WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Robust Botnet Detection Against Reinforcement-Learning Evasion: Transferability, Operational Cost and a Grounded Analyst Assistant

Netrang V Davey[1], Rishab P K[2], Dr. G. Paavai Anand[3]

[1–2] Undergraduate Students; [3] Faculty Guide
Department of Computer Science and Engineering, SRM Institute of Science and Technology, Vadapalani, Chennai, India
[1] RA2411003040039 [2] RA2411003040066
[1] nd6985@srmist.edu.in [2] pr0645@srmist.edu.in

ABSTRACT:

Tree-based classifiers are a common choice for flow-level botnet detection because they are accurate on benchmark traffic and cheap to run. Their weakness is that an attacker who can adjust a few flow statistics, without breaking the underlying connection, can push malicious flows across the decision boundary. The DReLAB benchmark packages exactly this scenario: it pairs original botnet flows with versions rewritten by deep reinforcement learning agents. We use it to ask three questions that earlier work on this dataset left open. First, when a detector is hardened with adversarial flows from some botnet families, does the protection carry over to a family it has never seen? Second, what does hardening cost on ordinary traffic, measured as false alarms? Third, do the features that the attacker edits coincide with the features the model actually relies on? A Random Forest trained on 171 thousand clean flows reaches 97.89% recall at 0.70% false-positive rate, yet detects only 25.51% of the adversarial flows. Mixing 10% adversarial samples into training lifts adversarial recall to 95.84% while the false-positive rate moves from 0.71% to 0.83% and clean F1 stays at 98.6%. In leave-one-family-out experiments the hardened model gains between 36 and 68 recall points on five unseen families, but only 11.7 points on Murlo. SHAP analysis shows that the two most heavily perturbed features rank first and third in the model's own importance ordering. We also describe a small LangGraph assistant whose language model can only report numbers returned by seven deterministic tools wrapped around the trained model, which lets an analyst question individual verdicts without risk of fabricated statistics.

Keywords: Botnet detection; adversarial evasion; reinforcement learning; adversarial training; transferability; false-positive rate; SHAP; network flow features; LangGraph; security operations.

Introduction:

A botnet is a set of compromised machines that take instructions from a controller. Its traffic is a mix of command-and-control exchanges, scanning and whatever payload the operator is running at the time, and much of it looks unremarkable at the packet level. Flow-level detection sidesteps deep packet inspection by summarising each connection into a short vector of statistics such as duration, byte counts and packets per second, and then asking a classifier whether the vector resembles known botnet behaviour. Random Forests and gradient-boosted trees handle this task well on public datasets, and on the clean portion of DReLAB we observe the same thing.

The difficulty is that these statistics are under the attacker's control to a degree. Adding padding changes bytes per packet; inserting a delay changes duration and rate. Apruzzese et al. [1] showed that a reinforcement learning agent can discover sequences of such edits that move a malicious flow to the benign side of a trained detector while keeping the flow functional. Venturi et al. [2] later released the resulting flows as the DReLAB dataset, pairing every adversarial flow with the original it was derived from. That pairing is what makes a controlled defence study possible without re-implementing the attack.

Background and Motivation:

The obvious remedy is adversarial training: include perturbed flows in the training set so that the model learns the perturbed region as well. Both prior papers note this and report that it helps. What they do not report is how far the help extends. A security team rarely has adversarial examples for every botnet it will face. If a model hardened on Neris and Rbot evasion samples also resists Menti evasion, hardening is a practical defence; if it does not, it is a per-threat patch.

There is also an operational question that benchmark papers tend to skip. An intrusion detector in a real network sees far more benign flows than malicious ones, so a small increase in the false-positive rate becomes a large number of alerts. A hardening method that doubled the false-positive rate might be rejected by the people who have to triage those alerts, however good its adversarial recall. We wanted the cost measured alongside the benefit.

Finally, we were curious whether the attacker's edits are interpretable. The RL agent is rewarded for lowering the victim's malicious score and knows nothing about feature semantics. If its most frequent edits land on the features a SHAP analysis flags as most influential, that is evidence the agent has effectively learnt the model's decision structure, and it also suggests which features a defender should treat as untrustworthy.

Problem Statement:

Given a flow-level botnet detector and a corpus of paired original and RL-perturbed malicious flows, we want to quantify (a) how much detection the detector loses under perturbation, (b) how much of that loss adversarial training recovers when the training perturbations come from different botnet families or different capture environments than the test perturbations, (c) the change in clean-traffic false-positive rate and F1 that accompanies hardening, and (d) the overlap between the model's most important features and the features the attacker changes most.

A secondary engineering goal is to make these results usable. The numbers above live in CSV files and a pickled model, which is fine for a report but not for an analyst asking why a particular flow was flagged. We built a conversational interface in which a language model orchestrates calls to the trained model and the saved results, and is constrained so that every figure in its answer originates from one of those calls.

Technical Challenges:

Three sources with different feature schemas have to be reconciled. The BOTNET capture exposes 33 columns, CTU-13 exposes 39 and CICIDS2018 only 14. We restricted the study to the 13 features present in all three, which rules out some potentially useful source-specific columns but is the only way to train one model and test it across sources.

Leakage is a real risk with paired data. If an original flow lands in the training set and its adversarial twin in the test set, the model has effectively seen the test flow with a few values changed. We assign a shared group identifier to each original and all of its adversarial variants and split by group, so that twins always travel together.

The clean data are heavily imbalanced in the raw files: CTU-13 alone contains about 2.58 million benign flows against a few hundred thousand malicious ones, and the smallest family, BOTNET Virut, has 643 malicious flows. We subsampled benign flows to 40,000 per source to keep training tractable, which means our clean false-positive rates are measured on a benign sample rather than the full population.

Lastly, a language model that can see numbers in its context can also paraphrase them inaccurately or supply a plausible number when a tool returns nothing. The assistant's prompt and tool design had to make that failure mode unlikely rather than merely discouraged.

Contributions:

The algorithms used here are standard. The contribution is in what is measured. We report the first leave-one-family-out and leave-one-source-out transferability results for adversarial training on DReLAB, with mean and standard deviation over three seeds; a robustness-versus-false-alarm trade-off curve at four adversarial mixing ratios; a rank comparison between SHAP feature importance and RL perturbation magnitude; and a grounded analyst assistant whose seven tools are documented well enough to be reused. We also report a negative result, the weak transfer to the Murlo family, rather than averaging it away.

Related Work:

Flow-Based Botnet Detection:

Flow statistics have been the dominant input for ML-based intrusion detection since the CTU-13 [3] and CICIDS [4] datasets made large labelled captures available. Ring et al. [5] survey the datasets in use and point out recurring problems: unrealistic traffic mixes, duplicated flows and labels that leak through timestamps or addresses. We dropped address, port-number and timestamp columns partly for this reason and kept only aggregate statistics and coarse port categories.

Reinforcement Learning Evasion:

Apruzzese et al. [1] framed evasion as a sequential decision problem. The agent observes the current feature vector, chooses one of a small set of edits such as increasing duration or appending bytes, and receives a reward when the victim's malicious probability falls. They trained Double Deep Q-Network and Deep SARSA agents against Random Forest and Wide-and-Deep victims, and found that the agents evaded most of the time with a handful of edits. Venturi et al. [2] published the generated flows as DReLAB, which is the data we use. Both papers evaluate attacks and retraining on the same family; neither asks whether retraining on one family protects another.

Adversarial Training and Transfer:

Adversarial training traces back to Goodfellow et al. [6] in the image domain, where it remains the most reliable empirical defence. In network security the perturbation space is far more constrained, because edits must preserve a working connection, so results from images do not transfer directly. Studies of defence transfer across attack types exist for image classifiers; we are not aware of one across botnet families on flow data, which is the gap this paper addresses.

Explainability:

Lundberg and Lee [7] unified several attribution methods under Shapley values, and the TreeExplainer algorithm [8] computes them exactly for tree ensembles in polynomial time. SHAP has been applied to intrusion detectors before, usually to justify a verdict to an analyst. Our use is slightly different: we compare the attribution ranking with the attacker's edit ranking to see whether the two agree.

Tool-Using Language Models:

The ReAct pattern [9] interleaves reasoning and tool calls so that a language model can look things up rather than recall them. LangGraph [10] implements this as an explicit state graph with a checkpointer for multi-turn memory. We adopted it because the graph makes the control flow auditable: there is one node that calls the model and one that calls tools, and nothing else.

Research Focus:

Taken together, the literature establishes that RL evasion works and that adversarial training helps on the same threat. The open questions are cross-threat generalisation, the alert cost of hardening, and whether the attack is interpretable. Those are the three research questions, labelled RQ1, RQ2 and RQ3 below.

Methodology:

Dataset and Preprocessing:

DReLAB consists of 59 CSV files. For each of the three sources it provides benign flows, original malicious flows per botnet family, and adversarial versions of the malicious flows generated by two RL agents (DDQN and SARSA) against two victim models (Random Forest and Wide-and-Deep). Table 1 summarises the raw counts we observed during the initial audit.

Table 1. Raw DReLAB composition by source (counts from the Stage 1 audit)

Source	Benign flows	Malicious flows	Families	Columns
CTU-13	2,582,434	124,366	Menti, Murlo, Neris, Rbot, Virut	39
CICIDS2018	405,280	285,920	Zeus/Ares	14
BOTNET	73,721	113,650	Menti, Murlo, Neris, Rbot, Virut	33

We kept the 13 features common to all sources: Dur, Proto, InBytes, OutBytes, TotPkts, TotBytes, BytesPerSec, BytesPerPkt, PktsPerSec, RatioOutIn and three one-hot indicators for the destination-port range (well-known, registered, private). Infinite values, which arise from division by zero duration, were replaced with the column's 99.9th percentile; missing values were filled with the median. Benign flows were subsampled to 40,000 per source. Every original malicious flow received a pair_id, and each adversarial flow inherited the pair_id of the original it was generated from.

The train/test split uses GroupShuffleSplit on pair_id with a 25% test fraction and a fixed seed, so a flow and all its adversarial twins are on the same side of the split. After splitting, the clean training set holds roughly 171 thousand flows and the clean test set 57,079 flows. Adversarial flows are partitioned according to the split of their originals, giving separate adversarial train and test pools.

Baseline Detectors:

Three classifiers were trained on clean training data only: logistic regression as a linear sanity check, a Random Forest with 100 trees of maximum depth 16, and an XGBoost model with 100 trees of depth 6 and learning rate 0.1. We did not tune hyperparameters beyond these defaults; the point of the baseline is to be representative, not optimal. All models use seed 42.

We report recall, precision, F1, false-positive rate (FPR) and the area under the precision-recall curve. Accuracy is omitted deliberately. With a large benign majority a classifier that labels everything benign scores well on accuracy while catching nothing, so accuracy says little about the quantity a defender cares about.

Evasion Audit:

The clean Random Forest was evaluated on the adversarial test pool, stratified by source and by family. We compare recall on clean malicious flows with recall on their adversarial counterparts and report the difference as the evasion drop. Because the adversarial pool contains flows generated against both victim architectures, the audit also implicitly tests transfer from the Wide-and-Deep victim to our Random Forest.

Adversarial Training (RQ2):

For RQ2 we retrained the Random Forest with adversarial flows drawn from the adversarial training pool and added to the clean training set at four ratios: 0%, 10%, 25% and 50% of the clean set size. Each configuration was run with three seeds, which control both the subsample and the forest. We report clean recall, precision, F1 and FPR on the clean test set and adversarial recall on the adversarial test pool, as mean and standard deviation.

Transferability (RQ1):

Leave-one-family-out (LOFO) holds out one botnet family. The baseline is a Random Forest trained on clean data only. The transfer model is trained on the same clean data plus up to 30,000 adversarial flows sampled from the remaining families. Both are tested on the held-out family's adversarial flows, and the difference in recall is the transfer gain. To keep eighteen forests tractable we used 60 trees of depth 14 in this experiment, again over three seeds. Leave-one-source-out (LOSO) repeats the procedure with the three capture sources as the groups. Note that Zeus/Ares appears only in CICIDS2018, so the LOFO row for Zeus/Ares and the LOSO row for CICIDS2018 describe the same held-out data.

Explainability (RQ3):

SHAP values were computed with TreeExplainer on the clean Random Forest. For a flow x with feature set F, the Shapley value of feature i is the average over all orderings of the change in model output when i is added:

$$\varphi_i(x) = \Sigma_{S \subseteq F \setminus \{i\}} |S|!(|F|-|S|-1)!/|F|! \cdot [\, f(S \cup \{i\}) - f(S) \,] \qquad (1)$$

Global importance is the mean absolute SHAP value over a sample of test flows. For the attacker side we computed, for each feature, the mean absolute difference between an original flow and its adversarial twin divided by the feature's standard deviation on clean data:

$$m_i = mean_pairs\ |x_i^{adv} - x_i^{orig}| / \sigma_i \qquad (2)$$

Dividing by σ makes features with different scales comparable. We then rank features by both quantities and compare the rankings. The same explainer is exposed to the assistant through an explain_flow function that returns the top-k signed contributions for a given flow.

Grounded Analyst Assistant:

Figure 1 shows the overall pipeline. The assistant sits at the end of it and reads from the saved models and results rather than recomputing anything. It is a LangGraph state graph with two nodes. The agent node sends the conversation and a system prompt to a language model that has seven tools bound to it. If the model's reply contains tool calls, a conditional edge routes to the tool node, which executes them and appends the results; control then returns to the agent node. When the reply contains no tool calls the graph terminates. A MemorySaver checkpointer keyed by thread id preserves prior turns, so a follow-up such as "and for Neris?" resolves against the previous question.

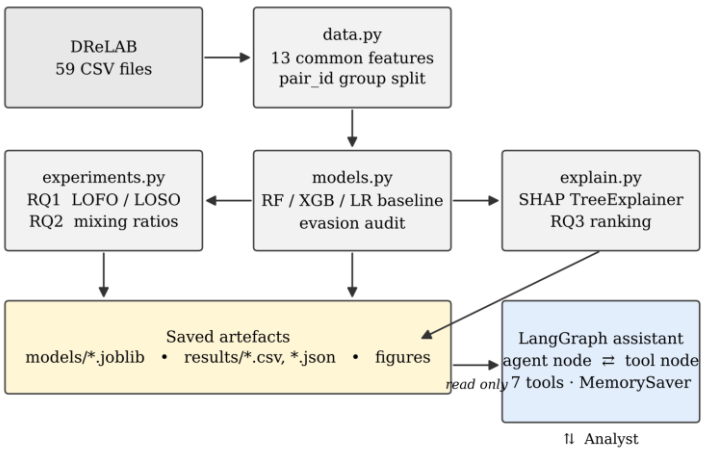


Fig 1. Processing pipeline from raw DReLAB files to the analyst assistant. The Python modules write saved models and result files; the assistant only reads them.

The seven tools are get_prediction, explain_prediction, compare_original_vs_adversarial, get_model_metrics, get_robustness_summary, list_flows and feature_glossary. Each is a plain Python function with typed arguments that returns a dictionary; the first two load the pickled forest and run inference and SHAP on demand, the rest read CSV or JSON files produced by the experiments. The system prompt instructs the model that every number must come from a tool result and that it should say so when a tool returns an error. The language model provider is selected by environment variable; we ran the evaluation with Google's Gemini Flash-Lite, with Anthropic's Claude as a fallback, and the graph is provider-agnostic. A --no-llm mode routes questions to tools by keyword matching so the tools can be tested without any API key.

Experimental Setup and Results:

Experimental Environment:

All experiments ran in Python 3.13 with scikit-learn, XGBoost, SHAP and LangGraph, on a single workstation without GPU. Seeds are fixed throughout. The repository contains one module per stage (data.py, models.py, experiments.py, explain.py, qlearning.py and the agent package), five notebooks that read saved results and produce the figures, and a pytest suite that checks the output schema of each tool. Every number in this section is read from a file in the results directory.

Baseline Results:

Table 2. Clean-traffic performance on 57,079 held-out flows

Model	Recall	Precision	F1	FPR	PR-AUC
Logistic Regression	80.53%	75.19%	77.77%	23.98%	0.845
Random Forest	97.89%	99.21%	98.55%	0.70%	0.999
XGBoost	96.55%	98.52%	97.53%	1.31%	0.997

The Random Forest is the strongest baseline on every metric and was used for all subsequent experiments. The gap to logistic regression is instructive: a 24% false-positive rate means the benign and malicious classes are not linearly separable in these thirteen features, so the trees are doing real work rather than thresholding one variable.

Evasion Audit:

Figure 2 and Table 3 show what happens to the same Random Forest when it is given the adversarial twins of the test flows. Recall on clean malicious flows is between 94% and 100% for every family. On adversarial flows it is between 15% and 54%.

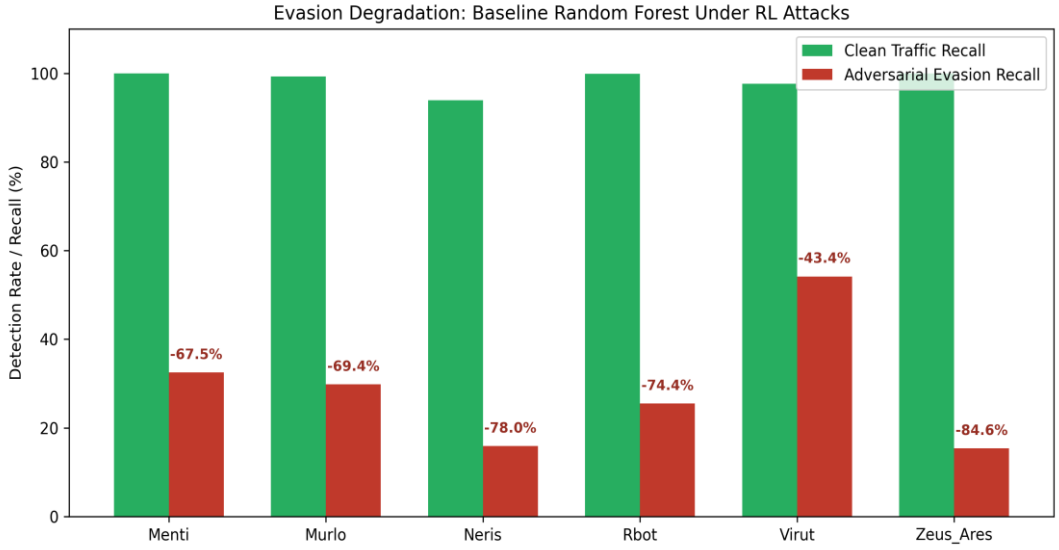


Fig 2. Random Forest recall on clean malicious flows (green) and their adversarial twins (red), by botnet family.

Table 3. Evasion drop by source and family

Group	Clean recall	Adversarial recall	Drop	Adv. flows
BOTNET	98.81%	30.61%	−68.20	42,626
CTU-13	96.43%	24.00%	−72.43	42,316
CICIDS2018	99.95%	15.36%	−84.58	15,016
Menti	100.00%	32.47%	−67.53	10,397
Murlo	99.24%	29.80%	−69.45	6,296
Neris	93.87%	15.92%	−77.96	27,684
Rbot	99.90%	25.52%	−74.37	29,423
Virut	97.60%	54.16%	−43.44	11,142
Zeus/Ares	99.95%	15.36%	−84.58	15,016

Pooled over all adversarial test flows, recall falls from 97.89% to 25.51%. Virut is noticeably more resistant than the others; we suspect this is because Virut's clean flows already sit closer to the benign region, leaving the agent less room to move them further without exceeding its edit budget, but we have not tested that explanation. It is worth stressing that about half the adversarial flows were generated against a Wide-and-Deep victim, not a Random Forest, and they evade our forest almost as well. The attack is not specific to the architecture it was trained against.

Cost of Hardening (RQ2):

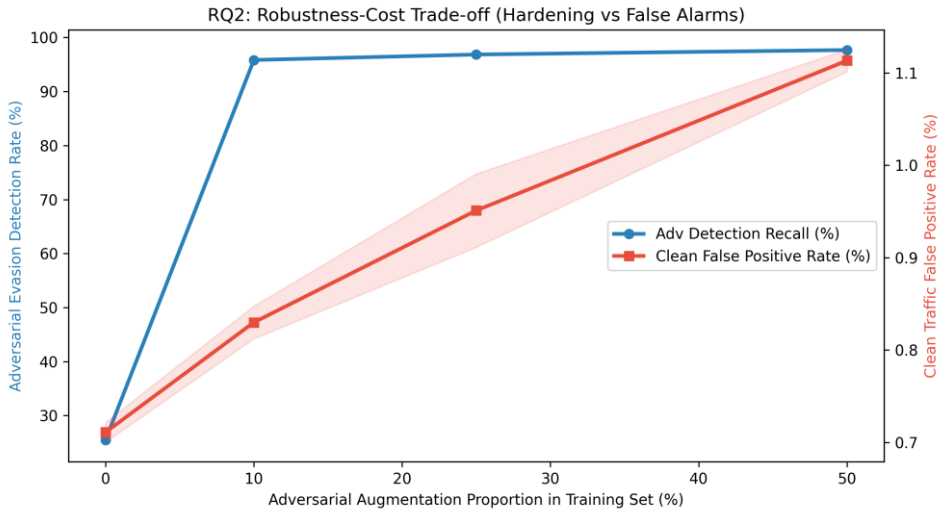


Fig 3. Adversarial recall (left axis) and clean false-positive rate (right axis) as the adversarial share of training data increases. Shading shows ±1 s.d. over three seeds.

Table 4. Hardening trade-off, mean ± s.d. over three seeds

Adversarial mix	Adv. recall	Clean recall	Clean F1	Clean FPR
0%	25.51 ± 0.43	98.01 ± 0.11	98.60 ± 0.05	0.71 ± 0.01
10%	95.84 ± 0.07	98.14 ± 0.01	98.60 ± 0.01	0.83 ± 0.02
25%	96.85 ± 0.08	98.25 ± 0.05	98.59 ± 0.02	0.95 ± 0.04
50%	97.68 ± 0.09	98.44 ± 0.07	98.60 ± 0.03	1.11 ± 0.01

Most of the benefit arrives with the first 10%. Adversarial recall rises by seventy points while the false-positive rate rises by 0.12 points and F1 does not move. Going from 10% to 50% buys a further 1.8 points of adversarial recall for a further 0.28 points of FPR. The standard deviations are small, so the

ordering of the four configurations is not a seed artefact. Whether an extra 0.4% FPR is acceptable depends on traffic volume; on a link carrying a million benign flows a day it is roughly 4,000 additional alerts, which is why we would recommend the 10% point rather than the 50% point for deployment.

Clean recall actually improves slightly with hardening, from 98.01% to 98.44%. The adversarial flows are, after all, additional labelled malicious examples, and some of them appear to fill in regions of the malicious class the clean data under-represented.

Transferability (RQ1):

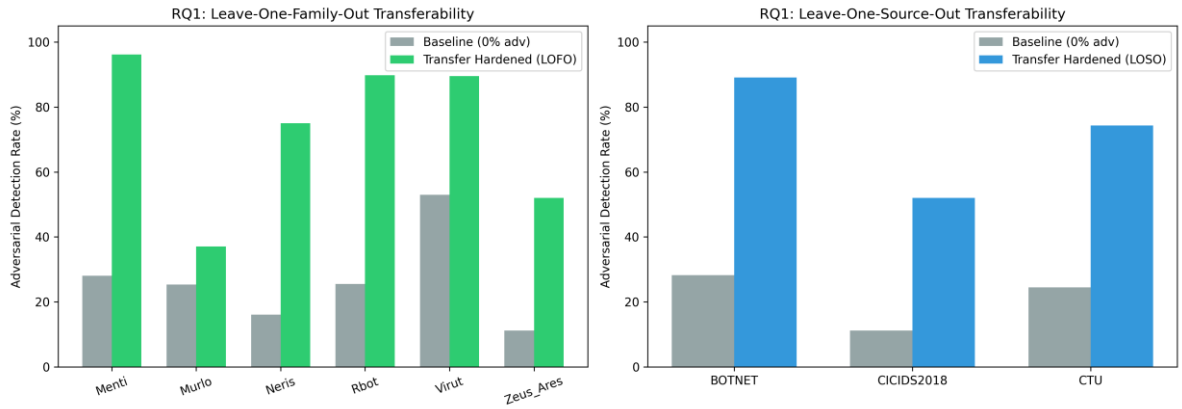


Fig 4. Adversarial recall on the held-out group for the clean baseline (grey) and the model hardened on the other groups (colour). Left: leave-one-family-out. Right: leave-one-source-out.

Table 5. Leave-one-family-out transfer, mean ± s.d. over three seeds

Held-out family	Baseline recall	Transfer recall	Gain
Menti	28.08 ± 2.86	96.21 ± 0.45	+68.13
Rbot	25.57 ± 1.34	89.81 ± 0.58	+64.24
Neris	16.09 ± 2.02	75.01 ± 0.42	+58.92
Zeus/Ares	11.17 ± 6.49	52.05 ± 0.19	+40.88
Virut	52.95 ± 0.85	89.49 ± 0.72	+36.53
Murlo	25.32 ± 0.52	37.06 ± 4.05	+11.74

Table 6. Leave-one-source-out transfer, mean ± s.d. over three seeds

Held-out source	Baseline recall	Transfer recall	Gain
BOTNET	28.27 ± 0.59	89.10 ± 1.98	+60.84
CTU-13	24.44 ± 2.32	74.31 ± 0.95	+49.87
CICIDS2018	11.17 ± 6.49	52.05 ± 0.19	+40.88

Hardening transfers, but unevenly. Five of six families gain between 36 and 68 recall points from adversarial samples they never saw. Murlo gains under 12, and its transfer recall has the largest standard deviation in the table, which suggests the hardened model is near a boundary for Murlo rather than firmly on one side. Murlo is also the smallest family after BOTNET Virut, with about 6,900 original flows across both sources, and its traffic is characterised by low packet rates that the other families' perturbations do not reproduce. We note the result rather than explain it; a Murlo-specific analysis would need more than the three seeds we ran.

Cross-source transfer is of the same order as cross-family transfer, which we did not expect. The CTU-13 and BOTNET captures were collected years apart on different networks, yet adversarial flows from one teach the model to catch adversarial flows from the other. One reading is that the RL agents, despite being trained separately per family and per victim, converge on similar edit strategies, so the perturbed region of feature space has a shared shape. RQ3 offers some support for that.

SHAP Versus Attacker Edits (RQ3):

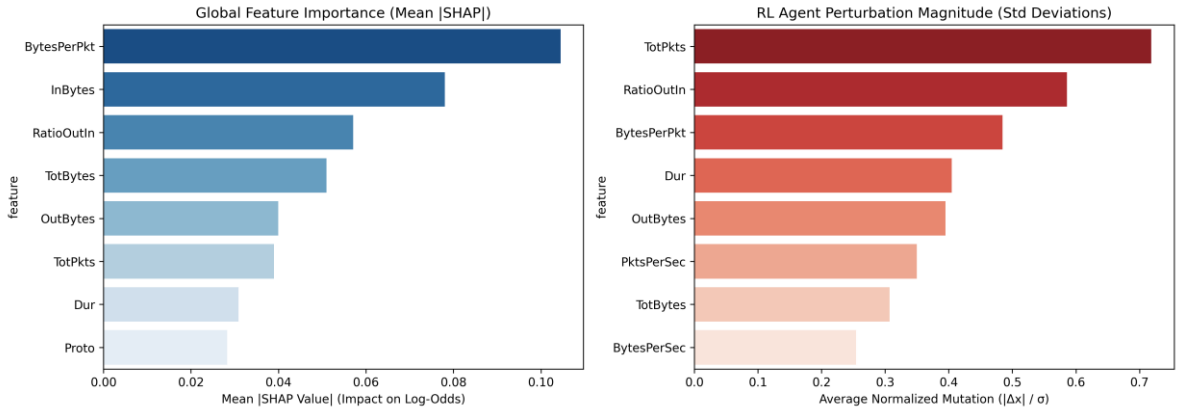


Fig 5. Left: mean |SHAP| per feature for the clean Random Forest. Right: mean normalised perturbation per feature across original–adversarial pairs. Only the top eight of thirteen features are shown in each panel.

Table 7. Feature ranking by model importance and by attacker perturbation

Feature	Mean |SHAP|	SHAP rank	Normalised edit	Edit rank
BytesPerPkt	0.1045	1	0.484	3
InBytes	0.0780	2	0.254	10
RatioOutIn	0.0571	3	0.586	2
TotBytes	0.0510	4	0.307	7
OutBytes	0.0399	5	0.395	5
TotPkts	0.0390	6	0.719	1
Dur	0.0309	7	0.405	4
Proto	0.0283	8	0.000	13

The overlap at the top is clear: BytesPerPkt and RatioOutIn are the model's first and third most important features and the attacker's third and second most edited. TotPkts is the attacker's favourite edit but only sixth for the model, which makes sense if packet count is being changed mainly as a side effect of changing bytes per packet while holding total bytes roughly fixed. InBytes is the one feature the model relies on heavily that the attacker leaves largely alone; inbound bytes are controlled by the server side of the connection, which the agent cannot edit without breaking functionality. Proto is never touched at all, as expected, since changing the protocol would change the connection rather than disguise it.

We did not compute a formal rank correlation because thirteen features is a small sample for one and the two tails behave differently from the head. The practical reading is that an attacker rewarded only on the model's output score ends up editing the features that drive that score, and that a defender who knows which features are attacker-controlled can anticipate where the perturbations will land.

Analyst Assistant:

The assistant was exercised with the six example questions from the specification plus follow-ups. Asked why flow 10 was flagged, it calls get_prediction and explain_prediction and reports the baseline and hardened probabilities together with the three largest SHAP contributions and their signs. Asked what the adversarial version of a flow changed, it calls compare_original_vs_adversarial and lists each edited feature with the original value, the new value and the relative shift, along with the drop in malicious probability. Asked about a family, it returns the LOFO figures from Table 5.

We tested the grounding rule by asking for quantities no tool provides, such as the model's AUC on a nonexistent split or metrics for a flow id outside the dataset. In the runs we tried, the assistant reported the tool error rather than inventing a value. This is a small informal check, not a benchmark, and a stronger evaluation would need a labelled set of adversarial questions. The deterministic tools themselves are covered by the pytest suite, which verifies the schema of every return value.

Discussion:

Key Findings:

The headline numbers are easy to state. A good flow classifier loses three quarters of its detections to RL evasion; adding one adversarial flow for every ten clean ones restores almost all of them at a cost of about one extra false alarm per thousand benign flows; and that protection mostly carries over to botnets and networks the model was never hardened against. The exception, Murlo, is as informative as the successes, because it shows that transfer is a property of the particular perturbation distribution rather than a guarantee of the method.

The RQ3 result ties the others together. If RL agents attacking different families and different victims all gravitate to the same handful of features, then adversarial flows from one family are a reasonable proxy for adversarial flows from another, which is what the RQ1 numbers show. Murlo's weak transfer would then correspond to an agent that found a different route, and checking that is the obvious next experiment.

Limitations and Validity:

The adversarial flows are pre-generated against two specific victims. Our hardened model has not been attacked by an agent trained against it, and an adaptive attacker would likely recover some evasion. The results should be read as robustness to the DReLAB perturbation distribution, not to RL evasion in general.

Flow statistics are a lossy summary. An edit that looks harmless in feature space, such as a longer duration, may or may not be achievable at the packet level under real network conditions. We inherit this limitation from the dataset.

Benign flows were subsampled to 40,000 per source, so the false-positive rates are estimates on a sample, and the Zeus/Ares family coincides with a single source, so its LOFO and LOSO rows are not independent evidence. Three seeds are enough to show that the main effects are far larger than seed noise, but not enough to characterise the Murlo variance properly.

The assistant's grounding was checked informally. A language model can still misread a tool output or round a number in a misleading way; the design reduces the chance of fabrication but does not eliminate the need for an analyst to check the underlying tool result when a decision depends on it.

Future Work:

The most useful next step is an adaptive evaluation: train an RL agent against the hardened model and measure how much evasion returns, then iterate. A Murlo-focused study with more seeds and a look at which features its adversarial flows actually change would test the explanation we offered. On the engineering side, the assistant could be connected to a SIEM so that it answers questions about live alerts rather than benchmark flows, and the grounding check could be turned into a proper evaluation set of questions with known tool-derived answers.

Conclusion:

Using the paired structure of DReLAB, we measured how a Random Forest botnet detector fails under reinforcement-learning evasion and how much of that failure adversarial training repairs. The repair is large and cheap: a 10% adversarial mix moves adversarial recall from 25.5% to 95.8% for a 0.12-point increase in false-positive rate. It also generalises, with gains of 36 to 68 recall points on five unseen botnet families and 41 to 61 points on unseen capture sources, though one family, Murlo, gains little. SHAP analysis shows the attacker concentrates its edits on the features the model depends on most, which is consistent with the transfer we observe. A LangGraph assistant built on seven deterministic tools makes these results and the individual predictions behind them queryable in plain language without exposing analysts to invented statistics.

References:

[1] G. Apruzzese, M. Andreolini, M. Marchetti, A. Venturi, and M. Colajanni, “Deep reinforcement adversarial learning against botnet evasion attacks,” IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 1975–1987, 2020.

[2] A. Venturi, G. Apruzzese, M. Andreolini, M. Colajanni, and M. Marchetti, “DReLAB – Deep REinforcement Learning Adversarial Botnet: A benchmark dataset for adversarial attacks against botnet intrusion detection systems,” Data in Brief, vol. 34, 106631, 2021.

[3] S. García, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods,” Computers & Security, vol. 45, pp. 100–123, 2014.

[4] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in Proc. 4th International Conference on Information Systems Security and Privacy (ICISSP), pp. 108–116, 2018.

[5] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, “A survey of network-based intrusion detection data sets,” Computers & Security, vol. 86, pp. 147–167, 2019.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in Proc. International Conference on Learning Representations (ICLR), 2015.

[7] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in Advances in Neural Information Processing Systems, vol. 30, pp. 4765–4774, 2017.

[8] S. M. Lundberg et al., “From local explanations to global understanding with explainable AI for trees,” Nature Machine Intelligence, vol. 2, pp. 56–67, 2020.

[9] S. Yao et al., “ReAct: Synergizing reasoning and acting in language models,” in Proc. International Conference on Learning Representations (ICLR), 2023.

[10] LangChain Inc., “LangGraph documentation,” 2024. https://langchain-ai.github.io/langgraph/

[11] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[12] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794, 2016.