing procedure creates an input matrix or vectors that make a lookup of feature weights faster by augmenting the string comparison with a hash value comparison. Applying hashing to text features improves performance and scalability in the big data predictive analytics lacking in the existing SQLIA signature-based detection [3].

Alternative to hashing, the N-grams can be used to transform the text or a numerical feature vector into numeric values. This feature construction transforms each query into a series of N-grams (binary-based and frequency-based). An N-gram is an N-character slice of a string [70], and its analysis has been successfully applied to intrusion detection. Furthermore, it is a fully automatic method and requires no prior knowledge about the target web application and target attacks.

Juvonen et al. [65] applied the N-gram analysis for feature extraction and dimensionality reduction. Their paper focuses on the HTTP log data, that was previous reviewed by Ingham and Inoue [71]. As a test case, Uwagbole et al. [21] built a web application that expects a dictionary word list as variables. They used hashing tables and N-grams for preprocessing.

Yan et al. [26] presented a Hybrid Deep Learning Network (HDLN), a more efficient model, based on the approach by Xiao et al. (X [72]. In addition to the already existing features in the dataset, Xiao et al. suggest to extracts the new features from the Abstract Syntax Tree of JavaScript in hybrid applications through the feature space generation and feature selection.

The feature selection process can be manual or automated. Manually they can be selected via trial and error, while automatically, the features can be selected using Recursive Feature Elimination, LassoCV, etc. In 2019 [36], proposed a deep learning approach as a viable strategy to design a mobile traffic classifiers based on automatically-extracted features, able to cope with the encrypted traffic, and reflecting their complex traffic patterns. For example:

- Recursive Feature Elimination [73] [18]
- Lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) [74].
- Statistical tests:
  o *SelectKBest*[19]
  o *Feature importance*[20]

4.2.3. Mathematical dimensionality reduction

Pretreatment and feature selection and/or engineering structure the data and reduce the volume of data for training. For example, CICFlowMeter[21] allows to reduce traffic recordings (*.pcaps) to a text-file dataset of biflow traffic features (78–82 features). This manipulation reduces the size of the dataset from Gigabytes of network traffic files into a spreadsheet, containing all the features required for a successful network traffic flow analysis.

With the features already extracted and dimensionally reduced (e.g., using principle component analysis, Kmeans clustering, etc.), a smaller neural network can be used, to analyse the patterns faster, without the need to detect any new features. For example [67], use the principal component analysis (PCA) as a form of preprocessing for Support Vector Machines (SVM). With a smaller dataset, the bigger neural networks will not train as effectively due to insufficient samples. The CICIDS2017 dataset contains sessions described in 78 features, however, PCA can reduce the number of values down to 10 (principal components) without a feasible loss in accuracy [63]. Gao et al. [69] use PCA for the experiments with NSL-KDD.

To address the size of datasets and reduce resource requirements, the mathematical analysis can extract additional correlations between features (usually involving frequency analysis) and reduce the number of values in the dataset. Very popular methods for the dimensionality reduction in the Network Intrusion Detection are K-means clustering and PCA.

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for the cluster analysis in data mining. PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

Tomar et al. [3] split dimensionality reduction into the Hard Dimension Reduction Problem and Soft Dimension Reduction Problem. A study by Abdulhammer et al. [63] uses PCA as one of the methods for dimensionality reduction in Intrusion Detection.

4.2.4. Negative impact of feature-based preprocessing

Identification, selection, extraction, and engineering of features can cause the lack or surplus of “knowledge” for the machine learning algorithm to be trained, that results in training biases. The selection of features that describe data has been addressed in many academic papers. Incorrect selection of features may result in unwanted randomness and “confusion” for the neural networks, and thus, reduced accuracy. Furthermore, decision to use the excessive number of features increases the complexity of the preprocessing process, and the size of the dataset, resulting in the reduced training speed for neural networks.

Similar to the selection of excessive features, the data enhancement method and feature engineering add classification attributes that inflate data. Those methods may result in adding drawbacks, instead of improving accuracy.

4.3. Encoding-based preprocessing

The intuitive approach to the encoding of the text-based data would be to use the natural language processing or simple per-symbol encoding. However, preprocessing for the network packet headers, payloads, and code is different, as, unlike in the natural language, it cannot interpret a simple set of symbols between two white spaces or between a white space and a punctuation symbol.

To finalise the full preprocessing cycle - the transformation of the extracted features into numeric values, that the machine learning methods can train on - we suggest the following there substages (Fig. 4).

4.3.1. Transformation

It is common for the pattern of symbols and numbers to be directly converted to the sequence of numbers. In the context of cyber security, encoding can transform a single item (word or symbol in the header, payload, or line of code) into one of the following:

- single digit or value;
- sequence of digits or values;
- pair with a digit or value and a classification marker;
- hash, further converted into one of the above.

[18] https://www.scikit-yb.org/en/latest/api/model_selection/rfecv.html.

[19] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.SelectKBest.

[20] https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html.

[21] https://www.unb.ca/cic/research/applications.html.

For example, if every symbol in an SQL query is converted into a relevant charcode, it does not enhance data, and potentially may lead to the reduction of the quality of data, as changing the case of symbols may lead to different numeric sequence (i.e., “A" is 101, “a" is 141). In cases when the use of this transformation is detrimental to the research, the previously mentioned issue can be mitigated by converting every symbol into lower or upper case. Another example, Ling and Wu [75] convert a flag type data into single digits (tcp-1, icmp-2, and so on).

Conversion of values like IP addresses can be another cornerstone. The intuitive conversion of IPv4 into a decimal value, and further division by 10 power 10, would not work as intended. For example, 192.168.1.1 would be into 3232235777, and 192.167.1.1 into 3232170241. Once scaled between 0 and 1, those two decimal values are very close to each other, even though they represent different subnetworks, and the neural network may not be able to notice any difference.

Keywords and special characters. Multiple methods try to address the conversion via the detection of keywords, operators, and escape symbols [20,23,27].

The method by Cheon et al. [20] presents a pre-processing algorithm (“converter”), that dissects SQL query into keywords and identifies them by the position of blank spaces: right side, both, or none. The converter is used to convert the HTTP parameters into numeric attributes. These attributes are supplied for the Classifier as features (length of parameters and the number of keywords). The keywords contain words and symbols in the SQL statements, like commas, equal signs, quotation marks, “SELECT”, “UNION” and so on. There are three types of keywords according to the SQL statements.

The research by Uwagbole et al. [21] uses a combination of data composed of the extracted dictionary wordlist with words and unique SQL tokens extracted from the MSSQL reserved keywords. The dataset items are labelled based on the exhibition of the SQLIA types characteristics which are: the presence of the SQL tokens in the injection point, disjointed text, single quotes, semicolons, comments, hex, etc. The data set items labelling is represented in the binary values of 0 (SQL negative) or 1(SQL negative).

4.3.2. *Scaling*

After all the features are presented in a numeric code, each value has to be further scaled between 0 and 1 using built-in functions (e.g., Numpy interp, Sklearn MinMaxScaler), or manually (e.g., with mathematical manipulations).

Abdulhammed et al. [63] use their equation to re-scale the features in the dataset based on the minimum and maximum values of each feature. Some features in the original dataset vary between [0, 1] while other features vary between [0,∞). Therefore, these features are normalised to restrict the range of the values between 0 and 1, which are then processed by the auto-encoder for feature reduction.

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

4.3.3. *Data augmentation through encoding*

The goal of data augmentation can be either to improve the accuracy (additional data is added), or to make the real-life data resemble closer the training dataset (as some of the variables might be missing).

Data can be enhanced with additional knowledge. For example, words can be converted into a single value in a specific range and special symbols into values in a completely different, more recognizable range, artificially increasing the difference between the groups (or classes) of values. Furthermore, additional values may be added for the classification to establish a new sequence of values.

Oppose to Refs. [20,27] classify and augment data with additional knowledge, converting the keywords and symbols into pairs of values (value, type).

Table 3
Classification summarising the existing approaches to the data preprocessing. Source: authors.

	Pretreatment						Feature-based preprocessing			Encoding-based preprocessing		
	Cleaning	Unification	Balancing	Shuffling	Splitting	Labelling	Feature extraction	Feature engineering	Dimensionality reduction	Transforming	Scaling	Augmentation
[3]	V		.				V	.	V	V	V	
[64]	V	V					V	.		.		
[67]	.				.	.	.	V	V			
(Y [11].	V	.				V	V	V	V	V		.
[28]	.	V				.	V	V		V		V
[21]	.	.	V	.	V		V	V		V		
[68]	.		V		V	V	V					
[63]	.	V	V		V	V	V		V		V	
[69]	.	.	V			.	V	V	V	V	V	
[24]	.	V					V	.		V		
[23]	.	V				.	V			V		V
[20]	.	V				.	V	V		V		V
[27]	.	V			V	V	V	V		V		V
[30]	.	V					V	V		V		
[65]		V					V	V	V	V		
[75]		V					V			V	V	
[44]	V	.	.				V	V		V	.	
[33]		.					.	V	.	V	.	
[74]	V	.					V	V	V	.		
[81]	V	V					.	V		.	V	

V – explicitly mentioned in detail or unique approach suggested.
. – mentioned or performed.
[empty space] – not mentioned.

Table 4
Percentage of preprocessing stages mentioned in the literature.

Stage	Substage	At least mentioned	Explicit mention or unique method
Pretreatment	Cleaning	85%	30%
	Unification	85%	55%
	Balancing	30%	20%
	Shuffling	5%	0%
	Splitting	25%	20%
	Labelling	45%	20%
Feature-based preprocessing	Feature Extraction	100%	85%
	Feature Engineering	80%	65%
	Dimensionality Reduction	40%	35%
Encoding-based preprocessing	Transformation	85%	70%
	Scaling	35%	25%
	Augmentation	25%	20%

There is a variety of methods on the network data visualisation, that can be used to generate static images, and then further use various automated tools (e.g., imgaug[22], AutoAugment [4], K-correct [5]) and image augmentation techniques presented in the surveys by Shorten and Khoshgoftaar [76] and Mikolajckzyk and Grochowski [77] to further improve the detection rates.

4.3.4. Other encoding-based methods

Automated approach. As oppose to the manual tool development of encoding methods, it is always possible to use the already existing methods of parcing data from the raw files to numeric sequences. For example, several parsers for the SQL dialects were used by Ref. [61] to obtain a parse tree. As stated by Bockermann et al. [30], the basic idea of Buehrer et al. [61] is to detect SQL injection attacks by means of changes in a queries syntax tree. Usually complex parsers are automatically generated based on a given grammar description using tools such as *yacc, antlr* or *javacc*. However, those parsers did not provide a satisfactory numeric sequence, and had to be further vectorised.

Gao et al. [69] use one-Hot-encoding to process text values into binary sequence.

Translation using natural language approach. Traditional ways of preprocessing are limited to a conversion of any data into a set of numeric values. However, in unique applications additional case-specific steps are required.

Translation of natural language into commands, images, lines of code, and samples for the dataset has been researched since the early days of programming languages. To illustrate the mechanism in the context of the survey, we have studied the SQL query as an example. The approach of the natural language translation into SQL queries can be potentially abused by attackers to bypass filtering.

Cai et al. [25] outlined a method of the Natural language conversion into the SQL queries using CNN and RNN, as well as provided techniques for preprocessing and postprocessing of the input data. Alternatively, a sequence-to-sequence model can be used for semantic preprocessing (C. [78], (L [79–81].

Masking is another method, used for the natural language processing. Liang et al. [82] used masking for symbolic parsing by storing key-variable pairs in the memory. Masking presented by Cai et al. [25] supports more complex operations, covering both short-term and long-term dependencies. Moreover, the authors emphasize that the grammar structure of SQL is known to be more complicated than the logical forms used in semantic parsing.

4.3.5. Negative impact of encoding-based preprocessing

Incorrect encoding may result in misrepresentation of data for the Machine Learning algorithm to correlate features. Typically, simple conversion of payload symbols or values into charcode is an intuitive choice. However, for the code injection detection, this approach may give results close to random (50–60% accuracy of models). We have also identified a few ways in which the improper encoding might reduce the accuracy:

- Scaling in a wrong range of values causes unintended clustering of features, which results in wrong correlation in features.
- Introduction of additional mathematical conversions, such as hashing the values, may create misrepresentation of features.
- In a variety of automated solutions, the raw data is often unclean and confusing, and automated preprocessing may detect non-existent features.

Those negative effects and their combinations impact the performance of the final system in some very specific ways, yet the outcomes may be unpredictable and counter-intuitive during troubleshooting and debugging.

4.4. Classification of existing preprocessing methods

The revised sources highlighted a wide diversity in the approaches to all machine learning stages preliminary to the training, ranging from the use of raw data to the synthetically engineered features, from the manual to automated preprocessing. The in-depth analysis of the 20 academic publications, selected based on their relevance to the code injection attack detection, and suggested machine learning methods have demonstrated that preprocessing was successfully used to maximise the effectiveness of their system.

The following similarity characteristics were observed:

- Only a few methods explicitly use preprocessing for data augmentation in order to improve performance and/or accuracy.
- None of the papers describes every step of the proposed classification of the full preprocessing cycle.
- Pretreatment is almost exclusively mentioned for Cleaning and Unification.
- Feature-based preprocessing is mentioned both as Feature extraction and Feature Engineering.
- Feature extraction is the only step that is mentioned in every publication.
- Dimensionality reduction is normally mentioned as a unique optional method, not as an essential one.
- Encoding-based preprocessing is typically mentioned as transformation of data into numeric values.
- None of the papers show samples of the dataset in the state between preprocessing and training.

It is our understanding that to ensure maximum efficiency the full preprocessing cycle should be respected. This cycle is suggested based on our previous research in the respective area [27]. To maximise the performance efficiency of machine learning for the code injection detection, we propose six stages of Pretreatment, three stages of Feature-based preprocessing, and three stages of Encoding-based preprocessing. This approach, however, can be also applied to all other cyber security methods in general. Revealing the present inconsistencies and to harmonise the preprocessing methodology, we map the existing approaches against our proposed classification to identify the gaps.

The above classification demonstrates that some of the stages were used explicitly while the others were only briefly mentioned without detail or not mentioned at all, potentially indicating that they were omitted. Based on Table 3 we conduct a statistical assessment of the use of preprocessing stages in the analysed works (Table 4).

[22] ImgAug, GitHub, https://github.com/aleju/imgaug.

As per the assessment, Feature-based preprocessing is the most represented stage. Feature extraction is mentioned in 100% of the analysed works, while shuffling of the dataset is the least mentioned substage. Cleaning, Unification, and Encoding Transformation are mentioned in 85% of the methods, followed closely by Feature Engineering in 80%. The remaining stages and substages are mentioned in less than a half of the analysed publications.

The above observations reveal the missing knowledge that could enhance the performance and/or accuracy of the designed intrusion detection systems.

5. Conclusion

Based on the selected academic publications, the survey explored the existing approaches to the applications of machine learning for the detection of code injection attacks, with special attention to deep learning. We identified at least 13 different methods of code injection detection using various types of machine learning. Deep learning is observed being the most used approach.

The stages of machine learning for intrusion detection have been further revised and least researched have been identified. The findings confirm that the scarcity of the datasets remains one of the biggest and most common challenges for the adoption of machine learning in cyber security. The findings also revealed that data preprocessing being one of the most critical stages in machine learning, lacks consistency in the approaches in the context of code injection attack detection.

Limitations and negative impacts in which preprocessing may reduce overall performance and detection rate have been also collected and documented in the study.

The suggested classification of a full preprocessing cycle for the code injection detection will result in the harmonisation of the approaches to this stage and will improve the accuracy and performance of the machine-learning-based Intrusion Detection Systems. The proposed consecutive stages of data preprocessing can be further used by machine learning researchers and practitioners for other cyber security needs, such as network traffic analysis and intrusion detection. And finally, the presented classification will allow to better understand the role of deep learning in the code injection attack detection process in machine-learning-based Intrusion Detection Systems.

Credit author statement

The research was conducted by Dr Stanislav Abaimov during the final year of the PhD programme under the supervision and guidance of Professor Giuseppe Bianchi. The presented survey was initiated as a joint idea by both authors and as a continuation of the previous research, conducted by Dr Stanislav Abaimov and Professor Giuseppe Bianchi, which was published as a novel method.

Declaration of competing interest

The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Acknowledgments

This work is partially supported by the EU Commission in the frame of the Horizon 2020 project SPARTA (grant #830892).

References

[1] Pyle Dorian. Data preparation for data mining. Morgan Kaufmann Publishers; 1999.
[2] Ramírez-Gallego Sergio, Krawczyk Bartosz, García Salvador, Woźniak Michał, Herrera Francisco. A survey on data preprocessing for data stream mining: current status and future directions. Neurocomputing 2017;239(May):39–57. https://www.sciencedirect.com/science/article/abs/pii/S0925231217302631.
[3] Tomar Divya, Agarwal Sonali. A survey on pre-processing and post-processing techniques in data mining. International Journal of Database Theory and Application 2014;7(4):99–128. https://doi.org/10.14257/ijdta.2014.7.4.09.
[4] Cubuk Ekin D, Zoph Barret, Mane Dandelion, Vasudevan Vijay, Quoc V Le. AutoAugment: learning augmentation policies from data. May, http://arxiv.org/abs/1805.09501; 2018.
[5] Hoyle Ben, Michael Rau Markus, Bonnett Christopher, Seitz Stella, Weller Jochen. "Data augmentation for machine learning redshifts applied to SDSS galaxies," january. 2015. https://doi.org/10.1093/mnras/stv599.
[6] Ray Donald, Ligatti Jay. Defining code-injection attacks. ACM SIGPLAN notices, vol. 47. New York, NY, USA: ACM PUB27; 2012. p. 179–90. https://doi.org/10.1145/2103621.2103678.
[7] Mitropoulos Dimitris, Spinellis Diomidis. Fatal injection: a survey of modern code injection attack countermeasures. PeerJ Computer Science 2017;2017(11):e136. https://doi.org/10.7717/peerj-cs.136.
[8] Mitropoulos Dimitris, Louridas Panos, Polychronakis Michalis, Dennis Keromytis Angelos. Defending against web application attacks: approaches, challenges and implications. IEEE Trans Dependable Secure Comput 2019;16(2): 188–203. https://doi.org/10.1109/TDSC.2017.2665620.
[9] Edalat Ehsan, Sadeghiyan Babak, Ghassemi Fatemeh. ConsiDroid: a concolic-based tool for detecting SQL injection vulnerability in android apps. http://arxiv.org/abs/1811.10448; 2018. 1, 10.
[10] Fielding Roy, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, Berners-Lee Tim. Hypertext transfer protocol–HTTP/1.1. 1999.
[11] Dong Ying, Zhang Yuqing, Ma Hua, Wu Qianru, Liu Qixu, Wang Kai, Wang Wenjie. An adaptive system for detecting malicious queries in web attacks. Sci China Inf Sci 2018;61(3). https://doi.org/10.1007/s11432-017-9288-4.
[12] Bishop Christopher M. Pattern recognition and machine learning (information science and statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc; 2006.
[13] Goodfellow Ian, Bengio Yoshua, Courville Aaron. Deep learning (adaptive computation and machine learning series). Nature 2016;521. https://doi.org/10.1038/nmeth.3707.
[14] Sutton Richard S, Barto Andrew G. Reinforcement learning: an introdcution. MIT Press; 2018.
[15] Dong Ying, Zhang Yuqing. "Adaptively detecting malicious queries in web attacks," january. 2017. https://doi.org/10.1007/s11432-017-9288-4.
[16] Cireşan Dan, Meier Ueli, Juergen Schmidhuber. Multi-column deep neural networks for image classification. February; 2012. http://arxiv.org/abs/1202.2745.
[17] Gu Jiuxiang, Wang Zhenhua, Kuen Jason, Ma Lianyang, Shahroudy Amir, Shuai Bing, Liu Ting, et al. Recent advances in convolutional neural networks. Pattern Recogn 2018;77(May):354–77. https://doi.org/10.1016/J.PATCOG.2017.10.013.
[18] Nagpal Bharti, Chauhan Naresh, Singh Nanhay. A survey on the detection of SQL injection attacks and their countermeasures. Journal of Information Processing Systems 2017;13(4):689–702. https://doi.org/10.3745/JIPS.03.0024.
[19] Halfond William GJ, Orso Alessandro, Abdoulaye Kindy Diallo, Sakib Khan Pathan Al. AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. Int J Commun Network Inf Secur 2013;5. https://doi.org/10.1145/1101908.1101935.
[20] Cheon Eun Hong, Huang Zhongyue, Yon Sik Lee. Preventing SQL injection attack based on machine learning. International Journal of Advancements in Computing Technology 2013;5(9):967–74. https://doi.org/10.4156/ijact.vol5.issue9.115.
[21] Uwagbole Solomon Ogbomon, Buchanan William J, Fan Lu. Applied machine learning predictive analytics to SQL injection attack detection and prevention. In: Proceedings of the IM 2017 - 2017 IFIP/IEEE international symposium on integrated network and service management; 2017. https://doi.org/10.23919/INM.2017.7987433. 1087–90.
[22] Alwan Zainab S, Younis Manal F. Detection and prevention of SQL injection attack: a survey. Int J Comput Sci Mobile Comput 2017;6(8):5–17. https://www.ijcsmc.com/docs/papers/August2017/V6I8201701.pdf.
[23] Valeur Fredrik, Mutz Darren, Vigna Giovanni. A learning-based approach to the detection of SQL attacks. https://doi.org/10.1007/11506881_8; 2005. 123, 140.
[24] Düssel Patrick, Gehl Christian, Laskov Pavel, Rieck Konrad. In: Incorporation of application layer protocol syntax into anomaly detection. Berlin, Heidelberg: Springer; 2008. p. 188–202. https://doi.org/10.1007/978-3-540-89862-7_17.
[25] Cai Ruichu, Xu Boyan, Zhang Zhenjie, Yang Xiaoyan, Li Zijian, Liang Zhihao. An encoder-decoder framework translating natural language to database queries. In: IJCAI international joint conference on artificial intelligence 2018-july; 2018. 3977–83.
[26] Yan Ruibo, Xiao Xi, Hu Guangwu, Peng Sancheng, Jiang Yong. New deep learning method to detect code injection attacks on hybrid applications. J Syst Software 2018;137(March):67–77. https://doi.org/10.1016/j.jss.2017.11.001.
[27] Abaimov Stanislav, Bianchi Giuseppe. CODDLE: code-injection detection with deep learning. IEEE Access 2019;7:128617–27. https://doi.org/10.1109/access.2019.2939870.

[28] Fang Yong, Yang Li, Liu Liang, Huang Cheng. DeepXSS. In: Association for computing machinery (ACM); 2018. p. 47–51. https://doi.org/10.1145/3194452.3194469.

[29] Ferrag, Amine Mohamed, Maglaras Leandros, Moschoyiannis Sotiris, Janicke Helge. Deep learning for cyber security intrusion detection: approaches, datasets, and comparative study. Journal of Information Security and Applications 2020;50(February):102419. https://doi.org/10.1016/j.jisa.2019.102419.

[30] Bockermann Christian, Martin Apel, Meier Michael. Learning SQL for database intrusion detection using context-sensitive modelling (extended Abstract). In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) 5587 LNCS; 2009. p. 196–205. https://doi.org/10.1007/978-3-642-02918-9_12.

[31] Vincent Pascal, Larochelle Hugo, Lajoie Isabelle, Bengio Yoshua, Manzagol Pierre-Antoine. Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. Undefined 2010. https://www.semanticscholar.org/paper/Stacked-Denoising-Autoencoders%3A-Learning-Useful-in-Vincent-Larochelle/e2b7f37cd97a7907b1b8a41138721ed06a0b76cd.

[32] Pan Yao, Sun Fangzhou, White Jules, Douglas C Schmidt, Jacob Staples, Lee Krause. Detecting web attacks with end-to-end deep learning. Acm 2019;1–14. https://www.dre.vanderbilt.edu/~schmidt/PDF/machine-learning-feasibility-study.pdf.

[33] Sun Fangzhou, Zhang Peng, White Jules, Schmidt Douglas, Jacob Staples, Lee Krause. "A feasibility study of autonomically detecting in-process cyber-attacks." in *2017 3rd IEEE international Conference on cybernetics (CYBCONF)*, 1–8. IEEE; 2017. https://doi.org/10.1109/CYBConf.2017.7985745.

[34] Lopez-Martin Manuel, Carro Belen, Sanchez-Esguevillas Antonio. Application of deep reinforcement learning to intrusion detection for supervised problems. Expert Syst Appl 2020;141(March):112963. https://doi.org/10.1016/j.eswa.2019.112963.

[35] Cova Marco, Balzarotti Davide, Felmetsger Viktoria, Vigna Giovanni. Swaddler: an approach for the anomaly-based detection of state violations in web applications. Recent Advances in Intrusion Detection 2007:63–86. https://doi.org/10.1007/978-3-540-74320-0_4.

[36] Aceto Giuseppe, Ciuonzo Domenico, Montieri Antonio, Pescape Antonio. Mobile encrypted traffic classification using deep learning: experimental evaluation, lessons learned, and challenges. IEEE Transactions on Network and Service Management 2019;16(2):445–58. https://doi.org/10.1109/TNSM.2019.2899085.

[37] Yao Hongyi, Ranjan Gyan, Tongaonkar Alok, Liao Yong, Mao Zhuoqing Morley. SAMPLES: self adaptive mining of persistent LExical snippets for classifying mobile application traffic. In: Proceedings of the 21st annual international conference on mobile computing and networking - MobiCom '15. New York, New York, USA: ACM Press; 2015. p. 439–51. https://doi.org/10.1145/2789168.2790097.

[38] Li Yuancheng, Ma Rong, Jiao Runhai. A hybrid malicious code detection method based on deep learning. International Journal of Security and Its Applications 2015;9(5):205–16. https://doi.org/10.14257/ijsia.2015.9.5.21.

[39] Molina-Coronado Borja, Mori Usue, Alexander Mendiburu, Miguel-Alonso José. Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process," january. 2020. http://arxiv.org/abs/2001.09697.

[40] Kolosnjaji Bojan, Demontis Ambra, Biggio Battista, Maiorca Davide, Giacinto Giorgio, Eckert Claudia, Roli Fabio. Adversarial malware binaries: evading deep learning for malware detection in executables. In: European signal processing conference. European Signal Processing Conference, EUSIPCO; 2018. https://doi.org/10.23919/EUSIPCO.2018.8553214. 2018-Septe:533–37.

[41] Sadeghi Alireza, Bagheri Hamid, Garcia Joshua, Malek Sam. A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. IEEE Trans Software Eng 2017;43(6):492–530. https://doi.org/10.1109/TSE.2016.2615307.

[42] Dainotti Alberto, Pescape Antonio, Claffy Kimberly. Issues and future directions in traffic classification. IEEE Network 2012;26(1):35–40. https://doi.org/10.1109/MNET.2012.6135854.

[43] Wei Wang, Zhu Ming, Zeng Xuewen, Ye Xiaozhou, Sheng Yiqiang. Malware traffic classification using convolutional neural network for representation learning. In *2017 international Conference on information networking (ICOIN)*, 712–17. IEEE; 2017. https://doi.org/10.1109/ICOIN.2017.7899588.

[44] Wang Wei, Zhu Ming, Wang Jinlin, Zeng Xuewen, Yang Zhongzhen. End-to-End encrypted traffic classification with one-dimensional convolution neural networks. In: 2017 IEEE international conference on intelligence and security informatics (ISI). IEEE; 2017. p. 43–8. https://doi.org/10.1109/ISI.2017.8004872.

[45] Lopez-Martin Manuel, Carro Belen, Sanchez-Esguevillas Antonio, Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. IEEE Access 2017;5. https://doi.org/10.1109/ACCESS.2017.2747560. 18042–50.

[46] Lotfollahi Mohammad, Shirali Hossein Zade Ramin, Jafari Siavoshani Mahdi, Saberian Mohammdsadegh. "Deep packet: a novel approach for encrypted traffic classification using deep learning," september. 2017. http://arxiv.org/abs/1709.02656.

[47] Salgado By Roberto. SQL injection optimization and obfuscation techniques. 2013.

[48] Kreuk Felix, Barak Assi, Aviv-Reuven Shir, Moran Baruch, Pinkas Benny, Joseph Keshet. "Deceiving end-to-end deep learning malware detectors using adversarial examples," february. 2018. http://arxiv.org/abs/1802.04528.

[49] Russell Rebecca, Kim Louis, Hamilton Lei, Lazovich Tomo, Jacob Harer, Ozdemir Onur, Paul Ellingwood, McConley Marc. Automated vulnerability detection in source code using deep representation learning. In: Proceedings - 17th IEEE international conference on machine learning and applications, ICMLA 2018. Institute of Electrical and Electronics Engineers Inc; 2019. https://doi.org/10.1109/ICMLA.2018.00120. 757–62.

[50] Li Xusheng, Hu Zhisheng, Fu Yiwei, Chen Ping, Zhu Minghui, Liu Peng. ROPNN: detection of ROP payloads using deep neural networks. " July; 2018. http://arxiv.org/abs/1807.11110.

[51] Sun Yiwei, Wang Suhang, Tang Xianfeng, Hsieh Tsung-Yu, Honavar Vasant. "Node injection attacks on graphs via reinforcement learning," september. 2019. http://arxiv.org/abs/1909.06543.

[52] Biggio Battista, Corona Igino, Maiorca Davide, Nelson Blaine, Nedim Srndic, Laskov Pavel, Giacinto Giorgio, Roli Fabio. Evasion attacks against machine learning at test time. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) 8190 LNAI (PART 3); 2017. p. 387–402. https://doi.org/10.1007/978-3-642-40994-3_25.

[53] Vabalas Andrius, Gowen Emma, Poliakoff Ellen, Casson Alexander J. Machine learning algorithm validation with a limited sample size. PloS One 2019;14(11): e0224365. https://doi.org/10.1371/journal.pone.0224365.

[54] Probst Philipp, Bischl Bernd, Anne-Laure Boulesteix. Tunability: importance of hyperparameters of machine learning algorithms. J Mach Learn Res 2018;20 (February). http://arxiv.org/abs/1802.09596.

[55] Probst Philipp, Wright Marvin N, Laure Boulesteix Anne. Hyperparameters and tuning strategies for random forest." wiley interdisciplinary reviews: data mining and knowledge discovery. Wiley-Blackwell; 2019. https://doi.org/10.1002/widm.1301.

[56] Zhou Yadi, Cahya Suntara, Combs Steven A, Nicolaou Christos A, Wang Jibo, Desai Prashant V, Shen Jie. Exploring tunable hyperparameters for deep neural networks with industrial ADME data sets. J Chem Inf Model 2019;59(3):1005–16. https://doi.org/10.1021/acs.jcim.8b00671.

[57] Hamed Tarfa, Ernst Jason B, Kremer Stefan C. A survey and taxonomy of classifiers of intrusion detection systems. In: Computer and network security essentials. Springer International Publishing; 2017. p. 21–39. https://doi.org/10.1007/978-3-319-58424-9_2.

[58] Brundage Miles, Avin Shahar, Clark Jack, Toner Helen, Eckersley Peter, Garfinkel Ben, Allan Dafoe, et al. "The malicious use of artificial intelligence: forecasting, prevention, and mitigation," february. 2018. http://arxiv.org/abs/1802.07228.

[59] Liu Hongyu, Lang Bo. Machine learning and deep learning methods for intrusion detection systems: a survey. Appl Sci 2019;9(20):4396. https://doi.org/10.3390/app9204396.

[60] Chitraa V, Davamani Dr Antony Selvdoss. A survey on preprocessing methods for web usage data. CoRR 2010;abs/1004.1. http://sites.google.com/site/ijcsis/.

[61] Buehrer Gregory T, Weide Bruce W, Sivilotti Paolo AG. Using parse tree validation to prevent SQL injection attacks. In *Proceedings of the 5th international Workshop on software Engineering and middleware - sem '05*, vol. 106. New York, New York, USA: ACM Press; 2005. https://doi.org/10.1145/1108473.1108496.

[62] Gould C, Su Z, Devanbu P. Static checking of dynamically generated queries in database applications. In: Proceedings. 26th international conference on software engineering, 645–54. IEEE Comput. Soc; 2004. https://doi.org/10.1109/ICSE.2004.1317486.

[63] Abdulhammed Razan, Hassan Musafer, Ali Alessa, Faezipour Miad, Abuzneid Abdelshakour. Features dimensionality reduction approaches for machine learning based network intrusion detection. Electronics 2019;8(3):322. https://doi.org/10.3390/electronics8030322.

[64] Jayaprakash Sujith. A comprehensive survey on data preprocessing methods in web usage minning, vol. 6; 2015. www.ijcsit.com.

[65] Juvonen Antti, Sipola Tuomo. Anomaly detection framework using rule extraction for efficient intrusion detection. 2014. October, http://arxiv.org/abs/1410.7709.

[66] Ehrlinger Lisa, Rusz Elisa, Wolfram Wöß. "A survey of data quality measurement and monitoring tools," july. 2019. http://arxiv.org/abs/1907.08138.

[67] Raja M Chithik, Munir Ahmed Rabbani M. Combined analysis of support vector machine and principle component analysis for IDS. In: Proceedings of the international conference on communication and electronics systems, ICCES 2016. Institute of Electrical and Electronics Engineers Inc; 2016. https://doi.org/10.1109/CESYS.2016.7889868.

[68] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. In: Journal of artificial intelligence research; 2011. https://doi.org/10.1613/jair.953. June.

[69] Gao Xianwei, Shan Chun, Hu Changzhen, Niu Zequn, Liu Zhen. An adaptive ensemble machine learning model for intrusion detection. IEEE Access 2019;7: 82512–21. https://doi.org/10.1109/ACCESS.2019.2923640.

[70] Cavnar William B, Trenkle John M. N-Gram-Based text categorization. https://www.semanticscholar.org/paper/N-gram-based-text-categorization-Cavnar-Trenkle/1c610a7e67b578de78436e8959b3ea462ca3e56d; 1994.

[71] Ingham Kenneth L, Inoue Hajime. Comparing anomaly detection techniques for HTTP. In: Recent advances in intrusion detection. Berlin, Heidelberg: Springer Berlin Heidelberg; 2007. p. 42–62. https://doi.org/10.1007/978-3-540-74320-0_3.

[72] Xiao Xi, Yan Ruibo, Ye Runguo, Li Qing, Peng Sancheng, Jiang Yong. Detection and prevention of code injection attacks on HTML5-based apps. In: Proceedings - 2015 3rd international conference on advanced cloud and big data. Institute of Electrical and Electronics Engineers Inc; 2016. https://doi.org/10.1109/CBD.2015.48. CBD 2015, 254–61.

[73] Guyon Isabelle, Weston Jason, Barnhill Stephen, Vapnik Vladimir. Gene selection for cancer classification using support vector machines. Mach Learn 2002;46(1–3): 389–422. https://doi.org/10.1023/A:1012487302797.

[74] Tibshirani Robert. Regression shriknage and selectino via the Lasso. J Roy Stat Soc B 1996;58(1):267–88. https://doi.org/10.2307/2346178.

[75] Ling Jie, Wu Chengzhi. Feature selection and deep learning based approach for network intrusion detection. In: Proceedings of the 3rd international conference on mechatronics engineering and information Technology (ICMEIT 2019). Paris, France: Atlantis Press; 2019. https://doi.org/10.2991/icmeit-19.2019.122.

[76] Shorten Connor, Khoshgoftaar Taghi M. A survey on image data augmentation for deep learning. Journal of Big Data 2019;6(1):60. https://doi.org/10.1186/s40537-019-0197-0.

[77] Mikolajczyk Agnieszka, Grochowski Michal. Data augmentation for improving deep learning in image classification problem. In: 2018 international interdisciplinary PhD workshop (IIPhDW), 117–22. IEEE; 2018. https://doi.org/10.1109/IIPHDW.2018.8388338.

[78] Xiao Chunyang, Dymetman Marc, Gardent Claire. Sequence-based structured prediction for semantic parsing. Proceedings of the 54th annual Meeting of the Association for computational linguistics, vol. 1. Stroudsburg, PA, USA: Association for Computational Linguistics; 2016. https://doi.org/10.18653/v1/P16-1127. Long Papers), 1341–50.

[79] Dong Li, Lapata Mirella. "language to logical form with neural attention," january. 2016. http://arxiv.org/abs/1601.01280.

[80] Guu Kelvin, Pasupat Panupong, Liu Evan Zheran, Liang Percy. "From language to programs: bridging reinforcement learning and maximum marginal likelihood," april. http://arxiv.org/abs/1704.07926; 2017.

[81] Chen Ding, Yan Qiseng, Wu Chunwang, Zhao Jun. SQL injection attack detection and prevention techniques using deep learning. In: Journal of physics: conference series. IOP Publishing Ltd; 2021. https://doi.org/10.1088/1742-6596/1757/1/012055. 1757:12055.

[82] Liang Chen, Berant Jonathan, Le Quoc, Forbus Kenneth D, Ni Lao. "Neural symbolic machines: learning semantic parsers on freebase with weak supervision," october. 2016. http://arxiv.org/abs/1611.00020.