



Research paper

Transport Layer Security attack detection using hardware performance counters side-channel

Tomasz Głowka^a, Cheng-Feng Hung^b, Wojciech Mazurczyk^a, Shin-Ming Cheng^b^a Warsaw University of Technology, Plac Politechniki 1, Warsaw, 00-661, Poland^b National Taiwan University of Science and Technology, No. 43, Sec. 4, Keelung Rd., Da'an Dist., Taipei, 106335, Taiwan

ARTICLE INFO

Keywords:

Transport Layer Security

Side-channel

Hardware performance counters

Deep neural network

Heartbleed

ABSTRACT

Cryptographic protocols form the backbone of secure communication in various environments, including the Internet of Things (IoT) and non-IoT settings. One of the most popular and widely used cryptographic protocols is Transport Layer Security (TLS). However, TLS implementations remain vulnerable to implementation flaws, such as the Heartbleed bug, which affected a significant proportion of deployed TLS 1.2 servers. Many detection approaches rely on network traffic inspection, which is ineffective in encrypted environments. In this paper, we present a non-intrusive detection framework that collects hardware performance counter (HPC) side-channel data from a TLS server process without modifying the application or requiring access to network packets. To the best of our knowledge, no prior work has combined HPC data with a deep neural network (DNN) in a zero-modification testbed for TLS attack detection. The training dataset was collected in a simplified single-machine setup, while evaluation was conducted on physically separated machines to simulate real-world deployment. Recall ranges from 0.590 to 0.753, and Specificity from 0.698 to 0.850 across six evaluation configurations, confirming the feasibility of detection while identifying generalization as the key challenge for future work. To validate the choice of DNN architecture, it was additionally compared against classical machine learning baselines such as Logistic Regression, Random Forest, and Extreme Gradient Boosting (XGBoost), achieving the highest combined Recall and Specificity in all six configurations.

1. Introduction

Cryptographic protocols aim to provide a secure communication channel between peers. One of the most popular and widely used cryptographic protocols is Transport Layer Security (TLS) (IETF, 2025c), currently in version 1.3. The TLS protocol is a successor to the Secure Sockets Layer (SSL) specification developed by Netscape Communications. Despite its role in protecting others, the TLS protocol is an active target of attacks. TLS itself is just a specification, meaning there are many general-purpose implementations, such as OpenSSL (OpenSSL, 2025) and BoringSSL (BoringSSL, 2025), as well as dedicated implementations for embedded systems and Internet of Things (IoT) products, including WolfSSL (WolfSSL, 2025). Each implementation and the specification itself are vulnerable to design and programming errors. Throughout the years, numerous confirmed attacks and vulnerabilities have been discovered in the TLS protocol, including Heartbleed, BEAST, CRIME, BREACH, DROWN, SLOTH, ROBOT, and Lucky 13 (Ivanov et al., 2018). Some attacks rely on the exploitation of side-channel information (Lucky 13 (Al Fardan and Paterson, 2013)), while

other attacks were caused by bugs in the software implementation (unauthorized memory access in Heartbleed (Heartbleed, 2025)).

One of the best strategies to mitigate an attack is to use the latest version of the software. However, despite widespread adoption of the latest TLS version, many platforms still support older versions. According to SSL Labs (SSL Labs, 2025) (Monthly Scan: June 02, 2025), TLS support of the surveyed sites is as follows: 75.3% support TLS v1.3, 100% support TLS v1.2, 25.2% support TLS v1.1, 23.5% support v1.0, 1% support SSL v3.0, and 0.1% support SSL v2.0. From an IoT perspective, adopting the latest TLS version appears somewhat concerning. Support for TLS v1.3 in selected IoT devices across ten categories (Router, Printer, Camera, DVR, NVR, VoIP, NAS, Modem, TV, Thermostat) varies around 10% (in fact, most are less than 10%) (Siwakoti and Rawat, 2023).

Due to widespread support for older TLS versions and numerous attack vectors, this paper proposes a non-intrusive detection framework to detect the Heartbleed attack. We consider the Heartbleed attack a good starting point for this relatively narrow and unexplored research area: detecting TLS attacks using hardware performance counter (HPC)

* Corresponding author.

E-mail address: tomasz.glowka.dokt@pw.edu.pl (T. Głowka).

side-channel information and a deep neural network (DNN). This choice is driven by the attack's significant impact, simplicity, and reproducibility. To the best of our knowledge, no prior work has combined HPCs with a DNN in a non-intrusive, zero-modification testbed specifically designed for TLS attack detection. The work most similar to ours (Torres and Liu, 2016) employs HPC data with Support Vector Machine (SVM) classifiers and requires instrumenting the OpenSSL library with `perf_events` API calls. Our framework avoids any server-side modification, instead using the standard Linux `perf` tool to monitor the server process externally. This constitutes the research gap the present work addresses. The main novel contributions of this work are as follows:

- The proposed framework does not require external sensors, kernel modifications, or changes to the running application.
- The introduced solution captures side-channel information, such as HPCs, and converts it into features that are used to feed the DNN model.
- We systematically evaluated various DNN models to assess the feasibility of an HPC-based attack-detection method for the Heartbleed vulnerability.

The rest of this article is structured as follows. In Section 2, we explain crucial terms and definitions related to the article, including TLS, side-channel information, HPCs, the Heartbleed attack, and DNNs. Then, Section 3 presents related work. Sections 4, 5, and 6 present the proposed framework, the experimental methodology, and the results. Finally, Section 7 concludes our work.

2. Background

In this section, we provide the background information necessary to understand the proposed framework. We briefly explain the TLS cryptographic protocol, side-channel information, HPCs, the Heartbleed attack, and DNN.

2.1. Transport Layer Security

Transport Layer Security (IETF, 2025c) is a cryptographic protocol designed to provide secure communication over an insecure channel. The established secure channel should provide three important features: authentication, confidentiality, and integrity. Authentication is a process in which a peer demonstrates its identity. Confidentiality means that data sent over the channel is visible only to the intended recipients or peers. Integrity ensures that data sent over the channel cannot be modified by an adversary without detection. To achieve these features, TLS uses cryptographic primitives such as hash functions, symmetric-key cryptography, asymmetric-key cryptography, digital signatures, message authentication codes (MACs), and key derivation functions. The latest version of TLS is 1.3, released in August 2018, and is a successor to version 1.2, released in August 2008. The latest version introduced improvements across various aspects of the protocol, including reduced network latency, forward security, reduced information transmitted in the clear, and the refresh of cryptographic primitives (Ristić, 2022).

TLS is based on the handshake protocol. The handshake protocol aims to agree upon and exchange cryptographic parameters, validate the server's identity (optionally, the client's identity as well), and ensure that the handshake itself has not been modified by an adversary. Fig. 1 presents the simplified TLS handshake protocol. The handshake consists of three phases: Key Exchange, Server Params, and Auth. The Key Exchange phase establishes shared keying material and selects cryptographic parameters (communication after this phase is encrypted). The Server Params phase establishes additional handshake parameters (e.g., whether the client is authenticated). The Auth phase authenticates the server and provides key confirmation and handshake integrity.

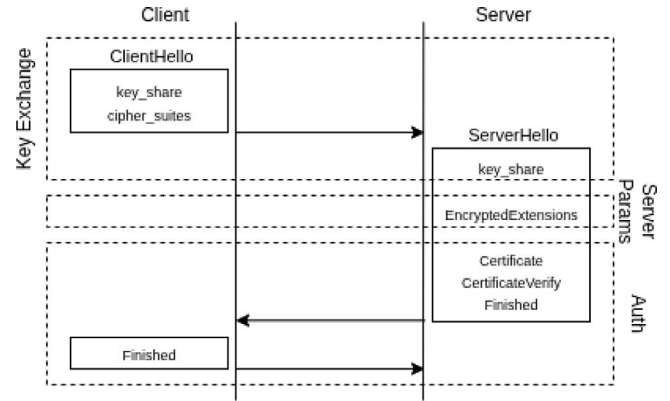


Fig. 1. TLS 1.3 handshake phases (RFC 8446 (IETF, 2025c)). Note: The Heartbleed vulnerability affects only TLS 1.2 and earlier implementations. The heartbeat extension (RFC 6520 (IETF, 2025b)) was not carried forward into TLS 1.3. TLS 1.3 is shown here as the current protocol standard.

First, the Client sends the ClientHello message, which includes a set of Diffie–Hellman key shares, cipher suites, and other necessary handshake details (omitted for clarity). The server responds with the ServerHello message, which includes fields for all three handshake phases: the selected key share, the EncryptedExtensions, and the Certificate, CertificateVerify, and Finished fields. Finally, as the last step of the Auth phase, the Client sends the Finished message to complete the handshake. Once all three phases are successful, the client and server can exchange messages over a protected channel.

The continued universal deployment of TLS 1.2 (as mentioned in Section 1) means that potentially Heartbleed-vulnerable configurations remain in production, motivating active detection research.

2.2. Side-channel information

The term side-channel refers to an observable source of information arising from the execution of computing tasks in electronic devices. When running a program, the machine consumes time, power, radiates an electromagnetic field, produces noise, etc. (Tsalis et al., 2019). Observation and detailed analysis of this data source can reveal valuable system properties to an adversary that can eventually be used to prepare an attack. This type of threat is called a side-channel attack. However, side-channel information can also be used to detect attacks or anomalies (Zhang et al., 2020). Another source of side-channel information could be collected using HPCs (Uhsadel et al., 2008; Bhattacharya and Mukhopadhyay, 2015; Park et al., 2019), which are described in 2.3. In this work, HPC data serve as the side-channel observable used to distinguish Heartbleed attack traffic from benign TLS sessions.

2.3. Hardware performance counters

Hardware performance counters are special registers built into processors that store various hardware- and software-related events (Malone et al., 2011), such as cache misses, branch misses, CPU cycles, context switches, and more. HPCs can be applied in various areas, including debugging, dynamic software profiling, CPU power modeling, vulnerability research, and malware defense (Das et al., 2019). To obtain HPC data, the `perf_events` (man7.org, 2025), or PAPI (Mucci et al., 1999) tools can be used. These properties make HPC a promising non-intrusive data source for detecting the memory-access anomalies characteristic of TLS attacks, such as the Heartbleed attack.

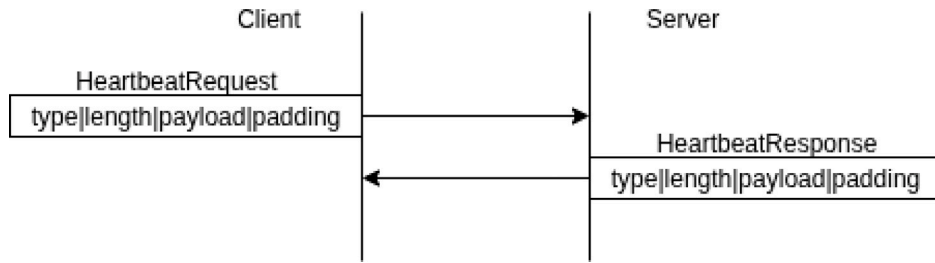


Fig. 2. The Heartbeat request and response scenario (IETF, 2025b).

2.4. Heartbleed

Heartbleed (Heartbleed, 2025) (official reference: CVE-2014-0160) is a vulnerability in the OpenSSL library that allows a malicious client to read the memory of the system where the vulnerable TLS server runs. This type of vulnerability is called a buffer over-read (Wang et al., 2015) and occurs when a program reads adjacent memory beyond the buffer boundary. The vulnerability itself is not related to the architecture of the TLS protocol; this is solely an implementation problem, that is, a programming mistake. The vulnerability exploits the heartbeat feature implemented in OpenSSL via the TLS Heartbeat Extension (IETF, 2025b).

The Heartbeat request and response scenario is shown in Fig. 2. The request and response consist of four parts: type, length, payload, and padding. The server responds to the client's request by sending back the same payload of the given length. However, the Heartbleed attack exploits the fact that the server trusts the client length part and does not validate it against the payload size. The attacker sends a malicious request specifying a length part larger than the number of bytes in the payload part, which eventually causes the TLS server to perform an unauthorized memory access. Because the length part is two bytes, the server responds with up to ~64 KB of memory to the adversary. OpenSSL 1.0.1 through 1.0.1f (inclusive) are vulnerable. The unusual pattern of memory-read operation caused by Heartbleed is expected to produce a distinctive HPC signature, which is the basis for the detection approach proposed in this work.

2.5. Deep neural networks

In recent years, neural networks (and artificial intelligence in general) have gained significant public attention, exemplified recently by large language models. However, they have myriad other applications, such as speech recognition, image processing, pattern recognition, and classification. An artificial neural network is a collection of interconnected processing elements (called neurons) inspired by the human brain in analyzing and processing data (Dastres and Soori, 2021). The neurons are organized into layers, as illustrated in Fig. 3. The example presented consists of an input layer, two hidden layers, and an output layer. An artificial neural network is called a deep neural network when it has more than one hidden layer.

The input signals are processed through weighted connections. Each neuron computes an activation function based on the provided inputs. Thanks to the activation function, the neural network can achieve non-linearity, which is crucial for capturing complex patterns. The weights are adjusted during training; therefore, selecting the appropriate training dataset and environment is crucial for optimal results. In this work, various DNN architectures are trained and evaluated to learn the HPC signature distinguishing Heartbleed attack from benign TLS traffic.

3. Related work

Attack detection using HPCs is an active area of research (Li and Gaudiot, 2019; Singh et al., 2017; Sayadi et al., 2024; Li and Gaudiot, 2022; Foreman, 2018; Prada et al., 2019; Carnà et al., 2023; Bazm et al.,

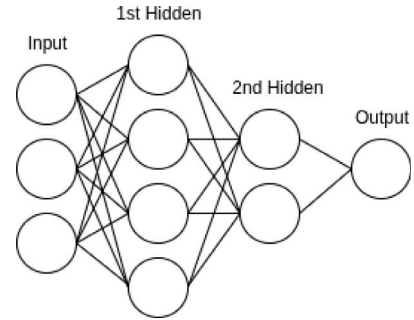


Fig. 3. The artificial neural network.

Table 1

Performance comparison of rule-based approach and machine learning classifiers (Amodei et al., 2023).

Method	Acc.	Prec.	Sen.	Spec.	F1-Score	AUC
DT	1.000	1.000	1.000	1.000	1.000	1.000
GNB	0.950	1.000	0.890	1.000	0.940	0.990
K-NN	0.900	0.890	0.890	0.910	0.890	0.970
MLP	0.900	0.890	0.890	0.910	0.890	0.970
RF	1.000	1.000	1.000	1.000	1.000	1.000
RBA	0.982	0.999	0.964	0.999	0.981	0.982

2018; Kosasih et al., 2024; Putrevu et al., 2023; Demme et al., 2013; Chiappetta et al., 2016; Zhang et al., 2016). When it comes to detecting a specific attack in a cryptographic protocol, such as Heartbleed, using artificial intelligence, this is somewhat of a niche topic. However, prior work has proposed various detection techniques for identifying the Heartbleed attack.

In Amodei et al. (2023), the authors consider IoT and resource-constrained systems and propose a rule-based solution to detect Heartbleed-like attacks. The authors propose three rules, RL1, RL2, and RL3, based on measurements of network traffic features (obtained from a custom testbed), including the maximum, average, and standard deviation of packet lengths. The feature data are collected using open-source traffic monitoring software CICFlowMeter. They defined threshold values for each rule, and if all rules are true, then a possible Heartbleed attack is identified. The results obtained from the rule-based approach (RBA) were also compared with machine learning classifiers, including Decision Tree (DT), Gaussian Naive Bayes (GNB), K-Nearest Neighbor (K-NN), Multi-Layer Perceptron (MLP), and Random Forest (RF). The comparison is presented in Table 1. It shows that even lightweight classifiers achieve near-perfect accuracy on network-traffic features, motivating the question of whether comparable performance is achievable with HPC side-channel features alone.

Another work on Heartbleed attack detection is (Hossain et al., 2020), which used the CICIDS2017 dataset. This dataset comprises various attacks, including cross-site scripting, SQL injection, denial-of-service, and Heartbleed. In the experiment, the dataset was split into 80% for training and 20% for testing, which is considered a good starting point. The experiment used a Long Short-Term Memory (LSTM)

Table 2The hardware performance events considered in [Torres and Liu \(2016\)](#).

Event Name	Description
RET	Near return instructions retired
MISP_BR	Mispredicted branch instructions
LOAD	Load instructions retired
MISP_BR_C	Mispredicted conditional branches
STORE	Store instructions retired
MISS_ITLB	I-TLB misses
STLB_HIT	Shared TLB hits after i-TLB misses
MISS_DTLB	DTLB-misses
CALL_ID	Indirect near call instructions retired
MISS_ICACHE	I-Cache misses
CALL_D	Direct near call instructions retired
MISS_LLC	Last Level Cache misses

network for multiclass classification. To achieve the best classification results, the authors focused on fine-tuning various LSTM hyperparameters, including optimizers, loss functions, learning rates, and activation functions. The results for the Heartbleed attack were impressive: Precision, F1 score, and Recall were equal to 1.00. However, the chosen dataset lacks extensive evaluation data for the Heartbleed attack, as it contains only 11 records related to the attack among 692,703 records.

The next research ([Torres and Liu, 2016](#)) is closest to our solution because the authors also used HPCs to detect the Heartbleed attack. They evaluated four classification methods in the detection module: single-event/single-threshold, multiple-events/multiple-thresholds, multi-class SVM, and one-class SVM. The first method classifies individual samples as good or bad based on a threshold corresponding to each hardware event. The second method is similar to the first, except that the final decision is based on the total number of events that suggest the same decision. The third method trains and tests an SVM model on a dataset containing both good and bad samples. The fourth method is the same as the third, except that the training set consists of only good samples. The experiment setup consists of the server application, which is included within the OpenSSL toolkit. The OpenSSL library was modified to use the `perf_events` calls to obtain HPCs at runtime. The HPCs considered in the study are shown in [Table 2](#). The performance of the first and second methods was assessed using the Receiver Operating Characteristic (ROC) and Area Under the Curve (AUC) for individual hardware events, with leaked gaps ranging from 1KB to 64KB. The AUC value for the RET, LOAD, STORE, and MISS_LLC events exceeded 0.98 at a 4KB gap. The third method (based on the two-class SVM model) achieved a detection accuracy of 92.8% and a false-negative rate of 0.99%. However, the fourth method achieved only 70.88% detection accuracy in the same scenario.

In [Berebaru and Giannuzzi \(2024\)](#), the authors used the Autoencoder neural network model to detect variants of the Heartbleed attack. For that purpose, the CICIDS2017 dataset and a custom one were used. The custom testbed created for the testing model consists of 3 Ubuntu client VMs and 1 Ubuntu server VM. The server ran a Docker container with a vulnerable OpenSSL library and an Apache server. The client VMs generated benign traffic by running a script that gets the page from the Apache server. The Heartbleed traffic was generated by using a Kali Linux machine and Metasploit. Feature selection (starting with 84 features from the CICIDS2017 dataset) was performed using a Random Forest classifier. Finally, 20 features were selected, including flow duration in microseconds, maximum backward packet size, and standard backward packet size, among others. The performance of the proposed model was evaluated in two ways: first, using the CICIDS2017 dataset alone, and second, combining it with custom Heartbleed datasets. We focus on the latter, as it has more attack samples and is therefore more reliable. The training phase included only benign traffic from the CICIDS2017 dataset. Then, it was tested with the custom Heartbleed dataset. The model achieved 88% of balanced accuracy.

Despite similarities to the related works described above, none of the prior works combine all of the following properties: (a) HPC-based

detection without modifying the monitored server; (b) a DNN classifier replacing threshold/SVM detection; and (c) a systematic multi-architecture evaluation across various deployment configurations differing in complexity/difficulty level. The summary of the related works is presented in [Table 3](#). This design choice is empirically validated against classical machine learning baselines in [Section 6.7](#).

4. Design of a non-intrusive detection framework

This section briefly describes the adversary model ([Section 4.1](#)) related to the given problem and presents how each contribution stated in [Section 1](#) is realized in practice: non-intrusive HPC collection without server modification ([Sections 4.2–4.4.1](#)), correlation-based dataset labeling ([Section 4.4.2](#)), and the systematic DNN training and evaluation framework ([Section 4.4.3](#)).

4.1. Adversary model

To exploit the Heartbleed vulnerability, no special requirements are needed. The only prerequisite for a successful attack is the ability to communicate with the vulnerable server. From the adversary's point of view, only minor request modifications are needed (as shown in [Section 2.4](#)), which is extremely easy from the programming perspective. The adversary can also send multiple requests to retrieve chunks of memory from the server. The leaked memory may contain sensitive data such as passwords, private keys, and personal information. The attack surface comprises numerous web, mail, database, and XMPP servers, among others ([Durumeric et al., 2014](#)). The adversary model assumes the attacker is unaware of HPC monitoring. An adaptive adversary aware of the detection mechanism could attempt to reduce the HPC footprint by rate-limiting Heartbleed requests or interleaving them with benign traffic. This scenario is partially represented by the Curl-obfuscated configurations (see [Section 5.1](#), configurations 1 and 4). Full adversarial robustness analysis, including deliberate HPC pattern manipulation, is left for future work.

4.2. Chosen hardware performance metrics

To detect Heartbleed attacks, we propose using multiple DNN architectures trained on specific datasets (see [Section 5.1](#)). The training dataset comprises HPCs collected using our custom implementation, which is described in detail later in this section. We used two sets of metrics, referred to as MS1 and MS2.

The MS1 metrics set was chosen to be versatile and to capture a broad range of activities (i.e., HPCs not focused on a single aspect of a running program, such as cache-related). The rationale behind MS1 was to provide a holistic view of system execution, allowing the deep learning model to learn relevant attack patterns. The MS2 metrics set was chosen to focus specifically on cache and memory access behavior and anomalies. Because the Heartbleed attack relies on unusual memory access behavior and previous research ([Torres and Liu, 2016](#)) utilized cache-related performance counters, we decided that composing the MS2 metrics set to be purely cache-related would be a reasonable choice for detecting the Heartbleed attack. [Table 4](#) presents which HPCs constitute the MS1 and MS2 metrics set.

4.3. Architecture

The high-level architecture of the proposed solution is illustrated in [Fig. 4\(a\)](#). There are three separate modules, namely the Collector ([Fig. 4\(b\)](#)), the Transformer ([Fig. 4\(c\)](#)), and the DNN-Factor ([Fig. 4\(d\)](#)). Each module was used in both the training and the evaluation phases.

The Collector executes an attack scenario in a controlled environment and collects logs from both the attacker and the victim, following a standard client-server approach. The client can send both benign and

Table 3
The summary of the related works.

Work	Data source	Detection model	Server modification required?
Amodei et al. (2023)	Network traffic (CICFlowMeter)	Rule-based + ML classifiers	No
Hossain et al. (2020)	Network traffic (CICFlowMeter)	LSTM	No
Torres and Liu (2016)	HPC	Threshold/SVM	Yes
Berbecaru and Giannuzzi (2024)	Network traffic (CICFlowMeter)	Autoencoder	No
This work	HPC	DNN	No

Table 4
The MS1 and MS2 metrics set.

Metric name	Description	MS1	MS2
cycles	Total cycles	X	
instructions	Retired instructions by the CPU	X	
branches	Retired branch instructions	X	
branch-misses	Mispredicted branch instructions	X	
cache-misses	Last level cache misses	X	X
cache-references	Cache accesses made by the CPU	X	X
L1-dcache-load-misses	Data cache load misses in the L1 data cache	X	X
dTLB-load-misses	Data Translation Lookaside Buffer load misses	X	
context-switches	Context switches	X	
major-faults	Major page faults	X	
minor-faults	Minor page faults	X	
cpu-clock	Reports the CPU clock, a high-resolution per-CPU timer	X	
task-clock	Clock count specific to the running task.	X	
L1-dcache-loads	Measures Level 1 Data Cache loads		X
L1-dcache-stores	L1 data cache writes		X
L1-icache-load-misses	Instruction fetches that missed the Level 1 Instruction Cache		X
LLC-loads	Load accesses to the Last Level Cache		X
LLC-load-misses	Load misses in the Last Level Cache		X

malicious traffic. The server starts under the control of the HPCs collector (either for the MS1 or the MS2 sets of metrics); hence, counters are collected for the vulnerable server.

The Transformer converts the raw data generated by the Collector into input suitable for the DNN-Factory module, producing a dataset for training or evaluating a DNN model. First, the `data.raw` file is transformed to the `data.transformed` file. Second, the `data.transformed` file is correlated with the `heartbleed.log` file and the final `data.correlated` file is produced. The `data.correlated` is designed to be ingested directly by the DNN-Factory.

The DNN-Factory is the last module of the high-level architecture. Given the input data (datasets) from the Transformer module, the primary objective is to train various DNN models and ultimately assess them on the provided evaluation datasets.

4.4. Implementation

In this section, we discuss the implementation details of the architecture. We present the implementation of the Collector, the Transformer, and the DNN-factory.

4.4.1. Collector

There are three crucial components in the Collector's implementation details: the malicious client, the vulnerable server, and the tool used to collect HPCs. The malicious client was implemented by adding a single commit to the OpenSSL 1.0.1h fork (the idea was taken from [Ristić, 2022](#)). Alongside the OpenSSL-based malicious client, there is one more custom malicious client written in Python. Contrary to the OpenSSL-based client, the Python implementation performs the Heartbleed attack right after exchanging hello phase messages: client

hello and server hello ([IETF, 2025a](#)) (both clients write logs to the `heartbleed.log` file). On the other hand, the vulnerable server is simply OpenSSL in version 1.0.1f. For the OpenSSL client, the `openssl s_client` command was used, while for the OpenSSL server, the `openssl s_server` command was used.

To collect HPCs, we decided to use Perf, the performance analysis tools for Linux ([man7.org, 2025](#)). Perf allows obtaining HPC metrics using a simple command-line interface. The server process was launched under perf stat control, specifying the selected metric set via the `-e` flag (either the MS1 or MS2 event list), a fixed sampling interval via the `-I` flag, and directing output to the `data.raw` file via `-o` flag. The interval value, CPU model, and full event strings used in the experiments are documented in Section 5.

4.4.2. Transformer

The Transformer module consists of two Python scripts: `transform.py` and `correlate.py`. The former transforms the `data.raw` file to the `data.transformed` file, whereas the latter creates the `data.correlated` file by correlating the `heartbleed.log` with `data.transformed`. To understand the correlation process, the files `data.raw`, `data.correlated` and `heartbleed.log` are presented below (`data.transformed` is essentially the same as `data.correlated` but without the last column, `is_attack`):

```
data.raw file:
time          counts  events
10.000615373  864680  cache-references
10.000615373  435265  cache-misses
```

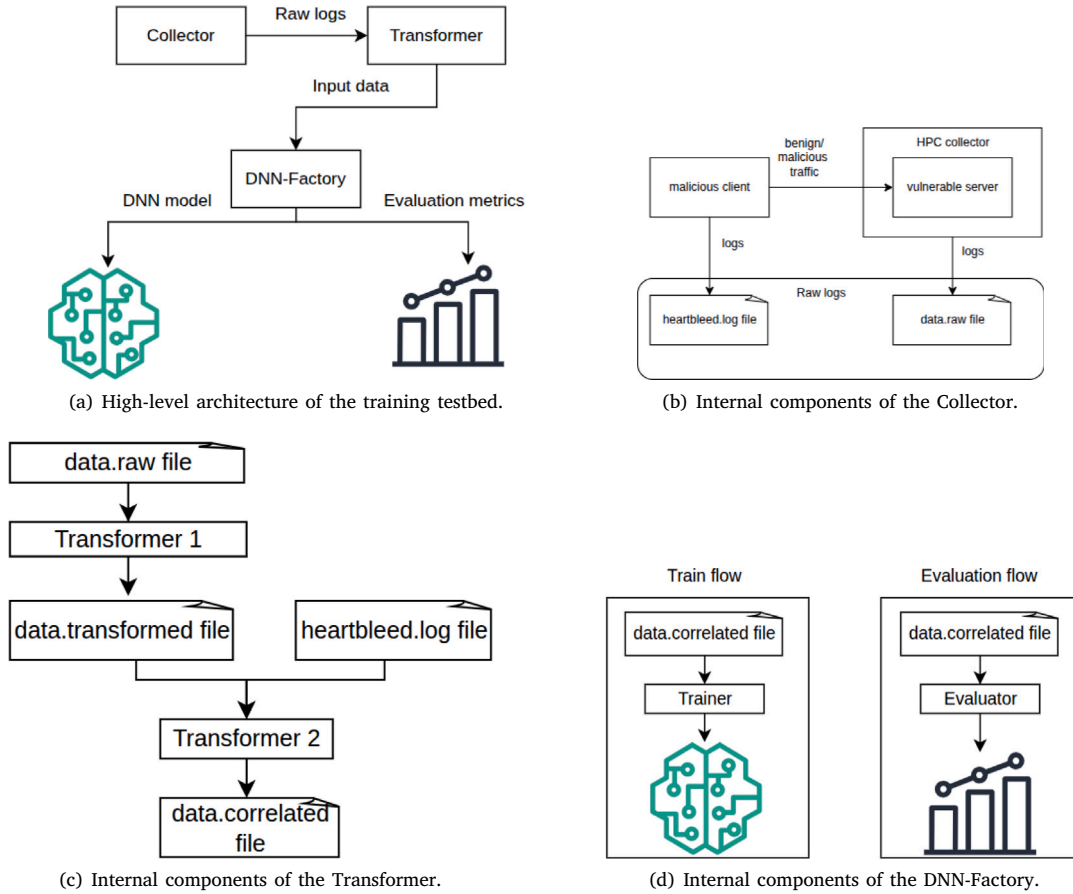


Fig. 4. The high-level architecture of the training testbed and the internals of each component.

```
data.correlated file:
time|cache-references|...|LLC-load-misses|is_attack
2025-05-22 06:53:27.000615|864680|...|16150|0
2025-05-22 06:53:37.001358|259982|...|6002|1
```

```
heartbleed.log file:
2025-05-21 14:41:01.133|sendHeartbleedOpenSSL
2025-05-21 14:41:05.332|sendRandomDataOpenSSL
2025-05-21 14:41:09.587|sendQuitOpenSSL
```

4.4.3. DNN-factory

The DNN-Factory module implements two functionalities: deep neural network training and evaluation. Implementation was done in Python v3.12.3, including libraries such as Keras v3.8.0, Pandas v2.2.3, scikit-learn v1.6.1, and TensorFlow v2.18.0. During the training phase, the selected metric (e.g., binary accuracy) is monitored, and the best checkpoint is saved when the metric improves over consecutive epochs. The training dataset was split into an 80:20 train/validation partition (see Section 5 for full dataset statistics). Training proceeded for a fixed 50 epochs with a batch size of 64; early stopping was not applied because the checkpoint mechanism ensures that the best-performing epoch is retained. During inference, the saved checkpoint is loaded and applied independently to each sample in the evaluation dataset, producing a binary `is_attack` prediction for each sample. Results are stored in a log file and subsequently aggregated, producing final metrics reported in Section 6.

5. Experimental methodology

In this section, we describe the training and evaluation datasets and the DNN architectures assessed in our experiment. The experiments were conducted on Intel Core i5-7300HQ, 16 GB RAM, running

Ubuntu 16.04 and 24.04. The perf stat sampling interval was 5s, 10 s, and 30 s during the training phase, and 10 s during the evaluation phase. The complete list of perf events for MS1 is: cycles, instructions, branches, branch-misses, cache-misses, cache-references, L1-dcache-load-misses, dTLB-load-misses, context-switches, major-faults, minor-faults, cpu-clock, task-clock, while for MS2: cache-references, cache-misses, L1-dcache-loads, L1-dcache-load-misses, L1-dcache-stores, L1-icache-load-misses, LLC-loads, LLC-load-misses (Table 4). HPC values are microarchitecture-dependent; hence, results on processors with different cache hierarchies or branch-predictor designs may vary.

5.1. Training and evaluation datasets

The training dataset setup was the same as presented in Fig. 4(b). Both clients (in the training scenario, two: one OpenSSL-based and one Python-based) and the server were configured to run on the same laptop using Ubuntu 24.04. Datasets for the MS1 and MS2 metric sets were collected separately.

However, the setup of the evaluation dataset was slightly different. First, there were two separate machines involved, one for the vulnerable server and one for the malicious client. We used two physical machines connected via an Ethernet cable to simulate a real-world setup in which the client and server are typically not on the same machine. Second, we collected evaluation datasets for six configurations and assessed the DNN models. The six configurations are based on two factors: (a) server OS version (Ubuntu 16 vs. Ubuntu 24), testing robustness across OS environments; and (b) traffic complexity (Nginx server combined with Curl obfuscation, Nginx only, or OpenSSL `s_server` only), spanning from near-production conditions (Configuration 1 and 4) to a controlled baseline identical to the training setup (Configuration

6). This design isolates the contribution of each environmental variable to the difficulty of detection. The summary of each configuration is presented below:

- **Configuration 1:**
 - Client: Ubuntu 24, malicious OpenSSL and benign Curl clients
 - Server: Ubuntu 16, Nginx server
- **Configuration 2:**
 - Client: Ubuntu 24, malicious OpenSSL client
 - Server: Ubuntu 16, Nginx server
- **Configuration 3:**
 - Client: Ubuntu 24, malicious OpenSSL and Python clients
 - Server: Ubuntu 16, OpenSSL server
- **Configuration 4:**
 - Client: Ubuntu 24, malicious OpenSSL and benign Curl clients
 - Server: Ubuntu 24, Nginx server
- **Configuration 5:**
 - Client: Ubuntu 24, malicious OpenSSL client
 - Server: Ubuntu 24, Nginx server
- **Configuration 6:**
 - Client: Ubuntu 24, malicious OpenSSL and Python clients
 - Server: Ubuntu 24, OpenSSL server

Configurations 3 and 6 aim to test Heartbleed attack detection using the same server setup as the training one, but on two machines, and also for older Ubuntu distributions (for Configuration 3). The other configurations introduce Nginx as a vulnerable server. According to the W3Techs report (W3Techs, 2025), “Nginx is used by 33.9% of all the websites whose web server we know”. Given Nginx’s popularity, we decided to use it as a real-world test case to detect Heartbleed attacks. Curl clients are used to test the detection in an obscure environment: instead of the OpenSSL client sending both malicious and benign traffic, Curl clients send only benign requests to the server.

The divergence between the training environment (single machine, OpenSSL server, no obfuscating traffic) and the evaluation environment (two physical machines, Nginx server, Curl obfuscation) is a deliberate design choice to test generalization. Its impact on detection performance is analyzed in Section 6.9. Table 5 presents the training and evaluation sample counts, including the attack and benign counts and the total number for each configuration.

5.2. Tested DNN architectures and baselines

To obtain DNN evaluation candidate models, we conducted an experiment on 15 different DNN architectures across each set of metrics and evaluation configurations. The training dataset was split into an 80:20 ratio (20% for validation). During the training phase, various monitoring metrics were established to ensure that the best model checkpoint was saved. The monitored metrics with their definitions are presented in Table 6:

Different numbers of hidden layers and neurons were tested, while the number of epochs (50) and batch size (64) were fixed. Using 50 epochs and a batch size of 64 provided the model with sufficient iterations to learn meaningful patterns, offering a balance between training stability and computational efficiency. Moreover, in our experiment, different batch sizes and a larger number of epochs did not yield better results; hence, we fixed the epochs to 50 and the batch size to 64.

Table 5

Training and evaluation samples count.

Metrics set	Configuration	Attacks count	Benign count	Total
MS1	6 (training)	65036	80282	145318
MS2	6 (training)	45482	57733	103215
MS1	1 (eval)	563	2055	2618
MS1	2 (eval)	711	2236	2947
MS1	3 (eval)	709	1256	1965
MS1	4 (eval)	498	1536	2034
MS1	5 (eval)	460	1503	1963
MS1	6 (eval)	776	1388	2164
MS2	1 (eval)	631	2116	2747
MS2	2 (eval)	443	1488	1931
MS2	3 (eval)	799	1328	2127
MS2	4 (eval)	708	2272	2980
MS2	5 (eval)	650	2276	2926
MS2	6 (eval)	686	1240	1926

Table 6

Monitored metrics used to save the best model checkpoint in the training phase.

Metric name	Definition
Binary accuracy	The number of correct predictions divided by the total number of predictions (Terven et al., 2025)
Loss	Loss functions measure the error between predicted and actual outputs.
Precision	Represents the ratio of true positives to all predicted positives (Powers, 2020)
Recall	Measures the proportion of true positive values out of all actual positives (Powers, 2020)
AUC-ROC	Area under the receiver operating characteristic curve (Terven et al., 2025). The Receiver Operating Characteristic (ROC) curve illustrates the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) across a range of classification thresholds.
F1-Score	Statistical metric that combines Precision and Recall into a single value, providing an overall assessment of a classification model’s performance (Powers, 2020)

The following listing presents hidden layer arrangements we assessed (subsequent values represent the number of neurons in hidden layers, e.g., 128, 64, 32 means 3 hidden layers with 128 neurons in the first layer, 64 in the second, and 32 in the third):

- Three layers: 128, 64, 32; 64, 32, 16; 32, 16, 8; 16, 8, 4; 8, 4, 2
- Two layers: 128, 64; 64, 32; 32, 16; 16, 8; 8, 4
- One layer: 128; 64; 32; 16; 8

To validate the selection of the DNN architecture, three classical machine learning baselines were evaluated using the same training and evaluation datasets, the same 80:20 train/validation split, and the same six evaluation configurations: Logistic Regression, Random Forest, and XGBoost, all using default hyperparameters to establish a representative baseline.

5.3. Evaluation metrics

We decided not to use binary accuracy as an evaluation metric because the evaluation datasets were clearly imbalanced (as shown in Table 5). For example, the second row of the table yields 75.87% binary accuracy by classifying all samples as benign. While binary accuracy is misleading under class imbalance, Recall and Specificity remain interpretable as they are computed per class independently of overall class proportions. However, both metrics are sensitive to the classification threshold; the default threshold of 0.5 was used for consistency across all 15 architectures and 6 configurations. Threshold tuning to optimize the operational Recall–Specificity trade-off for a specific deployment is a noteworthy direction for future work.

To choose the best candidates for further investigation, we propose the six criteria. Criteria 1–3 impose a minimum constraint of 0.5 on both Recall and Specificity before optimizing, ensuring that neither metric collapses to a trivial classifier. Criteria 4–6 relax this constraint, revealing the unconstrained performance ceiling (max Recall, max Specificity) and the global sum optimum. Together, the six criteria provide a full characterization of the Recall–Specificity trade-off space for each configuration. The summary of each criterion is as follows:

• **Criterion 1:**

$$(Recall > 0.5 \wedge Specificity > 0.5) \wedge (max(Recall + Specificity))$$

• **Criterion 2:**

$$(Recall > 0.5 \wedge Specificity > 0.5) \wedge (max(Recall))$$

• **Criterion 3:**

$$(Recall > 0.5 \wedge Specificity > 0.5) \wedge (max(Specificity))$$

• **Criterion 4:**

$$max(Recall + Specificity)$$

• **Criterion 5:**

$$max(Recall)$$

• **Criterion 6:**

$$max(Specificity)$$

For comparability with the DNN results, the same Criterion 4 was used to select the reported result for each baseline model.

6. Evaluation results and discussion

Tables 7–12 present the results obtained according to the above criteria. The first column refers to the configurations listed in Section 5.1. Then, Recall, Specificity, and their sum values are presented. The fifth column represents the DNN architecture, the sixth column lists the metrics, and the last column shows the monitoring metric used during training. For example, the first row of Table 7 can be interpreted as follows:

- For the Configuration 1 (i.e., evaluation data collected using Ubuntu 24 running a malicious OpenSSL and benign Curl clients, and Ubuntu 16 running a vulnerable Nginx server, see Section 5.1),
- Considering the

$$Recall > 0.5 \wedge Specificity > 0.5$$

and

$$max(Recall + Specificity)$$

as the selection criterion for the best evaluation model,

- The best model achieved $Recall = 0.652$ and $Specificity = 0.659$,
- The best model architecture had 3 hidden layers with 8 neurons in the first layer, 4 in the second, and 2 in the third,
- The best model was trained on the metrics set MS1,
- The metric used to save the best checkpoint during training was F1-Score.

Below, we describe the best outcomes for each criterion.

6.1. Criterion 1: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Recall + Specificity)$

Table 7 presents the best architectures for each configuration considering the criterion: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Recall + Specificity)$. It can be seen that for each configuration, the best architecture was obtained using the MS1 metrics set. If configurations

Table 7

The best architectures for each configuration considering Criterion 1: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Recall + Specificity)$.

Conf.	Rec.	Spec.	Sum	Arch.	Metrics	Mon.
1	0.652	0.659	1.311	8,4,2	MS1	F1-Score
2	0.847	0.587	1.434	8,4,2	MS1	Loss
3	0.746	0.807	1.553	64	MS1	Loss
4	0.590	0.698	1.288	16	MS1	Loss
5	0.863	0.597	1.460	8,4,2	MS1	Loss
6	0.753	0.850	1.603	8,4	MS1	Binary accuracy

1–3 and 4–6 are considered separately, it can be observed that the Sum column shows increasing values (1.311, 1.434, 1.553 for 1–3; and 1.288, 1.460, 1.603). In general, the greater the Sum column, the better because the column sums the Recall and Specificity values. It makes sense because if we look closer at configurations 1–3 and 4–6, we can observe the following patterns:

- Configurations 1–3 and 4–6 are exactly the same, except for the Ubuntu distribution in which the vulnerable server runs.
- In terms of detection difficulty, we can distinguish:
 - Configurations 1 and 4 are the hardest due to Curl clients (which obscure network traffic) and Nginx as the server. They most closely mimic the production environment.
 - Configurations 2 and 5 are easier than Configurations 1 and 4 because they do not involve Curl clients.
 - Configurations 3 and 6 are the easiest because they do not involve Curl clients and run on the OpenSSL s_server utility instead of Nginx. Configuration 6 is identical to the one used to obtain the training dataset. Configuration 3 is identical to Configuration 6, except that it runs on Ubuntu 16.

The Recall and Specificity values for configurations 1 (0.652, 0.659) and 4 (0.590, 0.698) indicate that, even in the obscured environment in which the evaluation data were collected using a vulnerable Nginx server, the proposed solution detected some hidden patterns in the collected HPC metrics. Furthermore, the DNN architectures obtained are relatively small, given the network's overall number of neurons. All architectures have at most 16 neurons, except the architecture for Configuration 3 (64 neurons in the hidden layers). It is worth noting that the highest number of neurons we tested was 224 (3 hidden layers: 128, 64, 32). For this reason, it can be inferred that a complex, dense architecture is unnecessary for this type of detection problem. One additional point to note is the last column, where most of the best models were obtained for loss function monitoring during the training phase (except for configurations 1 and 6).

6.2. Criterion 2: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Recall)$

Table 8 presents the best architectures for each configuration considering the criterion: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Recall)$. The criterion provides both values (Recall and Specificity) to be greater than 0.5 and maximizes the value of Recall. The second and fifth rows are the same as in Table 7, which means that for configurations 2 and 5, no better Recall (with Specificity greater than 0.5) was present. However, other configurations (1, 3, 4, 6) had new representatives that achieved Recall values greater than in Table 7. With higher Recall values, there is a clear trade-off: a decrease in Specificity. In addition, the best models for configurations 3 and 6 were trained on the MS2 set of metrics. The number of neurons in hidden layers is still relatively small (as in Table 7). Similarly to Table 7, the Loss function prevails.

Table 8

The best architectures for each configuration, considering Criterion 2: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Recall)$.

Conf.	Rec.	Spec.	Sum	Arch.	Metrics	Mon.
1	0.689	0.545	1.235	32	MS1	Loss
2	0.847	0.587	1.434	8,4,2	MS1	Loss
3	0.809	0.551	1.360	8	MS2	F1-Score
4	0.675	0.514	1.189	8,4	MS1	Binary accuracy
5	0.863	0.597	1.460	8,4,2	MS1	Loss
6	0.825	0.509	1.334	8	MS2	Loss

Table 9

The best architectures for each configuration considering Criterion 3: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Specificity)$.

Conf.	Rec.	Spec.	Sum	Arch.	Metrics	Mon.
1	0.517	0.765	1.282	8,4,2	MS1	Recall
2	0.523	0.807	1.330	16,8,4	MS1	Recall
3	0.628	0.900	1.527	128,64,32	MS1	Loss
4	0.538	0.737	1.275	8	MS1	Recall
5	0.515	0.804	1.319	16,8,4	MS1	Recall
6	0.539	0.876	1.415	32,16,8	MS1	Loss

Table 10

The best architectures for each configuration, considering Criterion 4: $max(Recall + Specificity)$.

Conf.	Rec.	Spec.	Sum	Arch.	Metrics	Mon.
1	0.652	0.659	1.311	8,4,2	MS1	F1-score
2	0.847	0.587	1.434	8,4,2	MS1	Loss
3	0.746	0.807	1.553	64	MS1	Loss
4	0.590	0.698	1.288	16	MS1	Loss
5	0.863	0.597	1.460	8,4,2	MS1	Loss
6	0.753	0.850	1.603	8,4	MS1	Binary accuracy

6.3. Criterion 3: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Specificity)$

Table 9 is analogous to Table 8 and presents the best architectures for each configuration considering the criterion: $Recall > 0.5 \wedge Specificity > 0.5$ and $max(Specificity)$. Specificity values are much higher than Recall values, as expected given the criterion. The highest Specificity was 0.9 for Configuration 3. Interestingly, maximizing Specificity increased the number of neurons in the hidden layers. Moreover, the Recall monitoring metric (last column) was the most frequent for the criterion.

6.4. Criterion 4: $max(Recall + Specificity)$

Table 10 presents the best architectures for each configuration considering the criterion: $max(Recall + Specificity)$. The values are the same as in Table 7, which means that there was no single model that would have a greater sum of Recall and Specificity with Recall or Specificity less than half (e.g., Recall = 0.99, Specificity = 0.49). The number of neurons in the hidden layers was relatively small across most configurations.

6.5. Criterion 5: $max(Recall)$

Table 11 presents the best architectures for each configuration considering the criterion: $max(Recall)$. In contrast to the previous table, the results are not the same as in Table 8. However, the values for Configuration 5 are the same as in Tables 7 and 8. The Recall column values are very high for the criterion (0.922, 0.926, 0.964, 0.869, 0.863, 0.961), but the Specificity decreased dramatically (0.201, 0.317, 0.189, 0.235, 0.597, 0.181). Interestingly, all except Configuration 5 used the MS2 set of metrics. The number of neurons in the hidden layers was also higher (16, 8, 4 for all configurations except 5) than for the previous criteria.

Table 11

The best architectures for each configuration, considering Criterion 5: $max(Recall)$.

Conf.	Rec.	Spec.	Sum	Arch.	Metrics	Mon.
1	0.922	0.201	1.123	16,8,4	MS2	Loss
2	0.926	0.317	1.242	16,8,4	MS2	Loss
3	0.964	0.189	1.153	16,8,4	MS2	Loss
4	0.869	0.235	1.104	16,8,4	MS2	Loss
5	0.863	0.597	1.460	8,4,2	MS1	Loss
6	0.961	0.181	1.141	16,8,4	MS2	Loss

Table 12

The best architectures for each configuration, considering Criterion 6: $max(Specificity)$.

Conf.	Rec.	Spec.	Sum	Arch.	Metrics	Mon.
1	0.314	0.901	1.215	8,4,2	MS1	Precision
2	0.252	0.903	1.155	8,4,2	MS1	Precision
3	0.628	0.900	1.527	128,64,32	MS1	Loss
4	0.247	0.892	1.139	8,4,2	MS1	Precision
5	0.139	0.917	1.056	8,4	MS1	Precision
6	0.539	0.876	1.415	32,16,8	MS1	Loss

6.6. Criterion 6: $max(Specificity)$

Table 12 presents the best architectures for each configuration considering the criterion: $max(Specificity)$. This time, the best models for configurations 3 and 6 are the same as in Table 9. Similarly to Table 11, the Specificity values are very high (0.901, 0.903, 0.900, 0.892, 0.917, 0.876), whereas the Recall values decreased dramatically (0.314, 0.252, 0.628, 0.247, 0.139, 0.539). The last column presents Precision as the main monitoring metric during the training phase. The number of neurons in the hidden layers was relatively small across most configurations.

6.7. Baseline comparison

To validate the choice of DNN architecture, Table 13 and Fig. 5 compare the best DNN result according to Criterion 4 against three classical machine learning baselines across all six evaluation configurations. All baseline values in Table 13 (except XGB in configuration 5) come from MS1 (MS1 outperforms MS2 in baselines with regard to Criterion 4). The DNN achieves the highest $Recall + Specificity$ sum in every configuration, confirming that the additional complexity of the proposed DNN architecture search is justified relative to simpler alternatives.

6.8. Feature importance analysis

To assess whether the MS1 metric set could be reduced without substantial detection loss, we conducted a feature importance analysis using the XGBoost classifier (similar to Devan and Khare, 2020). This analysis was conducted exclusively for MS1, as it consistently outperformed MS2 in the evaluated criteria, making it a more relevant set of metrics to refine for practical deployment. Moreover, the analysis was confined to Configuration 6, as it is the only evaluation configuration that exactly replicates the training environment, ensuring that the resulting ranking reflects a genuine Heartbleed-related signal rather than additional environment noise present in the other configurations.

The classifier ranked the MS1 features as follows, from most to least important: context-switches, branches, cycles, cpu-clock, task-clock, instructions, branch-misses, dTLB-load-misses, cache-references, cache-misses, L1-dcache-load-misses, minor-faults, major-faults. Next, we retrained the best architecture identified for Configuration 6 (Table 7: two hidden layers with 8 and 4 neurons, MS1 metric set, binary accuracy monitored) on progressively reduced subset of features, removing the least important features one at a time. Performance remained close

Table 13
Comparison of the best DNN architecture with ML baselines.

Conf.	DNN			LogReg			RF			XGB		
	Rec	Spec	Sum	Rec	Spec	Sum	Rec	Spec	Sum	Rec	Spec	Sum
1	0.652	0.659	1.311	0.659	0.650	1.309	0.616	0.571	1.187	0.265	0.839	1.104
2	0.847	0.587	1.434	0.727	0.555	1.282	0.613	0.553	1.166	0.247	0.851	1.098
3	0.746	0.807	1.553	0.783	0.696	1.479	0.790	0.739	1.529	0.740	0.749	1.489
4	0.590	0.698	1.288	0.616	0.660	1.276	0.508	0.626	1.134	0.217	0.806	1.023
5	0.863	0.597	1.460	0.885	0.537	1.422	0.287	0.788	1.075	0.596	0.467	1.063
6	0.753	0.850	1.603	0.800	0.676	1.476	0.789	0.665	1.454	0.729	0.805	1.534

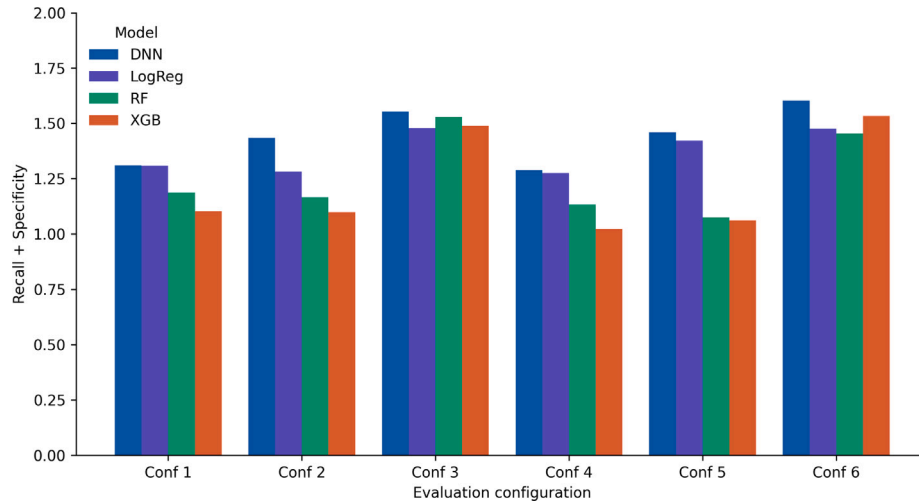


Fig. 5. Comparison of *Recall + Specificity* values for the best DNN architecture and the best classical ML baselines.

to the full-feature baseline (Recall = 0.753, Specificity = 0.850) until only a single feature (context-switches) remained, at which point Recall and Specificity dropped substantially to 0.406 and 0.628, respectively. With the two most important features retained (context-switches and branches), Recall and Specificity were 0.738 and 0.857, suggesting MS1 could be reduced from 13 to 2 counters without substantial performance loss.

6.9. Discussion and limitations

While this feature reduction demonstrates that the underlying signal can be compressed to two core counters for known execution states, the full-feature DNN remains critical for practical deployment. In unknown or evolving server environments, the specific counters that capture exploit signatures may shift; the DNN's value lies in its structural capacity to dynamically adapt and discover latent correlations across the entire counter spectrum, without requiring repeated manual feature-engineering re-runs by human experts.

The results presented must be carefully considered: a Recall range of 0.590–0.753 implies that between 25% and 41% of Heartbleed attacks would go undetected in the current form, which is too low to be considered as a standalone deployment as a production IDS. The proposed solution is best characterized as a proof-of-concept demonstrating the feasibility of HPC-based, non-intrusive TLS attack detection, with the results identifying the key challenges to be solved before practical deployment.

As can be observed from the presented results, the MS1 set of metrics prevailed over MS2, except Table 11 - 5 of 6 rows in Table 11 refer to MS2. It can be concluded that MS1 is more suitable for the given detection problem. The supremacy of the MS1 set can be attributed to the training sample count (see Table 5). The total number of training samples for the MS1 set was 145,318, and for the MS2 set, 103,215. The dominance of MS1 over MS2 also provides evidence that a holistic approach outperforms cache-only counters for Heartbleed detection.

This may happen because Heartbleed's primary observable signature involves not only cache behavior but also other patterns, such as cycle or instruction counts triggered by the buffer over-read operation.

Furthermore, it can be observed that monitoring Loss was the most frequent value in the last column, making it the best choice candidate for the given attack detection problem. In terms of architecture, the results indicate that a simple architecture involving fewer neurons is sufficient. Most of the time, fewer than or equal to 16 neurons were sufficient to balance the Recall and Specificity values, i.e., to maximize their sum and have their values as close as possible to each other.

The results obtained, although promising for some configurations, are still too poor to be considered production-ready. We believe that several improvements could increase the overall detection rate, and we discuss these here.

First, the training environment could be configured to more closely mimic the production environment. The training environment that we used diverged from the real-world client-server setup for TLS communication. As mentioned in Section 5.1, the training environment ran on one machine (both client and server), without obscuring network traffic, and the vulnerable server was based solely on OpenSSL (no Nginx). The proposed training environment undoubtedly has advantages in training dataset collection time (it is faster) and setup difficulty (it is much easier to set up the client and server on the same machine). However, the evaluation of the models in a more sophisticated environment (Configuration 1, Recall = 0.652, Specificity = 0.659, and Configuration 4, Recall = 0.590, Specificity = 0.698) turned out to be significantly worse than the results obtained from model evaluation in the environment used to collect the training dataset (Configuration 6, Recall = 0.753, Specificity = 0.850). The performance degradation observed in Nginx-based configurations likely reflects a fundamental mismatch between the HPC profile and the configuration. Nginx operates a multi-process worker pool model, whereas the training environment used OpenSSL simple `s_server` utility. Even when processing identical TLS traffic, the different execution contexts and process architectures produce distinct HPC distributions. This training-distribution

mismatch is a known challenge for side-channel-based classifiers and represents a concrete target for environment-aware training in future work.

Second, the number of training samples could be higher (for example, the authors of [Mushtaq et al. \(2018\)](#) used 1 million training samples). A larger number of training samples could improve the pattern recognition ability of the proposed models. Notably, the DNN already outperforms all baselines at the current dataset size; a larger dataset may further widen this gap, particularly relative to Logistic Regression, whose linear decision boundary has limited capacity to improve with additional data. On the other hand, it could also introduce issues such as increased time and computational costs. The size of the training dataset does not guarantee better model performance. The data must also be versatile and unbiased; hence, the training environment and scenarios would also be reconsidered.

Third, the proposed HPCs used in the MS1 and MS2 sets could not be optimal. Modern CPUs expose many performance counters; e.g., the Intel Lion Cove Performance Core exposes 309 of them ([Intel, 2026](#)). Moreover, the Linux kernel exposes software-based performance events even if no hardware support is available ([man7.org, 2026](#)). A detailed investigation of them would be necessary to choose the optimal subset for the Heartbleed attack detection problem.

Fourth, the HPC time series carries an implicit temporal structure (samples collected at fixed intervals), which feedforward DNNs do not explicitly model. Other architectures, such as LSTMs or 1D-CNNs, may better capture sequential attack signatures, for example, the temporal evolution of cache-related operations during a Heartbleed exploit. The present study uses feedforward DNNs; exploring sequential models is a concrete direction for future work.

Fifth, the reported results are from a single test run per model setup. Different model architectures produced noticeably different results, which means choosing the right architecture is important and not straightforward. The test datasets were also quite small, because they were designed to resemble the amount of data seen in a typical real user session rather than a large artificially generated dataset. Because the experiments were run only once and with limited data, the results are less statistically reliable than they would be with larger, repeated studies. Future work should improve confidence by running experiments multiple times (e.g., using cross-validation) and by collecting more data and using larger datasets.

Finally, we acknowledge that the dominance of “context-switches” in our feature importance analysis is likely a byproduct of the local loopback scheduling in our training environment. We caution that this feature may be less indicative of Heartbleed in distributed, network-separated environments, and advise future research to employ diverse cross-machine training data to mitigate such environmental overfitting.

The last part of the discussion focuses on the utility of the proposed solution. The Heartbleed vulnerability is exceptionally difficult to detect via HPC due to the overhead of surrounding TLS protocol operations. Despite this, the framework offers practical deployment scenarios in legacy systems where vulnerable TLS implementations remain in use, and network packet inspection is infeasible (e.g., end-to-end encrypted environments). The relatively low false-positive rate observed in the best configurations makes the system suitable as a supplementary alert mechanism within a layered security architecture, where it would not overwhelm security analysts with false alarms. Integration of the proposed solution requires only that the server process be launched under perf stat control — no kernel modifications, application code changes, or network sensors are needed. The perf tool is a standard Linux utility ([man7.org, 2025](#)) with well-characterized overhead and is generally considered acceptable for production monitoring purposes. Moreover, the proposed solution is intentionally attack-specific. Effective HPC-based detection requires targeted counter selection and a dedicated training dataset tuned to the expected HPC patterns of the target vulnerability. We believe the proposed framework can serve as a reusable template for building

attack-specific HPC detectors. The methodology presented here (non-intrusive collection, correlation-based labeling, and systematic DNN architecture search) is transferable to other vulnerabilities, even if the specific counters and trained models are not. Adapting the framework to detect different TLS- or memory-related attacks would require repeating the counter selection and dataset collection phases, with the expectation of discovering new attack patterns.

6.10. Comparison with previous work

The best candidate to compare our results with is ([Torres and Liu, 2016](#)), described in Section 3, because the authors used a custom testbed and trained a machine learning model (SVM) on 12 hardware events. However, they presented detection accuracy, whereas we presented Recall and Specificity. Moreover, we do not know the exact numbers of good and bad requests for their detection accuracy. Their results yield 92.8% for the 2-class SVM and 70.88% for the 1-class SVM. To make the comparison reasonable, we use results for Configuration 6, ensuring that the training and test samples share the same sources (as in [Torres and Liu, 2016](#)). We achieved the best accuracy of 82.52% (not mentioned earlier in the article, as our research is based on Recall and Specificity).

A second relevant comparison is with [Berbecaru and Giannuzzi \(2024\)](#), which applied an autoencoder to Heartbleed detection using a custom testbed and achieved 88% balanced accuracy. Our best result for Configuration 6 — the setup most comparable to [Berbecaru and Giannuzzi \(2024\)](#) in terms of matching training and evaluation environments — yields 82.52% accuracy. This suggests that the feedforward DNN approach does not outperform the autoencoder results reported in prior work, further motivating the exploration of alternative architectures and improved training environments discussed above. We note that direct comparison is limited by differences in testbed design, traffic generation, and overall evaluation methodology.

7. Conclusion

Detecting attacks using HPCs in complex cryptographic protocols such as TLS is undoubtedly challenging. The cryptographic protocols consist of various building blocks, including hash functions, public key cryptography, symmetric encryption, and message authentication codes, which obscure potential side-channel characteristics that could be used to detect the attack.

In this paper, we presented a non-intrusive detection framework that combines HPCs with a DNN to detect the Heartbleed attack. The experiment demonstrated that HPCs revealed hidden patterns that a carefully selected DNN model could discover. The proposed solution achieved (depending on the evaluation environment) a wide range of detection rates, from Recall = 0.590 and Specificity = 0.698 up to Recall = 0.753 and Specificity = 0.850.

While baseline models like Logistic Regression show competitive performance in stable environments, the DNN framework demonstrates higher adaptability to non-linear shifts in HPC patterns. The DNN provides a unified pipeline that does not require expert manual selection of counters (e.g., choosing between 2 and 13 features), which is a significant practical advantage for security administrators deploying it across heterogeneous hardware setups.

Several directions for future work emerge from the present study. First, improving generalization by configuring the training environment to better mirror real-world deployment conditions. On the detection side, tuning DNN model parameters is a reasonable approach to optimize Recall-Specificity results. Next, extending the experimental setup by introducing larger, more diverse datasets would strengthen the statistical confidence of the results. Finally, given the temporal nature of HPCs, investigating different model architectures, such as LSTMs or 1D-CNNs, could dramatically improve the overall detection rate.

Moreover, the plan for future work is to focus on exploiting side-channels in Post-Quantum Cryptography (PQC) algorithms used in the TLS protocol. Given the latest enhancements and the field's growing popularity, there is a need for a thorough analysis of PQC from a side-channel perspective. In addition, the National Institute of Standards and Technology (NIST) (NIST, 2026) standardized PQC algorithms that we plan to investigate in future research.

CRedit authorship contribution statement

Tomasz Głowka: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Resources, Methodology, Investigation. **Cheng-Feng Hung:** Writing – original draft, Investigation, Formal analysis. **Wojciech Mazurczyk:** Writing – review & editing, Validation, Supervision, Project administration. **Shin-Ming Cheng:** Supervision, Resources, Project administration, Conceptualization.

Declaration of competing 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.

Acknowledgments

This research was funded by the Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) program and Memorandum of 2024 Academic Cooperation Project Between 2024 National Taiwan University of Science and Technology and Warsaw University of Technology (WUT-Taiwan Tech-2024-05).

Data availability

Data will be made available on request.

References

- Al Fardan, N.J., Paterson, K.G., 2013. Lucky thirteen: Breaking the TLS and DTLS record protocols. In: Proc. 2013 IEEE Symposium on Security and Privacy. Berkeley, CA, USA, pp. 526–540. <http://dx.doi.org/10.1109/SP.2013.42>.
- Amodei, A., Capriglione, D., Cerro, G., Ferrigno, L., Miele, G., Tomasso, G., 2023. A measurement approach for inline intrusion detection of heartbleed-like attacks in IoT frameworks. IEEE Trans. Instrum. Meas. 72, 1–10. <http://dx.doi.org/10.1109/TIM.2023.3282662>.
- Bazm, M.-M., Sautereau, T., Lacoste, M., Sudholt, M., Menaud, J.-M., 2018. Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters. In: 2018 Third International Conference on Fog and Mobile Edge Computing. FMEC, pp. 7–12. <http://dx.doi.org/10.1109/FMEC.2018.8364038>.
- Berbecaru, D.G., Giannuzzi, S., 2024. On detecting anomalous TLS connections with artificial intelligence models. In: Proc. 2024 IEEE ISCC. Paris, France, pp. 1–6. <http://dx.doi.org/10.1109/ISCC61673.2024.10733669>.
- Bhattacharya, S., Mukhopadhyay, D., 2015. Who watches the watchmen?: Utilizing performance monitors for compromising keys of RSA on intel platforms. In: Güneysu, T., Handschuh, H. (Eds.), Proc. Cryptographic Hardware and Embedded Systems. CHES 2015, Berlin, Heidelberg, pp. 248–266. <https://eprint.iacr.org/2015/621>.
- BoringSSL, 2025. BoringSSL. <https://boringssl.google.com/boringssl>. (Accessed 13 July 2025).
- Carnà, S., Ferracci, S., Quaglia, F., Pellegrini, A., 2023. Fight hardware with hardware: Systemwide detection and mitigation of side-channel attacks using performance counters. Digit. Threat. 4 (1), <http://dx.doi.org/10.1145/3519601>.
- Chiappetta, M., Savas, E., Yilmaz, C., 2016. Real time detection of cache-based side-channel attacks using hardware performance counters. Appl. Soft Comput. 49, 1162–1174. <http://dx.doi.org/10.1016/j.asoc.2016.09.014>, URL <https://www.sciencedirect.com/science/article/pii/S1568494616304732>.
- Das, S., Werner, J., Antonakakis, M., Polychronakis, M., Monrose, F., 2019. SoK: The challenges, pitfalls, and perils of using hardware performance counters for security. In: Proc. 2019 IEEE Symposium on Security and Privacy. San Francisco, CA, USA, pp. 20–38. <http://dx.doi.org/10.1109/SP.2019.00021>.
- Dastres, R., Soori, M., 2021. Artificial neural network systems. Int. J. Imaging Robot. (IJIR) 21 (2), 13–25, URL <https://hal.science/hal-03349542>.
- Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S., 2013. On the feasibility of online malware detection with performance counters. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. ISCA '13, Association for Computing Machinery, New York, NY, USA, pp. 559–570. <http://dx.doi.org/10.1145/2485922.2485970>.
- Devan, P., Khare, N., 2020. An efficient XGBoost-DNN-based classification model for network intrusion detection system. Neural Comput. Appl. 32 (16), 12499–12514. <http://dx.doi.org/10.1007/s00521-020-04708-x>.
- Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., Halderman, J.A., 2014. The matter of heartbleed. In: Proc. of the 2014 Conference on Internet Measurement Conference. Vancouver, BC, Canada, pp. 475–488. <http://dx.doi.org/10.1145/2663716.2663755>.
- Foreman, J.C., 2018. A survey of cyber security countermeasures using hardware performance counters. arXiv:1807.10868.
- Heartbleed, 2025. Heartbleed vulnerability. <https://heartbleed.com/>. (Accessed 15 July 2025).
- Hossain, M.D., Ochiai, H., Fall, D., Kadobayashi, Y., 2020. LSTM-based network attack detection: Performance comparison by hyper-parameter values tuning. In: Proc. 2020 7th IEEE CSCloud. New York, NY, USA, pp. 62–69. <http://dx.doi.org/10.1109/CSCloud-EdgeCom49738.2020.00020>.
- IETF, 2025a. RFC 5246: The transport layer security (TLS) protocol version 1.2. <https://datatracker.ietf.org/doc/html/rfc5246section-7.4.1>. (Accessed 16 July 2025).
- IETF, 2025b. Transport layer security (TLS) and datagram transport layer security (DTLS) heartbeat extension. <https://datatracker.ietf.org/doc/html/rfc6520>. (Accessed 11 July 2025).
- IETF, 2025c. The transport layer security (TLS) protocol version 1.3, RFC 8446, IETF, 2018. <https://datatracker.ietf.org/doc/html/rfc8446>. (Accessed 15 July 2025).
- Intel, 2026. Intel core processors, arrow lake client - P-core events. <https://perfmon-events.intel.com/platforms/arrowlake/core-events/p-core/>. (Accessed 17 June 2026).
- Ivanov, O., Ruzhentsev, V., Oliynykov, R., 2018. Comparison of modern network attacks on TLS protocol. In: Proc. 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology. PIC S&T, Kharkiv, Ukraine, pp. 565–570. <http://dx.doi.org/10.1109/INFOCOMMST.2018.8632026>.
- Kosasih, W., Feng, Y., Chuengsatiansup, C., Yarom, Y., Zhu, Z., 2024. SoK: Can we really detect cache side-channel attacks by monitoring performance counters? In: Proceedings of the 19th ACM Asia Conference on Computer and Communications Security. In: ASIA CCS '24, Association for Computing Machinery, New York, NY, USA, pp. 172–185. <http://dx.doi.org/10.1145/3634737.3637649>.
- Li, C., Gaudiot, J.-L., 2019. Detecting malicious attacks exploiting hardware vulnerabilities using performance counters. In: Proc. 2019 IEEE COMPSAC, Vol. 1. Milwaukee, WI, USA, pp. 588–597. <http://dx.doi.org/10.1109/COMPSAC.2019.00090>.
- Li, C., Gaudiot, J.-L., 2022. Detecting spectre attacks using hardware performance counters. IEEE Trans. Comput. 71 (6), 1320–1331. <http://dx.doi.org/10.1109/TC.2021.3082471>.
- Malone, C., Zahran, M., Karri, R., 2011. Are hardware performance counters a cost effective way for integrity checking of programs. In: Proc. Sixth ACM Workshop on Scalable Trusted Computing. STC '11, Chicago, Illinois, USA, pp. 71–76. <http://dx.doi.org/10.1145/2046582.2046596>.
- man7.org, 2025. Perf - Linux manual page. <https://man7.org/linux/man-pages/man1/perf.1.html>. (Accessed 12 July 2025).
- man7.org, 2026. Perf event open(2) — Linux manual page. https://man7.org/linux/man-pages/man2/perf_event_open.2.html. (Accessed 17 June 2026).
- Mucci, P., Moore, S., Deane, C., Ho, G., 1999. PAPI: A portable interface to hardware performance counters.
- Mushtaq, M., Akram, A., Bhatti, M.K., Chaudhry, M., Lapotre, V., Gogniat, G., 2018. NIGHTS-WATCH: A cache-based side-channel intrusion detector using hardware performance counters. In: Proc. HASP '18: Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy. Los Angeles, United States, <http://dx.doi.org/10.1145/3214292.3214293>.
- NIST, 2026. NIST, post-quantum cryptography. <https://www.nist.gov/pqc>. (Accessed 12 June 2026).
- OpenSSL, 2025. OpenSSL. <https://www.openssl.org/>. (Accessed 13 July 2025).
- Park, J., Rahman, F., Vassilev, A., Forte, D., Tehranipoor, M., 2019. Leveraging side-channel information for disassembly and security. J. Emerg. Technol. Comput. Syst. 16 (1), <http://dx.doi.org/10.1145/3359621>.
- Powers, D.M.W., 2020. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. arXiv:2010.16061.
- Prada, I., Igual, F.D., Olcoz, K., 2019. Detecting time-fragmented cache attacks against AES using performance monitoring counters. arXiv:1904.11268.
- Putrevu, M.A., Putrevu, V.S.C., Shukla, S.K., 2023. Early detection of ransomware activity based on hardware performance counters. In: Proceedings of the 2023 Australasian Computer Science Week. ACSW '23, Association for Computing Machinery, New York, NY, USA, pp. 10–17. <http://dx.doi.org/10.1145/3579375.3579377>.
- Ristić, I., 2022. Bulletproof TLS and PKI. Feisty Duck.
- Sayadi, H., He, Z., Makrani, H.M., Homayoun, H., 2024. Intelligent malware detection based on hardware performance counters: A comprehensive survey. In: Proc. 2024 25th International Symposium on Quality Electronic Design. ISQED, San Francisco, CA, USA, pp. 1–10. <http://dx.doi.org/10.1109/ISQED60706.2024.10528369>.

- Singh, B., Evtyushkin, D., Elwell, J., Riley, R., Cervasato, I., 2017. On the detection of kernel-level rootkits using hardware performance counters. In: Proc. 2017 ACM on Asia Conference on Computer and Communications Security. Abu Dhabi, United Arab Emirates, pp. 483–493. <http://dx.doi.org/10.1145/3052973.3052999>.
- Siwakoti, Y.R., Rawat, D.B., 2023. Investigating security vulnerability related to exposure and TLS ecosystem in IoT devices. In: Proc. 2023 IEEE 24th IRI. Bellevue, WA, USA, pp. 7–12. <http://dx.doi.org/10.1109/IRI58017.2023.00009>.
- SSL Labs, 2025. Qualys SSL Labs—SSL pulse. <https://www.ssllabs.com/ssl-pulse/>. (Accessed 5 September 2025).
- Terven, J., Cordova-Esparza, D.-M., Romero-González, J.-A., Ramírez-Pedraza, A., Chávez-Urbiola, E.A., 2025. A comprehensive survey of loss functions and metrics in deep learning. *Artif. Intell. Rev.* 58 (7), <http://dx.doi.org/10.1007/s10462-025-11198-7>.
- Torres, G., Liu, C., 2016. Can data-only exploits be detected at runtime using hardware events? A case study of the heartbleed vulnerability. In: Proc. of the Hardware and Architectural Support for Security and Privacy 2016. Seoul, Republic of Korea, <http://dx.doi.org/10.1145/2948618.2948620>.
- Tsalis, N., Vasilellis, E., Mentzelioti, D., Apostolopoulos, T., 2019. A taxonomy of side channel attacks on critical infrastructures and relevant systems. In: Critical Infrastructure Security and Resilience: Theories, Methods, Tools and Technologies. Springer International Publishing, Cham, pp. 283–313. http://dx.doi.org/10.1007/978-3-030-00024-0_15.
- Uhsadel, L., Georges, A., Verbauwhe, I., 2008. Exploiting hardware performance counters. In: Proc. 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography. Washington, DC, USA, pp. 59–67. <http://dx.doi.org/10.1109/FDTC.2008.19>.
- W3Techs, 2025. Web server usage statistics - W3Techs. <https://w3techs.com/technologies/details/ws-nginx>. (Accessed 11 July 2025).
- Wang, J., Zhao, M., Zeng, Q., Wu, D., Liu, P., 2015. Risk assessment of buffer “heartbleed” over-read vulnerabilities. In: Proc. 2015 IEEE/IFIP International Conference on Dependable Systems and Networks. Rio de Janeiro, Brazil, pp. 555–562. <http://dx.doi.org/10.1109/DSN.2015.59>.
- WolfSSL, 2025. WolfSSL. <https://www.wolfssl.com/>. (Accessed 13 July 2025).
- Zhang, Z., Zhan, Z., Balasubramanian, D., Li, B., Volgyesi, P., Koutsoukos, X., 2020. Leveraging EM side-channel information to detect rowhammer attacks. In: Proc. 2020 IEEE Symposium on Security and Privacy. SP, pp. 729–746. <http://dx.doi.org/10.1109/SP40000.2020.00060>.
- Zhang, T., Zhang, Y., Lee, R.B., 2016. CloudRadar: A real-time side-channel attack detection system in clouds. In: Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J. (Eds.), *Research in Attacks, Intrusions, and Defenses*. Springer International Publishing, Cham, pp. 118–140.