Journal Homepage: -www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI:10.21474/IJAR01/24558
DOI URL: http://dx.doi.org/10.21474/IJAR01/24558

RESEARCH ARTICLE

VULNERABILITY OF DEEP NETWORK INTRUSION DETECTORS TO EVASION VARIES WITH THE TRAINING SEED

Ibrahima Toure, Moustapha Kaba, Boris Sourou Viwahoué Jerson Zannou, Amara Camara and Binko Mamady Toure

1. Department of Computer Engineering, Mamou Institute of Technology, Mamou, P.O. Box 63, Republic of Guinea.

Manuscript Info

Manuscript History
Received: 10 September 2026
Final Accepted: 12 October 2026
Published: November 2026

Key words:-
Intrusion detection, deep learning, adversarial evasion, decision-based attack, temporal evaluation, underspecification.

Abstract

Evaluations of deep network intrusion detectors usually rest on one trained model, a random train–test split and an attacker that learns nothing from one flow to the next. Method. We drop these three habits on the corrected CIC-IDS-2017: a per-class chronological split, a decision-based attacker that may only pad, delay or add packets to its own flows (at most 1,000 label queries per flow, two budgets), and many training runs of the same detector attacked on the same flows. Findings. A chronological split lowers the macro F1 of a 1D CNN by only 0.042 but raises its false-positive rate from 0.003% to about 2%; about 85% of these false alarms are flows that the dataset correction labels Attempted. Nine CNNs that differ only in their seed reach a test macro F1 of 0.87 to 0.95, yet the same attack gets through 0% to 99% of a family's flows depending on the detector, which explains 96% to 100% of the sum of squares across 15 attack conditions. Thirty-three MLPs behave alike, a budget five to ten times smaller keeps the ranking of the detectors, the ranking carries over to CSE-CIC-IDS-2018 flows, and random forests and gradient-boosted trees show the same seed effect. The closed-form feature updates of the attacker agree with an independent CICFlowMeter implementation to within 0.5%. Detection metrics do not predict this vulnerability, whereas attacking validation flows does (r_S = 0.60 to 1.00). Contribution. Vulnerability should be measured over several training runs and used to choose the deployed model.

"© 2026 by the Author(s).Published by IJAR under CC BY 4.0. Unrestricted use allowed with credit to the author."

Introduction:-

On CIC-IDS-2017, flow-based deep detectors often exceed an F1 of 0.98 [1], [2]. These scores rest on three habits. The split mixes past and future traffic, which inflates results [3], [4], and the false-positive rate, which sets the analysts' workload, is seldom given [5], [6]. Evasion is studied with access to the model, or with changes an attacker could not make to real traffic [7], [8]. Finally, one trained model stands for the whole method, although a training pipeline can produce models that score alike on held-out data and differ elsewhere [9], [10].We drop the three habits at once. Our main finding is that detectors trained the same way, with only the seed changed, differ almost entirely

Corresponding Author:-Ibrahima Toure
Address:-Department of Computer Engineering, Mamou Institute of Technology, Mamou, P.O. Box 63, Republic of Guinea.

in how easily the same attack evades them; their detection scores do not tell them apart, but an attack on validation flows does. We also locate the false alarms that a chronological split reveals and state which flow statistics the attacker's actions change. Code, models, flow sets and per-flow results are public.

Contributions:-

Most evasion studies attack a single trained model, often with gradient access or with feature changes no attacker could make [7], [8], [16], [25]. We depart from that in four ways: (i) it shows that, under a fixed and traffic-constrained black-box attack, the training seed of a detector decides most of its vulnerability, for CNNs, MLPs, random forests and gradient-boosted trees; (ii) it shows that detection metrics do not reveal this, but an attack on validation flows does; (iii) it traces the false alarms of a chronological evaluation to Attempted flows and reports usable operating points; (iv) it checks the attacker's feature updates against a flow extractor and reports the cost of the evaluation.

Related work:-

CIC-IDS-2017 and CSE-CIC-IDS-2018 were relabelled and their flows rebuilt by Engelen et al. [11] and Liu et al. [12]; Lanvin et al. [13] and Flood et al. [14] measured how dataset defects bias results. Pierazzi et al. [7] separate feature-space perturbations from attacks that can be carried out on real objects, and Apruzzese et al. describe realistic NIDS adversaries [8] and evasion under concept drift [15]. Label-only attacks such as HopSkipJumpAttack [16] spend many queries on each input; statefuldefences watch for such query streams [17], and adaptive attackers get around them [18]. Underspecification [9] predicts that equally accurate models differ off the validation distribution; we test it for evasion, keeping the attack identical across models.

Generic decision-based attacks, such as the Boundary attack [25] and HopSkipJumpAttack [16], work on unconstrained continuous inputs and usually need thousands of queries per input. Traffic morphing [26] showed long ago that packet sizes and timing can be reshaped to fool statistical traffic classifiers. Our attacker is restricted to traffic-preserving actions and capped at 1,000 queries per flow, and we compare it with blind and uniform random searches in Section 6.4. Tree ensembles can also be evaded with small, exactly computed perturbations [23], and robust tree training has been proposed [24]; both study one model at a time, not how robustness varies from one training seed to the next.

Setting:-

Flows and chronological split:-

CICFlowMeter groups the packets of a connection into a flow and summarises the sizes, gaps and flags of its packets by counts, sums, extremes, means, deviations and rates; after cleaning, a flow is a vector of 82 numbers. With m forward packets of sizes s_j, m' backward packets and $K = m + m'$ packets,

$$D = t_K - t_1, \qquad B_{fwd} = \sum_{j=1}^{m} s_j, \qquad B_{bwd} = \sum_{j=1}^{m'} s'_j \tag{1}$$

$$\text{Flow Bytes/s} = \frac{10^6\,(B_{fwd} + B_{bwd})}{D}, \qquad \text{Flow Packets/s} = \frac{10^6\,K}{D} \tag{2}$$

where the duration D is in microseconds, as in the dataset. Every feature goes through a signed logarithm and a min-max scaling fitted on training data. Each attack family of CIC-IDS-2017 was captured on a single day, so the split is made inside each class c: oldest 70% for training, next 15% for validation, last 15% for test,

$$\max_{x \in \text{Train}(c)} t(x) \;\le\; \min_{x \in \text{Val}(c)} t(x), \qquad \max_{x \in \text{Val}(c)} t(x) \;\le\; \min_{x \in \text{Test}(c)} t(x) \qquad \forall c \tag{3}$$

This tests later traffic of known classes; it does not simulate a single deployment date.

New attacks are a harder test, so we also use a global chronological split: all flows are ordered by time and cut at the 70% and 85% quantiles. Training then contains BruteForce, DoS and Infiltration, validation adds Bot and PortScan, and the test set (Friday afternoon) contains only DDoS and PortScan attacks, which the detector has never seen in training. A multiclass score means nothing there; what matters is whether an attack flow raises any alert at all (Section 6.8).

Threat model and actions:-

The attacker owns the source of a malicious flow that the detector flags and learns only the predicted label of each query. It cannot touch the server's answers and may only add to its forward traffic, through an action $a = (b, \delta, \kappa)$: b padding bytes per forward packet, a delay δ between forward packets, and κ dummy packets per forward packet.

Actions are sampled in $[0, 1]^3$ and squared, so small changes are tried more finely:-

$$a(u) = \left(u_1^2\, \bar{b}(\beta),\ u_2^2\, \bar{\delta}(\beta),\ u_3^2\, \bar{\kappa}(\beta)\right), \qquad \mathcal{A}(\beta) = \left[0, \bar{b}(\beta)\right] \times \left[0, \bar{\delta}(\beta)\right] \times \left[0, \bar{\kappa}(\beta)\right] \qquad (4)$$

Table 1.Upper limits of the three actions, per forward packet, at each budget.

Budget β	**Padding** $\bar{b}$	**Delay** $\bar{\delta}$	**Dummy-packet ratio** $\bar{\kappa}$
Low	100 bytes	10 ms	0.25
Medium	500 bytes	100 ms	1.0
High	1,000 bytes	1 s	2.0

A flow counts as evaded when it is classified benign within $L = 1{,}000$ queries. We modify the feature vector, not packets: every original forward packet is padded, the delay precedes each original forward packet but the first, and dummy packets carrying $\bar{s} + p$ bytes fall inside existing forward gaps. With $P_{max} = 1{,}460$ bytes,

	$p = \max\left(0,\ \min(b,\ P_{max} - s_{max})\right), \qquad k = \text{round}(\kappa m), \qquad B'_{fwd} = B_{fwd} + m\,p + k\,(\bar{s} + p)$	(5)
	$m^{\star} = m + k, \qquad D' = D + (m - 1)\,\delta, \qquad \bar{s}' = \bar{s} + p, \qquad \text{Down/Up}' = \dfrac{m'}{m + k}$	(6)

These conventions give closed forms for 22 features, which we recompute from the stored raw flow; on 2,000 synthetic packet traces they match the statistics of the modified traces to a relative error below 10^{-5}. Six position-dependent features follow rules of thumb, 28 that real traffic would change stay as they are, and 26 are set by the server or the protocol. This checks our algebra, not CICFlowMeter itself, so we speak of feature-space evasion under traffic constraints.

Why these budgets and this query limit. The low budget keeps every padded packet far below the 1,460-byte payload limit of a 1,500-byte MTU, adds at most 10 ms per packetof the order of ordinary Internet jitterand at most one dummy packet for every four real ones. The medium budget, five to ten times larger, still keeps a padded packet within one MTU and delays within common TCP retransmission timeouts (at least 200 ms in Linux), so a delayed connection is not torn down. The high budget of Table 1 is given for reference only and is not used in the results, because delays of one second per packet would change the nature of attacks such as DoS. The cap of 1,000 queries is well below what generic decision-based attacks use [16], [25], because each query corresponds to replaying a modified connection; in practice the cap rarely bites, because most successes come from replaying an action that already worked (Section 6.11).Section 6.10 checks these closed forms against an independent CICFlowMeter implementation on modified packet traces, and measures how much the features that we keep unchanged would really move.

Attacker and protocol:-

The adaptive attacker keeps a memory $\mathcal{M}$ of up to 20 actions that worked on earlier flows of the campaign (Algorithm 1). On a new flow it first replays them, then spends half of what is left on Gaussian perturbations of stored actions, and the rest on random draws whose scale grows with the query index. Once an action works, six bisection steps on its amplitude bring it close to the edge of the evading region.

Algorithm 1. Adaptive attacker on one flow

1: candidates ← stored actions, then their mutations, then escalating random draws, L in total
2: try the candidates in order; if none is classified benign, **return** failure
3: shrink the winning action u by six bisection steps on $t \in [0,1]$, keeping the smallest t that still evades
4: put $\min(\mathbf{1}, 1.25\,t\,u)$ first in $\mathcal{M}$, keep 20 entries, **return** success

A campaign attacks one family's flows in a set order, starting with an empty memory. To compare detectors fairly we fix four things. The flows: 100 test flows per family (21 for Bot), drawn once for all detectors. The success criterion: only a benign decision is an evasion, so every flow that raises an alert is attacked, and flows already judged benign count as evaded at no cost. The randomness: flow i gets its own generator, seeded by attack seed, attacker, family and

i, so all detectors see the same random numbers on the same flow. The order: chronological, or one of two fixed shuffles. Five attack seeds and three orders make 15 attack conditions. Before each campaign we check that the null action leaves every decision unchanged.

Terminology. Alert: a decision other than BENIGN. Evasion rate: share of the alerted flows of a family that the attacker turns into a benign decision within L queries. Attack condition: one attack seed and one flow order at one budget. Training seed: the seed passed to the training procedure, which fixes all of its randomness. False-positive rate (FPR): share of benign test flows that raise an alert. We use them in this sense throughout.

Experiments:-
We use the corrected CIC-IDS-2017 [11], [12] with Attempted flows labelled benign, and keep 20% of each day (419,976 flows, 7 classes, 76.0% benign). The CNN stacks convolutions with 64, 128 and 256 filters, a global average and two dense layers (157,319 weights); the MLP has hidden layers of 256, 128 and 64 units (62,855 weights). Training uses Adam with a learning rate of 10^{-3} and batches of 512, and keeps the epoch with the best validation macro F1. Two models of the same kind differ only in their seed, which sets the initial weights, the batch order and the dropout masks. We attack nine CNNs (seeds 1–9), 33 MLPs (seeds 0–32) and three CNNs without the destination port, in 3,870 campaigns (Fig. 1).

Preprocessing : From each of the five capture days we keep a 20% sample of the flows, drawn uniformly at random with a fixed seed (419,995 flows). We remove 2 rows with infinite or missing values, one constant column, and the two classes too rare to split in time (WebAttack, 15 flows; Heartbleed, 2 flows). The 2,452 Attempted flows are assigned to the benign class (see Section 6.7 for the consequence). The labels are mapped to seven classes: BENIGN (318,996 flows), DoS (34,230), PortScan (31,956), DDoS (18,992), Infiltration (14,338), BruteForce (1,331) and Bot (135). Each feature x becomes $\text{sign}(x)\cdot\log(1+|x|)$ and is then min–max scaled with bounds fitted on the training set.

Architectures and training : The CNN treats the 82 features as a signal of length 82 with one channel: three convolutional layers with 64, 128 and 256 filters of width 3 (ReLU, zero padding), a max-pooling layer halving the length after the second, a global average over positions, a dense layer of 128 units with ReLU and dropout 0.3, and a 7-unit softmax output. The MLP has hidden layers of 256, 128 and 64 units with ReLU, dropout 0.3 after the first two, and a 7-unit softmax output. Both minimise the unweighted cross-entropy; training stops after 10 epochs without improvement of the validation macro F1 (minimum 10, maximum 60 epochs) and keeps the best epoch. The optimiser settings, batch size and architectures were fixed in advance from common practice and were not tuned; the only choice made on preliminary experiments was to leave the loss unweighted, because class weighting raised the false-positive rate. The seed fixes the initial weights, the batch order and the dropout masks.

Tree ensembles : To test whether the seed effect is specific to neural networks (Section 6.6), we also train random forests (RF) [20] with 100 trees, maximum depth 20, $\sqrt{82}$ candidate features per split and bootstrap sampling (seeds 0–4), and histogram gradient boosting (HGB) [21], [22] with learning rate 0.1, at most 300 iterations, 31 leaves, 255 bins and early stopping on a 10% hold-out of the training set (seeds 0–2), with scikit-learn 1.6.1. In RF the seed selects the bootstrap samples and candidate features; in HGB, which uses all features, it only selects which 10% of the training flows are held out for early stopping. We attack them on the same kind of test flows, with three attack seeds per budget.

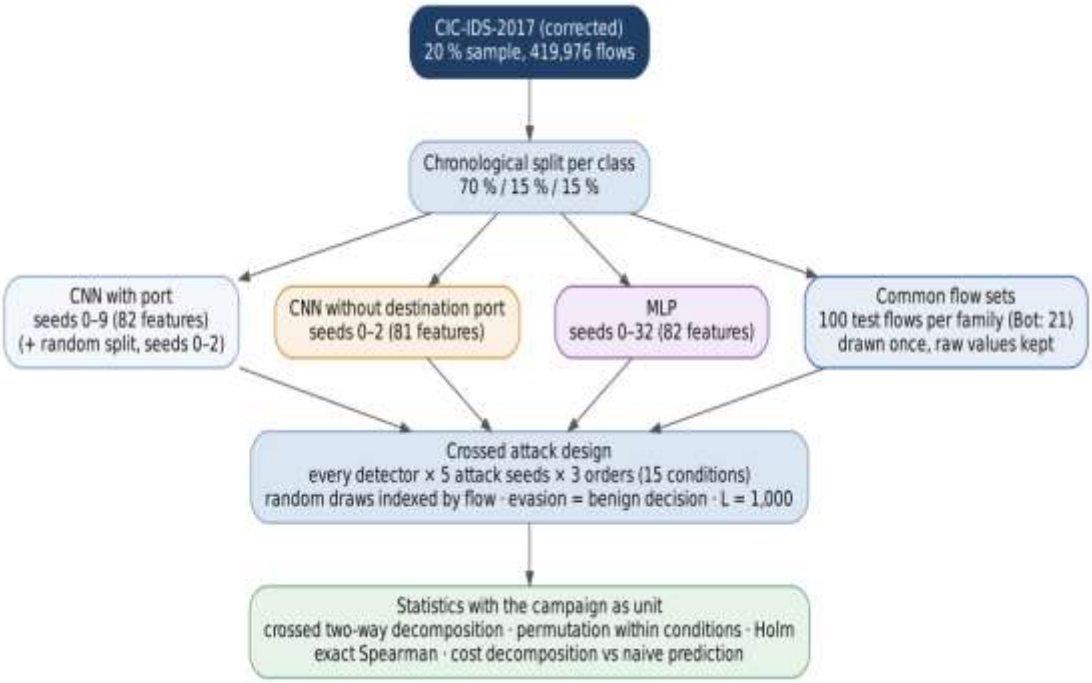


Fig. 1. Overview of the design: one sample, one chronological split, one set of attack flows shared by all detectors.

Because flows within an adaptive campaign depend on each other, the unit of analysis is the campaign. For one family, the evasion rate E_{kj} of detector k in condition j is modelled as

$$E_{kj} = \mu + \alpha_k + \gamma_j + \varepsilon_{kj}, \qquad \alpha_k \sim (0, \sigma_D^2), \quad \gamma_j \sim (0, \sigma_C^2), \quad \varepsilon_{kj} \sim (0, \sigma_E^2) \tag{7}$$

and the detector's share of variance is

$$\hat{\sigma}_E^2 = MS_E, \quad \hat{\sigma}_D^2 = \max\left(0, \frac{MS_D - MS_E}{J}\right), \quad \hat{\sigma}_C^2 = \max\left(0, \frac{MS_C - MS_E}{K}\right) \tag{8}$$

$$\hat{\eta}_D = \frac{\hat{\sigma}_D^2}{\hat{\sigma}_D^2 + \hat{\sigma}_C^2 + \hat{\sigma}_E^2} \tag{9}$$

We report it with a bootstrap over detectors and conditions, and test whether detectors differ by shuffling their labels inside each condition (10,000 shuffles). Links between detection metrics and evasion use Spearman's r_S with permutation p-values, and the tests behind one claim are corrected with Holm's method [19].

Reading the statistics. A detector share $\hat{\eta}_D$ close to 1 means that knowing which detector is attacked tells almost everything about its evasion rate, while repeating the attack with other random draws or another flow order changes almost nothing. The model assumes additive effects and needs no normality for the point estimate. Because a family has only 100 test flows (21 for Bot), we also give, for each detector, a Wilson 95% interval of its evasion rate pooled over the attack conditions [28]: two detectors whose intervals do not overlap differ beyond the sampling noise of the flows.

Results:-

Detection and false positives:-

Moving the CNN from a random to a chronological split lowers the macro F1 from 0.982 to 0.940 and raises the false-positive rate (FPR) from 0.003% to 2.01%. All 16 originally trained detectors have an FPR between 1.92% and 2.17%. On the last capture day, 832 benign test flows go to port 8080; every detector calls 95% of them PortScan, and they make up 77% to 85% of all false alarms (Fig. 2). Without them the FPR drops to 0.31–0.52%. This is a concentration of errors on one service, not a proven cause, and allow-listing the service would also let attacks through it.

Section 6.7 shows what these flows are: 831 of the 832 are flows that the dataset correction labels Attemptedattack attempts that failedand that we had folded into the benign class. The concentration of false alarms is therefore mainly a labelling convention rather than a property of a benign service.

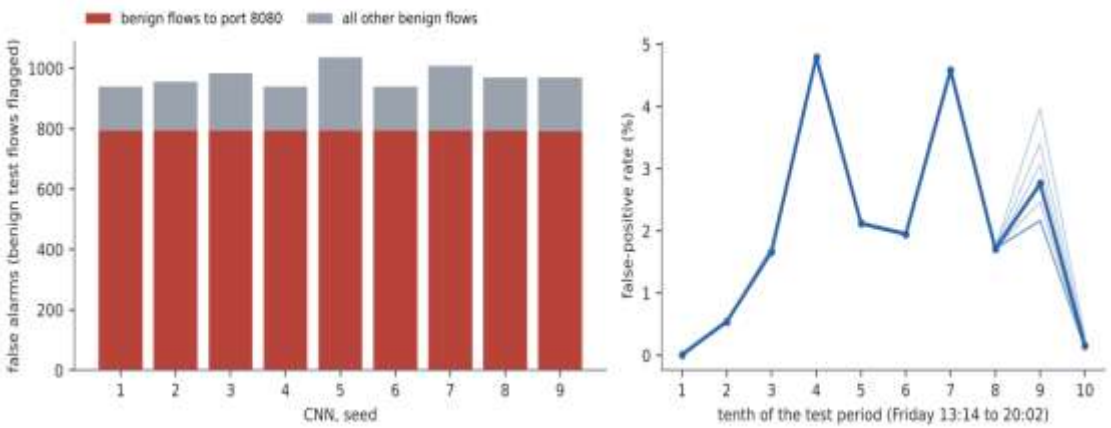


Fig. 2. False alarms of the nine CNNs: (a) share due to port 8080; (b) FPR in each tenth of the benign test period.

Same attack, same flows, different detectors:-

The nine attacked CNNs have an FPR of 1.96% to 2.17% and, seed 3 apart, a test macro F1 of 0.93 to 0.95. Under the same 15 conditions and on the same flows (Fig. 3, Table 2), their evasion rates span 5–100% for Bot, 14–100% for BruteForce, 9–100% for DDoS, 8–93% for DoS and 0–99% for PortScan. Repeating the attack on one detector moves its rate by 3.8 to 15 points at most, except for BruteForce on the CNN of seed 7 (1% to 41%), the one case where the attacker's luck matters.

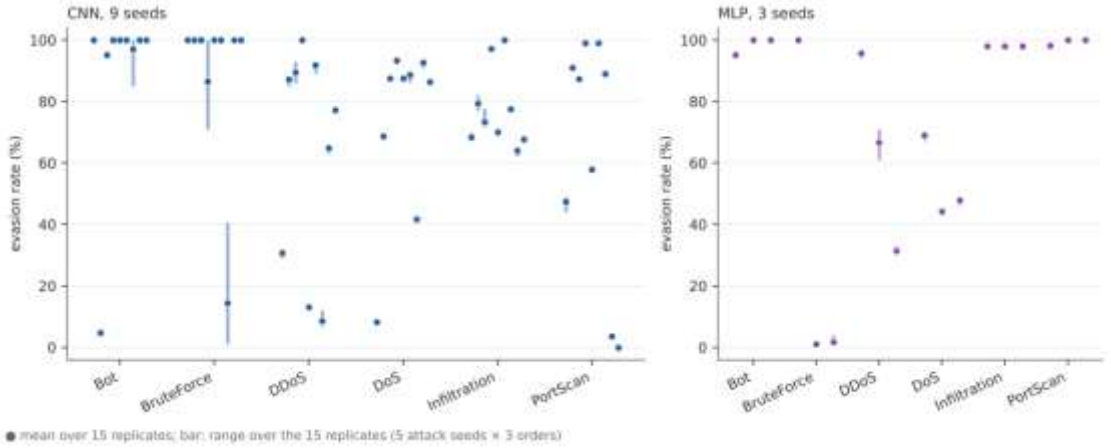


Fig. 3. Evasion at the medium budget on the same flows: nine CNNs (left) and three MLPs (right); a dot is a detector's mean over 15 attack conditions, its bar their range.

Table 2. For each of the nine CNNs: validation and test macro F1, FPR, and mean evasion rate per family (medium budget, 15 conditions).

CNN seed	Val. F1	Test F1	FPR	Bot	Brute Force	DDoS	DoS	Infiltration	Port Scan
1	0.983	0.944	1.96%	100%	100%	31%	8%	68%	47%
2	0.976	0.950	2.00%	5%	100%	87%	69%	79%	91%
3	0.982	0.874	2.05%	95%	100%	90%	88%	73%	87%
4	0.981	0.942	1.96%	100%	86%	100%	93%	97%	99%
5	0.982	0.933	2.17%	100%	100%	13%	88%	70%	58%
6	0.980	0.937	1.96%	100%	100%	92%	89%	100%	99%
7	0.970	0.939	2.10%	97%	14%	9%	42%	78%	89%

CNN seed	Val. F1	Test F1	FPR	Bot	Brute Force	DDoS	DoS	Infiltration	Port Scan
8	0.981	0.946	2.03%	100%	100%	65%	93%	64%	4%
9	0.965	0.937	2.03%	100%	100%	77%	86%	68%	0%

Table 3 splits this variation. The detector carries 13 to 39 points of standard deviation, the attack condition at most one point and the residual 0.6 to 5.3 points; the detector holds 96–100% of the sum of squares, and shuffling detector labels never produces a gap as large ($p < 0.001$ for each family, Holm). Counting already-benign flows as evaded, or keeping only flows flagged by all nine CNNs, shifts no share by more than 0.1 point.

Table 3. Variance of the evasion rate over nine CNNs and 15 attack conditions: standard deviations by source, detector share of the sum of squares, $\hat{\eta}_D$ with bootstrap interval and Holm-corrected permutation p-value.

Family	Range of the detector means	SD detector	SD condition	SD interaction and residual	Sum of squares, detector	$\hat{\eta}_D$ [95% interval]	p (Holm)
Bot	5–100%	31.5	0.0	1.4	99.8%	1.00 [0.26, 1.00]	< 0.001
BruteForce	14–100%	28.3	1.0	5.3	96.3%	0.97 [0.47, 1.00]	< 0.001
DDoS	9–100%	35.7	0.0	1.1	99.9%	1.00 [0.99, 1.00]	< 0.001
DoS	8–93%	29.2	0.0	0.6	100.0%	1.00 [0.99, 1.00]	< 0.001
Infiltration	64–100%	12.9	0.0	0.9	99.5%	1.00 [0.94, 1.00]	< 0.001
PortScan	0–99%	39.4	0.0	0.6	100.0%	1.00 [1.00, 1.00]	< 0.001

The 33 MLPs, with a test macro F1 of 0.895–0.946, spread just as widely: for BruteForce, 16 let 90% or more of the flows through and 17 at most 5%; DDoS spans 5–100% and DoS 4–91%; the detector holds at least 99.8% of the sum of squares. Dropping the destination port leaves detection unchanged (macro F1 0.928) and makes the three CNNs without port somewhat more vulnerable at the low budget, a trend that three models per group cannot establish (Holm-corrected $p = 0.30$).

Is the number of seeds sufficient? The central claimthat equally accurate models can differ greatly in evasion—is an existence claim: two detectors with non-overlapping intervals suffice, and every family has such pairs (for PortScan, 0% against 99%, each over 100 flows and 15 conditions). Estimating the full distribution of vulnerability over seeds would require many more runs; the 33 MLPs give a first picture of it, and the tree ensembles of Section 6.6 confirm the pattern with another kind of randomness. Bot has only 21 test flows, so its rates have wider intervals and its correlations are the least reliable.

A smaller budget:-

At the low budget, on all 42 detectors and the same flows (Fig. 4), no detector is more vulnerable than at the medium budget, but the drop depends on the family and the architecture. Infiltration and PortScan barely move. Bot separates the architectures: the median CNN now lets none of its alerted Bot flows through, against all of them at the medium budget, while the median MLP still lets 95% through. BruteForce, let through on more than half of the flows by 24 detectors at the medium budget, is let through that much by only five. DDoS and DoS lose 20 to 25 points in median. The detector still explains at least 98% of the variation of each family, and the two budgets rank the detectors alike (r_S from 0.56 for BruteForce to 0.96 for PortScan), except for Bot (0.28), which has only 21 test flows.

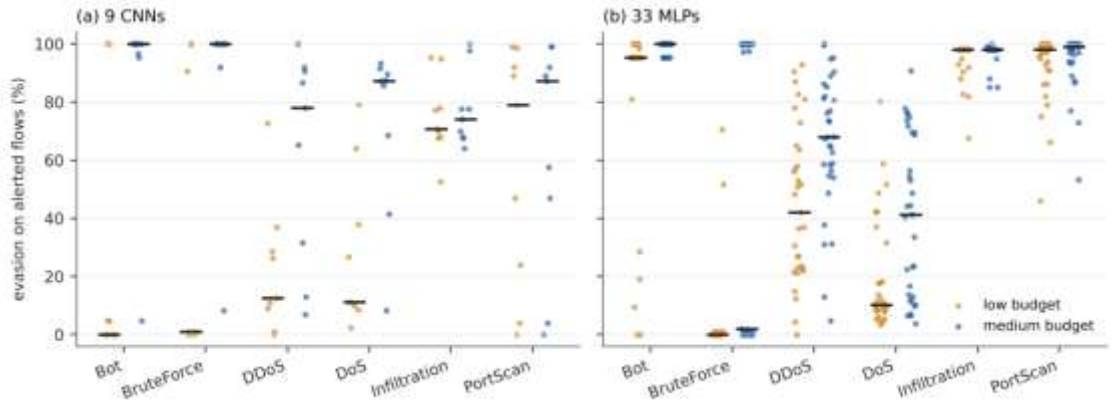


Fig. 4. Evasion of the 42 detectors at the low budget (orange) and at the medium budget (blue) on the alerted test flows, three attack seeds; a dot is a detector, a bar the median.

Does memory help?:-

Against a blind search with growing scale, the adaptive attacker is cheaper by a median factor of 39 for BruteForce, 15 for Bot and 1.4 to 3.1 elsewhere. This flatters memory, because the blind schedule wastes queries on tiny actions: over the nine CNNs, a uniform random search costs 2.57 million queries, as much as the adaptive attacker. Memory pays only when one action suits many flows, as for BruteForce, where an adaptive variant with uniform draws needs 58,000 queries against 229,000–253,000 for the other attackers.

Can vulnerability be measured before deployment?:-

Detection metrics do not reveal it. Across the nine CNNs, none of 18 correlations between a detection metric and an evasion rate is significant, and for the 33 MLPs validation macro F1 and FPR stay within $|r_S| \leq 0.38$. An attack does. We attacked every detector on 100 validation flows per family (Fig. 5). Ranked by validation evasion and by test evasion, the detectors come out in nearly the same order: r_S = 0.60–0.99 for the MLPs ($p < 0.001$ in each family, Holm) and 0.72–1.00 for the CNNs (Holm $p \leq 0.04$), and in every family the CNN with the lowest validation evasion is also the least vulnerable on the test flows. A cheaper probe, the same 20 random actions on each validation flow (2,000 queries per family), already correlates with test evasion at 0.64–0.95 for the MLPs.

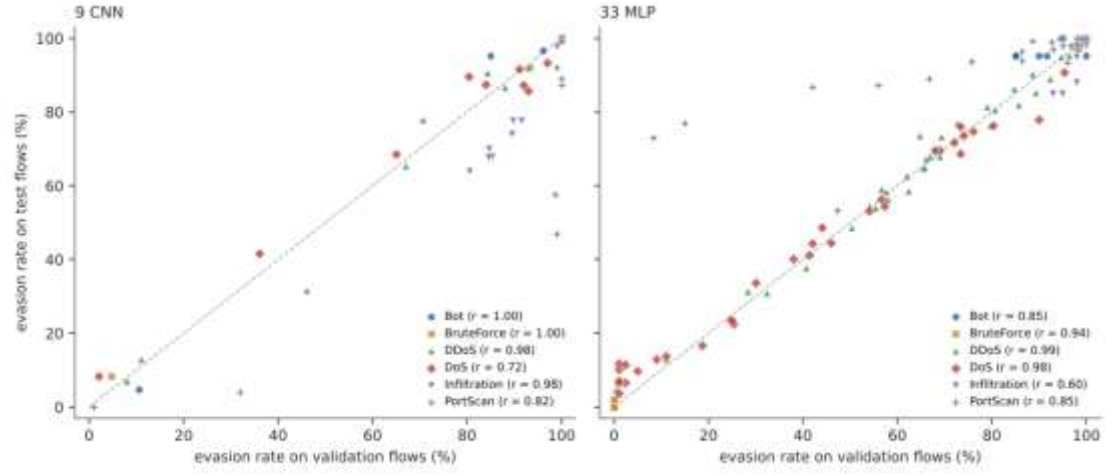


Fig. 5.Validation against test evasion, one point per detector and family; nine CNNs (left) and 33 MLPs (right).

Beyond neural networks: tree ensembles:-

Fig. 6 and Table 4 give the evasion rates of five random forests and three gradient-boosting models under the same attack. The five forests have test macro F1 between 0.936 and 0.938 and an FPR of 1.94%–1.95%. Yet three of them let through none of the alerted BruteForce flows and two let through all of them (0% [0, 1] against 100% [99, 100]); on DDoS the forest with seed 0 is evaded on 46% of the flows and the others on 2%–19%. Among the gradient-

boosting models, whose seed only chooses the early-stopping hold-out, BruteForce goes from 0% to 79% and 92%, and PortScan from 73% to 100%. Across the eight tree ensembles the detector explains $\hat{\eta}_D \geq 0.98$ of the variance in every family and budget, except DoS at the medium budget (0.93), which is almost never evaded; the attack seed changes the outcome of only 0.35% of flow–detector pairs. The two tree families differ in no single direction: at the low budget every forest is evaded on 92%–100% of PortScan flows and every gradient-boosting model on none, while for DDoS it is the reverse. So the seed effect is not a quirk of neural networks or of stochastic gradient training.

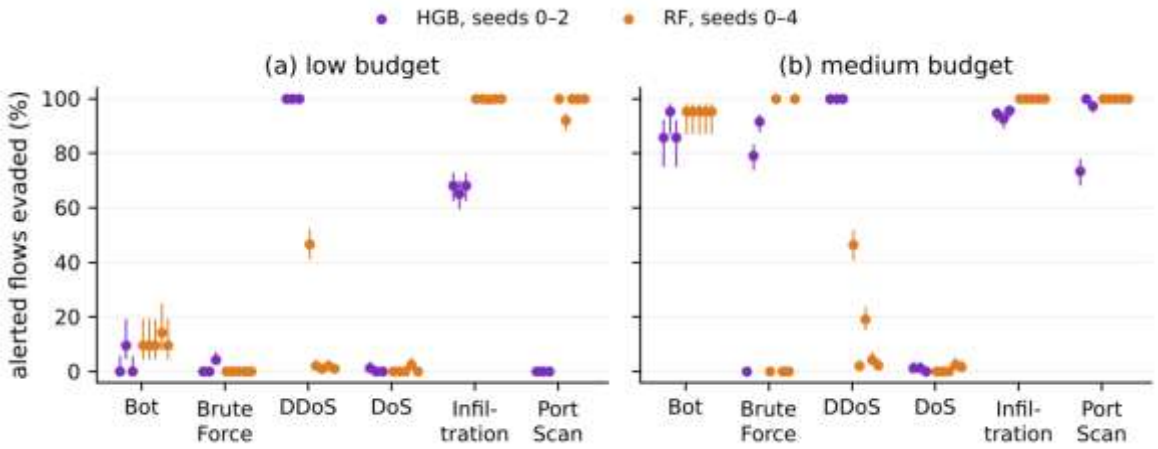


Fig. 6. Evasion of the alerted test flows by the same attack on five random forests and three gradient-boosting models (three attack seeds pooled; bars: Wilson 95% intervals). (a) Low budget. (b) Medium budget.

Table 4. Tree ensembles at the medium budget: test macro F1 and evasion rate per family (%, three attack seeds pooled) with Wilson 95% interval. Alerted flows per detector and condition: Bot 21, BruteForce 95–100, DDoS 100, DoS 80–82, Infiltration 100, PortScan 100.

Detector	Test F1	Bot	BruteForce	DDoS	DoS	Infiltration	PortScan
RF, seed 0	0.936	95 [87, 98]	0 [0, 1]	46 [41, 52]	0 [0, 2]	100 [99, 100]	100 [99, 100]
RF, seed 1	0.936	95 [87, 98]	100 [99, 100]	2 [1, 4]	0 [0, 2]	100 [99, 100]	100 [99, 100]
RF, seed 2	0.936	95 [87, 98]	0 [0, 1]	19 [15, 24]	0 [0, 2]	100 [99, 100]	100 [99, 100]
RF, seed 3	0.938	95 [87, 98]	0 [0, 1]	4 [3, 7]	2 [1, 5]	100 [99, 100]	100 [99, 100]
RF, seed 4	0.936	95 [87, 98]	100 [99, 100]	2 [1, 5]	2 [1, 4]	100 [99, 100]	100 [99, 100]
HGB, seed 0	0.941	86 [75, 92]	0 [0, 1]	100 [99, 100]	1 [0, 4]	95 [92, 97]	73 [68, 78]
HGB, seed 1	0.943	95 [87, 98]	79 [74, 83]	100 [99, 100]	1 [0, 4]	93 [89, 95]	100 [99, 100]
HGB, seed 2	0.933	86 [75, 92]	92 [88, 94]	100 [99, 100]	0 [0, 2]	96 [93, 97]	97 [95, 99]

Where the false alarms come from, and operating points:-

The benign test set contains 831 flows that the corrected dataset labels Attempted. They coincide with the benign test flows to port 8080 of Section 6.1, all recorded on the afternoon of the last day. Every model family flags most of thema median of 794 of the 831 for the networks and 747 to 793 for the tree ensemblesand they make up a median of 82% (CNN), 85% (MLP), 85% (RF) and 84% (HGB) of all false alarms. Without them the FPR is 0.30%–0.52% for every detector (Fig. 7a), and three MLPs retrained after removing Attempted flows from all splits reach 0.30%–0.33% with a test macro F1 of 0.934–0.948. The 0.3%–0.5% that remains is what an operator would actually face from benign traffic. How Attempted flows are labelled should therefore be stated in every evaluation on the corrected datasets, because it moves the FPR by a factor of six.

Mitigation by thresholds. Setting the decision threshold on validation data for a target FPR of 0.1% (Fig. 7b) gives very different outcomes on the later test data: the gradient-boosting models reach 0.01% FPR while still alerting on 91% of attack flows, the MLPs 0.44% for 84%, and the CNNs 0.08% but for only 47% of attack flows (range 10%–94%); the random forests cannot move because their probability estimates are coarse. Strict thresholds also silence the rarest families first: at the 0.1% target the median CNN alerts on none of the Bot and BruteForce test flows. Any threshold has to be checked, family by family, on data later than the data used to set it.

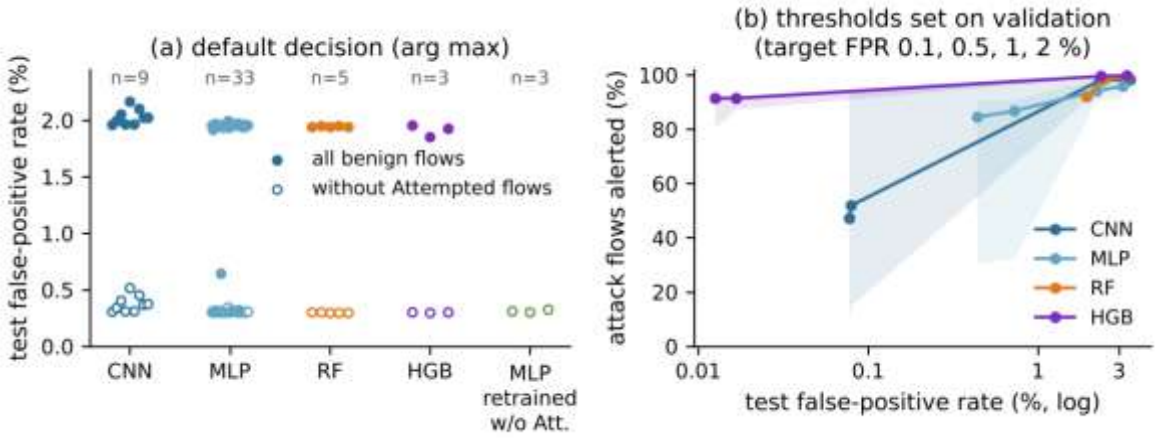


Fig. 7. False alarms under the per-class chronological split. (a) Test FPR of each detector on all benign flows (filled) and without the 831 Attempted flows (open); last column: three MLPs retrained without Attempted flows. (b) Alerted attack flows against test FPR when the threshold is set on validation data for a target FPR of 0.1%, 0.5%, 1% and 2% (median over detectors; band: range).

Unseen attack families: a global chronological split:-

Under the global split (Section 3.1), the test attacks are DDoS and PortScan, never seen in training. Table 5 shows that the random forest and the MLP still raise an alert on 98%–99.5% of them at an FPR below 0.1%; they give the wrong attack name, but it is the alert that sends an analyst to look. The default gradient-boosting model misses almost all DDoS flows (0.3% alerted), and a threshold set on validation data restores 99.4% of alerts at 0.31% FPR. The per-class split used for the attacks thus tests later traffic of known families, while the global split shows that the binary alert decision degrades much less than the multiclass score suggests.

Table 5. Detection under the three splits (one model per family for the random and global splits). * Multiclass score not meaningful: the test families are absent from training. † Threshold set on validation data for a target FPR of 0.5%.

Split	Model	Test macro F1	Test FPR	Attack flowsalerted
Random	RF / HGB / MLP	0.981–0.984	0.00%–0.01%	99.9%–100%
Per-class chronological	RF (5 seeds)	0.936–0.938	1.94%–1.95%	92.3%–92.9%
Per-class chronological	HGB (3 seeds)	0.933–0.943	1.85%–1.95%	93.4%–93.5%
Global chronological	RF	0.19*	0.00%	98.2%
Global chronological	HGB	0.09*	0.07%	43.1% (99.4% at 0.31% FPR†)
Global chronological	MLP	0.17*	0.02%	99.5%

Transfer to CSE-CIC-IDS-2018:-

We applied the 42 detectors, trained on CIC-IDS-2017, to a sample of 448,579 flows of the corrected CSE-CIC-IDS-2018 [12], with the 2017 normalisation, and attacked them at the low budget on 100 flows per family with three attack seeds. Detection does not transfer: the test macro F1 falls to 0.12–0.44 and most DoS, DDoS and BruteForce flows no longer raise an alert. On the two families that still do, the seed effect persists. On Infiltration, where every detector still alerts on 86 to 91 of the 100 flows, the evasion rate ranges from 7% to 100% across the 42 detectors, the detector explains 99.9% of its variance, and the ranking of the detectors agrees with their ranking on 2017 flows (r_S = 0.74,

permutation $p < 0.001$; 0.92 for the nine CNNs). On Bot, for the 34 detectors that alert on at least 20 flows, evasion ranges from 0% to 100% and the ranking again agrees with 2017 ($r_S = 0.59$, $p < 0.001$). Vulnerability, in other words, belongs partly to the trained model and follows it to another network; it is not only a matter of the test flows.

Are the modified features consistent with a flow extractor?:-

CIC-IDS-2017 provides flows, not the packet sequences behind them, so the attack modifies stored feature vectors. To test the conventions of Section 3.2 against an extractor rather than against our own algebra, we generated 2,000 synthetic TCP flows packet by packet (1–60 forward and 0–60 backward packets, payloads from 0 to 1,460 bytes, mean gaps from 0.1 ms to 1 s), applied a random medium-budget action to the packets themselves, and summarised both versions with an independent open-source implementation of CICFlowMeter [27] (Python package cicflowmeter 0.5.0). We compared our closed-form prediction, computed from the original flow's statistics only, with the extractor's output on the modified packets (Fig. 8a), and measured how the features that we keep unchanged really move (Fig. 8b).

The 24 features updated in closed form agree with the extractor on every trace: nine exactly (counts, size extremes, header bytes, down/up ratio), the duration, rates and forward inter-arrival total to within about 10^{-5}, and byte totals, means and dispersions to within 0.5%, the residual coming from rounding the dummy-packet size to whole bytes. Server-side and protocol features indeed do not change. Position-dependent statistics do: inter-arrival means, maxima and standard deviations change in 81%–98% of the modified flows, by a median of 20%–47%, and PSH and ACK counts and forward bulk statistics in 67%–89%. Leaving them untouched, as a feature-space evaluation does, cannot reproduce exactly what an extractor would compute. Because the error can go either way, the measured evasion rates are neither upper nor lower bounds for live traffic.

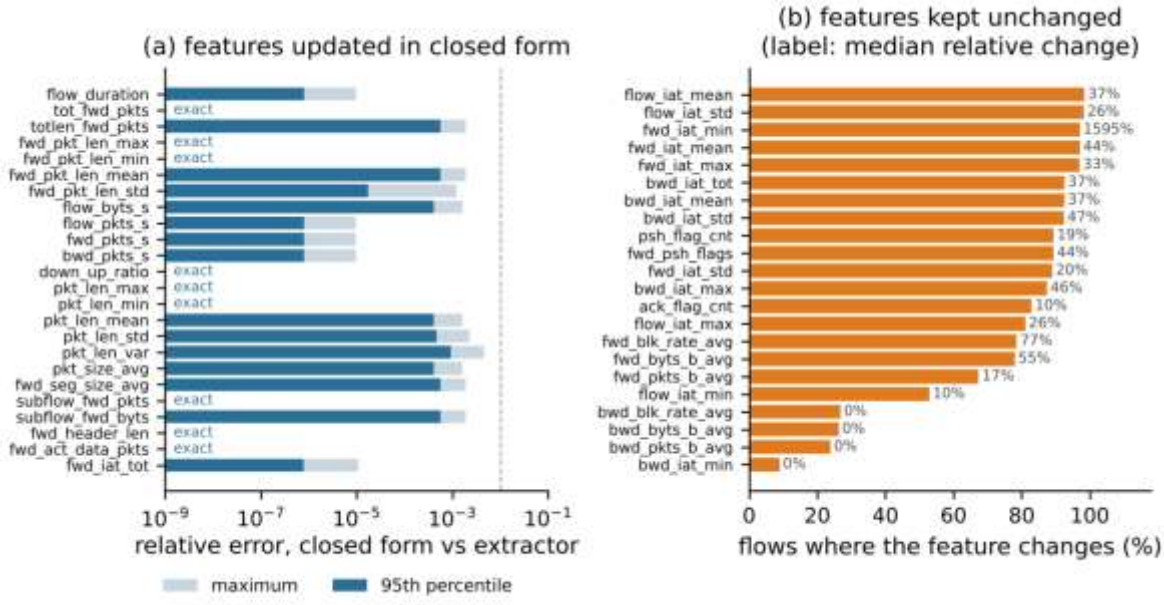


Fig. 8. Flow-level check on 2,000 synthetic traces, medium budget. (a) Relative error between the closed-form update and the extractor's output on the modified packets (95th percentile and maximum; dashed line: 1%). (b) Share of flows in which a feature kept unchanged by the feature-space attack actually changes, with the median relative change.

Computational cost:-

Table 6 gives the inference cost of each model family on one Kaggle CPU core. Per-flow latency matters for an inline detector and for the attacker, whose every query is one inference. Evaluating one tree ensemble against the attack took 0.31 to 0.82 million queries at the medium budget and 0.76 to 1.0 million at the low budget (three attack seeds, six families), i.e. about two hours of CPU time per detector at most; ten training runs per model family take hours to evaluate, not days. On successful flows the attacker needed a median of one query, because 95% of successes come from replaying a remembered action; DoS is the exception, with a median of about 245 queries.

Table 6. Inference cost on one CPU core, size of the stored model and training time per run.

Model	Latency, one flow (median)	Throughput, batches of 10,000	Model size	Training time
MLP	0.12 ms	440,000 flows/s	0.25 MB	2.4–3.7 min
CNN	0.47 ms	3,400 flows/s	0.63 MB	2–3.6 h
RF	7.5 ms	74,000 flows/s	4.3 MB	69–73 s
HGB	7.2 ms	35,000 flows/s	0.62 MB	38–44 s

Discussion and limits:-

Two models with the same accuracy may differ by almost 100 points in how often one attack gets through, and their validation scores will not say which. Attacking candidate models on a separate set of validation flows, or at least probing them with random actions, is a cheap way to choose; the final test set must stay out of that choice. For statefuldefences, an adaptive campaign stands out at its start, when one flow receives hundreds of queries, but later sends about one query per flow.

Why does the seed change robustness? The attacker's actions move a flow along a few directions of feature space—longer duration, larger and more forward packets, lower rates—that are sparsely covered by the training flows of each family. Training data constrain the decision boundary where flows are dense, so equally accurate models agree there; along the attacker's directions the boundary is underdetermined and its position is set by incidental choices of the training run, as underspecification predicts [9]. For networks these choices are the initial weights, the batch order and the dropout masks; Section 6.6 shows that much less randomness suffices—in gradient boosting, only the choice of the early-stopping hold-out. BruteForce flows are nearly identical, so the boundary falls either just beyond the reachable actions or just inside them: all or nothing. In more diverse families, the share of flows within reach shifts gradually from one model to the next.

Using the finding in practice. Three cheap steps follow. (i) Measure robustness over several training runs and report its spread, not a single value. (ii) Select the deployed model with the fixed attack run on validation flows that are not reused for the final evaluation; with the latencies of Table 6 this costs a few CPU hours per candidate. The selected model is the most robust against that attack and must be re-evaluated if the attacker's action space changes, for instance if it gains control of the server side. (iii) State the false-alarm budget: report the FPR with and without Attempted flows, set the threshold on data later than the training data, and check family-level recall at that threshold.

Does the modified traffic still attack? Padding adds bytes that the receiving application must tolerate: harmless where the protocol has free-form fields (HTTP headers, TLS padding, the payload of a scan probe), harmful where every byte is parsed (an SSH or FTP authentication exchange), in which case it must be carried by dummy packets instead. Delays slow the attack: a delayed brute-force campaign tries fewer passwords per hour and a delayed flood is no longer a flood. For DoS and DDoS, any delay lowers the attack rate, so their evasion rate measures how easily the detector is fooled rather than how harmful the evading flow remains. Only replaying modified attacks against real services can settle the question.

The results come from one laboratory dataset of five days, split per class, and validation and test flows come from the same days. The attacks act on features: 28 features that would change in real traffic are not updated and no modified flow was replayed through CICFlowMeter, so the measured evasion is neither an upper nor a lower bound for live traffic. A failure within L queries does not prove that no evading action exists, and we measure one attack policy at two budgets.

Future work. We will replay modified flows from packet captures through the corrected Java CICFlowMeter, extend the attacks to UNSW-NB15 and to detectors trained on CSE-CIC-IDS-2018 itself, study concept drift over longer periods [15], and evaluate defences against adaptive attackers: adversarial training, ensembles over several training runs, and stateful query detection [17], [18].

Conclusion:-

A chronological split barely moves the macro F1 of a deep detector but multiplies its false-positive rate by more than six hundred. Nine CNNs and 33 MLPs that differ only in their seed reach similar accuracy, yet the same attack on the same flows gets through almost none of them or almost all, depending on the model and the family, at two

attack budgets. Detection scores miss this; an attack on validation flows does not. Evaluations of evasion should cover several training runs with a fixed attack, and use that attack to pick the deployed model.

Taken together, the results show that the training seed decides most of a flow-based detector's vulnerability to a realistic black-box attack—for CNNs, MLPs, random forests and gradient-boosted trees, at two budgets and on a second dataset—together with a practical procedure: report robustness as a spread over training runs, choose the deployed model with an attack on validation flows, and state the false-alarm budget, including how Attempted flows are labelled.

Data availability.The corrected CIC-IDS-2017 is available athttps://www.kaggle.com/datasets/ernie55ernie/improved-cicids2017-and-csecicids2018. The 45 trained detectors used in the attacks are published at https://www.kaggle.com/datasets/boriszannou/ids-modeles and the training and attack notebooks, flow sets, per-flow results and manifests (SHA-256 digests and software versions) at https://www.kaggle.com/boriszannou/code.

References:-

[1] I. Sharafaldin, A. HabibiLashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP), 2018, pp. 108–116, doi: 10.5220/0006639801080116.
[2] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, “Deep learning approach for intelligent intrusion detection system,” IEEE Access, vol. 7, pp. 41525–41550, 2019, doi: 10.1109/ACCESS.2019.2895334.
[3] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, “TESSERACT: Eliminating experimental bias in malware classification across space and time,” in Proc. 28th USENIX Secur. Symp., 2019, pp. 729–746.
[4] D. Arp et al., “Dos and don’ts of machine learning in computer security,” in Proc. 31st USENIX Secur. Symp., 2022, pp. 3971–3988.
[5] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in Proc. IEEE Symp. Secur. Privacy (SP), 2010, pp. 305–316, doi: 10.1109/SP.2010.25.
[6] G. Apruzzese, P. Laskov, and J. Schneider, “SoK: Pragmatic assessment of machine learning for network intrusion detection,” in Proc. IEEE 8th Eur. Symp. Secur. Privacy (EuroS&P), 2023, pp. 592–614, doi: 10.1109/EuroSP57164.2023.00042.
[7] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, “Intriguing properties of adversarial ML attacks in the problem space,” in Proc. IEEE Symp.Secur. Privacy (SP), 2020, pp. 1332–1349, doi: 10.1109/SP40000.2020.00073.
[8] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, “Modeling realistic adversarial attacks against network intrusion detection systems,” Digit. Threats: Res. Pract., vol. 3, no. 3, pp. 1–19, 2022, doi: 10.1145/3469659.
[9] A. D’Amour et al., “Underspecification presents challenges for credibility in modern machine learning,” J. Mach. Learn. Res., vol. 23, no. 226, pp. 1–61, 2022.
[10] X. Bouthillier et al., “Accounting for variance in machine learning benchmarks,” in Proc. Mach. Learn. Syst. (MLSys), vol. 3, 2021.
[11] G. Engelen, V. Rimmer, and W. Joosen, “Troubleshooting an intrusion detection dataset: The CICIDS2017 case study,” in Proc. IEEE Secur. Privacy Workshops (SPW), 2021, pp. 7–12, doi: 10.1109/SPW53761.2021.00009.
[12] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, “Error prevalence in NIDS datasets: A case study on CIC-IDS-2017 and CSE-CIC-IDS-2018,” in Proc. IEEE Conf. Commun. Netw.Secur. (CNS), 2022, pp. 254–262, doi: 10.1109/CNS56114.2022.9947235.
[13] M. Lanvin, P.-F.Gimenez, Y. Han, F. Majorczyk, L. Mé, and É. Totel, “Errors in the CICIDS2017 dataset and the significant differences in detection performances it makes,” in Risks and Security of Internet and Systems (CRiSIS 2022), ser. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2023, pp. 18–33, doi: 10.1007/978-3-031-31108-6_2.
[14] R. Flood, G. Engelen, D. Aspinall, and L. Desmet, “Bad design smells in benchmark NIDS datasets,” in Proc. IEEE 9th Eur. Symp. Secur. Privacy (EuroS&P), 2024, pp. 658–675, doi: 10.1109/EuroSP60621.2024.00042.
[15] G. Apruzzese, A. Fass, and F. Pierazzi, “When adversarial perturbations meet concept drift: An exploratory analysis on ML-NIDS,” in Proc. Workshop Artif. Intell.Secur. (AISec), 2024, pp. 149–160, doi: 10.1145/3689932.3694757.
[16] J. Chen, M. I. Jordan, and M. J. Wainwright, “HopSkipJumpAttack: A query-efficient decision-based attack,” in Proc. IEEE Symp. Secur. Privacy (SP), 2020, pp. 1277–1294, doi: 10.1109/SP40000.2020.00045.

[17] S. Chen, N. Carlini, and D. Wagner, "Stateful detection of black-box adversarial attacks," in Proc. 1st ACM Workshop Secur.Privacy Artif.Intell. (SPAI), 2020, pp. 30–39, doi: 10.1145/3385003.3410925.
[18] R. Feng, A. Hooda, N. Mangaokar, K. Fawaz, S. Jha, and A. Prakash, "Stateful defenses for machine learning models are not yet secure against black-box attacks," in Proc. ACM SIGSAC Conf. Comput. Commun.Secur. (CCS), 2023, doi: 10.1145/3576915.3623116.
[19] S. Holm, "A simple sequentially rejective multiple test procedure," Scand. J. Statist., vol. 6, no. 2, pp. 65–70, 1979.
[20] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
[21] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 30, 2017, pp. 3146–3154.
[22] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.
[23] A. Kantchelian, J. D. Tygar, and A. D. Joseph, "Evasion and hardening of tree ensemble classifiers," in Proc. 33rd Int. Conf. Mach. Learn. (ICML), 2016, pp. 2387–2396.
[24] H. Chen, H. Zhang, D. Boning, and C.-J.Hsieh, "Robust decision trees against adversarial examples," in Proc. 36th Int. Conf. Mach. Learn. (ICML), 2019, pp. 1122–1131.
[25] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," in Proc. Int. Conf. Learn. Represent. (ICLR), 2018.
[26] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in Proc. Netw. Distrib. Syst. Secur. Symp.(NDSS), 2009.
[27] A. HabibiLashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Tor traffic using time based features," in Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy (ICISSP), 2017, pp. 253–262, doi: 10.5220/0006105602530262.
[28] E. B. Wilson, "Probable inference, the law of succession, and statistical inference," J. Amer. Statist. Assoc., vol. 22, no. 158, pp. 209–212, 1927, doi: 10.1080/01621459.1927.10502953.