People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Research Center on Scientific and Technical Information (CERIST)

Analysis of Simulated Malicious HTTP Traffic at CERIST

Internship Report – Bachelor's Degree in Computer Science
Option: Network Infrastructure and Cyber Security
Numidia Institute of Technology

Students :
Tarek BENNOUAR
Nour El Houda HANI

Supervisors :
Omar NOUALI
Amina HAMZAOUI

Academic Year 2024–2025

Contents

List of Abbreviations

CERIST	Research Center on Scientific and Technical Information
PCAP	Packet Capture (network capture file)
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
CSV	Comma-Separated Values
EDA	Exploratory Data Analysis
ML	Machine Learning
API	Application Programming Interface
IDS	Intrusion Detection System
WAF	Web Application Firewall
LFI	Local File Inclusion
SSTI	Server-Side Template Injection
SQLi	SQL Injection
XSS	Cross-Site Scripting
RCE	Remote Code Execution
CRS	Core Rule Set (ModSecurity/OWASP ruleset)
PyShark	Python wrapper for TShark (Wireshark)
Wireshark	Open-source network packet analyzer
TF-IDF	Term Frequency - Inverse Document Frequency
NaN	Not a Number (missing value)

1 Introduction

Securing information systems and leveraging network data have become major challenges, both in academia and industry. In this context, the detection, analysis, and classification of malicious HTTP traffic represent a major technical challenge, especially in the face of the growing number and sophistication of cyberattacks.

This internship, carried out at CERIST, is part of this thematic and aims to design and document a complete pipeline for extracting, transforming, and analyzing HTTP traffic from attack simulations.

2 Presentation of CERIST

The **Research Center on Scientific and Technical Information** (CERIST) is a leading Algerian public institution under the Ministry of Higher Education and Scientific Research.

Founded in 1985, it is now a key player in the development of information technologies in Algeria.

Its missions cover scientific and technological research in the field of scientific and technical information, management of national databases, network security, and support for the academic community and the information society.

CERIST has specialized research divisions and an extensive regional network, enabling it to carry out innovation projects and support the modernization of the sector.

3 Internship Topic and Problem Statement

The internship focuses on the design and implementation of a technical pipeline aimed at extracting, cleaning, transforming, and analyzing network data from attack simulations, more specifically HTTP traffic captured in PCAP files.

Problem Statement: *How can the extraction of malicious HTTP traffic from raw PCAP captures be made reliable and structured, in order to enable rigorous and reproducible analysis for research or operational security?*

The main objective is to obtain a homogeneous, clean, and usable dataset from heterogeneous and noisy sources, while respecting the constraints and best practices of network data analysis.

4 Specifications

The internship specifications are structured around the following points:

- Extract only HTTP traffic (application layer) from PCAP files simulating different attacks (LFI, SQLi, SSTI, etc.).
- Automate the conversion of this traffic into structured files in JSON and then CSV format, ensuring traceability and reproducibility of the steps.
- Apply minimal required cleaning (structure, artefacts), without advanced enrichment or transformation not planned.
- Generate, from the obtained data, web log files (Apache format) for further tests or simulations.
- Carry out a descriptive exploratory analysis of the resulting data, without proceeding to modeling or machine learning.
- Document each step precisely, including technical choices and encountered limitations.

5 State of the Art / Literature Review

Network traffic analysis in cybersecurity is traditionally divided into two main fields:

- **Detection in operational environments:** Network traffic analysis in cybersecurity relies on packet capture and extraction tools, notably Wireshark and its programmatic interfaces such as PyShark [1, 2], as well as the use of robust pipelines for preparing and cleaning structured data for research and intrusion detection [3, 4, 5].
- **Scientific structuring of data:** Modern approaches rely on data science tools such as pandas [6], and on data mining concepts applied to cybersecurity [7]. The dataset prepared in this internship is inspired by these works, proposing a clean, traceable, and usable corpus in academic or prototyping contexts.

Emphasis is placed on reproducibility, structuring quality, and the use of open and documented tools, in line with scientific literature recommendations [3, 4].

Within this internship, the choice of tools such as **PyShark** (Python interface for Wireshark/TShark) meets the need for reliability and completeness when extracting the HTTP layer. The data cleaning and transformation steps are inspired by data science best practices, with special attention to traceability, documentation, and reproducibility of processing (`pandas`, `json`, etc.).

The objective is not real-time detection of attacks, but the construction of a reliable corpus for exploratory analysis or future applications in machine learning.

6 Technical Implementation

This section describes in detail the processing and analysis pipeline implemented, following the progression of each technical notebook in the project.

6.1 STEP1_Pcap_To_Json_Pyshark.ipynb

- **Purpose:** Extract only HTTP frames from PCAP files simulating different attacks and save each capture as a JSON file for further analysis.
- **Used libraries:** `pyshark` (Python wrapper for tshark/wireshark), `os`, `json`
- **Main actions:**
  - Automated processing of all `.pcap` files in a given folder.
  - For each `.pcap`:
    - Extraction of HTTP packets (application layer) only.
    - Extraction of fields: HTTP method, URL, headers, bodies, IPs, ports, timestamp, etc.
    - Serialization of each HTTP capture as a list of dictionaries.
    - Save as a `.json` file (one per `.pcap`).
- **Technical choices:**
  - Use of `pyshark` to ensure validity and completeness of HTTP extraction.
  - Batch processing possible (processing a whole folder at once).
  - Raw extraction at this stage, no advanced cleaning/filtering.

6.2 STEP2_Cleaning_Flattening_Json.ipynb

- **Purpose:** Clean the raw JSON files (from pyshark) to keep only relevant information and transform them into a flat format.
- **Used libraries:** `os`, `json`
- **Main actions:**
  - Remove useless artefacts (extra fields, non-standard headers, etc.).
  - Keep only essential HTTP headers.
  - Flatten the nested structure into flat columns.
  - Remove duplicates.

  - Save results at each step.

- **Technical choices:**
  - Strict selection of fields to keep.
  - Flat JSON format for easy CSV conversion.

6.3 STEP3_Json_To_CSV.ipynb

- **Purpose:** Convert the flat/deduplicated JSON files into CSV files usable for analysis or ML.
- **Used libraries:** `pandas`, `os`, `json`
- **Main actions:**
  - Read each deduplicated JSON file.
  - Convert to pandas DataFrame.
  - Save as CSV.
- **Technical choices:**
  - Homogenize columns to facilitate global merging.

6.4 STEP4_Combine_CSV.ipynb

- **Purpose:** Merge all individual CSVs into a single global CSV file.
- **Used libraries:** `pandas`, `os`
- **Main actions:**
  - Read all previously generated CSVs.
  - Concatenate all DataFrames.
  - Shuffle rows randomly.
  - Save the final CSV file.

6.5 STEP5_CSV_to_Apache_log.ipynb

- **Purpose:** Generate an Apache-like log file from the global CSV, to facilitate simulation or feeding of classic log systems.
- **Used libraries:** `pandas`, `os`, `datetime`
- **Main actions:**
  - Read the global CSV.
  - Transform each row into an Apache log entry.
  - Save the `.log` file.
- **Technical choices:**
  - Configurable log format for maximum compatibility.

6.6 STEP6_Exploratory_Data_Analysis.ipynb

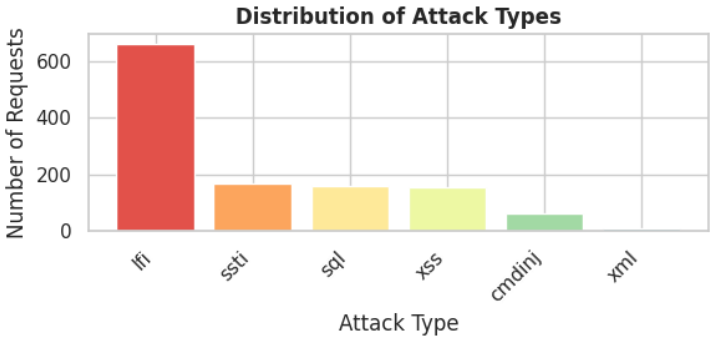
- **Purpose:** Carry out a rapid exploratory data analysis (EDA) to extract main properties/statistics.
- **Used libraries:** `pandas`, `matplotlib`, `seaborn`, `numpy`
- **Main actions:**
  - Visualize main distributions (HTTP methods, attacks, timestamps, headers, etc.).
  - Descriptive statistics (counts, diversity, anomalies, etc.).
- **Technical choices:**
  - Varied visual representation and focus on discriminative columns.

7 Results and Discussion

The exploratory analysis highlights several characteristic properties of the generated dataset:

- **Distribution of attack types:**

  The distribution is highly unbalanced, with a clear dominance of `lfi` attacks, followed by `ssti`, `sql`, `xss`, and then rarer families (`cmdinj`, `xml`).

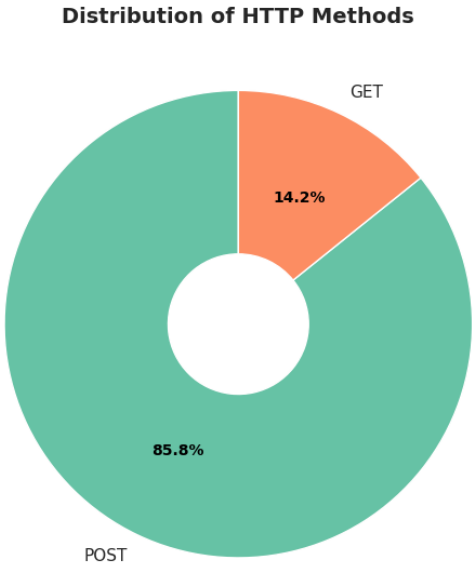


Distribution of attack types (`attack_tag`).

This imbalance must be considered for any comparative analysis or supervised modeling.

- **HTTP methods used:**

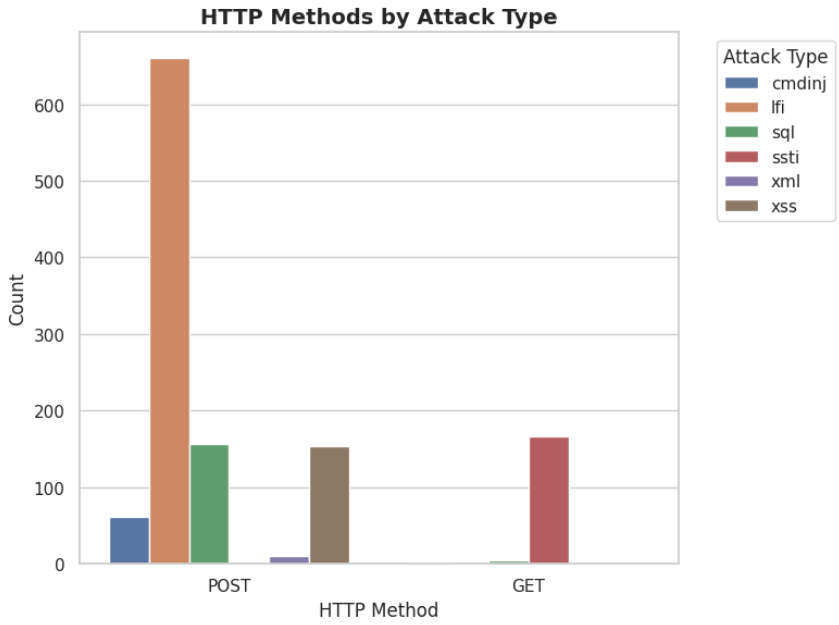
  The following figure shows the overall distribution of HTTP methods in the dataset, distinguishing the share of each type of request.



Distribution of HTTP methods (`GET` vs `POST`).

A clear dominance of the `POST` method (about 86%) over `GET` (14%) is observed, which reflects the nature of the simulated attacks: most injection payloads require a request body, thus a `POST` request.

The next figure details the distribution of HTTP methods by attack type.

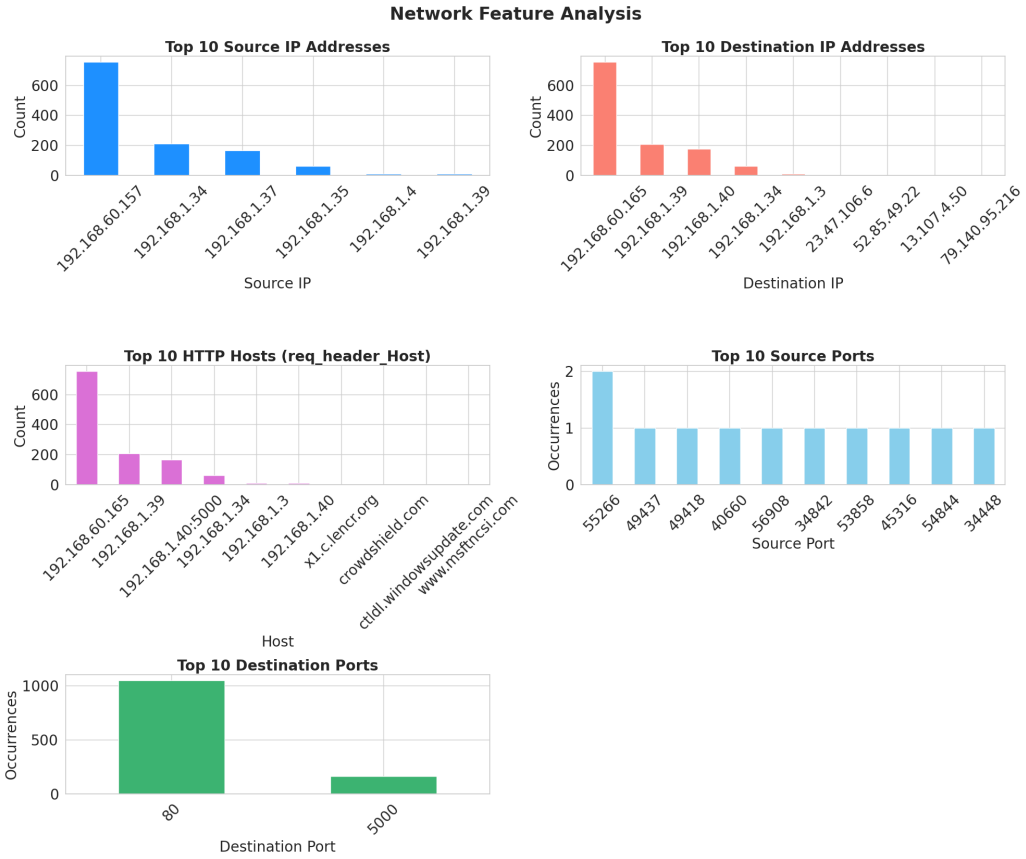


HTTP methods by attack type.

Some attack families (notably LFI, XSS, SQLi) are mostly conveyed by `POST`, while others such as SSTI also significantly use the `GET` method. This correlation between injection technique and HTTP method is typical of synthetic datasets in cybersecurity and reflects the designed simulation scenarios.

- **Diversity of IP addresses and ports:**

  The analysis of source/destination IPs and ports shows low diversity, typical of a controlled test environment.

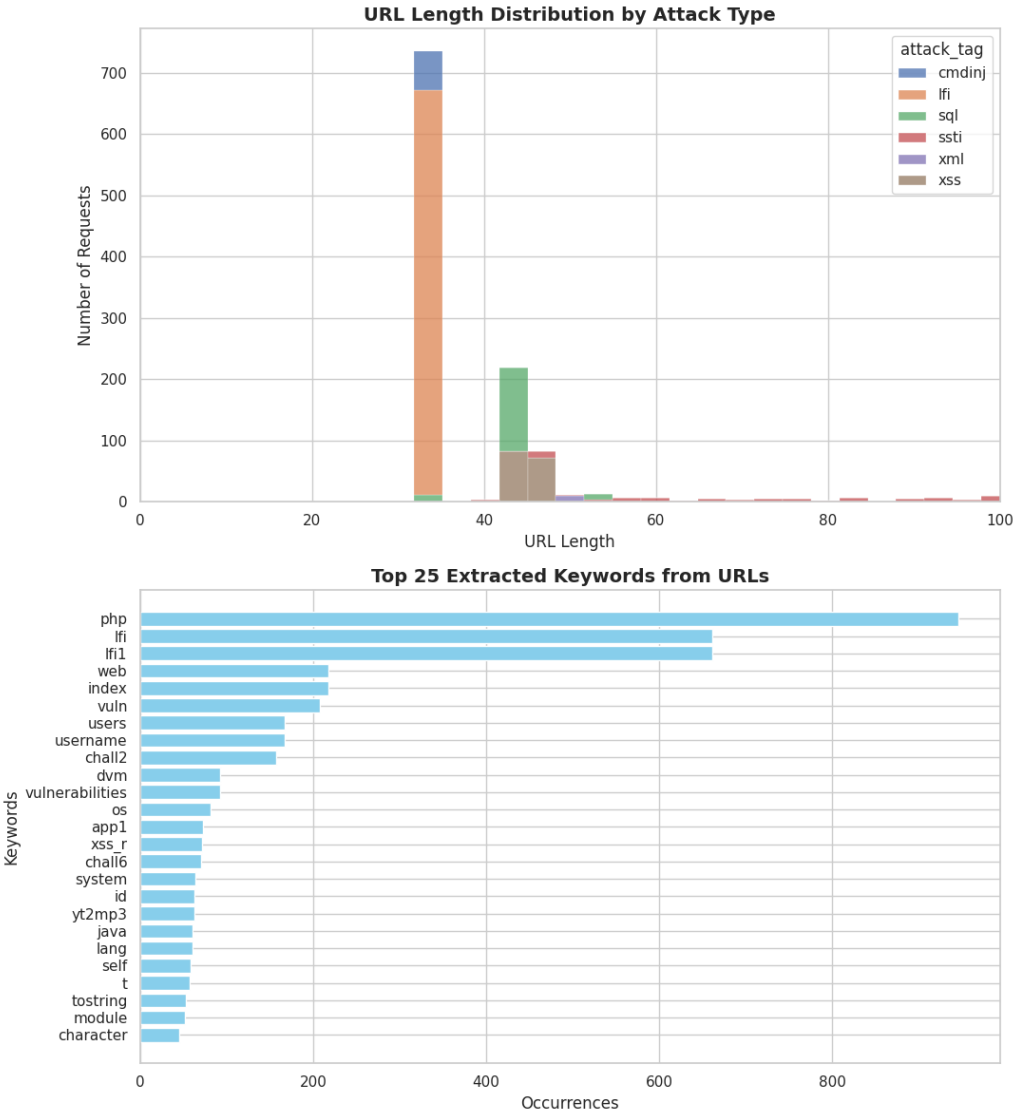


Top source, destination IPs and ports.

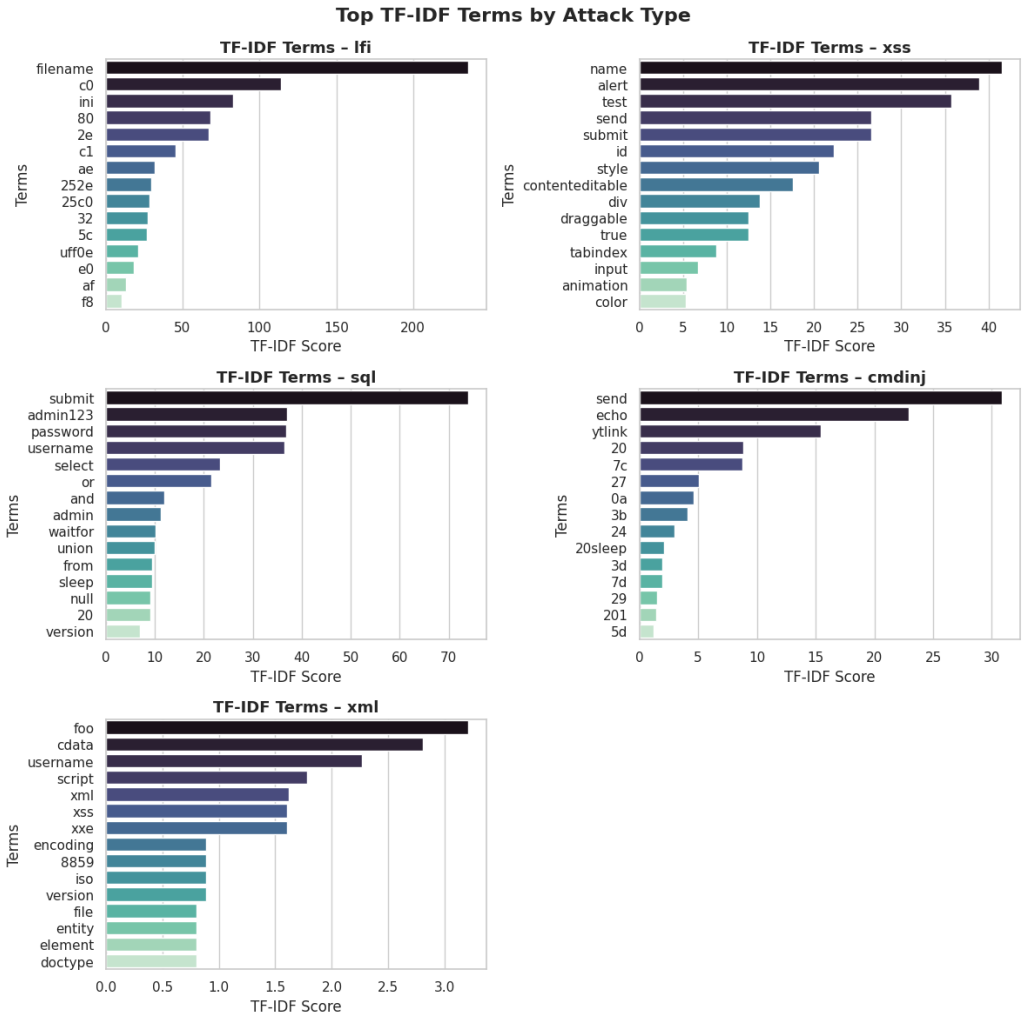
This reflects a configuration adapted to a simulation context, but limits representativity for real applications, where IP and port diversity is much higher. Nevertheless, this low diversity is not problematic in our case, since IP addresses and ports are not part of the attack payloads and do not participate in the injection or exploitation logic.

- **Analysis of URLs and tokens:**

  URL lengths, token frequency, and keywords (TF-IDF) vary by attack family. For example, LFI typically features frequent file paths, SQL has injection-related keywords.



URL analysis: length, token distribution, main keywords.

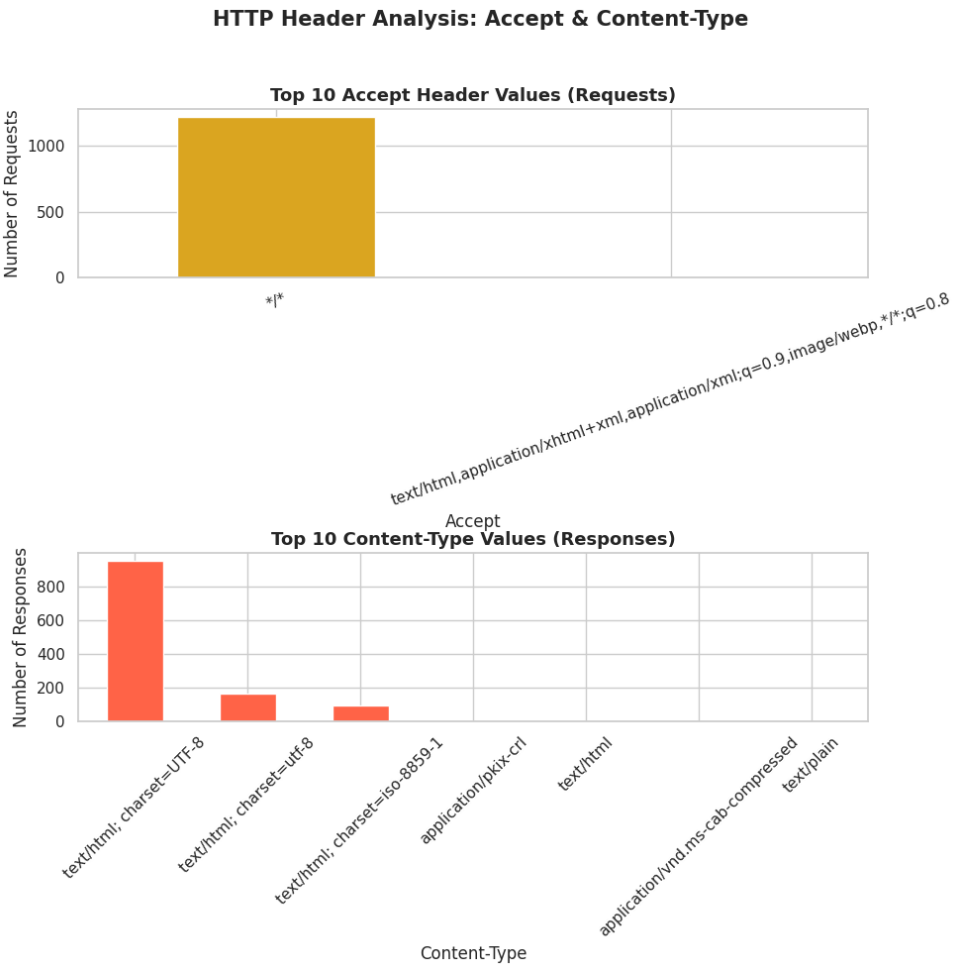


Top TF-IDF terms extracted by attack family.

TF-IDF analysis and extracted keyword distribution reveal strong specialization of vocabulary depending on attack type: for example, LFI requests massively use tokens related to file paths (`filename`, `ini`, `php`), while SQL requests contain many injection terms (`select`, `union`, `admin123`). This per-family structuring is also reflected in the distribution of URL lengths: LFI attacks usually have short but repetitive paths, while SSTI or XSS attacks generate longer and more varied URLs. Finally, the most frequent keywords in URLs highlight recurring patterns specific to each attack scenario, which validates the quality of the simulated corpus for differentiated analysis by injection technique.

- **Analysis of HTTP headers:**

  The figure below shows the distribution of main HTTP header values, notably `Accept` in requests and `Content-Type` in responses.



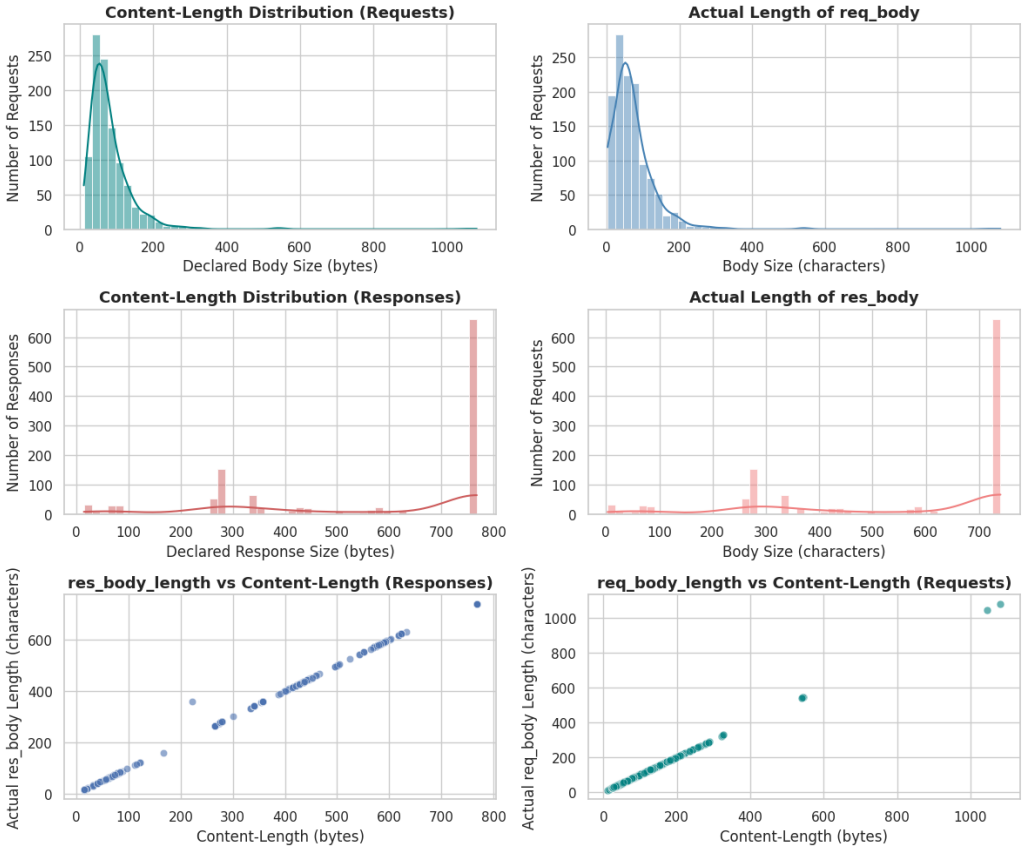
Distribution of HTTP header values (`Accept` and `Content-Type`).

There is a very strong dominance of a few values: for example, the `Accept` header is overwhelmingly set to `*/*` in requests, while responses are mostly `text/html; charset=UTF-8`. This homogeneity reflects the use of automation tools during the simulations, generating highly standardized and little-varied requests in terms of headers.

Note that some apparent "varieties" in values (`utf-8` vs `UTF-8`, or format differences) are more about a lack of normalization than true diversity of use. This can bias statistical analysis if these values are not harmonized, but in this context, it faithfully reflects the reality of a synthetic dataset produced by automation.

- **Properties of bodies and content-length:**

  The figure below shows the cross-distribution between the value declared by the `Content-Length` header and the actual length of request and response bodies, for the entire dataset.



Analysis of the `Content-Length` header and actual body size.

The distributions of the `Content-Length` header and the actual body lengths (`req_body`, `res_body`) show good global consistency: most values are close and follow a similar distribution. However, a significant proportion of zero lengths (`length = 0`) is noted for `req_body`, which is natural: most HTTP `GET` requests have no body, while the `Content-Length` field is often set only for `POST` requests with data to transmit. The few discrepancies observed are due to the synthetic nature of the simulations and to differences in encoding or formatting during capture.

- **Global limitations and biases:**

  Despite the structural quality and reproducibility of the pipeline, several **limitations and biases** remain:

  – **Controlled simulation:** The dataset is exclusively composed of simulated traffic, generated according to pre-defined scenarios. It includes **no benign**

or "normal" background traffic. However, this is not a major weakness in academic or prototyping contexts: this kind of dataset is common in cybersecurity, where capturing real attacks is complex and time-consuming. **It remains entirely possible to combine this attack corpus with a benign traffic dataset** (e.g., CICIDS, UNSW-NB15, or in-house capture), to obtain a mixed and more realistic base for modeling or detection testing.

- **Low diversity of IPs/ports:** Almost all exchanges originate from a very limited number of IP addresses and ports, corresponding to a "closed" test environment. This does not affect the relevance for studying payloads, as **IP and port are not involved in injection**, but limits the ability to assess pipeline robustness in more heterogeneous infrastructures.
- **Marked imbalance between attack types:** The class distribution (LFI, SSTI, SQLi, etc.) is highly unbalanced (LFI is ultra-majority, XML is very rare). This **distribution bias** impacts any statistical modeling or machine learning: it is essential to consider it to avoid overfitting or hasty conclusions.
- **Homogeneity of HTTP headers and methods:** Headers such as `Accept` or `Content-Type` and the `POST` method largely dominate. This homogeneity reflects the massive use of **automation tools** (`Wfuzz`, `curl`...), which does not always mimic the diversity of real traffic seen on the Internet. Poor normalization (e.g.: `utf-8` vs `UTF-8`) can also **introduce false positives** in diversity analysis.
- **Structuring specific to simulated scenarios:** The specialization of tokens (TF-IDF) and URL length distributions are directly related to how the attacks were generated. Some injection "rules" or attack patterns may therefore be **over-represented** (LFI, SQL, etc.), while other techniques or more subtle variants are absent.

In summary, this pipeline offers a solid foundation for the exploration and exploitation of HTTP attack data, but it **mainly reflects the limits of an academic/simulated environment**. Any extrapolation to production detection, machine learning, or generalization of results must take into account:

- The absence of real benign traffic (can be addressed by merging with other datasets)
- The class imbalance
- The parameter homogeneity
- The specialization induced by simulation tools

To go further, it would be **essential to introduce**:

- Heterogeneous background traffic
- Greater diversity in tools, techniques, and capture contexts
- Better normalization of text fields

– More varied and realistic attack scenarios

8 Conclusion

This internship at CERIST allowed us to tackle, in a concrete and methodical manner, all the steps required for building, structuring, and leveraging a network dataset dedicated to HTTP security analysis. The design and realization of the complete pipeline — from PCAP extraction to detailed statistical exploration — gave us solid experience in cleaning automation, structured data handling, and descriptive analysis of simulated malicious traffic.

The in-depth analysis showed that the generated dataset, although usable and homogeneous, is representative of a controlled simulation environment:

– Strong imbalance between attack types,
– Homogeneity of IP addresses, ports, and HTTP headers,
– Absence of real benign traffic or background context,
– Structuring guided by automation tools and simulated scenarios.

These properties, typical of synthetic datasets in cybersecurity, are both a strength (control, reproducibility, traceability) and a limitation (generalization, diversity of real cases). Any future use — whether for automated detection, learning models, or robustness tests — should consider these biases, and ideally combine this corpus with real benign flows and more varied attack scenarios.

This work also highlights the importance of data quality, rigorous documentation, and reflexivity regarding the biases introduced at each step. It lays a solid technical foundation for extending the pipeline to advanced uses (machine learning, IDS/WAF testing, dataset enrichment), while reminding of the need for critical vigilance about the representativity of simulated datasets.

Appendix

– **Project GitHub repository:**
  - github.com/Noah-213/cerist-http-malicious-dataset
    This repository contains all Python scripts, Jupyter notebooks, intermediate datasets, and generated figures during the project (complete pipeline, cleaning, EDA, visualizations, logs, etc.).

9 Bibliography

[1] Wireshark Foundation. *Wireshark User's Guide*. https://www.wireshark.org/docs/wsug_html_chunked/. Accessed: 2024-06-07. 2024.

[2] KimiNewt. *PyShark Documentation*. https://github.com/KimiNewt/pyshark. Accessed: 2024-06-07. 2024.

[3] Matthias Ring et al. "A survey of network-based intrusion detection data sets". In: *Computers & Security*. Vol. 86. Elsevier, 2019, pp. 147–167. URL: https://doi.org/10.1016/j.cose.2019.06.010.

[4] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)* (2015), pp. 1–6. URL: https://ieeexplore.ieee.org/document/7348942.

[5] Bayu Priyambadha Tama, Kyung-Hyune Rhee, and Taehoon Hwang. "Network intrusion detection system using undersampling and ensemble learning". In: *Information Sciences* 484 (2019), pp. 357–365. URL: https://doi.org/10.1016/j.ins.2019.01.079.

[6] Wes McKinney. "pandas: a foundational Python library for data analysis and statistics". In: *Python for Data Analysis* (2012). URL: https://wesmckinney.com/pages/book.html.

[7] Salvatore J. Stolfo et al. "Data Mining and Machine Learning in Cybersecurity". In: *Springer US* (2017), pp. 1–20. URL: https://doi.org/10.1007/978-1-4899-7641-3_1.