





# InspectorLog: A New Tool for Offline Attack Detection over Web Log Trace Files

Jesús E. Díaz-Verdejo<sup>1</sup><sup>a</sup>, Javier Muñoz-Calle<sup>2</sup><sup>b</sup>, Rafael Estepa Alonso<sup>2</sup><sup>c</sup>  
and Antonio Estepa Alonso<sup>2</sup><sup>d</sup>

<sup>1</sup>*Dpt. of Signal Theory, Telematics and Communications, University of Granada, Granada, Spain*

<sup>2</sup>*Dpt. of Telematics Engineering, University of Seville, Seville, Spain*

**Keywords:** Network Security Monitoring, Intrusion Detection Systems, Web Attacks Detection.

**Abstract:** InspectorLog is a novel tool for offline analysis of HTTP logs. The tool processes web server logs to identify attacks using diverse rule sets, focusing primarily on the URI field. It is compatible with standard rule formats from systems such as Snort, Nemesida, and ModSecurity. This paper describes InspectorLog functionalities, architecture and applications to the scientific community. We also experimentally validate InspectorLog by comparing its detection power with that of the IDS from which rules are taken. Inspector log fills a gap in available tools in cybersecurity practices in forensic analysis, dataset sanitization, and signature tuning. Future enhancements are planned to support additional Web Application Firewalls (WAFs), new rule types, and HTTP protocol methods, aiming to broaden its scope and utility in the ever-evolving domain of network security.

## 1 INTRODUCTION


Web traffic is critical in today's digital era due to increased online activity, remote work, e-commerce, and Internet access. Many vulnerable web servers support web services and websites, attracting attackers. Recently, there's been a significant rise in web server attacks, heightening risks for organizations, as compromised servers can initiate sophisticated attacks (Husak, 2021). This trend highlights the rising interest in detecting web server attacks in both academic and commercial sectors.


Traditionally, the detection of web attacks has primarily been based on Intrusion Detection Systems (IDS) (Agarwal, 2018) and Web Application Firewalls (WAFs) (Palka, 2011). IDSs are broader in their detection capabilities, examining both network and host data for attacks, while WAFs are specifically designed to analyze requests sent to a web server and typically include filtering and blocking features. An advantage of WAFs is their ability to function without decrypting HTTPS traffic, especially when they are co-located within the Web server module.


Common IDSs (e.g., *Snort*) and WAFs (such as *Modsecurity* or *Nemesida*) operate primarily on signature detection (S-IDS), seeking known attack patterns through specific rules. Because this approach falls short against zero-day attacks, these systems often integrate anomaly detection (A-IDS), which helps identify unusual activities, thus bolstering defenses against new threats (Betarte, 2018), (Hajji, 2021).


These systems detect in real time potential attacks by looking for anomalies or matching against a database of known attack signatures, referred to as *rules*. This online operation is the standard practice for detecting web attacks. However, other tasks related to web cybersecurity are conducted offline such as:

- Forensic analysis: Following an incident, server logs are scrutinized to find attacks unnoticed by existing protections systems (if any). The precision of this forensic analysis improves with the size and quality of the rule sets used (Díaz-Verdejo, 2022).
- Signature tuning for S-IDS: The rule set used by S-IDS needs to be adjusted for a site to minimize false positives. This tuning process is crucial when employing free signatures –more prone to false positives– (Díaz-Verdejo, 2022), (Neminath, 2014), (Aldweesh, 2020). This task entails linking requests to the alerts produced (SIDS) across

<sup>a</sup> <https://orcid.org/0000-0002-8424-9932>

<sup>b</sup> <https://orcid.org/0000-0001-8146-8438>

<sup>c</sup> <https://orcid.org/0000-0001-8505-1920>

<sup>d</sup> <https://orcid.org/0000-0003-1841-3973>

a sufficiently period to ensure meaningful results for the website. Conducting this offline adjustment using gathered logs streamlines the process and cuts down on the time required.

- Adjustment A-IDSs operational point: Anomaly detection depends on models or profiles, usually derived from statistical or AI-based methods, which require careful tuning. Effective adjustment requires collecting attack-free, representative website traffic and significant attack data (Tavallae, 2010). In this process, offline sanitization of the HTTP server log file is needed (Diaz-Verdejo, 2020). This process includes an initial phase using various IDS and signature sets, followed by a semi-automatic phase handled by the operator.

These activities are carried out off-line and require web-related log data accumulated over time, typically recorded in text format in the server log. However, many web security testing tools (e.g., *AppScan*, *Burp-Suite*, *ZAP*, *ModSecurity*) function as an HTTP Proxy, while IDS and WAF tools directly analyze live or captured traffic. Consequently, these online tools require the reconstruction of traffic from the server log, which involves emulating the entire client-server interaction. This can introduce inaccuracies due to factors like encoding issues, missing values not captured in the log (such as `cookies` and `referrer`), and results in significant time consumption. While some online tools can import a *pcap* traffic file to streamline iterative analysis, this still doesn't circumvent the need to recreate traffic from the log files.

It is also essential to associate each alert with its corresponding web request by analyzing the application's output log. For traffic sanitization and forensics, employing multiple rule sets similar to Google's *VirtusTotal*, which uses various virus engines, can improve detection reliability. However, this approach faces challenges such as processing diverse log formats, increased CPU usage, and the need to match web requests with alerts uniquely across different software components.

This paper presents *InspectorLog* (IL, 2024), a novel tool written in C for offline analysis of (*Apache*) web server logs to identify attacks using a set of rules available. *Inspectorlog* focuses primarily on the URI field, which is included in over 85% of HTTP attack signatures by prominent WAF/IDS like *Snort* and *ModSecurity*. *InspectorLog* analyzes the HTTP log, applying rule sets from various IDS and WAF to the URI field, aiding in attack detection. This enables the use of *Inspectorlog* as an off-line substitute for those on-line tools just by applying the corresponding rulesets. Its applications include, among others,

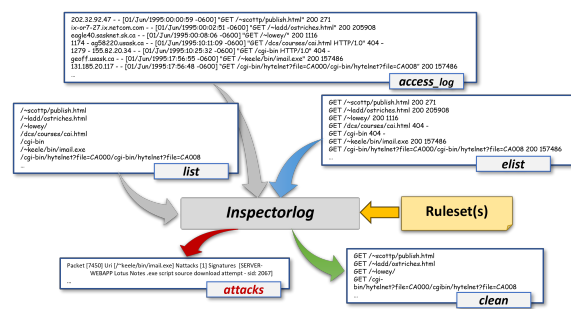


Figure 1: InspectorLog workflow.

forensic analysis, dataset sanitization, and signature tuning. To our knowledge, no other tool exists nowadays for these specific tasks. Past attempts include tools as *Scapl!* (Scalp, 2013), and *PHPIDS* (PHPIDS, 2012), both discontinued in 2013 and currently outdated. Nowadays, only *Modsecurity* version 3 is available to provide an API that can be used to develop tools for log file analysis but is restricted to use the *ModSecurity* *SecRules* format.

The paper is organized as follows. Section 2 describes *InspectorLog*'s functionality and architecture. Section 3 shows our experimental validation of the tool. Section 4 presents practical use cases and highlights the tool's limitations. Finally, Section 5 concludes the paper with a summary of findings and suggestions for future research and development.

## 2 DESCRIPTION AND ARCHITECTURE

*InspectorLog* uses string comparison and regular expressions to find rule-associated signatures in URIs. It processes log trace files in various formats and rule sets from S-IDS tools (Figure 1). The output of *InspectorLog* includes lists of log records that either trigger rules (with rule details) or don't trigger any, indicating likely legitimate requests. This process facilitates the creation of lists of clean and attack requests and the pairing between alerts and associated log registers.

The latest release of *InspectorLog*, version 3.6, available at (IL, 2024), comprises two main programs: *inspectorlog* and *ms-inspectorlog*. The former applies signatures akin to *Snort* and *Nemesida*, while the latter is compatible with *ModSecurity* rules.

*InspectorLog*, in its current version, is compatible with the following rule formats:

- *Standard Snort rules*: Rules written in the language originally defined by *Snort* related to HTTP. Both content and regular expression based

Table 1: Example of output file (U2URI format).

```
# inspectorlog v3.6.0.1
#--- Initializing Rules (SNORT) -----
# Rules directory : </data/uri-data/biblio-ds/tmp/reglas>
# Opening SNORT rule file ./http_uri-er-20220224-m2.rules... done
# Rules: read [5918], erroneous [8], URI [5910]
#--- Statistics (SNORT) -----
# Read [9514] Snort rules, [9500] http-related, [13] with errors
#--- Analysis results -----
# Alerts & signatures generated from: ...
Packet [62][A06] Uri [/day.css?o8r51f] Nattaacks [1] Signatures [882]
Packet [73][A09] Uri [/w.m.x/05-16.jpg] Nattaacks [1] Signatures [882]
Packet [85][A13] Uri [/c_m.css?o86dyb] Nattaacks [1] Signatures [882]
...
# N. packets [35934], [44] with alerts, N. Alerts [49]
```

fields for search patterns are supported. This covers rules from trusted sources as *Talos*, *VRT*, *Suricata* and *ET*.

- *Nemesida rules* (<https://rinfo.nemesida-security.com/>). The *Nemesida Community Edition* rules file is supported. This file includes white lists and black lists for fixed strings and regular expressions, which are properly supported.
- *ModSecurity SecRules format*: ModSecurity standard format for configuration and rules. The format for ModSecurity rules is standard and is managed through the corresponding configuration file.

*InspectorLog* processes various input log file formats (Fig. 1), including raw uri lists (*list*), selected fields (*elist* *telist*) and Apache standard log (*Apache*), among others.

Each line in these formats represents one record. An optional tag, enclosed in ' [] ', can be used at the start of each line for indexing and reference purposes.

The attack-related output is directed to a file (Table 1) and contains information about the loaded rules, statistics and a list of URIs identified as attacks, presented one record per line.

The log format for each attack register, named as U2URI format, contains information about the detected attacks and the URI triggering the detection as well as indexing information to match output-input registers.

The clean registers are optionally written to a file 'as-is', that is, the clean output file is a filtered version of the input one.

*InspectorLog* distribution includes several auxiliary tools to handle/convert labels and extract selected fields from input and output formats.

Further information about usage, parameters and file formats can be found in (IL, 2024). Additionally, sample trace and rule files for testing are available under the `tests` directory of the distribution.

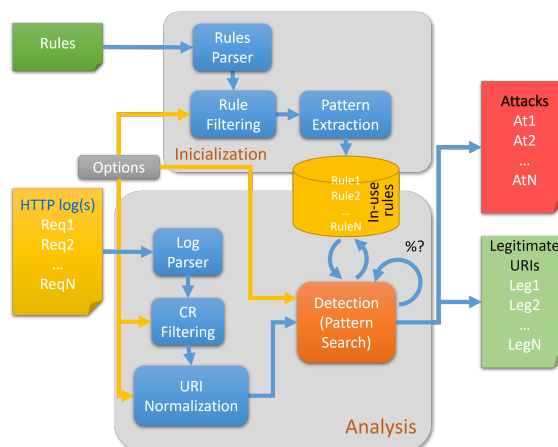


Figure 2: InspectorLog building blocks and architecture.

## 2.1 Building Blocks and Architecture

*InspectorLog* is structured using a modular architecture, as depicted in Figure 2, with the central component being the detection module. This module's role involves methodically applying a chosen group of rules to every request present in the traces.

The operational steps of *InspectorLog* are:

- **Initialization:** In this phase, the tool reads the rules from the files specified sequentially. It extracts various fields from each rule (using a rules parser module), focusing on those relevant to the HTTP service (through a rules filtering module). Non-relevant methods based on the trace format are also excluded. A signatures database is then created, including extracted patterns and search conditions (pattern extraction module).
- **Analysis:** Post initialization, the tool reads the input records in the trace file individually, extracting pertinent fields based on the input format (trace parser module). Request filtering is optionally done by response code (CR<300 or CR<400) — CR filtering module—. URIs undergo normalization (URI normalization module), particularly concerning percent encoding. This is done iteratively, to account for multiple percent encoding evasion techniques, with the first iteration unchanged and subsequent iterations decoding percent encoding sequences. This continues until all encoded characters are decoded or up to a preset fixed number of iterations. Each iteration produces a request for the detection module, which applies signatures in use to identify patterns. Detection at any stage results in an attack log, optionally including detailed detection information. If no detection occurs, the record is marked legitimate and added to the legitimate list.

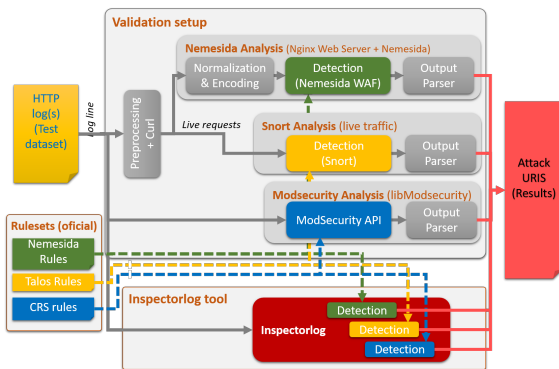


Figure 3: Validation testbed and methodology.

Despite the standard format outlined in RFC 3986, URI normalization regarding percent encoding is essential, as many systems overlook strict adherence to this standard. Some attacks also exploit percent encoding as a disguise, typically using `%25` to encode the `'%'` symbol or embedding one percent encoding into another (e.g. `%%341` to encode `'A'`). Multiple encoding with more than 3 levels, like using `%252525` for a single final `'%'` symbol, has been observed during the tool usage. Additionally, inconsistencies in percent encoding usage in *Talos* and *ETOpen* rules led to an optional process that applies, decodes, and re-applies rules to URIs with percent encoding until all encoding is resolved. Additionally, *InspectorLog* includes an option for handling case sensitivity in rules and character encoding, converting all characters to lowercase when activated. In its current implementation, the detection is based on the set of all the loaded rules, i.e., an OR operation is applied regarding the rules from different sources. This can be easily modified to allow for other operations as done in (Díaz-Verdejo, 2022).

### 3 EXPERIMENTAL VALIDATION

In order to ascertain the reliability of the outcomes provided by the tool, this section will undertake a comparative analysis of the results obtained with a real dataset using rulesets with the original Intrusion Detection Systems (IDS) or WAF vs applying these rules with *InspectorLog*. The objective is to validate the tool and its usability.

#### 3.1 Methodology Used

The methodology employed to conduct this study is depicted in Figure 3. The dataset used is Biblio-US17 (Biblio-US17, 2023), comprising 47,402,907 HTTP traffic requests, both attack-related and legitimate,

captured in a real-world environment. Biblio-US17 is a public dataset with selected fields of HTTP requests, collected from the web server traces of the University of Seville (Spain) Library, covering the period from January 1, 2017, to July 17, 2017. The server, developed using *Drupal CMS* (v7.96) and operating on *Apache* web server (v2.2), delivers a highly dynamic service focusing on HTTP requests related to available bibliographic resources. The dataset includes labels that establish a *ground truth*, distinguishing between legitimate requests (42,473,128) and attacks (327,906) based on URI.

Each experiment involves the comparison of the output (detections) of IL with a ruleset for one of the IDS tools—i.e. IL-Snort, IL-Nem and IL-MS—with those from the tool itself. Thus, the configurations for the experiments are:

- Snort validation: IL-Snort vs Snort (v2.9). Rulesets Talos Community (24/03/2022) and ETOpen (open version, 24/02/2022) were used.
- ModSecurity validation: IL-MS vs ModSecurity (v3.0). OWASP CRS 3.2.0 rule set was used with different paranoia levels (PL).
- Nemesida validation: IL-Nem vs. *Nemesida WAF CE* (v1.18). The *Nemesida Community Edition* rules published as of 09/01/2022 were used.

Given that *Snort* and *Nemesida WAF* only have the capability to analyze live HTTP traffic (online mode), it was necessary to employ a client/server web architecture for transmitting URIs carried in HTTP messages with minimal alterations compared to the original dataset (as shown in Figure 3). To circumvent modifications typically applied by web clients, such as character encoding or path interpretation, the web client *curl* was chosen for its ability to respect the original encoding. *Nginx* version 1.18 was used as the web server. On the other hand, since *ModSecurity* version 3 offers an API for direct reception, this option was utilized to avoid generating network traffic and potential artifacts introduced by the web client and server in the comparison.

#### 3.2 Results and Discussion

Table 2 presents the results of the attacks detected by each software using the specified rules. For *ModSecurity*, all four existing Paranoia Levels (PL) have been used. As we aim to compare the outcomes between the rules interpreted by *InspectorLog* (IL) and the rules interpreted by the original software upon receiving a request, the table does not account for repeated requests (all attacks are distinct). In column *Total (P%)*, we show the number of requests identi-



Table 2: Attack detection results.

S-IDS		# Attacks	
Tool	PL	Total (P%)	Unique (P%)
IL-Snort	-	784 (98.72)	24 (70.83)
Snort	-	768 (99.35)	8 (75.00)
IL-MS	PL1	1223 (96.89)	0
	PL2	1990 (88.04)	0
	PL3	142161 (87.57)	0
	PL4	147083 (84.77)	0
ModSec.	PL1	1223 (96.89)	0
	PL2	1990 (88.04)	0
	PL3	142161 (87.57)	0
	PL4	147083 (84.77)	0
IL-Nem	-	8843 (95.24)	7903 (91.70)
Nem.	-	971 (75.08)	31 (4.12)

fied as attacks by each software, with the (P)recision — $tp/(tp+fp)$ , being  $tp$  the number of true positives and  $fp$  the number of false positives— in percentage indicated in parentheses. Column *Unique (P%)* includes the count of URIs detected by each IL module but not by its equivalent Intrusion Detection System (IDS), and vice versa, such as attacks detected by IL-Snort but not by *Snort* (including Precision in parentheses). It should be noted that precision is primarily related to rules' quality, that is, it is not a measure of the performance of the tools.

Overall, *InspectorLog* demonstrates to perform notably precise. After more than 47 million requests have been analyzed and the error rate in classification, IL's behavior is very similar to that of the equivalent Intrusion Detection System (IDS). In the case of *IL-ModSec* and *Modsecurity* (both offline), no discrepancies are observed between the results, which perfectly validates the functioning of IL across various modes (PL1 to PL4).

In the cases of *Snort* and *Nemesida*, slight differences are observed between the detections of the *IL-Snort* and *IL-Nem* modules and their respective baseline SIDS. These discrepancies are primarily due to their different modes of operation, offline or online, which results in variations in the set of URIs effectively subjected to the signatures.

Most of the discrepancies between IL and their counterpart IDS can be attributed to:

- *IL* iteratively remove percent encoding from incoming URIs, a process not present in *Snort* and *Nemesida*.
- Before the application of signatures, *Snort* and *Nemesida* employ pre-processors (e.g. `http_inspect` in *Snort*) to network messages. This preprocessing may result in the blocking or modification of messages.
- The necessity of encapsulating URIs within an

HTTP message in online mode inevitably leads to minimal URIs alterations (URI Normalization). These changes may cause a URI to be no longer considered an attack or conversely.

In the case of *Snort*, the validation results vary slightly. Out of over 47 million requests, both systems identified the same 760 attacks, though *IL* detected an additional 24 and *Snort* an extra 8. In both instances, the accuracy of detection is similar, and it is postulated that this may be attributed to the handling of percent encoding. The following example illustrates a discrepancy arising from the treatment of percent-encoding in IL. Consider the URI

```
/shell?%75%6E%61%6D%65%20%2D%61
```

in *IL*, the preprocessing involves the substitution of percent-encoding, resulting in `/shell?uname=a`, whereas IDS *Snort* does not perform such substitution, consequently failing to detect this attack. The remaining cases are similar, leading to the conclusion that IL's interpretation of *Snort*'s rules is appropriate.

Regarding the outcomes from *Nemesida*, the primary discrepancy lies in 7,903 attacks identified by *IL* but not by *Nemesida*. Intriguingly, most of these attacks have been accurately detected, with a precision of 91.7%. On the other hand, the 31 attacks detected exclusively by *Nemesida* predominantly correspond to false positives. We have analyzed three relevant cases among the 7,903 attacks detected solely by *IL* that aid in elucidating the results:

- There are 6,907 requests identified as buffer overflow attacks with double encoding, i.e. % is encoded as %25. An example is  

```
...:translateBUS('es%252525...2525cen')
```

These requests have been recognized solely by *IL*. Either it is because the web server's processing of the request discarded it due to excessive length, or it is attributable to the preprocessing of percent encoding that was performed.

- Other requests not detected by *Nemesida* are those where the URI contains `'/./.'`. This scenario impacts 433 attacks only identified by *IL*. This might also be attributable to the preprocessing carried out by the Web server.
- A final case type not detected by *Nemesida*, potentially due to preprocessing and percent-encoding, would be:

```
/xup.php?x=...&login=go%21&H=
```

It should be noted that in this case, the rule SID 1145 is triggered when `'go!'` is encountered (as with *IL*, thanks to preprocessing). This case accounts for 128 attacks.

These three cases analyzed explain 7,468 out of the 7,903 attacks recognized by *IL-Nem* but not by *Nemesida*. The remaining cases are similar, and are presumably due also to artifacts introduced by the web client or by processing on the web server. Therefore, it can be asserted that the reliability of *InspectorLog* tool's interpretation of *Nemesida*'s rules is acceptable.

## 4 LIMITATIONS

Given the nature of the input (log files), the tool exhibits some notable limitations:

- *Inspectorlog* operates on a per-request basis. This means that rules that take into consideration the relationship between packets (e.g., flows, number of packets in a connection, connection state conditions) cannot be applied. This limitation may result in false positives and is insurmountable.
- In the current version, rule fields that determine relative positions between different rule components (e.g., distance) are disregarded. This can also lead to FPs.
- Challenges have been identified in rules containing %00 due to the line-by-line reading of files.
- Only the most commonly used methods incorporating URIs are considered in the current version: GET, POST, HEAD, PROPFIND, PUT, OPTIONS.

## 5 CONCLUSIONS AND FURTHER WORK

We have developed and validated a new tool for offline analysis of HTTP logs. To the best of our knowledge, there are no other tools currently available that are similar to *Inspectorlog*. The availability of this tool can significantly facilitate the tasks of cybersecurity officers and researchers by enabling the analysis of web service traces without the need to replicate the associated traffic. This tool exhibits acceptable accuracy and can be utilized for tasks such as sanitization, tuning of Anomaly-based Intrusion Detection Systems (AIDS), optimization of rulesets for a given scenario or forensic analysis.

We intend to make it available to the scientific community to further enhance its functionalities. Specifically, we aim to eventually support other WAFs, new types of rules and fields, and additional methods of the HTTP protocol.

## ACKNOWLEDGEMENTS

This work has been funded by the research grant PID2020-115199RB-I00 provided by the Spanish Ministry of Industry under the contract MICIN/AEI/10.13039/501100011033

## REFERENCES

- Agarwal, N. and Hussain, S. Z. (2018). A closer look at intrusion detection system for web applications. *Security and Communication Networks*.
- Aldweesh, A., Derhab, A. and Emam, A. Z. (2020). Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowledge-Based Systems*, 189, 105124.
- Betarte, G., Giménez, E., Martínez, R., Pardo, A. (2018). Improving web application firewalls through anomaly detection. In *17th IEEE Int. Conf. on Machine Learning and Applications*, pp. 779-784.
- Díaz Verdejo, J., et als. (2023). Labeled HTTP requests dataset: Dataset Biblio-US17. <https://idus.us.es/handle/11441/148254>
- Díaz-Verdejo, J.E., et als. (2020). A methodology for conducting efficient sanitization of HTTP training datasets. *Future Generation Computer Systems*, 109, 67-82.
- Díaz-Verdejo, et als. (2022). On the detection capabilities of signature-based intrusion detection systems in the context of web attacks. *Applied Sciences*, 12(2), 852.
- Hajj, S., El Sibai, R., Bou Abdo, J., Demerjian, J., Makhoul, A. and Gueyex, C. (2021). Anomaly-based intrusion detection systems: The requirements, methods, measurements, and datasets. *Trans. on Emerging Telecommunications Technologies*, 32(4), e4240.
- Hubballi, N. and Suryanarayanan, V. (2014). False alarm minimization techniques in signature-based intrusion detection systems: A survey. *Computer Communications*, 49, 1-17.
- Husák, M., Apruzzese, G., Yang, S. J. and Werner, G. (2021). Towards an efficient detection of pivoting activity. In *2021 IFIP/IEEE Int. Symposium on Integrated Network Management (IM)*, pp. 980-985.
- InspectorLog, available at [https://gitlab.com/neus\\_cslab/inspectorlog](https://gitlab.com/neus_cslab/inspectorlog) [Online; accessed 24-Jan-2024] (2024)
- Pałka, D. and Zachara, M. (2011). Learning web application firewall-benefits and caveats. In *IFIP WG 8.4/8.9 Int. Cross Domain Conf.*, ARES 2011, pp. 295-308.
- PHPIDS, available at <https://github.com/PHPIDS/PHPIDS> [Online; accessed 24-Jan-2024]
- Scap! available at <https://code.google.com/archive/p/apache-scalp/> [Online; accessed 24-Jan-2024]
- Tavallaee, M., Stakhanova, N. and Ghorbani, A. A. (2010). Toward credible evaluation of anomaly-based intrusion-detection methods. *IEEE Trans. on Systems, Man, and Cybernetics, Part C*, 40(5), 516-524.