



Early web application attack detection using network traffic analysis

Branislav Rajić¹ · Žarko Stanisavljević¹ · Pavle Vuletić¹

Published online: 26 October 2022

© The Author(s), under exclusive licence to Springer-Verlag GmbH, DE 2022

Abstract

The number of deployed web applications and the number of web-based attacks in the last decade are constantly increasing. One group of tools that gained the attention of cyber security specialists are Dynamic Application Security Testing (DAST) tools, which is used to assess the security posture of web applications. DAST tools have similar purpose for web applications as network scanners and mappers have for local networks and computers—to scan web applications, enumerate as much as possible information from them and this way potentially reveal existing vulnerabilities. The tools are not only used by security analysts but also by the attackers in the reconnaissance and enumeration phases of the attack. This paper analyses DAST tools' network behaviour patterns, characteristic features that distinguish them from other traffic and methods to detect their operation using classical supervised machine learning methods. Unlike most of the work related to web application security and web application attack detection, which relies on HTTP logs, the research presented here is based on network traffic traces and flow statistics. This allows malicious scanning detection on the network traffic path even in the case of encrypted web traffic. Experimental results show that an accurate and reliable detection of four analysed DAST tools, ZAP, Nikto, Vega and Arachni, is possible. Flow classification of the existing DAST tools has high precision because DAST tools still do not deploy any mechanisms to hide their operation and mimic web application browsing by human users. Additionally, the paper contains an analysis of fast malicious behaviour detection through an analysis of the detection of malicious behaviour, while the flows are still active. The experimental results show that it is possible to detect malicious behaviour with a relatively high accuracy after only 15 packets in a flow.

Keywords Web application attack · Web scanners · Machine learning · Intrusion detection · DAST tools

1 Introduction

In the last several years, since 2017, there has been a decreasing trend in the number of native applications in some of the most popular application stores (Google Play and Apple stores) and a gradual shift towards the use of web-based applications [1]. Also, the number of web sites is constantly growing. In a single year (2016–2017), this number grew from around 1 to 1.8 billion. The increase in the number of web sites and the complexity of web applications and their widespread services create challenges in securing them from

the threats with diverse motivations from financial or reputational damage to the theft of critical or personal information [2].

One strategy of securing web access is encrypting the data on the fly by using TLS. Google reported that since 2017, more than 90% of the web traffic towards their services is encrypted, and that today, 97% of the most popular web sites default to HTTPS [3]. The same trend of increased use of encrypted traffic towards web sites is observed in other networks. The use of HTTPS solved problems such as data-in-transit security and web site authentication. However, despite the increased use of encryption, ENISA reported a 52% increase in the number of web application attacks in 2019 compared with 2018, with an SQL injection, directory traversal, cross-site scripting (XSS), broken authentication and session management as the most common attack vectors [4].

Encrypting web traffic adds to the complexity of web traffic inspection and attack detection. With the use of TLS, all

✉ Branislav Rajić
rb215010p@student.etf.bg.ac.rs

Žarko Stanisavljević
zarko.stanisavljevic@etf.bg.ac.rs

Pavle Vuletić
pavle.vuletic@etf.bg.ac.rs

¹ School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, Belgrade, Serbia

HTTP methods and artefacts, which are in the research community commonly used to detect the web attacks, become available only on the web session end points. The use of middleboxes like web application firewalls (WAFs) as the most common web application protection strategy is more difficult. The operation and management of the WAF require additional skills in the web application provider company, and if the application provider decides to outsource its operation, there is an issue of adding more processors of potentially sensitive and private data used by the web application.

Unlike the main trends in web application attack detection research, which is based on HTTP methods analysis, in this paper, we explored the potential to detect malicious activities towards web applications by exploring web traffic as if it is encrypted, without looking into the content of the HTTP requests and responses. Our research is focussed on the first phases of the attack in which the attacker scans the application and enumerates the information about it using Dynamic Application Security Testing (DAST) tools. By detecting such activities, it is possible to prevent access from those IP addresses that are creating suspicious behaviour or to raise awareness that there is an interest from the potential attackers that can trigger web application security strengthening from the application owner. While there is a relatively large number of research efforts which aim to detect network and host scanning (e.g. ping sweeps, port scans and similar), there is a notable lack of papers which target the problem of scanning web sites. Since web application scanning tools are becoming increasingly popular and can be used for malicious purposes, we believe that there should be more attention on web application attack preparation phases. The detection method presented in this paper uses supervised machine learning-based classification and the obtained results even with the use of classical algorithms are very promising as it will be described in the remaining part of the text. Since the focus of the research is the early detection of web application attacks, we also present the results of dynamic detection analysis and the time needed to reliably detect web application scanning after this malicious behaviour has started.

The paper is organised as follows. Section 2 gives an overview of related work in the fields of machine learning-based intrusion detection, web application attack detection and web application scanners. It also provides an overview of the contribution this paper makes. Section 3 describes four web application scanners (DAST tools) that were used in this research. Section 4 gives details about the dataset, methodology and laboratory setup we used to extract the data and features from the captured and used network traffic. Classification and detection results are given in Sect. 5. Finally, Sect. 6 presents the results of the dynamic detection analysis, and Sect. 7 concludes the paper.

2 Related work

Detecting cyber attacks using machine learning classification methods based on various protocols and traffic artefacts is a widely used approach, especially in the last decade. The aim of this section is not to provide an exhaustive list of related works as papers in the field appear daily. We focussed only on those papers which are closely related to the research objectives presented in this paper: web application attack detection. For a more comprehensive overview of the research efforts in the fields of machine learning supported threat and attack detection, one might refer to [5–7].

2.1 Machine learning-based network attack detection

One branch of the research focuses on the detection of network-based attacks such as Denial of Service (DoS) attacks, SYN flooding, host surveillance and probing attacks like port scanning, and similar. The network-based attack classification predominantly relies on the datasets prepared by extracting features from network packet traces and the fact that various attacks have specific network statistics (e.g. rate, packet size, etc.). In these papers, the authors successfully used different classification methods ranging from the classical supervised to unsupervised machine learning methods for attack detection. Feature representation approaches, named cluster centre and nearest neighbour, are presented in [8]. Rough Set and K-nearest neighbours algorithms used for intrusion detection are compared in [9]. In [10], the authors combined particle swarm optimisation with the K-nearest neighbour algorithm for intrusion detection. Syarif et al. propose a novel similarity rule for the K-nearest neighbours algorithm when used for intrusion detection in [11]. In [12], authors use a hybrid approach for intrusion detection that could be successfully used in real time. This approach uses the Naive Bayes feature selection technique, an outlier rejection using Optimised SVM and prioritised K-nearest neighbours classifier. Saleh et al. use SVM with naive Bayes feature embedding for intrusion detection with the focus on improving the data quality to boost performance in [13]. Additionally, Liao and Vemuri [14] explored the classification of programme behaviour using frequencies of system calls and K-nearest neighbours classifier. The other group of papers related to network-based intrusion detection analysed the existing datasets and their suitability for this type of research. One of the modern datasets with a high number of citations in the research literature is CICIDS-2017 [15]. It is used in this paper as well. In [16], CICIDS-2017 dataset and the process of extracting features from packet traces were analysed and recognised as one of the datasets used most often for the intrusion detection system development. A method to eliminate big data challenges when

using the CICIDS-2017 dataset for intrusion detection system development by using recursive feature elimination was proposed in [17]. In [18], authors compared SVM, K-nearest neighbours and decision tree algorithms for DDoS detection using the CICIDS-2017 dataset and Fisher Score algorithm for selecting the best features and concluded that although the dataset was reduced 60% by selecting the best features, the accuracy of KNN increased, SVM decreased, and DT did not change. Stiawanet et al. presented how feature selection influences detection accuracy and execution time when using machine learning algorithms for intrusion detection [19].

2.2 Web application attack detection

In the previous work on web application attack detection and security, the authors focussed their research on the analysis of web server logs and HTTP methods. In a recent paper, Tekerek [20] used the convolutional neural network to detect web application attacks. Dataset used in that paper was a well-known CSIC2010v2 dataset, which consists of a set of benign and malicious HTTP requests and responses. The same method and dataset were used in [21], in which the authors in addition created their own synthetic dataset. Pan et al. explored the potential of unsupervised learning for web attack detection [22]. Goseva-Popstojanova et al. also performed their analysis based on features that they collected in a honeypot from server logs by analysing HTTP methods and sessions [23]. Here, the dataset was recorded in the honeypot developed and installed for the research.

2.3 Web application scanning

Web application scanning is used in two ways: as a part of vulnerability scanning by the white hat analysts but also as a part of the preparatory (enumeration and scanning) phases of the attack. As described in Sect. 3.1, there are various web application scanners freely available to both white and black hat hackers and the number of tools is increasing. There is a solid body of work that analyses either web application scanning tools [24–27], web application scanning examples [24], or web application scanning approaches [25,26,28]. In [24], authors presented the results of vulnerability testing of an organisation using different tools. A new methodology for comparison of vulnerability-scanning tools for SQLi and XSS vulnerability detection tested on three commercial scanning tools is proposed in [25]. In [26], the authors detected several limitations of web vulnerability scanners and presented a solution in the form of a proxy between the scanner and tested application. Eight web application vulnerability scanners were analysed and compared in [27]. In [28], authors present their web application security scanner and focus on different modes of its usage. The conclusion of these papers is that web application scanners are good in

finding known vulnerabilities while less successful for the zero day issues. This means that if an attacker uses a scanner and finds a vulnerability in the web application, the last final step of the actual attack is relatively simple—to execute this type of attack on a known vulnerability.

This paper makes several important contributions. It gives the analysis of DAST tools network behaviour patterns and characteristic features that distinguish them from other traffic. Further, it extends the approach of detecting attacks using machine learning classification from network packet trace information to a new set of malicious activities, which target web applications. The aim of the approach presented in this paper is to detect possible attacks before they happen. This is done by recognising automated web application scanning in the early phase of the attack, and to the best of our knowledge, this approach was not analysed previously in the research literature. A recent study with a similar goal is [29] in which the authors explored the detection of web site visits by artificial agents as opposed to human users using unsupervised learning. This paper also gives a comprehensive overview of the recent research in the domain of automated web bot detection. Unlike [29] and other work related to that paper which also uses the HTTP method and log analysis, our paper is based on network packet traces and flow feature analysis. The choice of the data source for web application attack detection brings some important differences in web application protection because it 1) determines the position of the detector: HTTP and log-based detection must be located on the web server, while in the case of network traffic-based detection, it can be in a network element in front of one or more web servers, 2) impacts data privacy: HTTP and log-based detection implicitly assume that the system for the detection has an insight into the user's data, while in the case of network traffic-based detection, the detection can be performed even in TLS-protected web communication (encrypted data) thus preserving the data privacy. Like in many other research papers, for the design and prototype evaluation, we used the CICIDS-2017 dataset and its network flow features. We evaluated some of the features in this dataset by inspecting the code of the CICFlowMeter tool that generates the features. Finally, we propose a method to detect malicious behaviour before the network flow carrying the attack is over and analyse detection accuracy for different flow lengths. This is again important for fast attack detection.

3 Web application scanning tools

One of the measures in web applications security testing is the use of software designed for automated testing, known as Dynamic Application Security Testing (DAST) tools. DAST tools scan web applications from the outside, create a sitemap and the content produced by the target and its behaviour is

scrutinised for known vulnerability patterns. A large number of web application scanners appeared in the last several years, both commercial and open-source, some of these described in this section. Also, there is a large number of online web sites that offer web site scanning. All these tools not only allow web application developers to scan and secure their web applications, but also the intruders to detect the potential security flaws and to attack the applications.

3.1 DAST tools operation

DAST tools use two approaches: passive and active. Passive scanning involves observing security flaws visible to any user interacting legitimately with the target through the HTTP headers and responses. Such issues are often caused by misconfiguration or leaked information (IP and email addresses, social security numbers) that could help a knowledgeable attacker. Active scanning incorporates malicious target probing by injecting carefully crafted inputs. The way the application responds to these probes enables inferring whether a security flaw is present. The inputs are directed towards all possible entry points-URLs, static resources, HTTP headers, AJAX (Asynchronous JavaScript and XML) calls, HTML forms, session and authentication mechanisms.

For any previous security checks to happen, scanners must interact with the target application and learn about its resources and behaviour. One way of feeding this information to the DAST software is using a built-in proxy server. The scanner operator should configure it to intercept all the traffic generated during the interaction with the target. Then, it is necessary to explore the web application through the browser, by navigating through the presentation layer and submitting HTML forms with different combinations of the input parameters. However, generating a sitemap manually has its drawbacks. There could be paths and code that are intentionally or mistakenly unreachable by browsing.

A more automated approach to this phase of scanning has been devised by leveraging web spiders. Such scripts can index the target application automatically. The DAST tool operator supplies the URLs which are to be used as a starting point, as well as a blacklist which should be avoided by the script. The initial links are added to the pool of previously unseen resources. The script takes an entry from the pool, issues an HTTP request and examines the contents of the response. The visited URL is added to the sitemap and if more paths are discovered, these are filtered using the blacklist provided in the configuration and added to the pool. The process repeats until the pool is empty.

Although spiders provide means of navigating paths that are not available in the browser, there are downsides as well. Two major hindrances for automation include session management and generating adequate parameters when interacting with HTML forms. The former entails the ability to

authenticate and recognise potential session expiration, while the latter pertains to the situation where a series of multi-step forms need to be populated with strict validation on the server side. Moreover, modern web applications rely on heavy use of AJAX calls to alter the presentation layer. Due to this fact, meticulously covering such navigation mechanisms is a cumbersome task and many intricate spiders fail in discovering the complete sitemap. Therefore, the best approach is to first navigate the application by hand, with the DAST tool listening through a proxy, and then resort to the automated script for a fuller coverage of the sitemap.

While acquiring the sitemap, either manually or in an automated manner, the DAST tool conveys passive scanning of the content. After that process is complete, active scans may be started. In this process, every entry point of the application that was located in the sitemap may be examined by applying payloads designed to invoke a particular behaviour when a vulnerability is present. Depending on the capabilities of the scanner, SQL Injection, XSS, Path Traversal, Cross-site Request Forgery (CSRF), Code Injection, XML external entity and other vulnerabilities are tested. The inputs are targeted at HTTP headers, body, URL query string or path in case of a RESTful interface.

3.2 Analysed DAST tools

In this research, we considered four open-source DAST tools from the OWASP reference list [30] with different features: Nikto, Vega, Arachni and OWASP ZAP. Nikto [31] is the simplest of the four, having no built-in proxy and spidering abilities. It performs only passive checks for occurrence of dangerous files, web server misconfiguration and version-specific vulnerabilities. All the other tools are more sophisticated. In addition to passive scanning, Vega [32], Arachni [33] and OWASP ZAP [34] include the active probing, as well as providing proxy and spider subunits. In this experimental setup, Arachni has proven to have the most powerful spider among the three, being able to build the most comprehensive sitemap of the targeted web applications. Concerning the active scanning vulnerability coverage, according to the documentation of the DAST tools [32], Vega has a more modest ability in comparison with OWASP ZAP and Arachni which boast to cover even more specific vulnerabilities, such as NoSQL injection and DOM XSS.

What is common for all these scanning tools and the existing scanning tools today is that automating security checks leaves a significant footprint in the network traffic between the tool and the targeted application. Without prior knowledge about the technology stack used by the target, a significant amount of payload data is injected towards the web site before an existing vulnerability is triggered. DAST tools still don't have low bandwidth or stealth operation modes, and in some extreme cases (e.g. Arachni tool

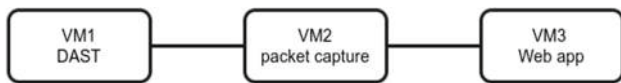


Fig. 1 Laboratory setup used for the web scanning analysis

described in the next section), intrusion detection systems can detect their behaviour as a denial of service attack. Even though some scanners offer to filter the security checks based on the known target database and web server providers, such filtering does not significantly reduce the intensity of the scanning, because the DAST tools must brute-force their way through every possible entry point of the target application. This implies that the web application scanning activities today greatly differ from usual behaviour exhibited by the user browsing the application, and that detection based on network traffic pattern analysis has a potential to be successful.

4 Methodology and dataset

The analysis of the datasets followed the well-known data science pipeline which consists of data engineering including data collection, cleansing, feature selection and machine learning: model learning and model validation. All the phases of the pipeline are described in detail in the subsequent sections.

4.1 Data source-collecting scanner behaviour

Malicious interactions were captured in a number of scanning sessions set up in a laboratory environment. The isolated testbed consisted of 3 Linux Ubuntu 20.04 virtual machines (VM1, VM2 and VM3) as depicted in Fig. 1. DAST tools were set up on VM1, and the attacked/scanned machine was on VM3. The traffic between these two hosts was routed by VM2 which captured it using the tcpdump utility.

Web scanning targets were four different web applications well known among the security professionals: WebGoat [35], DVWA [36], Gruyere [37] and Multillidae [38]. These applications, used for penetration testing and raising security awareness through exhibiting flaws such as SQL injection, path traversal, XML External Entity (XXE), XSS, CSRF and other commonly found vulnerabilities, are representatives of different software stacks and complexities. WebGoat is a Java application deployed on Tomcat web server, accessing an in-memory HyperSQL database. Mutillidae and DVWA use the frequent PHP-MySQL stack, while Gruyere consists of a handful of python scripts, relying on BaseHttpServer implementation to serve requests. Every application has a register and a login form with file upload being a common feature.

Unlike others, Gruyere does not have an SQL database as a datastore, but only keeps track of internal state using simple in memory data structures, since its only purpose is to memorise and present the snippets user entered through a form. Three other applications are more complex, serving many pages mostly presented as a series of guided steps through exploitable vulnerabilities.

For Nikto, OWASP ZAP and Vega DAST tools, capturing a single scanning session for all target web applications was sufficient, since the duration and the intensity of the activities were moderate. However, Arachni generated a large amount of network traffic, sometimes 2–3 orders of magnitude larger than for the other DAST tools - Table 1. Arachni scan of WebGoat application was performed in three separate intervals, with the tool being stopped each time using the built-in mechanism that allows it to continue the scanning seamlessly, without losing track of the session state. Besides these interactions, two additional situations were captured which showcased the use of built-in proxies for OWASP ZAP and Vega tools. All the sessions generated regular TCP flows except the sessions between the Arachni, Vega and ZAP tools and Multillidae application in which TCP flows were not terminated properly with the FIN handshake, and the flow analysis tool used timeouts to end the flow. This created a difference in the flow duration and throughput calculations, as shown in Table 1. However, since this is a behaviour we observed between these two applications, we kept this session in the dataset. The whole recorded dataset is publicly available [47].

On the other hand, obtaining sufficient quantities of benign data is challenging. Manual browsing depends on the web application and cannot produce enough traffic when compared to the vast quantities of TCP flows generated during the DAST scans which can lead to the heavily unbalanced dataset. Also, each web application has its own pattern of use which is different and depends on the application design and purpose. Sharafadin et al. [39] analysed available IDS datasets with the conclusion that generating benign traffic artificially is the most practical solution to the problem. They proposed a process to do that by recording the network activities and using X Means clustering to determine behaviour similarity. Finally, the centroid of each cluster is considered as the behaviour of users in that cluster and can be used to generate realistic interactions. CICIDS-2017 [37] dataset was created using this approach. To compensate for the low number of benign interactions, we used a hybrid approach: benign data points were populated with data extracted from the CICIDS-2017 dataset. This choice of benign dataset allows us to draw the conclusion about the potential to classify web scanning among the whole traffic in the network, which is a harder problem than to detect it among the traffic to and from the web server.

4.2 Data preprocessing and feature extraction

To differentiate between the malicious and benign interactions with the web server, a common approach is to examine statistical properties of the transport layer network traffic flows. CICFlowMeter [41,42] is a tool that extracts 83 flow features from raw packet captures. The software can be utilised in two scenarios: live packet capturing on an interface of choice, or in the offline mode, where the previously captured pcap files can be loaded into the program. When CICFlowMeter captures live packets, the per-flow statistics are computed upon flow termination. TCP flows can be terminated either due to the idle timeout (maximum period of inactivity, which is configurable) or if a FIN or RST control bits are set in the TCP header. UDP does not have session control mechanisms and the flows must be terminated upon idle timeout. With the emergence and wider use of QUIC-based web traffic [43], UDP flow analysis for web applications attack detection becomes increasingly important.

The network traffic features are tracked and calculated bidirectionally and the first packet in the flow determines the forward direction. In this paper, we used the same feature notation as in the official dataset repository [41]. The analysis of CICIDS-2017 flow features included also the analysis of the CICFlowMeter tool source code, as not all the features were documented in details and on the other side some of them had suspicious values in the dataset, especially those which are related to bulk rate (Fwd Byts/b Avg, Fwd Pkts/b Avg, Fwd Blk Rate Avg, Bwd Byts/b Avg, Bwd Pkts/b Avg, Bwd Blk Rate Avg), and subflows (Subflow Fwd Byts, Subflow Fwd Pkts, Subflow Bwd Byts, Subflow Bwd Pkts). Bulk rate features (burst of traffic consisting of four or more packets incoming in intervals not longer than 1 second is considered a bulk load) had a value 0 for all analysed flows, which makes them of no particular use for the classification. For the subflow features, the software implementation recognises a subflow by tracking the microsecond timestamps of two subsequent packets. If the interval is longer than one second, the tool considers that a new subflow has started. However, there is an issue in the CICFlowMeter code—only the second timestamp is divided by 106, and the difference between the moment of the arrival of the first and the second packet is always greater than 1 second. Furthermore, the Active and Idle features are affected by this same issue, rendering their usability questionable. When applying CICFlowMeter to our sessions, Active and idle time-related features (Idle Min, Idle Max, Idle Std, Idle Mean, Active Min, Active Max, Active Std, Active Mean) also produced an unexpected output, with cases where the calculated values were greater than the duration of the whole flow as given by the Flow Duration feature. Therefore, these features were excluded from the classification.

There is also a slight inconsistency in the way CICFlowMeter defines the end of the TCP flows that leads into reporting multiple flows instead of only one. CICFlowMeter Flow-Generator class implements the logic for closing an active flow. If a FIN control bit is set in the TCP header, the flow is closed immediately and archived. However, the remaining communication that is a part of the 4-way end of TCP session handshake and is exchanged before the session is closed is now observed as a beginning of a new short flow (only 2–3 packets), which expires upon the expiration of the idle timeout. In this case, the initiator of the flow is the side that received the packet with the FIN flag set, and the flow starts with an ACK flag. Such CICFlowMeter flow termination practice makes the direction of the second flow inverted, with the web server being considered the initiator of the that flow, rendering the generated features which take flow direction into account useless, since in a real-world scenario, the initiator of the flow is always the client (attacker). This further leads to the flow properties fragmentation and not entirely accurate flow statistics calculation. In the data preparation, this problem was addressed by removing the small flows that appear after the FIN flag. We created a preprocessing filter and applied it consistently to both DAST packet captures and CICIDS-2017 flows, with a predicate to keep only those flows which consist of at least two packets in the forward and one packet in the backward direction.

4.3 Feature selection and dimensionality reduction

The number of flows CICFlowMeter extracted from the DAST software sessions is presented in Table 1. The column “Flows before preprocessing” is the total number of data points produced by CICFlowMeter for each DAST session. After applying the aforementioned filter to remove the superfluous flows, for most of the sessions, the number of flows was halved, as expected. From Table 1, it can be seen that the Arachni tool produced much more flows than the other DAST tools, while all the DAST tools consistently produced a significant amount of data when scanning Gruyere. To capture the behaviour of all tools with a similar impact across all targeted applications, Arachni WebGoat sessions were chosen by picking totally 6,000 random flows from the whole population, divided in three different files. Similarly, for Gruyere, each scanning session was sampled by taking 10% of the total number of flows, as denoted in the “Flows kept” column. This way the number of flows of different DAST tools was kept relatively balanced. In addition to the scanning activity, 5% of total flows from CICIDS-2017 Monday session were randomly sampled to create the benign data represented by a total of 16,225 records. The malicious DAST flows accounted for 20,338 records, which means that the dataset is still slightly imbalanced. Table 1 shows two additional properties of the flows: average flow duration and average flow throughput.

Although, as said in the previous section, these parameters do not accurately reflect the real values of the flows, relative differences among the flow classes can still be assessed because the same flow processing methodology was used for all classes. As it can be seen, except in few cases, DAST tools create shorter flows with a lower throughput than the majority of benign flows, depending on the tool and scan phase.

After cleaning the data, the flows were labelled. CICIDS-2017 already contained the label benign (B) which was kept, while all scanning sessions were labelled as malicious (M). The dataset was divided according to the 60:20:20 ratio for the training, validation and testing sets, respectively. All the initial session files were split into three parts according to this ratio, and then, the appropriate datasets were assembled, so that every session is present in each generated dataset. This approach also allowed for later evaluation of the sensitivity of each machine learning model towards specific sessions and tools.

The next phase was the dimensionality reduction through the elimination of some of the flow features. In the first phase, we eliminated those features which are not suitable for the classification because of the nature of the analysed traffic. Seven attributes are not considered flow features, but rather flow identifiers: Flow Id, Src Port, Dst Port, Src IP, Dst IP, Protocol and Timestamp. A research paper [44] has used the Dst Port attribute from the CICIDS-2017 dataset. However, the malicious scanning in this paper was targeted at a single port each of the web application was configured to listen on (e.g. WebGoat was configured to use port 8080). Utilising the destination port as a feature would make the classification a trivial task in this case. Further several flow features describe the same flow property and are thus not bringing new information about the flow (e.g. Pkt Size Avg vs. Pkt Len Mean, which both give the information about the mean packet size in a flow).

All malicious flows from the automated web application scanning used TCP protocol only. On the other hand, benign flows extracted from CICIDS-2017 contain both TCP and UDP flows. The control bit flags and window sizes are a characteristic of the TCP protocol only, which makes all these attributes equal to zero for non-TCP traffic. This fact would be leveraged by any machine algorithms, making it an easy task to correctly classify all UDP interactions from the dataset as benign. However, there is an increasing trend of encapsulating HTTP in UDP flows (QUIC protocol). For such use cases, models trained on a dataset where the absence of control bits and window sizes means the traffic is benign would prove useless. Due to this fact, 14 features related to the TCP protocol were not considered.

This left 38 out of the initial 83 features that were used in the classification process. In the second phase of dimensionality reduction, we analysed the contribution of each

feature to the classification process. We tried two approaches: principal component analysis (PCA) and correlation. Both processes relied upon WEKA [45] implementation. The first approach produced worse classification results, so we used Pearson correlation to determine the correlation between each feature and the class label. The first step was to apply the Pearson correlation coefficient on the training data contained in the training dataset as noted in Table 2. High class label correlation with the features that are calculated from the packet lengths suggests that DAST flows express in these features significantly different behaviour than the rest of the network traffic. A detailed analysis revealed that DAST flows consist of packets, which are on average larger and more variable than the packets in CICIDS-2017 flows.

5 Classification and result evaluation

In this research, we used the WEKA implementation [45] of the K-Nearest Neighbours (KNN), Support Vector Machine (SVM) and Random Forest (RF) algorithms. All algorithms were used with the default set of hyperparameter settings. For KNN, the metric of choice was Euclidean distance, where the number of neighbours being relied upon for classifying new instances was $K = 1$. Search for the optimal number of nearest neighbours is possible through the built-in support for cross-validation. However, running this process has proved that changing the parameter K does not affect the classification performance positively. This suggests that malicious and benign data points using the chosen set of features are far away. Settings for the RF algorithm included generating 100 trees. When building each tree, the first step is to choose a random subset of all features available. The number of features for each subset was $\log_2(NF)+1$, where NF is the total number of features. By randomising the features which are chosen, overfitting is reduced. For the SVM algorithm, the hyperparameter C was set to 1 for all experiments.

Since the dataset is not balanced, simple classification accuracy measures do not accurately reflect the quality of trained models. In this case, F-measure is an appropriate measure because network flow classification is a binary classification. The set of scores used in this paper are precision, recall, and f1-score. Precision is defined as

$$Precision = \frac{truepositive}{truepositive + falsepositive} \quad (1)$$

Recall is defined as

$$Recall = \frac{truepositive}{truepositive + falsenegative} \quad (2)$$

Table 1 The number of flows used in the dataset

Session	Flows before preprocessing	Flows after preprocessing	Flows kept	Average flow duration [s]	Average flow throughput [Mbps]
Arachni WebGoat	537,047	266,280	6,000	1.12	2.6667
Arachni-DVWA	12,368	5,943	5,943	1.42	0.0694
Arachni-Gruyere	40,609	18,457	1,846	0.03	0.1229
Arachni-Mutillidae	1,264	1,237	1,237	13.62	0.0002
Nikto-WebGoat	1,876	909	909	0.01	2.9800
Nikto-DVWA	393	188	188	1.02	0.0348
Nikto-Gruyere	17,741	8,131	813	0.02	0.1090
Nikto-Mutillidae	483	234	234	1.20	0.0784
Vega-WebGoat	471	232	232	1.44	0.4600
Vega-DVWA	337	135	135	8.82	0.0082
Vega-Gruyere	8,132	3,445	344	0.03	0.0898
Vega-Mutillidae	152	87	87	19.73	0.0402
ZAP-WebGoat	2,988	1,237	1,237	3.71	4.1300
ZAP-DVWA	406	171	171	2.71	0.3195
ZAP-Gruyere	12,430	5,100	510	0.02	0.1187
ZAP-Mutillidae	1,028	452	452	15.04	0.1914
CICIDS-2017 Monday	26,459	16,225	16,225	10.39	1.54

Table 2 Feature rank by correlation with class label in training dataset

Rank	Feature	Correlation coefficient	Rank	Feature	Correlation coefficient
1	Bwd Pkt Len Std	0.4685	20	TotLen Fwd Pkts	0.1777
2	Pkt Len Min	0.4619	21	Bwd Pkts/s	0.1665
3	Bwd Pkt Len Min	0.4617	22	Flow IAT Std	0.1632
4	Pkt Len Std	0.4418	23	Bwd IAT Mean	0.1577
5	Fwd Pkt Len Min	0.4227	24	Flow IAT Mean	0.1514
6	Fwd Pkt Len Std	0.4205	25	Fwd IAT Mean	0.1447
7	Bwd Pkt Len Max	0.4009	26	Fwd Pkts/s	0.1359
8	Pkt Len Max	0.4008	27	Fwd IAT Min	0.1166
9	Pkt Len Mean	0.3706	28	Bwd IAT Min	0.1107
10	Bwd Pkt Len Mean	0.3538	29	Fwd IAT Min	0.1089
11	Fwd Pkt Len Mean	0.3419	30	Fwd IAT Std	0.1021
12	Bwd IAT Tot	0.2707	31	Tot Fwd Pkts	0.0905
13	Fwd IAT Tot	0.2665	32	Tot Bwd Pkts	0.0793
14	Pkt Len Var	0.2618	33	TotLen Bwd Pkts	0.0758
15	Fwd Pkt Len Max	0.2380	34	Down/Up Ratio	0.0666
16	Bwd IAT Max	0.2053	35	Fwd Act Data Pkts	0.0660
17	Fwd IAT Max	0.1839	36	Bwd Header Len	0.0078
18	Flow IAT Max	0.1828	37	Fwd Header Len	0.0077
19	Bwd IAT Std	0.1813	38	Fwd Seg Size Min	0.0076

f1-score is defined as the harmonic mean between precision and recall:

$$f1 - score = 2 \frac{precision * recall}{precision + recall} \quad (3)$$

In the experiments conveyed in this research, the flows labelled as malicious were treated as the relevant or positive class, while the benign flows were treated as the negative class.

All algorithms were trained on the training subset. We evaluated the classification algorithms with different sets of features. The set of features was determined based on the minimal Pearson correlation coefficient from Table 2. Pearson coefficients ranged from 0.5 to 0 with a step of -0.1, which created the following number of features, respectively: 11, 16, 30 and 38. The trained models were validated against the validation subset, enabling a comparative analysis of models with different data dimensionality. The results are presented in Fig. 2.

For KNN and RF algorithms, models built using 30 flow features exhibited the best f1-score, with the data dimensionality significantly reduced (less than the half of the original number of features), while the false positive rate is kept low. With the SVM algorithm, the model trained on 38 features had the highest f1-score (0.92), but still SVM had a significantly lower f1-score than the other two classifiers, especially for the sets of attributes with the higher number of attributes (16, 30 and 38 features). The false positive rate for the SVM

model trained on 30 features is the same as for the 38 selected features (0.016), and the f1-score is slightly lower (0.911) while having 10 features less. Hence, the model trained on 30 features was chosen for all three classifiers for testing. These models, which were previously trained on the training subset, are evaluated on a set of separate testing data- testing subset, with the results given in Table 3.

Results from Table 3, suggest that KNN and RF classifiers have a high accuracy, that RF is slightly more accurate and that the proposed detection model can be used in production environments successfully. As expected from the validation phase, SVM showed worse detection accuracy but still with relatively high precision and f1-score. In addition, we evaluated the accuracy of the trained model per each scanner and application. The results are given in Tables 4, 5 and 6.

Again, all KNN and RF models, except KNN, for Multitool detected more than 92% of the malicious flows, while SVM exhibited a more modest recall for the majority of DAST tools. Therefore, KNN and RF models are a better choice for the classification. Another factor that impacts the classifier choice is its suitability for use in real-time intrusion detection or prevention. Essentially, KNN algorithm calculates the distance from every entry in the training set to a new data point that is to be classified. This process can be slow, especially for larger datasets. For this reason, we argue that for the web application scanning detection, RF classifier is the most appropriate candidate for real-time detection. Also, it should be emphasised that detecting scanning on Multitool

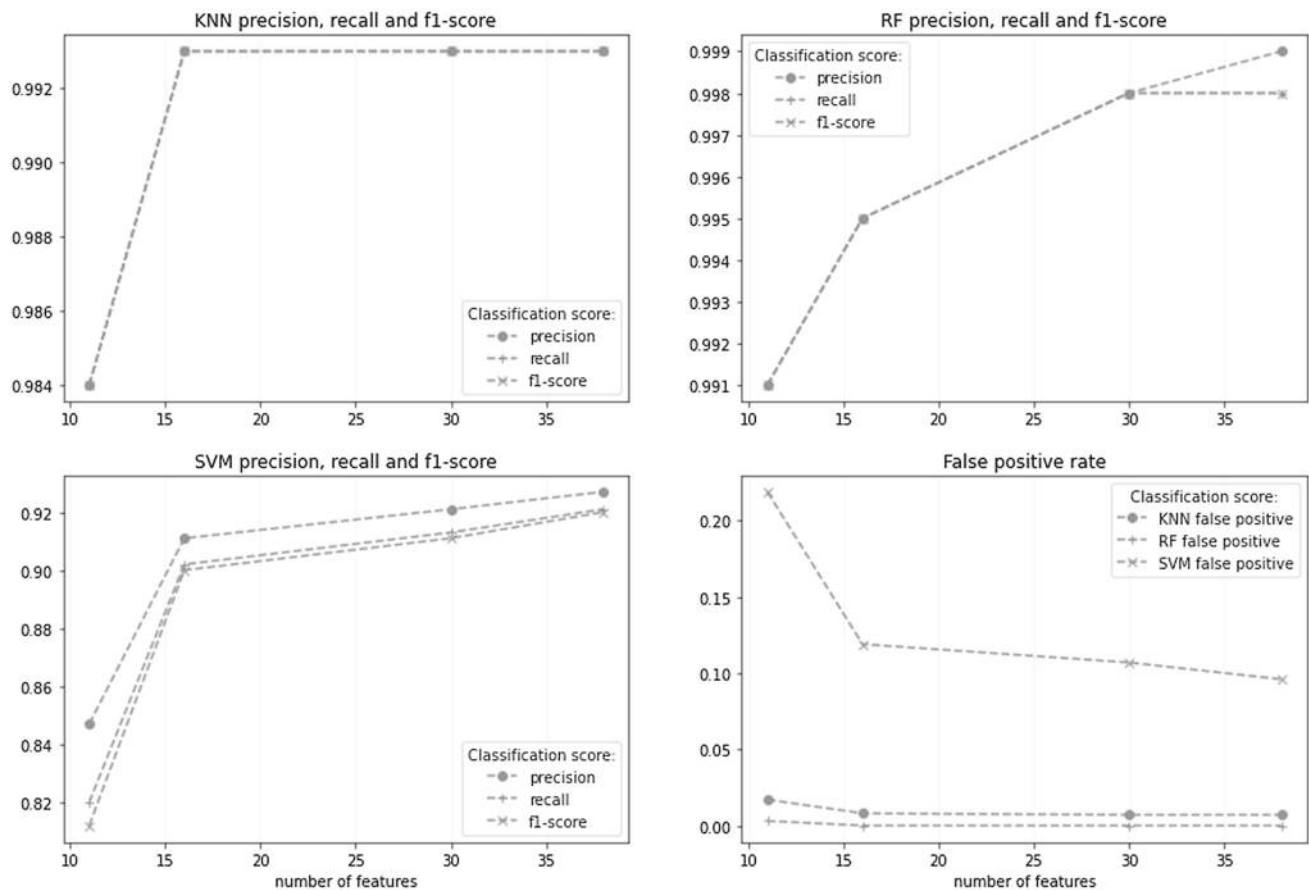


Fig. 2 Precision, recall, f1-score and false positive rate for chosen classifiers and the variable set of network flow features on the validation dataset

Table 3 Results of evaluating different models on testing dataset

Classifier	Number of attributes	False positive rate	Precision	Recall	F1-score
KNN	30	0.008	0.993	0.993	0.993
RF	30	0.002	0.998	0.998	0.998
SVM	30	0.108	0.920	0.911	0.910

dae web application gave worse results for all classifiers. As described in Sect. 4.1, some TCP sessions between the scanners and this web application were not properly terminated. This led to the flow termination by the monitoring tool after a predefined time out. Such practice corrupts the real flow statistics derived from the flow length (makes them longer and with less throughput) and has a negative impact on flow classification. However, even in this case, RF proved to be a robust and reliable classifier.

It is common for web security scanners to include proxies, which are used to explore the targeted web server manually. When traffic passes through the proxy, the DAST tool becomes aware of the contents and paths explored by the operator. This feature is useful when automated scanners cannot reach a certain portion of the web application, mak-

Table 4 KNN: Recall values per DAST tool and web application

	WebGoat	DVWA	Gruyere	Mutillidae
Arachni	0.999	0.988	1	0.984
Nikto	0.995	1	1	1
Vega	1	0.926	1	0.882
Zap	0.996	1	0.99	0.978

ing these more obscure parts available for further automated scanning. OWASP ZAP and Vega included such proxy tools. In the process of scanning the WebGoat application, two additional datasets were created separately to test the detection of this feature. These were fed to the models previously trained on 30 most correlated features. Based on the results

Table 5 RF: Recall values per DAST tool and web application

	WebGoat	DVWA	Gruyere	Mutillidae
Arachni	0.999	0.998	1	0.992
Nikto	1	1	1	1
Vega	1	0.963	1	0.941
Zap	0.996	1	1	0.989

Table 6 SVM: Recall values per DAST tool and web application

	WebGoat	DVWA	Gruyere	Mutillidae
Arachni	0.998	1	1	0.996
Nikto	1	0.973	1	0.936
Vega	1	1	1	0.353
Zap	0.98	1	1	0.802

Table 7 Classifying flows generated between DAST proxies and the WebGoat application

Session	Flows	KNN Ben	KNN Mal	RF Ben	RF Mal	SVM Ben	SVM Mal
Vega proxy	9	1	8	0	9	1	8
Zap proxy	862	271	591	381	481	391	471

presented in Table 7, it can be concluded that the models recognise these interactions as fully or partially malicious. The malicious labels are classified with M, while the benign labels are marked with B. In the case of Vega proxy, the detection accuracy was similarly high as in the case of scanning without proxy. On the other hand, classifiers could detect the majority of proxy flows for the ZAP tool, but with a significant number of proxy flows classified as benign. This suggests that the behaviour of the proxy flows is different from the behaviour of automated scanning, but that still the activity of the scanner, which runs through a proxy, will be detected as the majority of its flows will be marked as malicious.

6 Dynamic detection analysis

One of the goals of building a successful IDS system is to recognise malicious behaviour as soon as possible after it starts. This enables stopping the attack before it generates larger damage to the network and computing infrastructure. Because in the classical network setups network flows are being analysed on flow collectors upon flow completion, flow-based attack detection is capable of detecting malicious behaviour only after such behaviour is expressed in some of the network flows and probably already completed. This makes flow-based approaches more suitable for attack foren-

sics than for the attack prevention and mitigation. However, the analysis of the full packet captures and recent advances in the application of the programmable network elements allow the calculation of flow statistics while the flows are still active. Therefore, we explored whether it is possible to detect malicious behaviour before the flow is over, while it is still active and how quickly network flows express the behaviour which characterises them as malicious. Detecting malicious behaviour before the flow is completed would allow faster online malicious behaviour detection and attack mitigation.

Flows with up to 64 packets make around 96% of the flows in the general Internet traffic, but these flows are carrying less than 2% of the total traffic, as most of the content is in the longer ones [46]. Figure 3 shows very similar packet counts in the flows for the benign traffic in the CICIDS-2017 dataset to those reported in [46], while the DAST tools generated much more flows with the larger packet counts. Around 8.46% of the malicious flows had more than 50 packets and 9.47% of all malicious flows in our dataset lasted longer than 1 minute with the longest one which lasted for almost 2 hours. Longer flows are typically those with the larger packet counts. If the flows have approximately the same statistics within the first 50 packets as the whole flows, it would be possible to detect the malicious activity after approximately one minute, which is in average the duration of the 50-packet flow fragment.

We used the previously mentioned classification approach with the classifiers trained using 30 features and tested it on the features calculated from the fragments of the flows of different sizes for the WebGoat tool. All features were calculated on flow fragments that had 10, 15, 20, 25 and 50 packets in a flow. Of course, this methodology is applicable only on flows which are long enough to be treated this way.

Figure 4 depicts the result of the previously trained classifiers recall. As expected, the larger the number of packets in the network flow is taken for the feature calculation, the higher the precision of the classification is obtained because a larger sample means that the calculated features are closer to the values of the whole flow. Also, as in the previous section, KNN and RF classifiers were more successful, regardless of the number of packets being considered in a flow fragment. The RF model exhibits high sensitivity (greater than 90%) to Arachni and Nikto session flows after only 10 packets in a flow, while the fraction of relevant flows retrieved for Vega and ZAP is somewhat lower. This is because Arachni and Nikto consist of flows which have shorter than average duration (Table 1) and with lower packet counts than the other flows (Fig. 3). In such cases, characteristic flow behaviour is expressed quickly. Benign traffic classification for shorter flow fragments shows a higher number of false positives. However, for the RF classifier, the difference in recall between the 50 and 10 packets in the flow results in the drop of recall from 0.998 to 0.993, respectively, which is still an excellent result. When 50 packets are in the flow, clas-

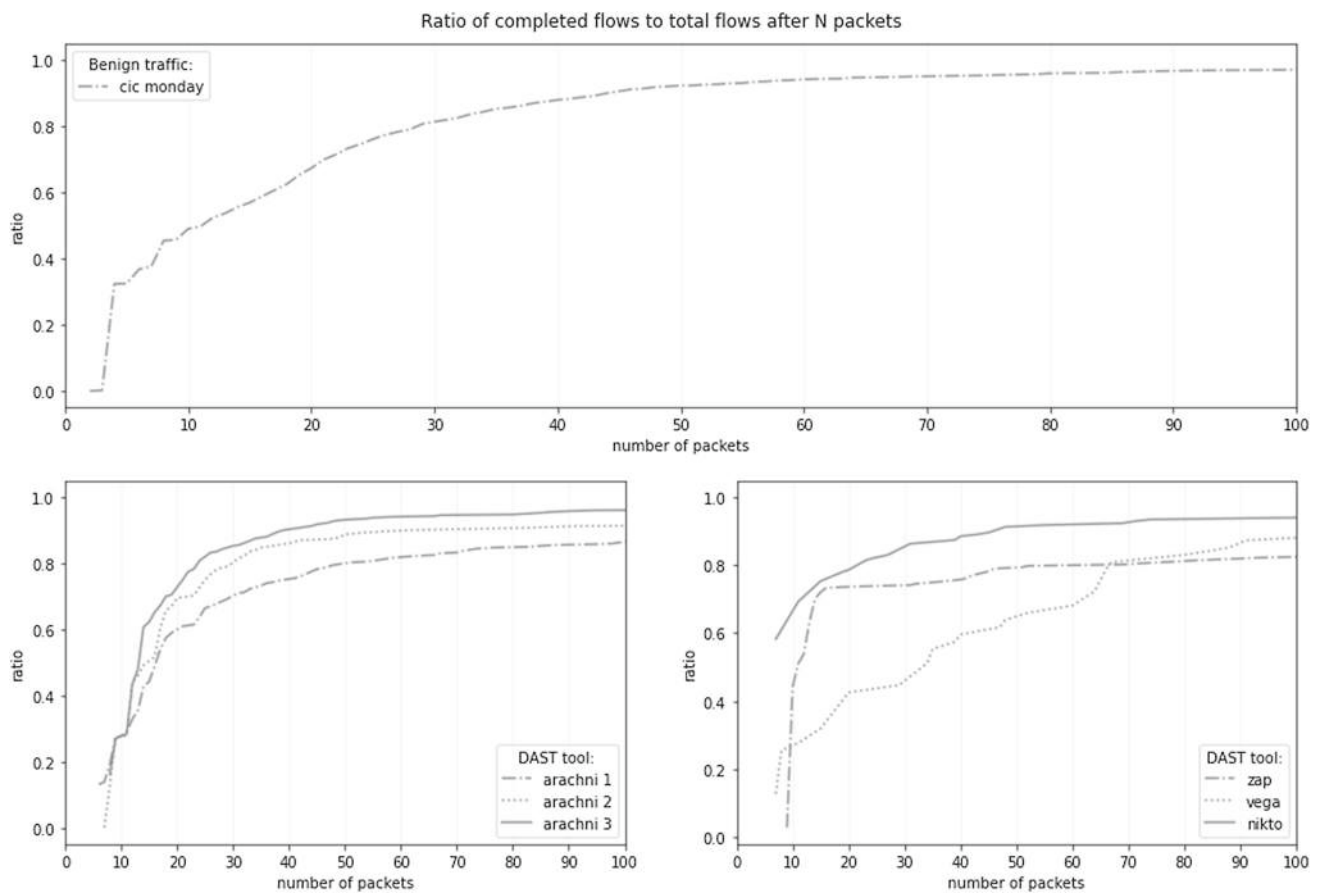


Fig. 3 Ratio of completed flows to total flows after N packets for each of the test sets

sification precision values for the benign traffic are the same as in Sect. 5 when the evaluation is based on full packet flows. This suggests that the intrusion detection systems should be able to detect the ongoing web application scanning activity after 50 packets in the flow with a high precision.

7 Conclusion

Results shown in this paper have confirmed the possibility to detect the preparatory phases of a hacker attack involving DAST tool scans using only network traffic analysis. The DAST tools in use ranged from basic ones such as Nikto and Vega, to more intricate ones such as Arachni and OWASP ZAP, which target a wide scope of security vulnerabilities present in modern web applications. Since detecting and preventing web scanning activities is not yet a commonly used practice, current DAST tools do not take measures to hide their behaviour and the detection accuracy was high. In a case that such practice becomes widely used, it can be expected that DAST tools start to change their behaviour to avoid detection and exhibit behaviour similar to the stealth port scans.

During the data preprocessing, after rigorously analysing all the features, we have considerably lowered the dimensionality of initial 83 features generated from CICFlowMeter to 38 that were relevant. Following the tendencies of encapsulating HTTP in UDP, we have eliminated TCP-specific features, to allow for a more general approach. Further, by determining the correlation of attributes with the class label, during the validation process, it was demonstrated that a subset of 30 features is adequate for achieving the best results. Several well-known classical machine learning algorithms have been employed (KNN, RF, SVM) with high accuracy in recognising suspicious activity. The RF model has the most potential for real-time use, since the false positive rate is kept low, and sensitivity (recall) to malicious flows is promising. Furthermore, RF does not suffer from the scalability issues such as the ones the KNN model does inherently experience due to the specifics of its training phase. Finally, the analysis in Sect. 6 has shown that it is possible to deploy the flow-based detection on flow fragments, while the flows are still active, which can speed up the detection process. We believe that this paper is a start in the new subfield of the research on web application scanning. There are various open topics that should be explored further like capturing and averaging the

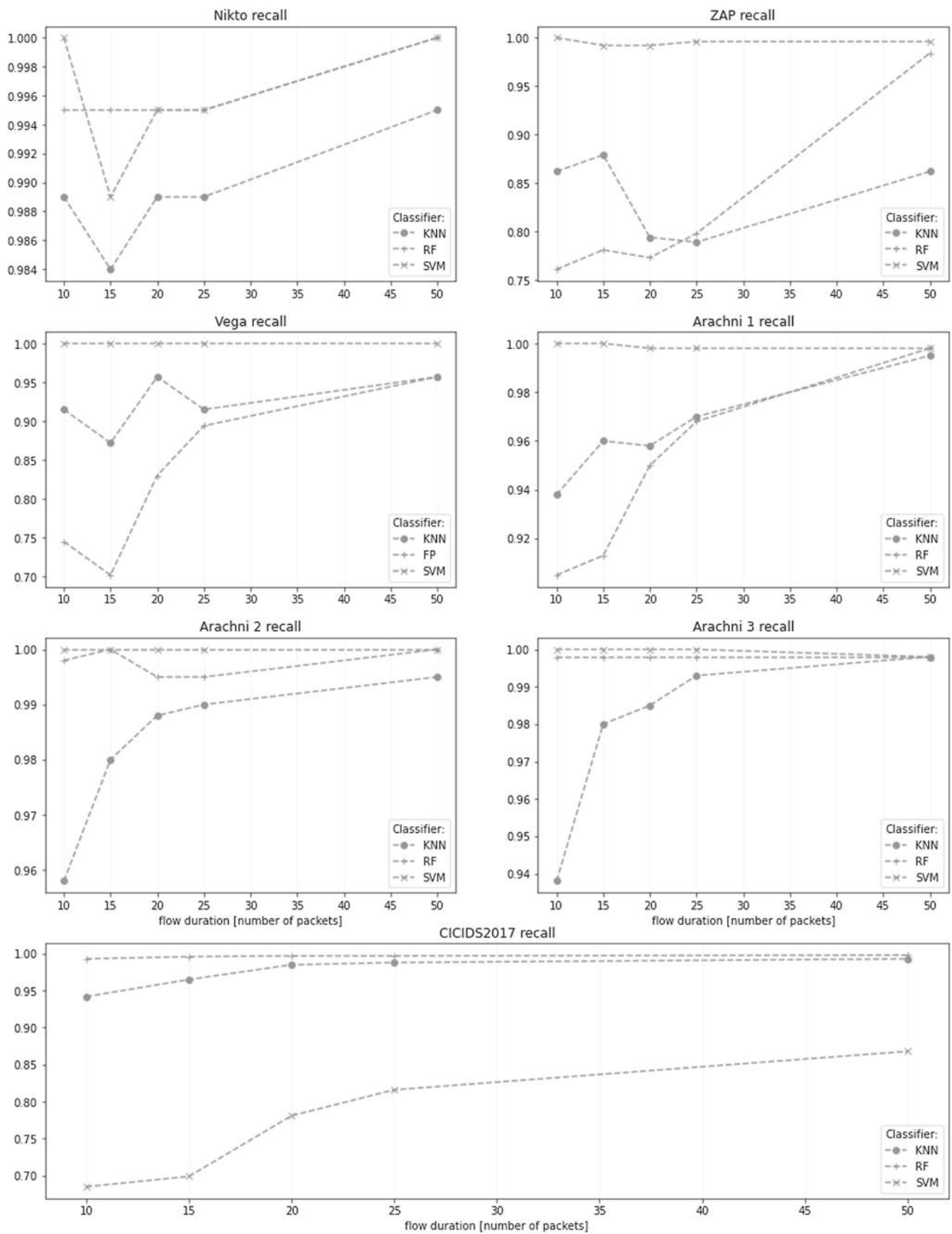


Fig. 4 KNN, RF and SVM classifier recall for the shortened flows, based on a model trained on the training dataset with 30 most correlated features

normal user behaviour on a wide variety of web applications that have different usage patterns, the analysis of the wider set of DAST tools and web-based scanners, classification of the DAST tool behaviour, creating strategies to enable stealth DAST operations, analysing the web applications' logs in order to get the set of features which are common across multiple web applications and which could improve the dataset and other.

Author Contributions BR contributed to data curation, investigation, methodology, software, writing—original draft. PV contributed to conceptualization, methodology, investigation, resources, writing—review and editing, validation, supervision ZS contributed to investigation, validation, writing—review and editing

Funding No funds, grants, or other support was received.

Data Availability Data is publicly available at [47]

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors

Code availability Not applicable

References

- Collins V.: The Decline Of The Native App And The Rise Of The Web App. <https://www.forbes.com/sites/victoriacollins/2019/04/05/why-you-dont-need-to-make-an-app-a-guide-for-startups-who-want-to-make-an-app/?sh=597b75f26e63> (2019). Accessed 11 April 2021
- The Future Is the Web! How to Keep It Secure? <https://www.acunetix.com/white-papers/the-future-is-the-web-how-to-keep-it-secure/>. Accessed 11 Aug 2021
- HTTPS encryption on the web, <https://transparencyreport.google.com/https/overview?hl=en>. Accessed 11 April 2021
- ENISA Threat Landscape Web application attacks, from January 2019 to April 2020, https://www.enisa.europa.eu/publications/web-application-attacks/at_download/fullReport. Accessed 11 April 2021
- Moustafa, N., Hu, J., Slay, J.: A holistic review of network anomaly detection systems: a comprehensive survey. *J. Netw. Comput. Appl.* **128**, 33–55 (2019)
- Gibert, D., Mateu, C., Planes, J.: The rise of machine learning for detection and classification of malware: research developments, trends and challenges. *J. Netw. Comput. Appl.* **2**, 153 (2020)
- Tahsien, S.M., Karimipour, H., Spachos, P.: Machine learning based solutions for security of Internet of Things (IoT): a survey. *J. Netw. Comput. Appl.* **2**, 161 (2020). <https://doi.org/10.1016/j.jnca.2019.102630>
- Lin, W.C., Ke, S.W., Tsai, C.F.: CANN: an intrusion detection system based on combining cluster centers and nearest neighbors. *Knowl.-Based Syst.* **78**, 13–21 (2015)
- Adetunmbi, A.O., Falaki, S.O., Adewale, O.S., Alese, B.K.: Network intrusion detection based on rough set and k-nearest neighbour. *Int. J. Comput. ICT Res.* **2**(1), 60–66 (2008)
- Syarif, A.R., Gata, W.: Intrusion detection system using hybrid binary PSO and K-nearest neighborhood algorithm. In: 2017 11th International Conference on Information and Communication Technology and System (ICTS), pp. 181–186. IEEE (2017)
- Ma Z., Kaban A.: K-Nearest-Neighbours with a novel similarity measure for intrusion detection. In: 2013 13th UK Workshop on Computational Intelligence (UKCI), pp. 266–271. IEEE (2013)
- Saleh, A.I., Talaat, F.M., Labib, L.M.: A hybrid intrusion detection system (HIDS) based on prioritized k-nearest neighbors and optimized SVM classifiers. *Artif. Intell. Rev.* **51**(3), 403–443 (2019)
- Gu, J., Lu, S.: An effective intrusion detection approach using SVM with naïve Bayes feature embedding. *Comput. Secur.* **2**, 103 (2021). <https://doi.org/10.1016/j.cose.2020.102158>
- Liao, Y., Vemuri, V.R.: Use of k-nearest neighbor classifier for intrusion detection. *Comput. Secur.* **21**(5), 439–448 (2002)
- Ferrag, M.A., Maglaras, L., Moschoyiannis, S., Janicke, H.: Deep learning for cyber security intrusion detection: approaches, datasets, and comparative study. *J. Inform. Secur. Appl.* **50**, 102419 (2020). <https://doi.org/10.1016/j.jisa.2019.102419>
- Panigrahi, R., Borah, S.: A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *Int. J. Eng. Technol.* **7**(3.24), 479–482 (2018)
- Ustebay S., Turgut Z., Aydin M.A.: Intrusion detection system with recursive feature elimination by using random forest and deep learning classifier. In: 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), pp. 71–76. IEEE (2018)
- Aksu D., Üstebay S., Aydin M.A., Atmaca T.: Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm. In: International Symposium on Computer and Information Sciences, pp. 141–149. Springer, Cham. https://doi.org/10.1007/978-3-030-00840-6_16 (2018)
- Stiawan, D., Idris, M.Y.B., Bamhdi, A.M., Budiarto, R.: CICIDS-2017 dataset feature analysis with information gain for anomaly detection. *IEEE Access* **8**, 132911–132921 (2020)
- Tekerek, A.: A novel architecture for web-based attack detection using convolutional neural network. *Comput. Secur.* **100**, 102096 (2021). <https://doi.org/10.1016/j.cose.2020.102096>
- Rong, W., Zhang, B., Lv, X.: Malicious Web Request Detection Using Character-Level CNN. In: Chen, X., Huang, X., Zhang, J. (eds.) *Machine Learning for Cyber Security. ML4CS 2019. Lecture Notes in Computer Science*, vol. 11806. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30619-9_2
- Pan, Y., Sun, F., Teng, Z., White, J., Schmidt, D.C., Staples, J., Krause, L.: Detecting web attacks with end-to-end deep learning. *J. Internet Serv. Appl.* **10**(1), 16 (2019). <https://doi.org/10.1186/s13174-019-0115-x>
- Goseva-Popstojanova, K., Anastasovski, G., Dimitrijević, A., Pantev, R., Miller, B.: Characterization and classification of malicious Web traffic. *Comput. Secur.* **42**, 92–115 (2014). <https://doi.org/10.1016/j.cose.2014.01.006>
- Daud, N.I., Bakar, K.A.A., Hasan, M.S.M.: A case study on web application vulnerability scanning tools. In: 2014 Science and Information Conference, pp. 595–600. IEEE (2014)
- Fonseca, J., Vieira, M., Madeira, H.: Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. In: 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007), pp. 365–372. IEEE (2007)
- Esposito, D., Rennhard, M., Ruf, L., Wagner, A.: Exploiting the potential of web application vulnerability scanning. In: ICIMP 2018 the Thirteenth International Conference on Internet Monitor-

- ing and Protection, Barcelona, Spain, 22–26 July 2018, pp. 22–29 IARIA (2018)
27. Bau, J., Bursztein, E., Gupta, D., Mitchell, J.: State of the art: automated black-box web application vulnerability testing. In: 2010 IEEE Symposium on Security and Privacy, pp. 332–345, IEEE (2010)
 28. Huang, Y.W., Tsai, C.H., Lee, D.T., Kuo, S.Y.: Non-detrimental web application security scanning. In: 15th International Symposium on Software Reliability Engineering, pp. 219–230, IEEE (2004)
 29. Rovetta, S., Suchacka, G., Masulli, F.: Bot recognition in a Web store: an approach based on unsupervised learning. *J. Netw. Comput. Appl.* **157**, 102577 (2020). <https://doi.org/10.1016/j.jnca.2020.102577>
 30. Vulnerability Scanning Tools, OWASP, https://owasp.org/www-community/Vulnerability_Scanning_Tools. Accessed 14 April 2021
 31. CIRT Nikto 2, <https://cirt.net/Nikto2>
 32. Subgraph Vega vulnerability scanner, <https://subgraph.com/vega/>
 33. Arachni, Web application security scanner framework, <https://www.arachni-scanner.com/>
 34. OWASP Zed Attack Proxy (ZAP), <https://www.zaproxy.org/>
 35. OWASP WebGoat, <https://owasp.org/www-project-webgoat/>
 36. DVWA - Damn Vulnerable Web Application, <https://github.com/digininja/DVWA>
 37. Google Gruyere, <https://google-gruyere.appspot.com/>
 38. OWASP Multillidae, <https://github.com/webpwnized/mutillidae>
 39. Sharafaldin, I., Gharib, A., Lashkari, A.H., Ghorbani, A.A.: Towards a reliable intrusion detection benchmark dataset. *Softw. Netw.* **5**, 177–200 (2017). <https://doi.org/10.13052/jsn2445-9739.2017.009>
 40. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISPP, ISBN 978-989-758-282-0; ISSN 2184-4356, pp. 108–116 (2018). <https://doi.org/10.5220/0006639801080116>
 41. Lashkari, A.H., Gil, G.D., Mamun, M.S.I., Ghorbani, A.A.: Characterization of tor traffic using time based features. *Science* **5**, 253–262 (2017). <https://doi.org/10.5220/0006105602530262>
 42. Lashkari, A.H., Gil, G.D., Mamun, M.S.I., Ghorbani, A.A.: Characterization of encrypted and VPN traffic using time-related features (2016). <https://doi.org/10.5220/0005740704070414>
 43. Iyengar, J., Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport, Internet draft (2021). <https://datatracker.ietf.org/doc/draft-ietf-quic-transport/>
 44. Kurniabudi, Stiawan D., Darmawijoyo, Idris M.Y.B., Bhamdi, A., Budiarto, R.: CICIDS-2017 dataset feature analysis with information gain for anomaly detection. *IEEE Access* **2**, 774 (2020)
 45. Eibe, F., Hall, M.A., Witten, I.H.: The WEKA Workbench. Online Appendix for “Data Mining: Practical Machine Learning Tools and Techniques”, 4th edn. Morgan Kaufmann, Burlington, MA (2016)
 46. Jurkiewicz, P., Rzym, G., Boryło, P.: Flow length and size distributions in campus Internet traffic. *Comput. Commun.* **167**, 15–30 (2021). <https://doi.org/10.1016/j.comcom.2020.12.016>
 47. Rajić B., Stanisavljević Ž., Vuletić P.: DAST scanning sessions dataset, Mendeley Data, V3 (2022). <https://data.mendeley.com/datasets/ctkh2zy6s3/3>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.