

# L8-Sentinel: Host Intrusion Detection and Prevention using Local Language Models

Pedro Fialho<sup>1,2</sup>[0009–0005–6403–7157], Rui Claro<sup>2</sup>[0000–0003–0176–2720], João Trindade<sup>2</sup>, and Miguel L. Pardal<sup>1</sup>[0000–0003–2872–7300]

<sup>1</sup> INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

<sup>2</sup> Layer8, Lisboa, Portugal [info@layer8.pt](mailto:info@layer8.pt)  
<http://www.layer8.pt>

**Abstract.** Cybersecurity threats increasingly target digital infrastructure and organized attack groups grow in capability and visibility. Attacks such as Phishing or ClickFix manipulate users to have them run commands that compromise their machines and, likely, the local network. Most security work has been focused on network anomaly detection, but using a host-based intrusion detection allows us to leverage Large Language Model technology for more proactive defense and prevention. We present L8-Sentinel, an intrusion detection system that captures host behavior with osquery and analyzes this telemetry using locally hosted language models. We also introduce a reproducible methodology for generating training and testing datasets from libraries of attack techniques. Experiments with Qwen3-1.7B and MiniCPM5-1B show that fine-tuning compact models improves accuracy over their base counterparts, supporting the viability of the proposed localized endpoint defense.

**Keywords:** Cybersecurity · Fine-Tuning · Generative Artificial Intelligence · LLM · Intrusion Detection.

## 1 Introduction

As cybersecurity threat actors harness recent developments in Artificial Intelligence (AI) to aid in the development and exploitation of new attacks, it is increasingly necessary to also use AI to enhance cyber defense capabilities.

Recently, in the field of intrusion detection, Large Language Models (LLMs) have been mainly used for Network-based Intrusion Detection Systems (NIDS). A 2024 survey by Satilmiş et al. [10] found that applications of machine learning approaches in Host-based Intrusion Detection Systems (HIDS) focused mainly on neural network architectures that predate the transformer model [12].

While traditional Network Intrusion Detection Systems (NIDS) are used to identify perimeter breaches, the increasing ubiquity of end-to-end encryption has significantly diminished their visibility into malicious payloads [14]. Modern threat actors frequently exploit this blind spot, bypassing network defenses via encrypted command-and-control channels or by employing “living-off-the-land” techniques that utilize legitimate system tools for illegitimate purposes.

This paradigm shift reinforces the need for a transition toward endpoint telemetry. By capturing host-level data via tools like *osquery*, systems gain granular, post-decryption visibility into process execution trees, file system modifications, and memory access patterns. This host-centric approach is critical for proactive threat detection, as it allows for the identification of sophisticated, localized behaviors that leave little to no actionable footprint on the network layer.

To study the changes in endpoints caused by attacks, the tool Atomic Red Team<sup>3</sup> (ART) was used. ART is an open source library of tests designed to test organizations’ security controls. The tests are simple to execute, requiring only a powershell command. When run, they generate the same side effects an attack would, without the compromising any data.

We present L8-Sentinel, a system that uses *osquery* to collect host data and uses a LLM to analyze the data and detect intrusions. L8 refers to Layer 8, which informally refers to the human/user layer beyond the seven-layer OSI (Open Systems Interconnection) model, to highlight the focus on human-driven security risks and behaviors<sup>4</sup>. We performed a data collection on a controlled attack environment using L8-Sentinel, and tested the detection capability of the system using the collected data and two selected language models. The results show that fine-tuning the models can improve accuracy and demonstrate the need for future work on both data collection and data processing before testing for intrusions.

The contributions of this work are: an *architecture* for capturing host data and processing it with large language models for intrusion detection and, a *testbed* that performs controlled attacks, captures the resulting data, and uses it to evaluate and fine-tune the language models for intrusion detection.

The structure of the remainder of this document is as follows. Section 2 goes into background and related work. Section 3 presents the architecture of the system. Section 4 presents the experimental testing and the results. Finally, Section 5 presents our conclusions and explores future work.

## 2 Background and Related Work

In this section we introduce the concepts of Intrusion Detection Systems (IDS), Large Language Models (LLMs) and *osquery*, necessary for the presentation of the L8-Sentinel system. Related work is also presented to motivate the system.

### 2.1 Intrusion Detection Systems

IDS are cyber security systems intended to detect the presence of malicious actions on a system, based on data collected from it. IDS can be categorized in two ways: by the data they consume, and by the mechanism used to detect intrusions. NIDS use data from the network, typically collected from a network

<sup>3</sup> Atomic Red Team (ART): <https://www.atomicredteam.io/>

<sup>4</sup> L8 also references the Layer8 company providing consulting, technology, and managed services on cybersecurity. More information at: <http://www.layer8.pt>

device, such as a firewall or a router. HIDS use data from endpoint devices, such as workstations or servers, collecting the data directly from the endpoint machines.

When categorizing by the detection mechanism, Signature-based IDS (SIDS), also known as knowledge, or rules-based, use a database of signatures for known attacks and search the data for these signatures. This type of IDS is very efficient and accurate at detecting known attacks, but usually unable to detect novel threats. Anomaly-based IDS (AIDS) use statistical or machine-learning methods to detect attacks. They are capable of detecting novel attacks, but are usually less accurate and more prone to false positives. Artificial Intelligence (AI) methods fall into the latter category.

## 2.2 Large Language Models

The work of Vaswani et al. [12] improved scalability of traditional neural networks. This allowed new neural networks to be trained using a significantly larger corpus of data. From this scalability appeared the concept of Large Language Models (LLMs), trained using more data and with more parameters than was possible with previous neural network architectures. After the introduction of ChatGPT<sup>5</sup> in 2022, much more attention has been paid to researching the capabilities of these models. This increased research focus applies to the field of cyber security, where several possible applications are being studied and developed. We focus on the applications to intrusion detection.

Steverson et al. [11] used a transformer model for intrusion detection using Windows logs. They achieved an F1-Score of 99% on data collected from the CyberVAN testbed environment;

Wu et al. [13] proposed a transformer architecture with the goal of network based intrusion detection. They achieve F1-Scores of 99.17% and 98.48%, respectively, in the CICIDS2017 [7] and CIC-DDoS2019 [8] datasets;

Ghourabi [2] used LightGBM and a BERT-based model for intrusion detection in the healthcare network. They achieved accuracies of 1000% and 99.8% using the BERT-based model in two datasets;

Hassanin et al. [5] proposed a transformer-based model for intrusion detection on satellite networks. Testing on two Internet of Things (IoT) datasets achieved accuracies of 100%;

Joraviya et al. [6] used Convolutional Neural Networks (CNNs), a model architecture preceding transformers, to detect intrusions in host data. Testing in the LID-DS-2019 [3] dataset achieved accuracy and precision of 98%;

Berguiga et al. [1] proposed an approach combining a CNN and a Long Short-Term Memory (LSTM) neural network, both architectures predating the transformer model, to detect intrusions in Internet of Medical Things (IoMT) devices. Their approach used host data transmitted from the host devices into a network of fog nodes that run the neural networks and detect the intrusions,

---

<sup>5</sup> Introducing ChatGPT: <https://openai.com/index/chatgpt/>

sending results to the cloud. The models were trained and evaluated using two IoT datasets, achieving both accuracy and precision above 99.8%.

These works tell us that LLMs already have the capacity to detect intrusions. However, information about the time taken by the models to analyze the data is not available in most works.

### 2.3 Osquery

Osquery is an open-source and multi-platform operating system instrumentation framework. It allows the user to obtain information about the operating system on which the tool is installed by querying the system using SQL-like queries. The queries can be run on demand through a command line tool (*osqueryi*) or configured to be run on a schedule by a service (*osqueryd*) and send the results to one of several available logging plugins. Scheduled queries can be run on an interval ranging from 1 second to 1 week.

Queries can also be set to log differentials or snapshots. If logging differentials (the default mode) queries will only log a result if a difference is found relating to the last time the query ran. If logging snapshots, the query will always log the entire result. As an example, we could use the following configuration on *osqueryd* to query the services running in a Windows machine, removing processes from *svchost*:

```
{
  "queries": {
    "All_Process_Events": {
      "query": "SELECT pid, path, cmdline, cwd, parent, time
               FROM processes
               WHERE path
               NOT LIKE '%\\Windows\\System32\\svchost.exe';",
      "interval": 10,
      "description": "Captures general process executions,
                     excluding noisy svchost processes."
    }
  }
}
```

Some works have looked into the integration of *osquery* in IDS. Park et al. [9] used *osquery* for data collection for SIDS. Their work focused on studying the capacity of *osquery* to collect data covering each of the techniques covered in the MITRE ATT&CK framework. They provide a detailed analysis of the coverage achieved per category of technique.

Haas et al. [4] used *osquery* to extend their existing Zeek IDS. Their work focused on extending their NIDS with host data by collecting data with *osquery* and correlating it with network data.

### 3 Architecture

L8-Sentinel can be divided into two main components. The endpoint service runs the osquery daemon, performing queries at regular intervals and sending the results to a *kafka* broker. We explain how the set of queries and intervals was selected in Section 3.1. The HIDS runs on a separate machine, consuming the query results from the kafka broker, processing them into a format that will then be sent into the LLM, along with a prompt requesting a classification as an intrusion or not. The LLM then performs the classification. Figure 1 shows a high level diagram of the system.

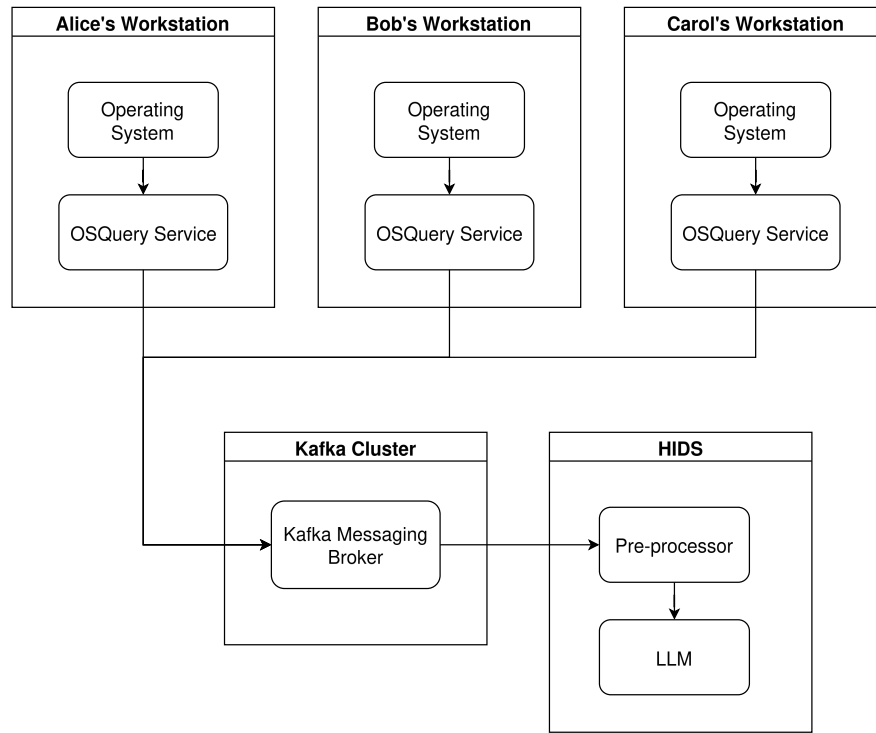


**Fig. 1.** Overview of the proposed solution. One or more workstations collect data using a osquery-based service, sending the data to kafka. The HIDS collects the data from kafka and processes it to provide de anomaly classifications.

#### 3.1 Query Selection

The queries used were selected according to two requirements. First, the queries should *cover the MITRE ATT&CK matrix as broadly as possible*. This aims

to keep the proposed system from becoming focused in some specific classes of intrusion and becoming blind to other classes. Second, the queries should *reduce redundant information to a minimum*. Since LLM inference is computationally expensive, it is important that we avoid redundant information so that the costs of running the system, as well as the time the system takes to detect an intrusion, can remain as low as possible.

The queries selected expose the following tables from osquery: Processes, socket events, registry, file events, listening ports, Windows crashes, USB devices, logged in users, scheduled tasks, startup items, WMI event filters, WMI CLI event consumers, and WMI script event consumers<sup>6</sup>

### 3.2 Processing

To be able to give as much context as possible each time we prompt the model, we decided to aggregate the query results into a system state. We classified the query results into two groups: *persistence* and *events*. Results that represent a change to the system that remains until another change reverts them are persistence, e.g. registry entries. Results that consist of one time changes that do not persist until they are explicitly reverted are considered events, e.g. windows crash reports.

The state then consists of a data structure that contains all the persistence information, along with rotating lists of the last  $N$  events.

To create the state, we chose to process the query results using the *kafka streams* API. This is a library that allows stream processing over kafka topics. We are able to consume one or more topics, perform the stream processing operations using the *Java* language, and output the results into another topic. The result of our stream processing is the aggregation of all query results into the state described above.

### 3.3 Model Choice

When deciding upon which model two use, we had two main requirements. First, *the model should be as recent as possible*. As this is a field in such rapid development, it seemed important to us that we use only models from the latest generations, in order to achieve the best possible results. Second, *the model should be small enough to be easy to run locally*. This allows us to run the system using our own hardware. This keeps operational costs as low as possible, while also allowing for reproducibility of the experimental environment.

For these reasons, we chose Qwen3.5-0.8B and MiniCPM5-1B for the first round of testing. However, due to the size of each line of input in our dataset, with some lines of input having over 100k tokens, and the system memory needed for the fine-tuning process scaling quadratically with the size of the input, two memory optimizations needed to be used: gradient check-pointing and flash attention. At the time of our testing, a known bug in flash attention affecting the

<sup>6</sup> WMI stands for Windows Management Instrumentation, and it is the framework for managing data and operations on Microsoft Windows operating systems.

Qwen3.5 family of models left us unable to use Qwen3.5-0.8B. Qwen3-1.7B was used instead.

## 4 Evaluation

In order to test the solution, an experimental environment was set up to gather data for the fine-tuning of the models and the testing of the classifications. This data collection required a realistic and controlled way to perform intrusions. For this, we chose to use the tests from Atomic Red Team. This solution was chosen both for being a recognized solution in the cyber security industry, and for its ease of use.

### 4.1 Procedure

A script was created, consisting of several ART tests run sequentially, each followed by the respective cleanup, to restore the system state to before the test. A certain interval was given between each test and the respective cleanup, as well as between cleanup and the following test. Plans for future work include randomizing test order, as well as introducing non-malicious workloads to be run instead of tests, with a defined probability.

During each run of the script, the osquery service is active in the machine, performing the queries and sending the results to kafka. The results from the kafka streams processor are read into a file during each run, and are then manually classified into which indicate an intrusion.

For the first iteration of our testing, this process was performed three times. This resulted in one dataset for fine-tuning, one for evaluation during the fine-tuning process, and another for the final testing. The final dataset was used for testing both the fine-tuned models and the base version.

### 4.2 Setup

The process of fine-tuning and testing was performed on an ASUS Ascent GX10<sup>7</sup> machine, with the following technical specs:

- **System-on-a-Chip:** NVIDIA GB10
- **System Memory:** 128GB Unified memory.

The data collection was performed on a virtual machine running Windows 11 version 10.0.26200 build 26200. Windows Defender was disabled in this machine so as to not block some of the Atomic Red Team attack scripts.

---

<sup>7</sup> ASUS Ascent GX10: <https://www.asus.com/networking-iot-servers/desktop-ai-supercomputer/ultra-small-ai-supercomputers/asus-ascent-gx10/>

### 4.3 Fine-tuning the models

The models were fine-tuned using the *transformers* library in the *python* language. As mentioned earlier, this first iteration of testing used very long inputs for the fine-tuning process, requiring the settings used to be optimized for lower memory usage, as memory usage for the fine-tuning process scales quadratically with the size of the inputs. For this, *flash attention* and *gradient check-pointing* were used. Each model was fine-tuned for 10 epochs, with a batch size of 1 and 8 gradient accumulation steps. Evaluation loss was used to select the epoch with the best performance.

### 4.4 Results

Before presenting the experimental results, a brief summary of the metrics used is warranted. We use *record* to describe a labeled, expected result, and *prediction* to describe the response given by the model when prompted.

*Accuracy* represents the ratio of all correct predictions. In this case, the ratio of correct predictions to the total set of records.

*Precision* represents how many of the positive predictions were correct. We obtain it by taking the ration of correct positive predictions to all positive predictions.

*Recall* represents how many positive records were correctly predicted. We obtain it by taking the ratio of correct positive predictions to all positive records.

*F1-Score* gives us an harmonic mean of precision and recall. Balancing these two metrics tells us how well the model performs with unbalanced data.

Table 3 shows the results from the testing. These results show an improvement in accuracy for the fine-tuned models, but also show problems with the data used for the fine-tuning and the testing. The times taken by each model to answer to prompts are shown in Table 4.

**Table 1.** Results from first testing round.

|               | Qwen3-1.7B<br>Base | Qwen3-1.7B<br>Fine-tuned | MiniCPM5-1B | MiniCPM5-1B<br>Fine-tuned |
|---------------|--------------------|--------------------------|-------------|---------------------------|
| Accuracy (%)  | 15.79              | 78.95                    | 15.79       | 84.21                     |
| Precision (%) | 15.79              | 25                       | 15.79       | 0                         |
| Recall (%)    | 100                | 16.67                    | 100         | 0                         |
| F1-Score (%)  | 27.27              | 20                       | 27.27       | 0                         |

While the fine-tuning process yielded a measurable increase in overall accuracy for both models, the metrics for the fine-tuned MiniCPM5-1B reveal a critical issue with the training distribution. Despite achieving an accuracy of

**Table 2.** Results from second testing round. For this round, the osquery results were used directly without the creation of a more complex system state.

|               | Qwen3-1.7B<br>Base | Qwen3-1.7B<br>Fine-tuned | MiniCPM5-1B | MiniCPM5-1B<br>Fine-tuned |
|---------------|--------------------|--------------------------|-------------|---------------------------|
| Accuracy (%)  | 88.45              | 96.64                    | 21.71       | 98.3                      |
| Precision (%) | 44.23              | 98.01                    | 45.40       | 97.81                     |
| Recall (%)    | 50                 | 85.57                    | 43.80       | 93.75                     |
| F1-Score (%)  | 46.94              | 90.59                    | 21.51       | 95.66                     |

**Table 3.** Results from third testing round. For this round, the ART script order was randomized, and benign payloads were introduced into the script.

|               | Qwen3-1.7B<br>Base | Qwen3-1.7B<br>Fine-tuned | MiniCPM5-1B | MiniCPM5-1B<br>Fine-tuned |
|---------------|--------------------|--------------------------|-------------|---------------------------|
| Accuracy (%)  | 12.89              | 95.99                    | 87.11       | 97.32                     |
| Precision (%) | 6.44               | 97.33                    | 43.56       | 97.92                     |
| Recall (%)    | 50                 | 84.76                    | 50          | 90.08                     |
| F1-Score (%)  | 11.42              | 89.75                    | 46.56       | 93.52                     |

84.21%, the model recorded 0% for Precision, Recall, and the F1-Score. This mathematical breakdown indicates that the True Positive rate was exactly zero; the model collapsed into exclusively predicting the negative, majority class (non-intrusions). In intrusion detection contexts, baseline system telemetry vastly outnumbers anomalous events. Without strict balancing during the fine-tuning phase, the model simply learned to maximize its accuracy by always predicting the baseline state, rendering it entirely blind to actual attacks. This phenomenon directly motivates the need for the dataset balancing strategies and workload randomization proposed in Section 5

However, the testbed environment performed as expected. We managed to create a reliable framework to collect host information and test language models for intrusion detection. The ease to reproduce the experiment allows us to easily experiment with new queries, ways to process the results, and prompts to the models.

## 5 Conclusion and Future Work

In this work, we presented L8-Sentinel, an IDS that collects and analyzes host data for detecting intrusions. We were able to successfully create a testbed for testing intrusion detection using host data collected using osquery and LLMs for

**Table 4.** Times taken by the model to answer to a prompt.

|             | Qwen3-1.7B<br>Base | Qwen3-1.7B<br>Fine-tuned | MiniCPM5-1B | MiniCPM5-1B<br>Fine-tuned |
|-------------|--------------------|--------------------------|-------------|---------------------------|
| Average (s) | 67.92              | 78.76                    | 50.16       | 56.89                     |
| Minimum (s) | 57                 | 66                       | 41          | 47                        |
| Maximum (s) | 101                | 113                      | 76          | 84                        |

classification. We can collect datasets for testing, fine-tuning and evaluation of the models, and perform classification, in a reproducible way.

The experimental evaluation allowed us to gather insight for future improvements. The use of long, detailed inputs caused difficulties with the fine-tuning process. The time the system takes to perform the final evaluation for each input was also too long. We intend to test with smaller, less detailed input lines to check if we will have better results with faster testing.

We aim to evaluate several ways of improving the data used for future experiments. Larger variety and quantity of atomic red team tests would allow for a larger sample size and variety of attacks for model training. Randomization of test order would help keep the training, validation, and testing data more differentiated, which should also help reduce overfitting. Addition of normal workloads between tests could help to avoid training a model only on malicious data.

## References

1. Berguiga, A., Harchay, A., Massaoudi, A.: Hids-iomt: A deep learning-based intelligent intrusion detection system for the internet of medical things. *IEEE Access* **13**, 32863–32882 (2025), <https://doi.org/10.1109/ACCESS.2025.3543127>
2. Ghourabi, A.: A security model based on lightgbm and transformer to protect healthcare systems from cyberattacks. *IEEE Access* **PP**, 1–1 (2022), <https://doi.org/10.1109/ACCESS.2022.3172432>
3. Grimmer, M., Kaelble, T., Nirsberger, F., Schulze, E., Rucks, T., Hoffmann, J., Rahm, E.: Dataset report: Lid-ds 2021. In: *Critical Information Infrastructures Security* (2022), [https://doi.org/10.1007/978-3-031-35190-7\\_6](https://doi.org/10.1007/978-3-031-35190-7_6)
4. Haas, S., Sommer, R., Fischer, M.: Zeek-osquery: Host-network correlation for advanced monitoring and intrusion detection. *ICT Systems Security and Privacy Protection* **580**, 248 – 262 (2020), [https://doi.org/10.1007/978-3-030-58201-2\\_17](https://doi.org/10.1007/978-3-030-58201-2_17)
5. Hassanin, M., Keshk, M., Salim, S., Alsubaie, M., Sharma, D.: Pllm-cs: Pre-trained large language model (llm) for cyber threat detection in satellite networks. *ArXiv abs/2405.05469* (2024), <https://doi.org/10.1016/j.adhoc.2024.103645>
6. Joraviya, N., Gohil, B.N., Rao, U.P.: DI-hids: deep learning-based host intrusion detection system using system calls-to-image for containerized cloud environment. *The Journal of Supercomputing* **80**, 12218 – 12246 (2024), <https://doi.org/10.1007/s11227-024-05895-3>
7. of New Brunswick, U.: Intrusion detection evaluation dataset (CIC-IDS2017) (2017), <https://www.unb.ca/cic/datasets/ids-2017.html>

8. of New Brunswick, U.: DDoS evaluation dataset (CIC-DDoS2019) (2019), <https://www.unb.ca/cic/datasets/ddos-2019.html>
9. Park, S.H., Yun, S.W., Jeon, S.E., Park, N.E., Shim, H.Y., Lee, Y.R., Lee, S.J., Park, T.R., Shin, N.Y., Kang, M.J., Lee, I.G.: Performance evaluation of open-source endpoint detection and response combining google rapid response and osquery for threat detection. *IEEE Access* **10**, 20259–20269 (2022), <https://doi.org/10.1109/ACCESS.2022.3152574>
10. Satilmiş, H., Akleyek, S., Tok, Z.Y.: A systematic literature review on host-based intrusion detection systems. *IEEE Access* **12**, 27237–27266 (2024), <https://doi.org/10.1109/ACCESS.2024.3367004>
11. Steverson, K., Carlin, C., Mullin, J.M., Ahiskali, M.B.: Cyber intrusion detection using natural language processing on windows event logs. 2021 International Conference on Military Communication and Information Systems (ICMCIS) pp. 1–7 (2021), <https://doi.org/10.1109/ICMCIS52405.2021.9486307>
12. Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: *Neural Information Processing Systems* (2017), <https://doi.org/10.48550/arXiv.1706.03762>
13. Wu, Z., Zhang, H., Wang, P., Sun, Z.: Rtids: A robust transformer-based approach for intrusion detection system. *IEEE Access* **10**, 64375–64387 (2022), <https://doi.org/10.1109/ACCESS.2022.3182333>
14. Zhou, J., Fu, W., Hu, W., Sun, Z., He, T., Zhang, Z.: Challenges and advances in analyzing tls 1.3-encrypted traffic: A comprehensive survey. *Electronics* (2024), <https://doi.org/10.3390/electronics13204000>