IoTEdu Core: Multi-IDS Correlation and Automated Containment of Attacks in Institutional IoT Networks

Emanuel Ferreira[1], **Matheus Ciocca**[1], **Douglas Fideles**[1,2]
Tuigg Barcelos[1,2], **Silvio Quincozes**[1,2], **Diego Kreutz**[1,2]

[1]AI Horizon Labs, Federal University of Pampa (UNIPAMPA)

[2]PPGES, Federal University of Pampa (UNIPAMPA)

{emanuelferreira,matheusciocca,douglasfideles,
tuiggbarcelos}.aluno@unipampa.edu.br

{silvioquincozes,diegokreutz}@unipampa.edu.br

Abstract. *IoTEdu Core is an open-source tool for institutional IoT networks that integrates Suricata, Snort, and Zeek alerts, correlates them by device, and enforces configurable containment through pfSense. Across 240 labeled windows covering ten attack classes and six benign profiles, orchestration increases recall from 0.67 to 0.81, while k-of-n consensus eliminates all false alarms. However, no engine detects SSH brute force, highlighting that consensus cannot address shared blind spots. We also analyze cross-family misclassifications and release the rule set, raw data, and replication scripts.*

1. Introduction

An institutional network, whether on a campus, in a hospital, or in a utility, may operate thousands of IoT devices with diverse protocols and often limited protection [Li et al. 2023, Kim and Lee 2024, Rahman et al. 2023]. These devices are recurring targets and vectors for denial-of-service attacks and lateral movement within organizational networks [Alqahtani et al. 2023, Nguyen et al. 2024]. Worldwide, their number exceeds 21 billion and continues to grow by approximately 13% per year [IoT Analytics 2025].

Intrusion Detection Systems (IDSs) are therefore essential for monitoring IoT networks, but detection alone does not contain an attack. Table 1 summarizes related IDS research for IoT. Comparative studies [Waleed et al. 2022, Boukebous et al. 2023, Qutqut et al. 2026] evaluate throughput and latency across Snort, Suricata, and Zeek, but do not integrate their alerts or trigger containment actions. Integration platforms [Alharbi and Khan 2023, Muhammad et al. 2023] aggregate and visualize events without an explicit mitigation policy, while machine-learning extensions [Ouiazzane et al. 2022] improve detection but remain passive.

Some approaches perform automated blocking [Katsura et al. 2022, Praptodiyono et al. 2023], but typically rely on a single IDS and do not correlate alerts across sources. Mature SIEM and XDR platforms, such as *Wazuh*, *Security Onion*, and *OSSIM*, integrate multiple security engines and provide active-response mechanisms. However, an important question remains largely unanswered: what is the operational cost of automated containment? In particular, existing evaluations rarely measure how many *legitimate* devices would be blocked under the same traffic conditions. This is critical in institutional

networks, where an incorrectly blocked device may be a sensor, a medical device, or a student's experiment.

A second issue concerns how alerts are translated into containment decisions. In existing platforms, this logic is often embedded in the implementation, making it difficult to isolate and quantify the benefit of combining multiple IDS engines. IoTEdu Core addresses these two gaps by associating alerts with a *registered device and owner*, rather than solely with a log source, and by exposing containment as an explicit, configurable policy. This design enables the trade-off between detection coverage and wrongful blocking to be evaluated directly.

We present IoTEdu Core[1], an open-source pipeline for IDS alert ingestion, normalization, cross-source correlation, containment decisions, and enforcement through *pfSense*. The broader IoTEdu project's device-registry and web layers[2] provide identity and audit infrastructure but are not contributions of this work. Our contributions are threefold: *(i)* a versioned public rule catalog for Suricata, Snort, and Zeek covering ten attack classes (Section 3); *(ii)* a configurable containment policy that supports decisions ranging from "block on any alert" to k-of-n agreement (Section 2.2); and *(iii)* an evaluation of both detection capability and wrongful blocking, including per-engine metrics, cross-family confusion analysis, and publicly released raw data and replication scripts (Section 4). The complete implementation and a demonstration are publicly available.[3]

Table 1. IDS-based approaches in IoT environments. Composition: homogeneous (single IDS) or heterogeneous. Evidence reported: L = latency, D = detection coverage, FP = false alarms under benign traffic, M = cross-family misclassification; n/r = not reported.

Reference	IDS	Composition	Operation	Evidence
[Qutqut et al. 2026]	Snort, Suricata	Heterogeneous	Detection	L, D
[Katsura et al. 2022]	Snort 2.9.9	Homogeneous	Detection + Response	L
[Praptodiyono et al. 2023]	Suricata	Homogeneous	Detection + Response	L, D
[Boukebous et al. 2023]	Snort2, Snort3, Suricata	Heterogeneous	Detection	L, D
[Muhammad et al. 2023]	Zeek, Slips	Heterogeneous	Detection	D
[Waleed et al. 2022]	Snort, Suricata, Zeek	Heterogeneous	Detection + Response	L, D
[Ouiazzane et al. 2022]	Suricata	Homogeneous	Detection	D
[Alharbi and Khan 2023]	Suricata, Zeek, Slips	Heterogeneous	Detection	D
[Wazuh 2026, Security Onion 2026, OSSIM 2026]	Snort, Suricata, Zeek	Heterogeneous	Detection + Response	n/r
IoTEdu Core (this work)	**Snort, Suricata, Zeek**	**Heterogeneous**	**Detection + Response**	**D, FP, M**

2. IoTEdu Core: Architecture and Detection-Response Pipeline

IoTEdu Core transforms alerts from heterogeneous IDSs into device-level firewall blocks, recording which device was blocked, on what evidence, and under which policy. When a registered laboratory sensor triggers a high- or critical-severity alert, the system blocks it and notifies its owner; operators can review and reverse every block through the web interface. The detection layer integrates Suricata, Snort, and Zeek, while `pfSense` enforces the containment rules issued by the API.

[1] `https://github.com/GT-IoTEdu/iotedu-core_sbseg26`. Components: Suricata (`https://suricata.io/`), Snort (`https://www.snort.org/`), Zeek (`https://zeek.org/`) and `pfSense` (`https://www.pfsense.org/`).

[2] `https://gt-iotedu.github.io`

[3] `https://github.com/GT-IoTEdu/iotedu-core_sbseg26`
`https://youtu.be/zjoRR-140Cc`

2.1. Event pipeline

Figure 1 illustrates the processing of a single alert. The collector monitors `eve.json` (Suricata), `alert_fast` (Snort), and `notice.log` (Zeek), forwarding new records to the API through Server-Sent Events. The API maps each record to the common schema in Listing 1, making alerts from the three engines directly comparable through fields such as the source engine, 5-tuple, shared attack class, severity, and raw evidence.

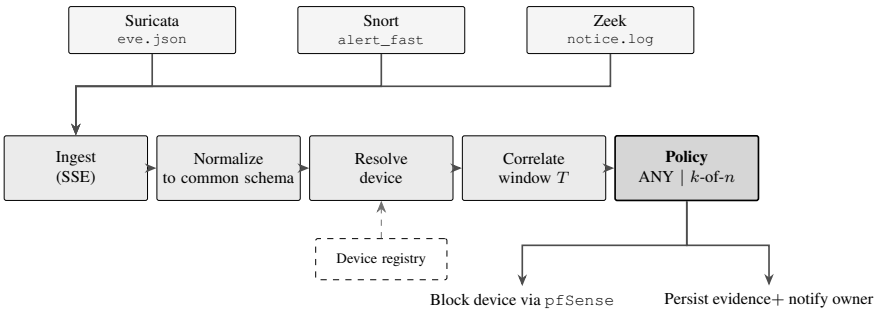

Figure 1. Detection, decision and enforcement pipeline of IoTEdu Core.

The API resolves the source IP address against the device registry, allowing events to be accumulated per *device* rather than per address, and maintains them in a sliding window of T seconds. Correlation is performed per device: the active policy from Section 2.2 is evaluated over normalized events associated with the same device within the last T seconds. Thus, *ANY* triggers on a single event with a severity of at least high, whereas k*-of-*n triggers when at least k distinct engines report the same `attack_class` for that device within the window.

The correlation horizon T is distinct from the per-rule aggregation window `window_s` in Listing 1, which is internal to an individual engine rule. As a deployment parameter, T must be at least as large as the widest aggregation window among the loaded rules, and it controls the trade-off between exposure time and response speed. The detection campaign does not exercise T: each labeled window runs a single isolated scenario with the containment path disabled, and the policy replay of Section 4.4 aggregates per attack class rather than within a window of T seconds, which is why its k-of-n coverage is reported as an upper bound. In a live deployment, when the active policy triggers, the API sends a blocking rule to `pfSense`, records the action and supporting evidence, and notifies the device owner.

```
{
  "source": "snort", // suricata | snort | zeek
  "timestamp": "2026-07-17T18:07:47.718Z",
  "src_ip": "178.12.186.219", // spoofed by the flood
  "dst_ip": "172.30.0.5",
  "device_id": null, // resolved in deployment only
  "signature_id": "1000068",
  "attack_class": "syn_flood", // shared taxonomy across engines
  "severity": "high", // low | medium | high | critical
  "evidence": {"syn_count": 252, "window_s": 12}
}
```

Listing 1. Normalized event produced by Snort for a SYN flood window of the detection campaign. The source address is forged by the attack.

2.2. Containment policies

The containment policy of IoTEdu Core is not hard-coded. The tool evaluates the policy over the sliding window of normalized events for each device, and the operator can change it without modifying the remaining pipeline. *ANY* blocks on the first alert with severity at least high, minimizing the exposure window but increasing susceptibility to false positives. k-*of*-n triggers only when at least k distinct engines report the same `attack_class` for the same device within T, trading exposure time for cross-engine agreement.

This policy is particularly meaningful because the engines are heterogeneous: a false positive specific to one engine's rule does not automatically result in a block. Moreover, the policy compares the attack class rather than merely the presence of an alert; therefore, two engines reporting different attack families do *not* satisfy the agreement condition. We set $k = 2$ as the minimum corroboration threshold above a single engine. With $n = 3$ heterogeneous engines, $k = 2$ is the smallest value that prevents a false positive from a single engine from triggering a block, whereas $k = 3$ requires unanimity and would exclude attack classes detected by fewer than three engines. Sections 4.3 and 4.4 quantify the trade-off associated with this stricter criterion using $k = 2$ and $n = 3$.

3. Detection Rules and Attack Coverage

We document, version, and publicly release the exact detection artifacts used by each engine. Table 2 presents five representative classes among the ten evaluated in Section 4. These examples cover the three detection regimes observed in our evaluation: classes detected only by the signature-based engines, classes detected only by Zeek, and classes detected by both. The remaining five classes, ICMP flood, UDP flood, DNS tunneling, cross-site scripting, and SSH brute force, follow the same conventions. The repository README provides the complete catalog, including rule bodies, Zeek scripts, and SHA-256 hashes for every file loaded by the pipeline.

Two aspects of the catalog are particularly relevant to the evaluation. First, Snort and Suricata use identical detection thresholds, ensuring that differences in their results reflect engine behavior rather than rule configuration. Second, Zeek uses a fundamentally different detection approach: instead of signature matching, its event-driven scripts maintain state and derive decisions from network-level observations. This independence makes k-of-n agreement meaningful, as agreement across engines reflects complementary detection mechanisms rather than repeated application of essentially the same test.

Table 2. Representative entries of the rule catalog: detection artifact per engine and severity. `SumStats` is Zeek's aggregation framework; `sfportscan` is Snort's port-scan preprocessor.

Attack	Indicator	Snort / Suricata	Zeek	Severity
SYN Flood	SYN rate per target	`flags:S` + `threshold` (100/5 s)	`SumStats` over `connection_attempt`	high
DoS HTTP	request rate	`flow:to_server` + `threshold` (200/10 s)	`SumStats` over `http_request`	high
SQL Injection	URI pattern	`http.uri`: `union`+`select`, `nocase`	`http_request` + URI normalization	critical
Port Scan	distinct ports per src	`sfportscan` / `threshold` on SYN	`PortScan` policy script	medium
Path Traversal	`../` sequences	`http.uri`: `..%2f`, `../`	`http_request` + path canonicalization	critical

4. Evaluation

The evaluation answers two questions. *Q1:* with the catalog of Section 3, what does each engine detect, and how often does it fire when nothing malicious is happening? *Q2:* what

does correlating three engines buy, in coverage and in wrongful blocks, compared with acting on a single engine? Both are answered from one detection-effectiveness campaign.

4.1. Testbed and workload

IoTEdu Core is deployed on a virtualized testbed, using the configuration marked [D] in Table 4, which corresponds to the demonstration setup described in Section 5. The detection campaign uses a complementary configuration designed to stress the rules in isolation. Malicious and benign traffic run inside a dedicated Docker network, while the three engines monitor its bridge in host mode. The targets expose only the services required by each scenario: HTTP at `172.30.0.5:80` and SSH at `172.30.0.6:22`. No blocking application is deployed, ensuring that containment does not interfere with the measurements.

The unit of analysis is a *labeled window*, defined as one isolated execution of a single scenario. Malicious and benign traffic are evaluated in separate windows rather than being superimposed, allowing each engine's false-alarm rate to be attributed directly to the corresponding benign profile. Each window is labeled per engine. On malicious traffic: a true positive (at least one alert of the correct family), a *cross-family misclassification* (an alert only for another family), or a false negative (no attack alert). On benign traffic: a false positive (any attack alert) or a true negative. From these labels, we compute precision, recall, specificity, F1, false-positive rate, and balanced accuracy, defined as the mean of recall and specificity. Repeated windows are aggregated using 95% Wilson confidence intervals. Table 3 summarizes the resulting capture.

Table 3. Detection campaign capture: 240 labeled windows, 6.88 h of traffic and roughly 10.3 million packets, with 15 windows per scenario. Blocks separated by the dashed rule are independent lists: rows are not paired.

Malicious (10): 150 windows				Benign (6): 90 windows			
Scenario	**Family**	**Windows**	**Packets**	**Profile**	**Traffic**	**Windows**	**Packets**
icmp-flood	flood-icmp	15	6,337,885	benign-mix	mixed	15	3,600
syn-flood	flood-syn	15	15,225	benign-http	HTTP	15	11,936
udp-flood	flood-udp	15	3,608,876	benign-ssh	SSH	15	859
dos-http-simple	dos-http	15	30,075	benign-dns	DNS	15	45
sql-injection	sqli	15	163,196	benign-icmp	ICMP	15	2,559
xss-scanner	xss	15	13,712	benign-scan	scan	15	2,633
idor-path-traversal	path-traversal	15	45,691				
port-scanner-tcp	scan	15	30,172				
ssh-bruteforce	ssh-brute	15	15,636				
dns-tunneling	dns-tunnel	15	6,150				

4.2. Q1: Attack coverage and false-alarm cost

Table 5 reports, for each engine and for the orchestrated system, attack coverage, standard detection metrics, and the frequency of alerts on benign traffic. The last column translates false alarms into an operational measure: the number of legitimate devices that *ANY* would incorrectly block.

Three findings stand out. First, no individual engine provides sufficient coverage: Suricata achieves the highest recall at 0.67, while orchestration reaches 0.81. Figure 2 explains this gain. Snort and Suricata detect the volumetric floods, with Suricata also covering HTTP-based attacks and DNS tunneling. Both miss TCP port scanning and path traversal, which are detected only by Zeek; Zeek also independently detects SQL

Table 4. Testbed parameters. Rows marked [D] belong to the demonstration testbed only; the detection campaign of Section 4 uses the remaining rows.

Component	Specification
Host[D]	Notebook, VirtualBox 7.1.12, Windows 11 Pro, Intel Core i7-1185G7, 32 GB RAM
Attacker VM[D]	Linux Mint 22.3 (Ubuntu 24.04 base), 1 vCPU, 2 GB RAM
Monitoring VM[D]	Ubuntu 24.04 x86_64, 1 vCPU, 2 GB RAM
Firewall VM[D]	pfSense 2.8.1-RELEASE (FreeBSD 15.0-CURRENT), 4 vCPUs, 8 GB RAM
Evaluated IDSs	Suricata 8.0, Snort 3.9.7, Zeek 8.1, running simultaneously with the catalog of Section 3
Capture point	Bridge of a dedicated Docker network (`172.30.0.0/24`); engines in host mode
Containment policies	ANY; k-of-n with k=2, n=3

injection. Second, only Zeek generates alerts on benign traffic, doing so in 8 of the 90 benign windows across the `dns`, `http`, `icmp`, `mix`, and `scan` profiles. Under *ANY*, these eight alerts become wrongful blocks. Because all false alarms originate from a single engine, requiring agreement from a second independent engine eliminates them entirely. Third, all three engines miss SSH brute force, making it the only attack class that orchestration cannot recover. This is a rule-coverage limitation rather than a failure of the orchestration mechanism.

Table 5. Detection quality and false-alarm cost over the detection campaign (family-correct detection, 240 windows, 95% Wilson aggregation). *Detected* = attack classes with at least one correct detection, of ten; *FA/h* = false alarms per hour on benign traffic; *Wrongful* = benign windows blocked under ANY.

Engine	Detected	Recall	Precision	F1	Bal. Acc.	FA/h	Wrongful (ANY)
Snort	6/10	0.460	1.000	0.630	0.730	0.0	0
Suricata	7/10	0.673	1.000	0.805	0.837	0.0	0
Zeek	7/10	0.400	0.882	0.550	0.656	2.6	8
Orchestration	**9/10**	**0.813**	**0.938**	**0.871**	**0.862**	2.6	8

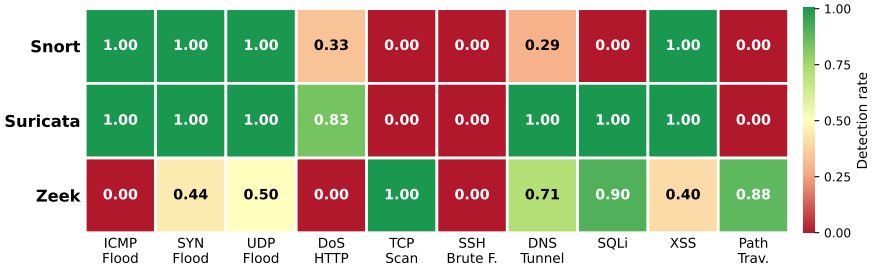

Figure 2. Per-class detection rate by engine over the detection campaign (green = high, red = low).

4.3. Cross-family misclassification

Detecting an attack and identifying its correct class are different tasks, and coverage alone does not capture this distinction. Across the 450 engine-window observations for malicious traffic, 57 contain alerts assigned exclusively to the signature of a different attack family. These cases are neither misses nor false positives, but *cross-family misclassifications*. Table 6 aggregates these outcomes, with three patterns accounting for most of them.

Snort classifies SQL injection as cross-site scripting in 15 windows, as both attacks traverse HTTP parameters and share related tokens. Snort and Suricata classify path traversal as a generic HTTP denial-of-service attack in 30 windows, triggering an HTTP anomaly rule instead of the traversal signature. Zeek classifies some floods, as well as its single SSH brute-force alert, as behavioral *scan* activity.

These results explain why the k-of-n policy matches `attack_class` rather than merely the presence of an alert. Agreement that an attack occurred is insufficient for corroborated containment; the engines must also agree on what was detected. Because containment is applied to the *resolved device*, rather than to the attack class, a misclassification does not cause the wrong device to be blocked. Under *ANY*, the source device is still contained, while the recorded evidence and owner notification carry the incorrect class. Under k-of-n, however, two detections with different classes do not form a match, potentially preventing a legitimate block. This is the same effect reflected in the 7/10 result of Section 4.4, which therefore represents an upper bound.

Table 6. Confusion matrix over the malicious windows of the detection campaign (intended family in rows, family actually signaled in columns; counts aggregated over the three engines, 450 observations). *None* = no attack alert. Diagonal in bold; SSH brute force has no diagonal because its signature never fired.

Intended \ Signaled	None	DNS	DoS	ICMP	SYN	UDP	Path	Scan	SQLi	XSS
DNS tunneling	16	**29**	0	0	0	0	0	0	0	0
DoS HTTP	23	0	**16**	1	4	0	0	1	0	0
ICMP flood	14	0	0	**30**	0	0	0	1	0	0
SYN flood	6	0	0	0	**35**	0	0	4	0	0
UDP flood	8	0	0	0	0	**37**	0	0	0	0
Path traversal	5	0	30	0	0	0	**10**	0	0	0
Port scan	34	0	0	0	0	0	0	**11**	0	0
SQL injection	4	0	0	0	0	0	0	0	**26**	15
SSH brute force	44	0	0	0	0	0	0	1	0	0
XSS	9	0	0	0	0	0	0	0	0	**36**

4.4. Q2: What the containment policy buys

Table 7 keeps the traffic fixed and varies only the containment policy, replaying the labeled detection outcomes under each decision rule. This makes the comparison directly attributable to the policy. A single engine leaves clear coverage gaps, as shown in Table 5: Snort and Suricata miss the classes detected only by Zeek, while Zeek misses the volumetric floods and generates all false alarms. *ANY* contains nine of the ten attack classes but also inherits Zeek's eight false alarms, resulting in eight wrongful blocks. Requiring agreement from two engines eliminates all eight false alarms while retaining automatic containment for seven classes, at the cost of losing port scanning and path traversal. The choice therefore depends on operational risk. Networks where an incorrect block could disconnect a laboratory sensor should prefer $k{=}2$ and leave the uncovered classes for analyst review. Networks that can tolerate a reversible block may prefer *ANY* to maximize automatic coverage.

Two caveats bound these results. In the replay, a class is considered contained under k-of-n when at least two engines produce a family-correct detection. Because this aggregation is performed per class and does not require both detections to occur within the same window T, the reported 7/10 is an *upper bound*. The wrongful-block result is exact in the opposite direction: no benign window triggered alerts from two engines. Therefore,

regardless of the choice of T, none of these benign windows would satisfy $k{=}2$, and the zero wrongful-block count remains unchanged.

Table 7. Containment policy trade-off, obtained by replaying the labeled outcomes of the detection campaign through each decision rule. *Attacks contained* counts attack classes, of ten, for which the policy would fire; *wrongful blocks* counts benign windows, of 90, that the policy would block.

Policy	Engines required	Attacks contained	Wrongful blocks
ANY (first alert from any engine)	1	**9/10**	8
k-of-n ($k = 2, n = 3$)	2	7/10	**0**

5. Availability, Reproducibility and Demonstration Plan

The repository listed below the title provides the complete source code and reproducibility artifacts, including the rule catalog with SHA-256 hashes, `pfSense` configuration, raw alerts, per-window results, and scripts to regenerate all tables and figures. The campaign can be reproduced without the firewall VM using the three documented steps: start the services with `docker compose`, run labeled windows with `run_window.sh`, and replay the logs with `replay_policies.py`.

The demonstration shows the complete containment workflow, from alert reception and policy evaluation to `pfSense` enforcement, notification, and block reversal, followed by a replay under both policies. It lasts approximately 10 minutes and requires only a projector or HDMI connection and a power outlet. All components run locally on a single notebook without network connectivity. A recording is available at the video URL listed below the title.

6. Conclusion, Limitations, and Future Work

IoTEdu Core closes the loop between heterogeneous IDS alerts and per-device containment in institutional IoT networks. Its contribution is to make this trade-off measurable through a public rule catalog, a swappable containment policy, and evaluation of both detection and wrongful blocking. Across the evaluated scenarios, *ANY* provides broader coverage but causes eight wrongful blocks, whereas $k{=}2$ eliminates them while retaining seven attack classes.

The results are bounded by isolated scenarios, lightweight emulated services, and the absence of concurrent benign load, production firmware, response latency, exposure time, ingestion flooding, scalability, orchestrator hardening, and distributed attacks. IP-based containment can be spoofed or changed through DHCP, and k-of-n reduces but does not eliminate this risk. Low-and-slow attacks are not evaluated; all engines miss SSH brute force, 57 malicious observations are cross-family misclassifications, and no zero-day coverage is claimed. Future work will evaluate production-like workloads, latency, scalability, and adversarial resilience, while extending rule and protocol coverage and exploring stronger device identity and adaptive containment policies.

Acknowledgments: This work was supported by the Hackers do Bem Program, executed by RNP and SENAI, coordinated by Softex, and funded by MCTI through the ICT Law (Law No. 8,248/1991), with additional support from CAPES (Código de Financiamento 001), CNPq (Grant No. 409743/2025-9), and FAPERGS (Grant Nos. 24/2551-0001368-

7 and 24/2551-0000726-1). The authors also acknowledge the use of Generative AI to support code and manuscript review and improvement.

References

Alharbi, S. and Khan, A. (2023). Ensemble defense system: A hybrid ids approach for effective cyber threat detection. In *2023 33rd International Telecommunication Networks and Applications Conference*, pages 267–270.

Alqahtani, S. et al. (2023). Cybersecurity threats in iot environments: Recent advances and challenges. *Journal of Network and Computer Applications*.

Boukebous, A. A. E., Fettache, M. I., Bendiab, G., and Shiaeles, S. (2023). A comparative analysis of snort 3 and suricata. In *2023 IEEE IAS Global Conference on Emerging Technologies (GlobConET)*, pages 1–6.

IoT Analytics (2025). State of iot 2025: Number of connected iot devices growing 13% to 21.1 billion globally. Accessed: 2026.

Katsura, Y., Sakarin, P., Yamai, N., Kimiyama, H., and Visoottiviseth, V. (2022). Quick blocking operation of firewall system cooperating with ids and sdn. In *2022 24th ICACT*, pages 393–398.

Kim, J. and Lee, H. (2024). Recent advances in iot architectures and applications: A comprehensive survey. *Future Generation Computer Systems*.

Li, S., Xu, L. D., and Zhao, S. (2023). Security challenges and opportunities in internet of things: A survey. *IEEE Internet of Things Journal*.

Muhammad, A. R., Sukarno, P., and Wardana, A. A. (2023). Integrated security information and event management (siem) with intrusion detection system (ids) for live analysis based on machine learning. *Procedia Computer Science*, 217:1406–1415. ISM 2022.

Nguyen, T. et al. (2024). Security and privacy challenges in iot-based smart environments. *Computer Networks*.

OSSIM (2026). OSSIM: AlienVault's open source SIEM. levelblue.

Ouiazzane, S., Addou, M., and Barramou, F. (2022). A suricata and machine learning based hybrid network intrusion detection system. In Maleh, Y., Alazab, M., Gherabi, N., Tawalbeh, L., and Abd El-Latif, A. A., editors, *Advances in Information, Communication and Cybersecurity*, pages 474–485, Cham. Springer International Publishing.

Praptodiyono, S., Firmansyah, T., Anwar, M. H., Wicaksana, C. A., Pramudyo, A. S., and Al-Allawee, A. (2023). Development of hybrid intrusion detection system based on suricata with pfsense method for high reduction of ddos attacks on ipv6 networks. *Eastern-European Journal of Enterprise Technologies*, 5(9 (125)):75–84.

Qutqut, M. H., Ahmed, A., Taqi, M. K., Abimanyu, J., Ajes, E. T., and Alhaj, F. (2026). A comparative evaluation of snort and suricata for detecting data exfiltration tunnels in cloud environments. *Journal of Cybersecurity and Privacy*, 6(1).

Rahman, M. et al. (2023). Edge computing for iot: Security and scalability challenges. *IEEE Communications Surveys & Tutorials*.

Security Onion (2026). Security onion: A free and open platform for threat hunting, network security monitoring, and log management. security onion solutions, llc.

Waleed, A., Jamali, A. F., and Masood, A. (2022). Which open-source ids? snort, suricata or zeek. *Computer Networks*, 213:109116.

Wazuh (2026). Wazuh: The open source security platform — unified XDR and SIEM protection for endpoints and cloud workloads.

A. Supplementary Results of the Detection Campaign

The per-window and per-scenario detail behind the aggregated tables of Section 4, regenerated by the released scripts from the same 240 labeled windows.

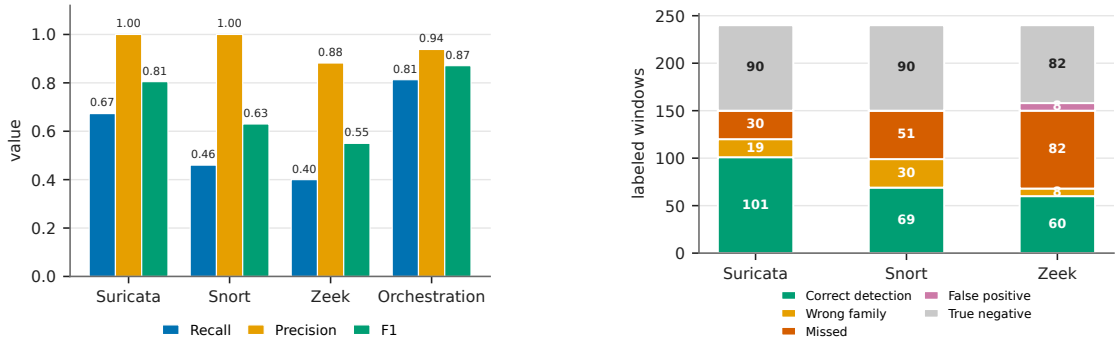

Figure 3. Per-engine summary. *Left:* family-correct detection metrics (Table 5). *Right:* window labels per engine; *wrong family* is neither a miss nor a false positive, and every false positive belongs to Zeek.

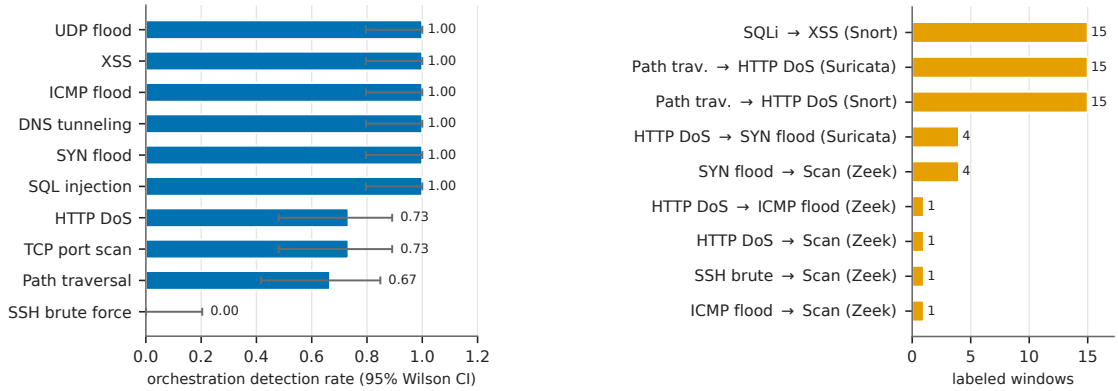

Figure 4. *Left:* orchestration detection rate per attack scenario ($n = 15$ windows each), 95% Wilson intervals. *Right:* the 57 cross-family misclassifications of Section 4.3, by pair and engine.

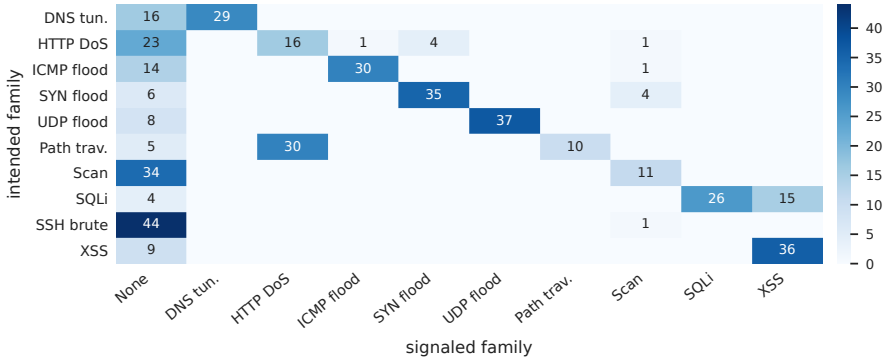

Figure 5. Confusion matrix over the malicious windows, aggregated over the three engines (450 observations); graphical form of Table 6.