On Detecting Anomalous TLS Connections with Artificial Intelligence Models

On Detecting Anomalous TLS Connections with Artificial Intelligence Models

Diana Gratiela Berbecaru
Department of Control and Computer Engineering
Politecnico di Torino, Italy
diana.berbecaru@polito.it

Stefano Giannuzzi
Politecnico di Torino, Italy
Corso Duca degli Abruzzi 24, 10129, Torino (ITALY)

***Abstract*—In recent years, anomaly-based intrusion detection systems using machine learning (ML) and deep learning techniques have started to be developed to mitigate cybersecurity attacks. An anomaly-based intrusion detection system performs traffic analysis by exploiting supervised or unsupervised ML algorithms and raises alerts if a suspicious pattern is encountered. In this paper, we exploit the Autoencoder neural network model to detect variants of a very famous attack discovered in 2014, namely Heartbleed. The attack was caused by an implementation flaw in the OpenSSL library, widely used in web servers, database systems, or e-mail servers to support the Transport Layer Security (TLS) protocol. To evaluate our model, we exploited the CIC-IDS2017 dataset and a custom one created on purpose. The proposed model recognized the anomalous TLS connections containing variants of the Heartbleed attack and distinguished them from the benign traffic in 85% of the cases.**

***Index Terms*—TLS protocol, attack detection, anomalous connection, Heartbleed**

I. Introduction

The TLS protocol is the most widely used security protocol nowadays. It provides security features, like data authentication and integrity, confidentiality, peer authentication, protection from replay and filtering attacks to many application-level protocols, such as secure web and e-mail. At the same time, attackers continuously investigate the TLS protocol to find ways to circumvent it. Consequently, the protocol suffered various modifications and enhancements, the most recent version is TLS 1.3 [1], although the TLS version 1.2 is still in use.

The TLS attacks have been classified in different classes in [2] [3]: core cryptography, crypto usage, TLS protocol functionality, TLS configuration errors, or TLS implementation flaws. In core cryptography attacks, an attacker aims to exploit the vulnerabilities of the cryptographic algorithms used in the TLS protocol, like RC4, or MD5. In the crypto usage attacks, an adversary tries to exploit the mode of operation rather than the algorithm used, such as the attacks against the Cipher Block Chaining (CBC) mode. In the TLS protocol functionality attacks, an attacker targets particular features like the compression of the TLS Record Layer or protocol extensions. Another class of attacks is due to TLS configuration errors. Since the TLS protocol exploits X.509 certificates [4], these must be properly configured on the end nodes (including the trusted CA certificates). At least for the TLS server, the certificate is mandatory, while on the client side, the certificate is optional as users may employ alternative user-friendly authentication methods at application level, such as static passwords, one-time password generated by crypto tokens, smartphones, or other authentication methods recognized at local, national, or enterprise level [5] [6]. A possible attack is the Man In The Middle (MITM) attack [7] in which the attacker succeeds in corrupting the trusted CA store on the client side or manages to issue a malformed or self-signed certificate for the server (or obtain it from a compromised CA) and then convinces the client that the fake TLS server he puts in place (and the corresponding certificate) is valid. The end user's behavior makes MITM attacks still possible because users wrongly accept such bogus X.509 certificates in transactions. In the attacks due to errors in the implementation of TLS libraries, the adversaries look for bugs or development flaws that could be catastrophic in some cases, like the Hearbleed attack that was even called a "security disaster" [8] due to the severe impact (and cost) it generated. Throughout the years, several TLS libraries occurred, like OpenSSL, NSS, Bouncy Castle, Polar SSL, CyaSSL, MatrixSSL, MbedTLS, wolfSSL, BoringSSL, and GnuTLS.

Several TLS vulnerabilities have been discovered so far, such as [9] [10] [11] [12] [13] [14], and different countermeasures can be adopted for defending systems from the various TLS attack types. In case of cryptographic algorithms' vulnerabilities, the only efficient way is to deprecate the algorithm itself and recommend not to use that algorithm anymore in applications and libraries. To counter attacks targeting the certificates issued by CAs, some authors have proposed to use blockchains as decentralized storage for certificate-related data, and the TLS endpoints have been adapted to fetch such data from the blockchains [15] [16]. For the TLS implementation flaws, a classical solution is performing the following steps: i) individuate the bug that makes the (TLS) library/code prone to attacks; ii) implement a patch to fix the individuated error; and iii) make that patch available as widely as possible, and notify potential victims to apply the patch in short time. The window of exposure varies depending on the time required to perform each of the above steps.

A famous bug that affected the implementation of TLS protocol is the Heartbleed attack, which is a buffer overflow attack. To protect from this attack, the users have been advised

to install an updated OpenSSL version. To check whether a host is vulnerable to Heartbleed, a solution is given in [8]. The authors have modified Zmap to send Heartbeat requests with no payload nor padding, and the length set to zero and then checked whether the library correctly discarded the result. Nevertheless, we observe that a node might be corrupted by an internal attacker (e.g., by installing an old vulnerable version) or the attack could still occur in other variants. While remote attestation of installed nodes could by done to check whether the end points runs the correct software and version [17] [18], for the second case, more efficient methods (other than simply applying a patch or remote attestation) are needed to detect Heartbleed attack variants.

We propose a model that can identify potential Heartbleed attacks or variants of the attack in anomalous TLS connections by analyzing the network traffic. The *anomaly-based* concept is similar to allow listing: when the system detects a behavior outside an acceptable range, it is considered an anomaly. For this reason, an anomaly-based Intrusion Detection System (IDS) is more efficient for detecting unknown or zero-day attacks compared to a signature-based IDS system, which recognizes (only) the attacks discovered in the past that have known attack signatures.

Many anomaly-based IDS models exploit *supervised* machine learning models [19] that classify objects in a pool using a set of known features. Unfortunately, these IDS types do not recognize unknown attacks because they classify the network traffic based on what the model has learned in the training phase. If there is an "unknown" attack, their accuracy decreases since they have not been trained (in advance) on its features. In contrast, the unsupervised machine learning techniques form groups among the objects in a pool by identifying their similarity and then using them to classify the unknown samples.

In this work, we apply an autoencoder-based model previously proposed in [20] [21], to identify Heartbleed attacks in anomalous TLS connections. The autoencoder [22] recognizes the unknown attacks based on several TCP flow characteristics. We used the CIC-IDS2017 dataset[1], the TORSEC datasets [23], and a custom Heartbleed dataset (described in Sect. III) created on purpose. Compared to the Zavrak model in [24], which has used the same strategy, the model in [21] could detect the attacks with the CIC-IDS2017 dataset more effectively. Thus, we have exploited the above model for the Heartbleed attack detection.

The paper is organized as follows: Section II describes the Heartbleed attack as well as related work on detecting network anomalies through machine learning. Section II-B describes the Autoencoder model, Section III describes the adopted model and the datasets used for Heartbleed attack detection. Finally, Section IV resumes the conclusions and indicates future works.

[1]https://www.unb.ca/cic/datasets/ids-2017.html

II. Related Work

A. Heartbleed TLS attack

The extension called *Heartbeat* allows either end point of a TLS connection to check whether its counterpart is still present. It is very useful in those cases where neither of the nodes performs any action (download or upload) for a considerable period of time. The extension was motivated by the need of session management in DTLS (Datagram TLS), a variant of the protocol running over unreliable transport protocols. The standard implementation of TLS running over TCP do not require such extension because the reliable transport protocol supports the session management.

An endpoint indicates support for the Heartbeat extension during the initial TLS handshake phase. After a certain time, following the negotiation of security parameters, one of the two nodes sends the other some encrypted data within a message called *Heartbeat Request*, to verify connectivity. The second node responds with an exact copy of the sequence encrypted data received, to prove that the connection is still active. Introduced originally in February 2012 in RFC 6520, the extension was first released in the OpenSSL version 1.0.1 implementation, and rendered available on March 14, 2012. In March 2014, a security engineer from Google Security discovered a bug related to the implementation this extension, which could lead to the theft of sensitive data, in particular the private memory, potentially including information transferred over the secure channel and cryptographic secrets.

The bug involved the length of the Heartbeat Request present in the message itself. When a computer receives a Heartbeat Request, it reads the information regarding the length of the message and allocates a memory buffer equal to that length in which it saves the encrypted data, finally reads the data again and sends it as a reply to the other computer. The vulnerability occurred because the OpenSSL implementation in question did not check the length of the message in order to verify that it was actually equal to the one declared inside. The attack was particularly damaging because in the extra data released could have been stored sensitive data, like a server's private key. Thus, as a consequence of this attack, a huge amount of certificates have been revoked. This operation was costly in terms of money (CloudFlare estimated a corresponding cost of approximately $400,000 per month [26]) and size of Certificate Revocation Lists (CRLs) that had to be downloaded by the clients, which has grown from a few KB to about 4.7 MB due to CloudFlare's revocations [8]. Moreover, Heartbleed had the potential to affect any service that used OpenSSL to establish TLS connections, including database servers, e-mail servers, or popular web sites. As stated in [8], "between 24-55% of HTTPS-enabled servers in the Alexa Top 1 Million were initially vulnerable, including 44 of Alexa Top 100".

B. Autoencoder-based model for anomaly detection

Autoencoder (AE) is an unsupervised Artificial Neural Network that learns to compress data and learns how to rebuild

it, reproducing the original input. The autoencoder has the characteristic that it ignores the input noise and lowers the dimensions of the dataset [22]. An early proposal to use Autoencoders for anomaly detection and condition monitoring is given in [27]. The article presented a problem of predicting bearing failure in a factory and used only the "known good" to train the autoencoder.

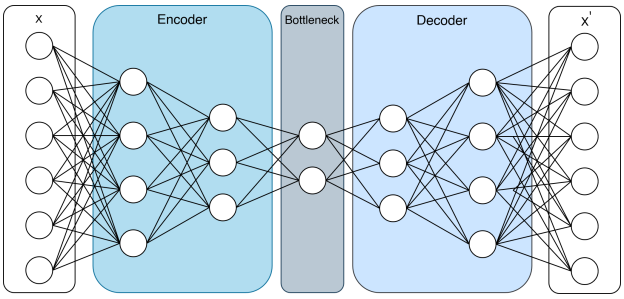

Fig. 1. Autoencoder model representation.

The autoencoder, "by design, reduces data dimensions by learning how to ignore the noise in the data" [28]. While the autoencoder creates a compressed representation of a dataset, it attempts to maintain enough variance to reconstruct the original data. As the autoencoder ignores the noise, it looks for correlations (unique) between various features and extracts those correlations for data reconstruction. As shown in Fig. 1, the autoencoder consists of three parts: an encoder, which is used by the model for learning to compress and reduce the dimensionality of input data, a bottleneck, which contains the data compressed and is the layer with fewer neurons in the model, and a decoder, which is used by the model for learning to reconstruct the original input data. At the end of the process, the Reconstruction Error (RE) is used to understand how well it performs the autoencoder: the trained AE will reconstruct normal input with very low RE, whereas it is unsuccessful to do so with anomalous data.

Several researchers have addressed the problem of detecting unknown attacks through anomaly-based solutions. In [29], researchers have shown how an ensemble model composed of different decision trees is more efficient in identifying an atypical attack, that is attacks with different profiles, than a single model like a Linear Support Vector Classifier or a Dense Neural Network. These models used only supervised learning techniques but they did not simulate how the model behaves if there is an unknown attack, i.e., attack for which the model was not trained.

To detect unknown attacks, the authors in [24] exploited two approaches, namely an autoencoder (AE) or a Variant Autoencoder (VAE), to detect network anomalies. The authors used a semi-supervised learning strategy, and only benign data flow was utilized in the construction of the models from the CIC-IDS2017 dataset. The AE and VAE models were evaluated by using both normal and anomaly data. Their results were quite good, the proposed AE model had an AUC of 73%, while the VAE model had an AUC of 76%. While the previous works detect generic attacks, malware detection in encrypted TLS traffic by using machine learning analysis has been addressed in [25].

III. Exploiting Autoencoder-based model for Heartbleed TLS attack detection

In the autoencoder-based models described in [24], the authors trained the model only on benign traffic and not on the attacks, meaning that the model recognizes only the benign traffic, and anything else is considered anomaly. We used the same procedure in our work. As explained in Section II-B, the autoencoder compresses data and learns how to rebuild it. In this process, it generates an RE. So, if the AE was only trained on benign traffic, then it recognizes it when (benign traffic) is encountered, making a very small RE. If the encoder encounters traffic that it has not been trained to recognize, it generates a large RE and over a threshold, it considers it as an "anomaly". More precisely, the AE only distinguishes normal and anomalous traffic. In the testing process, all the classes (DoS Slowloris, DoS SlowHttpTest, Bruteforce SSH, Heartbleed, Infiltration, Bot, ...) have been re-called "anomaly" to understand if the methods can distinguish the unknown attacks from normal traffic. For the single class evaluation, the authors selected the attack class they wanted to evaluate, they excluded the other attack classes. For example, to evaluate only the DoS Slowloris attack, the authors have got all the records with the label DoS Slowloris, next they combined them with the benign traffic, and then they processed the resulting set with their AE model.

Given the good results reported in [24], we have also adopted the AE model described therein given its ability to detect new types of attacks and, thus, its proven efficiency in anomaly-based detection. We have however increased the number of hidden layers to 4. We have also used additional datasets in the training and testing phases, as described below.

a) Datasets: We have exploited the CIC-IDS2017 dataset for the training, validation, and testing phases. This dataset covers several attacks like Denial of Service (DoS), Distributed Denial of Service (DDoS), Brute Force, XSS, SQL Injection, Infiltration, Port Scan, Botnet, and Heartbleed attacks. The dataset contains 84 features for each TCP connection, including the label that classifies the type of traffic created.

In the testing phase, we also used the Heartbleed dataset that we have created to test our model for this vulnerability. We used these two datasets to understand how our proposed model behaves with a different dataset for which it is not trained.

To increase the number of Heartbleed cases used in testing the model, we created the Heartbleed dataset by exploiting a lab testbed in which we have set up 3 Ubuntu virtual machines (VMs) and 1 Ubuntu VM server. On the Ubuntu Server machine, we installed a container docker (jas9reet/heartbleed [2]) that uses the OpenSSL version library (1.0.1) vulnerable to the Heartbleed attack, and an Apache server on port 8443. The clients running on the three Ubuntu VMs run a script written

[2]https://github.com/jas9reet/heartbleed-lab

in bash that gets the page from the Apache server (via HTTPS) at different times.

To generate benign traffic the clients have accessed the web server, which showed a login page asking to insert the login credentials. In this way, we simulated real traffic with access credentials. To increase benign traffic, this traffic was combined with some parts of the benign traffic of the TORSEC dataset [23], which contains traffic for the Bot and DoS attacks (GoldenEye, Hulk, Slowloris, and Httptest).

To generate Heartbleed traffic, we exploited a machine running Kali Linux distribution and Metasploit [3]. Then, the command for the heartbleed vulnerability exploit was used [4]. Finally, the generated traffic was captured with TCPdump tool which generated three pcaps (2 benign and 1 malicious) that were converted to .csv files by using the CICFlowMeter tool. Table I shows the characteristics of the exploited datasets.

Dataset	Label	Number of samples
CIC-IDS2017	BENIGN	1652527
	Denial of Service Hulk	171974
	Distributed Denial of Service	128011
	Denial of Service GoldenEye	10281
	FTP-Patator	5931
	Denial of Service Slowloris	5276
	Denial of Service SlowHttptest	5186
	SSH-Patator	3153
	PortScan	1922
	Web Attack Brute Force	1427
	Bot	1337
	Web Attack XSS	652
	Infiltration	36
	Web Attack SQL Injection	20
	Heartbleed	11
TORSEC	Denial of Service GoldenEye	343258
	Denial of Service Hulk	193718
	BENIGN	40514
	Denial of Service Slowloris	5500
	Denial of Service SlowHttptest	3838
	Bot	3598
Heartbleed	BENIGN	2713
	Heartbleed	866

TABLE I
DATASETS EXPLOITED.

b) Features: We randomly split the CIC-IDS2017 dataset into 64% for the training set, 16% for the validation set, and 20% for the test set. Starting from the 84 features of the CIC-IDS2017 dataset, we performed a feature selection using a random forest classifier to select the most promising attributes. We ignored features like the flow ID, source and destination IP addresses, protocol, and source and destination ports, as they do not bring relevant information. For each feature, we computed the distributions based on the value assumed by a label and the normalized frequency that the value has assumed. The frequency has been transformed into a value in the range [0,1]. If a significant difference between the benign traffic and its malicious traffic counterpart was encountered, then that feature was selected, otherwise, it was discarded. The remaining features that we used are shown in Table II.

[3]https://www.metasploit.com/

[4]https://github.com/jas9reet/heartbleed-lab

TABLE II
SELECTED FEATURES AFTER THE FEATURE SELECTION.

#	Feature name	Description
1	Flow_duration	Duration of flow in microseconds
2	Bckwd_pkt_len_max	Maximum size of packets in backward direction
3	Bckwd_pkt_len_std	Standard size of packets in backward direction
4	Flow_time_mean	Average time between two packets sent in the flow
5	Flow_time_std	Standard time between two packets sent in the flow
6	Flow_time_max	Maximum time between two packets sent in the flow
7	Fwd_time_tot	Total variation time between two packets sent in forward direction
8	Fwd_time_mean	Mean variation time between two packets sent in forward direction
9	Fwd_time_std	Standard variation time between two packets sent in forward direction
10	Fwd_time_max	Maximum time between two packets sent in the flow
11	Pkt_len_max	Maximum length of a packet (payload + header), the MTU (Maximum Transmission Unit)
12	Pkt_len_mean	Mean length of a packet (payload + header)
13	Pkt_len_std	Standard packet length (payload + header)
14	Pkt_len_var	Variance packet length (payload + header)
15	Bckwd_time_tot	Total variation time between two packets sent in backward direction
16	Pkt_size_avg	Average packet size (payload only), equal to MSS (Maximum Segment Size)
17	Bckwd_seg_size_avg	Average packet size (payload only) in the backward direction, equal to MSS
18	Fwd_pkt_len_std	Size of packets in forward direction (Standard)
19	Fwd_pkt_len_max	Size of packets in forward direction (Maximum)
20	Bckwd_pkt_len_min	Size of packets in backward direction (Minimum)

c) Implementation details: The Autoencoder-based model is similar to the one in [24], and is composed of three parts: 1) the encoder, composed of 4 layers of 512 neurons, 64 neurons, 16 neurons, and 8 neurons; 2) the bottleneck, composed of 2 neurons; 3) the decoder, composed of 4 layers of 8 neurons, 16 neurons, 64 neurons, 512 neurons. To avoid overfitting, for each layer excluding the input, the output, and the bottleneck layer, we put a dropout layer with a probability of 0.2 [30]. The training and validation are performed only on benign traffic, which means that all the attack labels are not considered. The flowcharts for AE training algorithm and for the AE-based anomaly detection algorithms are provided in [24]. The optimal hyperparameters were determined through trial and error and are $1e^{-4}$ for the learning rate and 512 for the batch size.

To implement the model, we used Python 3.9.0 with Tensorflow 2.9.1 and Keras 2.9.0. In particular, to reduce the training time and use the GPU, we used the Conda version of Tensorflow 2.9.1[5] with cudatoolkit 11.2[6]. For the

[5]https://www.tensorflow.org/install/pip#linux

[6]https://developer.nvidia.com/cuda-toolkit

TABLE III
AUC RESULTS OF TESTS WITH THE CIC-IDS2017 DATASET.

LABEL	RF	XGB	OUR MODEL	ZAVRAK [24]
Bot	*1.0*	*1.0*	0.51	0.62
Denial of Service GoldenEye	*1.0*	*1.0*	0.94	0.75
Denial of Service Hulk	*1.0*	*1.0*	0.96	0.83
Denial of Service Slowhttptest	*1.0*	*1.0*	0.92	0.85
Denial of Service Slowloris	*1.0*	*1.0*	0.81	0.84
FTP-Patator	*1.0*	*1.0*	0.70	0.74
Heartbleed	*1.0*	*1.0*	***1.0***	0.98
Infiltration	0.86	*0.98*	0.90	0.89
SSH-Patator	*1.0*	*1.0*	0.77	0.68
micro AUC	*1.0*	*1.0*	0.94	0.73
balanced accuracy	0.89	*0.91*	0.89	-

data preprocessing, as well as to create the confusion matrix and the ROC (Receiver Operating Characteristics) curve, we used scikit-learn, matplotlib, and seaborn[7]. For creating the Heartbleed dataset, the tool CICFlowMeter[8] was used to create the csv file from the pcap files. To open the csv file we used pandas[9] and numpy[10].

d) Results.: ***Tests with CIC-IDS2017 dataset.*** Next, the testing results obtained with the CIC-IDS2017 dataset are presented, as well as a comparison with Random Forest (RF) and (eXtreme Gradient Boosting) XGB models and the model in [24]. We trained first the proposed model only with the benign traffic in the CIC-IDS2017 dataset. Thus, all the anomalous traffic in this dataset was discarded in the training and validation phases. Then, in the testing phase, we considered only one attack class and excluded the others, i.e., DoS Slowloris, Next, we tested our model with that attack class joined with some benign traffic. Then, we repeated the same process for the other attack classes, including the Heartbleed attack class.

Since the proposed model has been trained to recognize only benign traffic, it generates a high RE when it encounters traffic not seen before. In this case, the error for benign traffic is lower than 0.10-0.15, while for anomaly traffic, it reaches 0.30. We chose a threshold with the trial and error strategy, which was set to 0.01. Under this threshold, the traffic is considered benign, otherwise, it is considered an anomaly.

For comparison, we have also trained and tested with the same dataset two supervised ML models, namely RF and XGB. Table III shows that the proposed model has a high accuracy (AUC) for the Heartbleed attack, comparable with the supervised models and better than the original model [24].

Tests with CIC-IDS2017 and Heartbleed datasets. In these tests, the proposed model was trained with the benign traffic from the CIC-IDS2017, and then it was tested with the Heartbleed dataset.

[7]https://seaborn.pydata.org/
[8]https://www.unb.ca/cic/research/applications.html
[9]https://pandas.pydata.org/
[10]https://numpy.org/

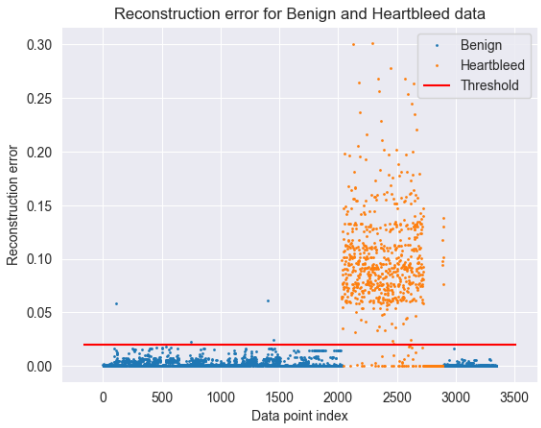

Fig. 2. RE of the proposed model tested with the Heartbleed dataset.

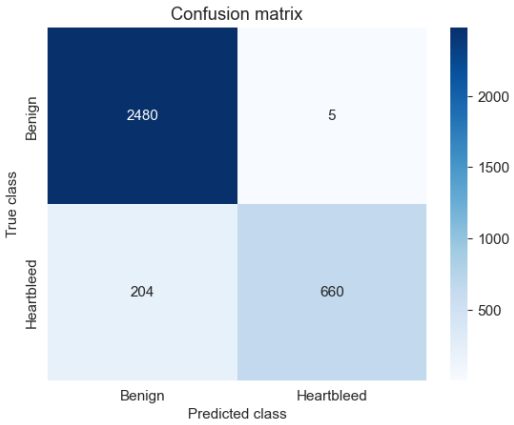

Fig. 3. Proposed AE-based model tested with the Heartbleed dataset.

By analysing the RE shown in Fig. 2 we observe that there is a distinct division between the benign and the Heartbleed anomalous traffic, just a few benign records are over the threshold, which is also set using the trial-and-error strategy.

Fig. 3 shows the confusion matrix of the proposed model for the testing phase with the Heartbleed dataset. In this case, there are few false negatives, if we consider the benign traffic, and just some false positives. The model recognizes the benign traffic (of the Heartbleed dataset), considering it has never seen that traffic before because it is trained with the benign traffic of the CIC-IDS2017 dataset. Moreover, it distinguishes the Heartbleed anomalies from the benign traffic. In this test, the proposed model has 88% of balanced accuracy.

The ROC curve, shown in Fig. 4, indicates that the model is able to distinguish the positive and negative classes. It achieves an AUC value of 0.85, which means that the model recognizes the anomalies caused by the Heartbleed and it distinguishes them from the benign traffic in 85% of cases.

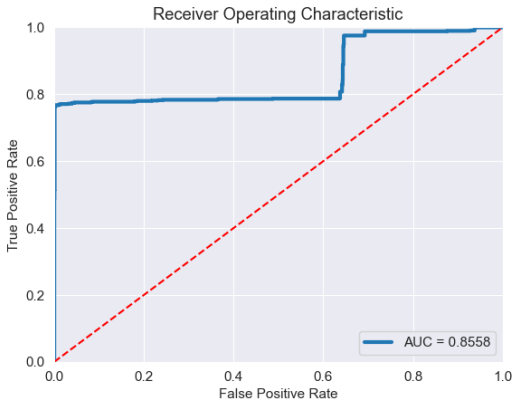

Fig. 4. ROC curve of the proposed model for the Heartbleed dataset.

IV. Conclusions

While the encrypted TLS connections are increasing continuously, more efficient solutions are required to detect the anomalous ones, which could be used for performing attacks, malware distribution, or data exfiltration. Herein, we considered the Heartbleed attack, which affected the OpenSSL library several years ago. To detect variants of the attack, we exploited an autoencoder unsupervised model trained in a semi-supervised learning manner. Through experimental tests with the considered datasets, we show that the model can detect the Heartbleed attacks with a performance comparable to the supervised learning methods.

Acknowledgments. Dr. Diana Gratiela Berbecaru carried out this study within the Ministerial Decree no. 1062/2021 and received funding from the FSE REACT-EU - PON Ricerca e Innovazione 2014-2020. Stefano Giannuzzi performed his work during preparation of his graduation thesis at Politecnico di Torino. This manuscript reflects only the authors' views, findings, conclusions, and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

[1] E. Rescorla, "The Transport Layer Security (TLS) Protocol: Version 1.3," IETF RFC 8446, https://tools.ietf.org/pdf/rfc8446.pdf.

[2] D. Stebila, "Attacks on TLS,"https://www.douglas.stebila.ca/research/presentations/tls-attacks/.

[3] D. G. Berbecaru and G. Petraglia, "TLS-Monitor: A Monitor for TLS Attacks," 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2023, pp. 1-6, doi: 10.1109/CCNC51644.2023.10059989.

[4] D. Cooper et al., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." IETF RFC 5280, https://www.rfc-editor.org/rfc/rfc5280.txt.

[5] D. G. Berbecaru, A. Lioy, and C. Cameroni, "Providing Login and Wi-Fi Access Services With the eIDAS Network: A Practical Approach," *IEEE Access*, 2020, vol. 8, pp. 126186-126200, doi: 10.1109/ACCESS.2020.3007998.

[6] D. G. Berbecaru, A. Lioy, and C. Cameroni, "On enabling additional natural person and domain-specific attributes in the eIDAS network," *IEEE Access*, 2021, vol. 9, pp. 134096-134121, doi: 10.1109/ACCESS.2021.3115853.

[7] S. Stricot-Tarboton, S. Chaisiri and R. K. L. Ko, "Taxonomy of Man-in-the-Middle Attacks on HTTPS," 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 2016, pp. 527-534, doi: 10.1109/TrustCom.2016.0106.

[8] Z. Durumeric et al., "The Matter of Heartbleed," In Proc. of the 2014 Conference on Internet Measurement Conference (IMC '14). ACM, New York, NY, USA, 475–488. doi: 10.1145/2663716.2663755.

[9] D. Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1," In: CRYPTO '98, pp. 112, London, UK, 1998, Springer-Verlag. doi: 10.1007/BFb0055715.

[10] N. Aviram et al., "DROWN: breaking TLS with SSLv2," In: 25th USENIX Security Symposium, August 10–12, 2016, Austin, TX (USA), https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_aviram.pdf.

[11] K. Bhargavan and G. Leurent, "Transcript collision attacks: breaking authentication in TLS, IKE and SSH," In: 23rd Annual Network and Distributed System Security Symposium, NDSS, Feb 2016, San Diego, United States, doi: 10.14722/ndss.2016.23418.

[12] Heartbleed Bug, https://heartbleed.com.

[13] Y. Gluck, N. Harris, and A. Prado. "BREACH: reviving the CRIME attack," 2013, http://breachattack.com/.

[14] M. Green, "A Diversion: BEAST Attack on TLS/SSL Encryption," 2011, https://blog.cryptographyengineering.com/2011/09/21/brief-diversion-beast-attack-on-tlsssl/.

[15] D. G. Berbecaru and L. Pintaldi, "Exploiting Emercoin Blockchain and Trusted Computing for IoT Scenarios: A Practical Approach," 2023 IEEE Symposium on Computers and Communications (ISCC), Gammarth, Tunisia, 2023, pp. 771-776, doi: 10.1109/ISCC58397.2023.10217961.

[16] S. R. Garzon, D. Natusch, A. Philipp, A. Küpper, H. J. Einsiedler, D. Schneider, "DID Link: Authentication in TLS with Decentralized Identifiers and Verifiable Credentials," May 2024, http://https://arxiv.org/abs/2405.07533v2.

[17] E. Bravi, D. G. Berbecaru and A. Lioy, "A Flexible Trust Manager for Remote Attestation in Heterogeneous Critical Infrastructures," 2023 IEEE Intl. Conf. on Cloud Computing Technology and Science (CloudCom), Naples, Italy, 2023, pp. 91-98, doi: 10.1109/CloudCom59040.2023.00027.

[18] D. G. Berbecaru et al., "Mitigating Software Integrity Attacks With Trusted Computing in a Time Distribution Network," in IEEE Access, vol. 11, pp. 50510-50527, 2023, doi: 10.1109/ACCESS.2023.3276476.

[19] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning Intrusion Detection: Supervised or Unsupervised?" In: Roli, F., Vitulano, S. (eds) Image Analysis and Processing – ICIAP 2005. ICIAP 2005. LNCS, vol 3617. Springer, Berlin, Heidelberg. doi: 10.1007/11553595_6.

[20] D. G. Berbecaru, S. Giannuzzi and D. Canavese, "Autoencoder-SAD: An Autoencoder-based Model for Security Attacks Detection," 2023 IEEE Symposium on Computers and Communications (ISCC), Gammarth, Tunisia, 2023, pp. 758-763, doi: 10.1109/ISCC58397.2023.10217930.

[21] S. Gianuzzi, "Artificial Intelligence for Security Attacks Detection," Master Degree Thesis, Politecnico di Torino, Italy, Dec. 2022. https://webthesis.biblio.polito.it/secure/25562/1/tesi.pdf.

[22] Y. Song, S. Hyun, and Y-G. Cheong, "Analysis of Autoencoders for Network Intrusion Detection," Sensors, 2021; 21(13):4294, doi: 10.3390/s21134294.

[23] D. Canavese, L. Regano, C. Basile, G. Ciravegna, and A. Lioy, "Data set and machine learning models for the classification of network traffic originators," *Data in Brief*, Vol. 41, 2022, 107968, doi: 10.1016/j.dib.2022.107968.

[24] S. Zavrak and M. İskefiyeli, "Anomaly-Based Intrusion Detection From Network Flow Features Using Variational Autoencoder," IEEE Access, vol. 8, pp. 108346-108358, 2020, doi: 10.1109/ACCESS.2020.3001350.

[25] B. Scarbrough, "Malware Detection in Encrypted TLS Traffic Through Machine Learning," https://www.giac.org/paper/gcia/14008/malware-detection-encrypted-tls-traffic-machine-learning/157200.

[26] M. Prince, "The Hidden Costs of Heartbleed," April 2014, https://blog.cloudflare.com/the-hard-costs-of-heartbleed.

[27] V. Flovik, "How to use machine learning for anomaly detection and condition monitoring," 2018, https://towardsdatascience.com/how-to-use-machine-learning-for-anomaly-detection-and-condition-monitoring-6742f82900d7.

[28] W. Badr, "Auto-Encoder: What Is It? And What Is It Used For? (Part 1)", 2019, https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726.

[29] U. Sabeel, S. S. Heydari, K. Elgazzar and K. El-Khatib, "Building an Intrusion Detection System to Detect Atypical Cyberattack Flows," in IEEE Access, vol. 9, pp. 94352-94370, 2021, doi: 10.1109/ACCESS.2021.3093830.

[30] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors,", 2012, doi: 10.48550/arXiv.1207.0580.