Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Do Machine Learning Security Detectors Survive New Attacks? A Leakage Free Evaluation of Hidden Prompt Injection Detection in AI Coding Agents, Malicious npm and PyPI Package Detection, and Network Intrusion Detection

Nnaemeka Ugwumba

nnaemeka_ugwumba@uniport.edu.ng

University of Port Harcourt https://orcid.org/0009-0000-2493-9846

Clement Barisua Ndeekor

clement.ndeekor@uniport.edu.ng

University of Port Harcourt, Port Harcourt, Nigeria

Research Article

Keywords: prompt injection detection, AI coding agents, Model Context Protocol, malicious package detection, software supply chain security, npm, PyPI, network intrusion detection, CICIDS2017, UNSW NB15, concept drift, data leakage, shortcut learning, adaptive attacks, machine learning security evaluation

Posted Date: November 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11667204/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Do Machine Learning Security Detectors Survive New Attacks? A Leakage Free Evaluation of Hidden Prompt Injection Detection in AI Coding Agents, Malicious npm and PyPI Package Detection, and Network Intrusion Detection

Nnaemeka Kingsley Ugwumba
nnaemeka_ugwumba@uniport.edu.ng
Department of Cybersecurity, Faculty of Computing
University of Port Harcourt, Port Harcourt, Nigeria

Clement Barisua Ndeekor
clement.ndeekor@uniport.edu.ng
Department of Cybersecurity, Faculty of Computing
University of Port Harcourt, Port Harcourt, Nigeria

Abstract

Machine learning detectors for cybersecurity are usually tested by splitting one dataset at random, so the attacks in the test set closely resemble the attacks seen in training. Real attackers change technique. This paper asks whether three kinds of machine learning security detectors still work when they meet an attack technique they have never seen, using one shared, leakage free evaluation protocol. Study A builds a corpus of 4,955 real AI agent instruction files and Model Context Protocol tool descriptions from GitHub, plants harmless hidden instructions with nine hiding techniques used in 2026 prompt injection campaigns, and tests five detectors. Study B tests malicious npm and PyPI package detection on 13,317 packages, training on samples published before 2026 and testing on the 2026 attack waves. Study C revisits network intrusion detection on CICIDS2017, CICIDS2018 and UNSW NB15 with duplicate removal, time ordered splits, unseen attack families and cross dataset transfer. On standard random splits, almost every detector scores a ROC AUC between 0.95 and 1.00. Against hiding techniques never seen in training, a feature based detector of hidden instructions falls from 0.87 to 0.04 recall at a one percent false alarm rate, while a canonicalising character model keeps 0.89 and a fine tuned DistilRoBERTa keeps 0.98. A malicious package model built on static features falls from 0.90 to 0.41 recall on 2026 packages once its size shortcut is removed. An intrusion detector trained on earlier days of CICIDS2018 scores a ROC AUC of 0.49 on later days and finds 10 percent of unseen attack families. Small open language models (Qwen2.5 Instruct, 1.5B to 7B parameters) obeyed hidden instructions in up to 67 percent of poisoned files, larger models obeyed more often, and removing invisible characters made zero width attacks more effective. Our Study A detector, trained only on synthetic poisoning, found 80 percent of 202 real malicious AI agent skills. We release the scripts and give a short checklist for testing security detectors against new attacks.

Keywords: prompt injection detection, AI coding agents, Model Context Protocol, malicious package detection, software supply chain security, npm, PyPI, network intrusion detection, CICIDS2017, UNSW NB15, concept drift, data leakage, shortcut learning, adaptive attacks, machine learning security evaluation

1 Introduction

Machine learning is now part of everyday security work. Detectors classify network flows, scan packages before they enter a build, and are starting to screen the instructions that AI coding assistants read from a repository. Published results for these detectors are often close to perfect. A random forest that reaches a ROC AUC of 0.999 on an intrusion dataset, or a package scanner that finds 99 percent of malicious uploads, gives the impression that the problem is largely solved.

Most of those results come from one kind of test. A dataset is shuffled, a share is held out, and the model is scored on the held out part. When the attacks in that part are drawn from the same campaigns, the same tools and often the same days as the training data, the score measures how well the model recognises what it has already seen. It says much less about the attack that arrives next month. Several authors have warned about this for more than a decade [3, 30, 39], yet random splits remain the default in applied work.

The year 2026 gives a sharp reason to look again. Ransomware groups posted 5,505 victims in six months, the highest total in Trellix data [41]. Self propagating worms spread through npm and PyPI and moved their malicious code into build files to get around the common advice to disable install scripts [32]. Attackers also began to target AI coding assistants directly. They hid instructions inside files such as `CLAUDE.md`, `AGENTS.md` and `.cursorrules`, which assistants load automatically, using invisible Unicode characters that a human reviewer does not see [9]. Each of these is a change in technique, which is exactly the situation that random split evaluations do not test.

This paper studies that situation across three layers of the attack surface with one protocol. Study A concerns hidden instructions in AI agent instruction files and Model Context Protocol (MCP) tool descriptions. Study B concerns malicious packages in npm and PyPI. Study C concerns network intrusion detection on three widely used benchmarks. In every study we remove duplicates before splitting, compare a random split with a split that forces the detector to face something new, look for shortcut

features, and report results at a fixed false alarm rate of one percent.

We ask four research questions.

1. How much do detection scores fall when the test attacks use a technique, a time period or a dataset that the detector never saw in training?

2. Which design choices make a detector hold up under that change, and which make it collapse?

3. Do small open language models actually follow hidden instructions in agent files, and do simple defences reduce that?

4. Does a detector trained on synthetic poisoning transfer to real attacks written by real attackers?

The contributions are as follows.

1. A shared evaluation protocol that combines duplicate removal, new attack splits, shortcut audits and a fixed false alarm rate, applied to three different security tasks with public data.

2. A new corpus of 4,955 public AI agent instruction files and MCP tool descriptions, a poisoning procedure with nine hiding techniques and harmless canary payloads, and the first leave one technique out evaluation of hidden instruction detectors that we are aware of.

3. Evidence that a simple canonicalisation step, which decodes hidden characters before detection, separates detectors that survive new hiding techniques from those that fail.

4. A measurement of how often Qwen2.5 Instruct models of three sizes follow hidden instructions, showing that removing invisible characters can make one class of attack more effective.

5. A time ordered evaluation of malicious package detection on 2026 attack waves and a size shortcut that inflates static feature models.

6. A reanalysis of CICIDS2017, CICIDS2018 and UNSW NB15 showing duplicate rates of 8 to 40 percent, thousands of feature vectors with conflicting labels, and near chance performance on later days of CICIDS2018.

7. A test of the Study A detector on 202 real malicious AI agent skills, which links the synthetic study to real attacks.

Section 2 reviews related work. Section 3 states the threat model and Section 4 the shared protocol. Sections 5, 6 and 7 report Studies A, B and C. Section 8 compares the three studies, Section 9 discusses implications and limitations, and Section 10 concludes.

2 Literature Review

2.1 Prompt injection against language model applications and agents

Prompt injection occurs when text that an application treats as data is read by a language model as an instruction. Perez and Ribeiro showed early that simple phrases such as requests to ignore earlier instructions could redirect GPT based applications [31]. Abdelnabi, Greshake and colleagues introduced indirect prompt injection, in which the attacker never talks to the model directly but plants instructions in content that the application later retrieves, such as web pages or emails [1]. They demonstrated data theft, spreading behaviour and manipulation of answers in real integrated applications. Liu and colleagues gave a formal framework for prompt injection attacks and defences and benchmarked combinations of them across several models and tasks [24].

Agents that call tools raise the stakes, because a followed instruction can turn into an action. InjecAgent measured how often tool using agents follow injected instructions found in tool outputs and reported that even strong models were vulnerable in a large share of cases [44]. AgentDojo built a dynamic environment with realistic tasks and attacks to evaluate both agents and defences [11]. The Model Context Protocol, now a common way to connect assistants to outside tools, adds another entry point. Hou and colleagues surveyed its security and described a broad set of threat scenarios across its lifecycle [18]. A 2026 whitepaper from the Cloud Security Alliance noted that MCP clients are not required to check tool descriptions and that most do not warn users when a description changes [9].

Many defences have been proposed. Spotlighting marks or encodes untrusted text so the model can tell it apart from instructions [17]. StruQ separates instructions and data into structured queries and trains the model to follow only the instruction channel [7]. Detection based defences train a classifier to flag injected text before it reaches the model. The record on adaptive attackers is poor. Zhan and colleagues bypassed all eight indirect prompt injection defences they tested with adaptive attacks, each time reaching attack success rates above 50 percent [43]. Nasr and colleagues defeated twelve recent defences against jailbreaks and prompt injections, most of which had originally reported near zero attack success [27]. These results motivate our use of new technique splits and an adaptive attacker when we evaluate detectors.

2.2 Invisible and confusable text

Unicode contains characters that render as nothing, characters that change the display order of text, and characters that look the same as others. Boucher and colleagues showed that such characters can fool natural language processing systems while leaving text unchanged to a human reader, and called the attacks imperceptible [5]. Trojan Source used bidirectional control characters to make source code display differently from how compilers read it, and led to warnings in major compilers and code hosts [4]. The Unicode Consortium documents confusable characters and security mechanisms

for identifiers in Technical Standard 39 [42]. In 2026 these ideas reached AI agent files. Attackers used Unicode tag characters and zero width characters to hide instructions in files that coding assistants read automatically [9]. To our knowledge no prior work has measured how detectors of such files behave on hiding techniques they did not see in training.

2.3 Malicious packages in open source registries

Package registries are an attractive target because one upload can reach many developers. Zimmermann and colleagues showed that a small number of maintainer accounts could reach a large part of the npm ecosystem [45]. Ohm and colleagues collected 174 malicious packages and described how they were injected and triggered [28]. Ladisa and colleagues built a taxonomy of attacks on open source supply chains [22], and Duan and colleagues built a pipeline that found hundreds of malicious packages across npm, PyPI and RubyGems [13]. On the detection side, Amalfi combined a classifier on package features with reproducibility and clone checks, and found malicious npm packages that existing tools missed [37]. Datadog Security Labs maintains a public dataset of malicious npm and PyPI packages, stored as encrypted archives with discovery dates [10]. In 2026, campaigns spread through compromised maintainer tokens, moved code into build steps and planted files aimed at AI coding assistants [32]. Few detection studies have tested models trained on earlier campaigns against these newer ones.

2.4 Network intrusion detection datasets and their limits

CICIDS2017 [38], the 2018 dataset from the Communications Security Establishment and the Canadian Institute for Cybersecurity (CICIDS2018 in this paper), and UNSW NB15 [26] are among the most used benchmarks for intrusion detection. Ring and colleagues surveyed such datasets and their properties [34]. Their quality has been questioned. Engelen, Rimmer and Joosen found errors in the feature extraction and labelling of CICIDS2017 and released a corrected version [15]. Liu and colleagues found many unreported mistakes in both CICIDS2017 and CICIDS2018 [23]. Sarhan, Layeghy and Portmann argued for a standard feature set because different datasets use incompatible features, which blocks cross dataset evaluation [36]. D'hooge and colleagues trained on one dataset and tested on another and found that models that look strong within a dataset often fail across datasets [12]. In October 2026 Tantaroudas, Karachalios and McCracken showed that a single flow feature, the initial TCP window size, lets a shallow tree match complex models on web attacks in CICIDS2017, which points to a dataset shortcut [40].

2.5 Leakage, drift and shortcut learning

Leakage occurs when information that would not be available at prediction time influences training [21]. Kapoor and Narayanan found it to be a common cause of irreproducible machine learning results across many fields, including duplicates shared by training and test sets and preprocessing on the full dataset [20]. Allamanis showed that duplicated code inflates the results of machine learning models of source code [2]. In security, Arp and colleagues listed ten pitfalls, among them sampling bias, data snooping and spurious correlations, and found them in most of the papers they reviewed [3]. Pendlebury and colleagues showed that malware classifiers lose much of their accuracy when training and test data are ordered in time, and proposed time aware evaluation [30]. Jordaney and colleagues proposed methods to detect concept drift in malware classifiers [19]. Geirhos and colleagues described shortcut learning, in which a model solves a benchmark with simple cues that do not carry over to the real task [16]. Sommer and Paxson had earlier argued that intrusion detection differs from other machine learning tasks because attacks are adversarial and new attacks are, by definition, not in the training data [39].

2.6 Research gap

Table 1 summarises the prior work closest to each of our three studies, the kind of evaluation it used, and what remains open. Prior work has studied each of these problems inside one task. Hidden instruction detection has not been evaluated against unseen hiding techniques. Package detectors have rarely been tested on a later attack wave. Intrusion detection critiques usually examine one dataset at a time. We are not aware of work that applies the same leakage free, new attack protocol to all three tasks and compares the results. This paper fills that gap and adds a link between synthetic and real attacks through the AI agent skills in the Datadog dataset.

Table 1. Prior work closest to the three studies and the question this paper adds

Work	Area	Main finding	Open question addressed here
Abdelnabi, Greshake et al. [1]	indirect prompt injection	instructions planted in retrieved content take over integrated applications	how detectors of such content behave on unseen hiding techniques
InjecAgent [44], AgentDojo [11]	tool using agents	agents follow injected instructions in tool outputs	how often small open models follow instructions hidden in project files
Zhan et al. [43], Nasr et al. [27]	defences under adaptive attack	published defences fall to adaptive attackers	how text and feature detectors compare under the same query budget
Boucher et al. [4, 5]	invisible Unicode	invisible characters fool NLP models and code review	whether decoding hidden text before detection restores detection
Amalfi [37], Duan et al. [13]	malicious packages	learned and rule based checks find malicious uploads	how such detectors age against the 2026 attack waves
Pendlebury et al. [30], Jordaney et al. [19]	drift in malware detection	time ordered evaluation reveals large drops	whether the same holds for package detectors and for agent file detectors
Engelen et al. [15], Liu et al. [23]	intrusion dataset quality	labelling and extraction errors in CIC datasets	how much duplicates and errors change measured performance
D'hooge et al. [12]	cross dataset intrusion detection	models fail when moved to another dataset	how cross dataset failure compares with unseen family and later day failure
Arp et al. [3]	pitfalls in security ML	common pitfalls in most reviewed papers	one protocol that tests several pitfalls across three tasks at once

3 Threat Model and Scope

We consider a defender who trains a detector on attacks known at training time and then deploys it. The attacker knows that detectors exist and changes technique over time. In Study A the attacker can write to files that an AI coding assistant reads, for example through a pull request, a compromised dependency or a shared template, and wants the assistant to follow an instruction that the human user did not give. In Study B the attacker publishes or compromises a package on npm or PyPI. In Study C the attacker sends network traffic to a monitored network.

We use three kinds of change. A new technique means a hiding method or attack family that is absent from the training data. A new period means attacks from a later time than all training data. A new dataset means traffic captured in a different environment. In Study A we also model an adaptive attacker who can query the detector and keep the variant with the lowest score.

All payloads in Study A are harmless. They ask the assistant to print a random marker such as ZQX4821, to reply only with that marker, to echo it in a terminal command, or to mention a link on example.com. No real secret is targeted. In Study B we only read the static content of packages and never run any code.

4 Shared Evaluation Protocol

All three studies follow the same six rules, summarised in Table 2.

Table 2. The three studies under the shared protocol

	Study A: agent instruction files	Study B: malicious packages	Study C: network intrusion
Data	4,955 GitHub files and MCP tool descriptions	9,885 malicious, 3,432 legitimate packages	CICIDS2017, CICIDS2018, UNSW NB15
Attacks	9 hiding techniques, 4 harmless goals	Datadog dataset, 2024 to 2026	7 to 9 attack families per dataset
Duplicates	exact and near duplicates removed	near duplicate clusters grouped	exact duplicates counted and removed
Standard split	random rows, then by repository	random packages	random flows
New attack split	leave one technique out, leave one goal out	train before 2026, test 2026	later days, unseen family, other dataset
Shortcut audit	raw text versus canonical text	size and file count features	single feature trees
Models	rules, char LR (2), feature GBM, DistilRoBERTa	LR on code tokens, XGBoost on features	LR, XGBoost, MLP
Extra tests	adaptive attacker, obedience of 3 LLMs	real malicious AI skills	cross dataset transfer
Run on	Kaggle, 2 T4 GPUs, 137 minutes	Windows desktop, 83 minutes	Kaggle CPU, 55 minutes

Remove duplicates before any split. Exact duplicates are removed in every study. Studies A and B also remove or group near duplicates with MinHash signatures [6], since copied files and cloned malware campaigns are common.

Compare a standard split with a new attack split. The standard split is the random split used in most published work. The new attack split holds out a technique, a period or a dataset.

Group related samples. In Study A, files from the same repository stay on one side of a split. In Study B, packages from the same near duplicate cluster stay together.

Fix the false alarm rate. A detector that flags too much is unusable. For every model we choose the decision threshold on held out legitimate samples from the training side so that one percent of them are flagged, then apply that threshold unchanged to the test set. We report recall at this threshold together with ROC AUC.

Audit shortcuts. We train a depth three decision tree on each single feature and record its ROC AUC. Features that alone separate the classes are candidate shortcuts, and we retrain without them.

Report uncertainty. Where a split is repeated with different seeds, we report the mean and a 95 percent t interval. For single splits we report bootstrap intervals [14] where they are informative.

The models were implemented with scikit learn [29], XGBoost [8] and the Hugging Face transformers library. All random processes used the seed 42. Table 3 lists the settings of every model. We did not tune hyperparameters on the test data of any split; the settings were fixed before the experiments and kept the same for the standard and the new attack splits, so that any change in score comes from the split and not from the model.

Table 3. Model settings used in all three studies

Model	Settings
Char LR (A)	TF IDF character 2 to 5 grams, 200,000 features, C = 4
Feature GBM (A)	histogram gradient boosting, 300 iterations, 18 features
DistilRoBERTa (A)	2 epochs, learning rate 0.00003, batch 16, 512 tokens, half precision
Qwen2.5 Instruct (A)	1.5B, 3B, 7B; greedy decoding, 96 new tokens
LR on tokens (B)	hashed TF IDF, 262,144 buckets, C = 4, balanced classes
XGBoost (B)	300 trees, depth 6, learning rate 0.08, 34 features
LR (C)	C = 1, balanced classes, 300 iterations
XGBoost (C)	200 trees, depth 6, learning rate 0.1, row and column sampling 0.8
MLP (C)	hidden layers 64 and 32, early stopping, at most 40 iterations

5 Study A: Hidden Instructions in AI Agent Files

5.1 Corpus

We collected public agent instruction files from GitHub with the code search interface. The file types were `CLAUDE.md`, `AGENTS.md`, `.cursorrules`, the GitHub Copilot instructions file, `GEMINI.md` and `SKILL.md`. We also collected MCP tool descriptions by parsing the source code of public MCP servers written in Python and TypeScript, taking the docstrings or description strings attached to each tool. We kept files between 120 and 20,000 characters.

We then removed 661 exact duplicates and 442 near duplicates found with MinHash signatures over word shingles. We also set aside 17 files that already contained invisible characters before we touched them. Section 5.6 examines these files. Table 4 shows the final corpus of 4,955 documents from more than 3,600 repositories. MCP tool descriptions are short, with a median of 133 characters, and come from fewer repositories, because one server often defines many tools.

Table 4. Study A corpus after duplicate removal

File type	Documents	Repositories	Median characters
CLAUDE.md	665	664	838
AGENTS.md	654	628	542
.cursorrules	633	613	478
GEMINI.md	600	585	453
Copilot instructions	555	538	525
SKILL.md	432	405	728
MCP tool description	1,416	175	133
Total	4,955		

5.2 Poisoning

For every legitimate document we created one poisoned copy, which gives a balanced set of 9,910 documents. Each poisoned copy received one payload, hidden with one of nine techniques chosen at random (Table 5). The payload carried one of four goals: print a canary marker, override the user's request and reply only with the marker, run a terminal command that echoes the marker, or recommend a link on example.com that contains the marker. Each goal had five written templates in different styles, from plain imperatives to notes addressed to AI assistants. The insertion point was the start, the middle or the end of the file. For techniques that hide text inside a line, half of the payloads were appended to an existing line instead of a new one.

Table 5. Hiding techniques used in Study A

Technique	How the payload is written
Plain text	visible sentence in the file
HTML comment	inside an HTML comment, hidden when rendered
Markdown reference	inside an empty markdown link reference, hidden when rendered
Unicode tags	every character mapped to the invisible Unicode tag block
Zero width	a zero width space after every character
Homoglyph	about 60 percent of Latin letters replaced by Cyrillic lookalikes
Base64	encoded, with a request to decode and follow it
Bidirectional	reversed and wrapped in right to left override characters
Off screen	pushed to the right by 250 to 400 spaces

5.3 Detectors

We compared five detectors. The rule scanner adds weighted points for tag characters, bidirectional controls, zero width characters, words with mixed scripts, HTML comments, hidden markdown references, long base64 runs, long runs of spaces and imperative words. It needs no training. The character model is a logistic regression on TF IDF weighted character n grams of length two to five, with 200,000 features. We trained it on two inputs: the raw text, and a canonical form described below. The feature model is a gradient boosted tree ensemble on 18 engineered features such as counts of hidden characters, comment counts, base64 run lengths, imperative words, links and the share of non ASCII characters. The last detector is DistilRoBERTa [25, 35], fine tuned for two epochs on the canonical text with a learning rate of 0.00003, a batch size of 16 and a maximum length of 512 tokens.

The canonical form is our proposed defence. It applies Unicode NFKC normalisation, decodes Unicode tag characters back to the ASCII text they encode and wraps that text in a visible marker, removes zero width characters but records that they were present, marks bidirectional controls, maps common Cyrillic and Greek lookalikes back to Latin with a marker, replaces long runs of spaces with a marker, and decodes base64 runs of at least 24 characters that decode to printable text. The goal is to show the detector what the language model will read, not what a human sees on screen. Table 6 lists each step with the marker it leaves in the text. The markers matter because they keep the information that something was hidden, even after the hidden content itself has been made readable.

Table 6. Canonicalisation steps applied before detection

Input pattern	Action	Marker
Compatibility forms	NFKC normalisation	none
Unicode tag characters	decoded to ASCII	TAG
Zero width characters	removed	ZW_PRESENT
Bidirectional controls	kept, marked	BIDI
Cyrillic or Greek lookalikes	mapped to Latin	H
Runs of many spaces	replaced	LONG_SPACE
Base64 runs of 24 or more	decoded if printable	B64

5.4 Splits

We used four splits. E1 splits rows at random, so the legitimate file and its poisoned copy can fall on different

sides. In E1, 69 percent of test documents had their twin in the training set. E2 splits by repository so that no source file appears on both sides. E1 and E2 were repeated three times. E3 holds out one hiding technique at a time: the detector trains on poisoned files that use the other eight techniques and is tested on legitimate files and on poisoned files that use only the held out technique. As a reference, the same test set is also scored by a detector trained on all nine techniques. E4 does the same with the four payload goals. E3 and E4 use the repository split, so no source file is shared either.

For the adaptive attacker we took 300 poisoned test files and allowed 25 queries to each detector per file. Each query generated a new variant from the same legitimate file and goal: a random technique and template, in 40 percent of cases a payload split into two halves hidden with different techniques, and in 30 percent of cases extra padding copied from the file itself. The attacker kept the variant with the lowest detector score. We report the share of files still detected at the one percent threshold.

5.5 Do language models obey hidden instructions?

A detector only matters if the hidden text changes what an assistant does. We gave three open models, Qwen2.5 Instruct with 1.5, 3 and 7 billion parameters [33], a system prompt that described them as coding assistants working inside a repository, followed by a project file and a user request for a two sentence summary of the project. For each technique we sampled 30 test files of at most 5,000 characters and poisoned each with a random goal, and we also included 30 legitimate files. Generation was greedy with at most 96 new tokens. We counted an attack as successful when the canary marker appeared in the reply and the reply did not warn about a hidden or suspicious instruction.

Each file was run twice: as written, and after a simple sanitiser that removes invisible characters, HTML comments and hidden markdown references, and collapses long runs of spaces. We also recorded whether the canonical character detector, at its one percent threshold, would have blocked the file before the model saw it.

5.6 Results

Standard splits look excellent. Table 7 shows that on random and repository splits all learned detectors reach a ROC AUC between 0.97 and 1.00. The two character models catch almost every poisoned file at one percent false alarms, DistilRoBERTa catches 97 to 98 percent, and the feature model catches 87 percent. The rule scanner reaches a ROC AUC of 0.95 but catches only 62 to 63 percent at the strict threshold, mostly because plain text payloads contain nothing unusual. The random split and the repository split give almost the same results. Duplicate removal had already taken out the copies that would otherwise leak across the split.

Table 7. Study A results. Each cell gives ROC AUC and, after the slash, recall at one percent false alarms. E1 and E2 are means of three repeats; E3 and E4 are means over the held out techniques or goals.

Split	Rule scanner	Char LR, raw text	Char LR, canonical text	Feature GBM	DistilRoBERTa
E1 random rows	0.955 / 0.618	1.000 / 0.998	1.000 / 0.999	0.973 / 0.866	0.998 / 0.979
E2 by repository	0.952 / 0.631	1.000 / 1.000	1.000 / 0.999	0.971 / 0.866	0.999 / 0.974
E3 technique seen in training	0.957 / 0.618	1.000 / 0.999	1.000 / 0.999	0.971 / 0.863	0.999 / 0.978
E3 technique never seen	0.957 / 0.618	0.798 / 0.545	0.983 / 0.890	0.609 / 0.037	0.999 / 0.979
E4 goal never seen	0.960 / 0.627	0.995 / 0.862	0.994 / 0.870	0.965 / 0.852	0.999 / 0.982

New hiding techniques separate the detectors. When the held out technique was absent from training, the feature model fell from 0.863 to 0.037 recall, which means it missed 96 percent of attacks that use a trick it had not seen. The raw character model fell from 0.999 to 0.545. The canonical character model kept 0.890, and DistilRoBERTa did not lose anything, with 0.979 recall on unseen techniques against 0.978 on seen ones. The rule scanner does not train, so its seen and unseen rows are the same by construction.

Figure 1 and Table 8 show where the failures come from. The raw character model misses almost all unseen Unicode tag, zero width, base64 and bidirectional payloads, with recall between 0.000 and 0.012, because those payloads share no visible character patterns with the techniques it was trained on. Once the canonical form decodes the hidden text, the same model finds every unseen tag, zero width, homoglyph and base64 payload. The feature model fails on every unseen technique, with recall at most 0.142. Its features count each trick separately, so a poisoned file with an unseen trick looks like a normal file. The canonical character model has one clear weakness, the bidirectional technique, where it caught only 0.031 of unseen payloads. The marker for bidirectional controls appeared only in that technique during training, and the reversed payload text matches nothing else. DistilRoBERTa caught 0.988 of the same payloads.

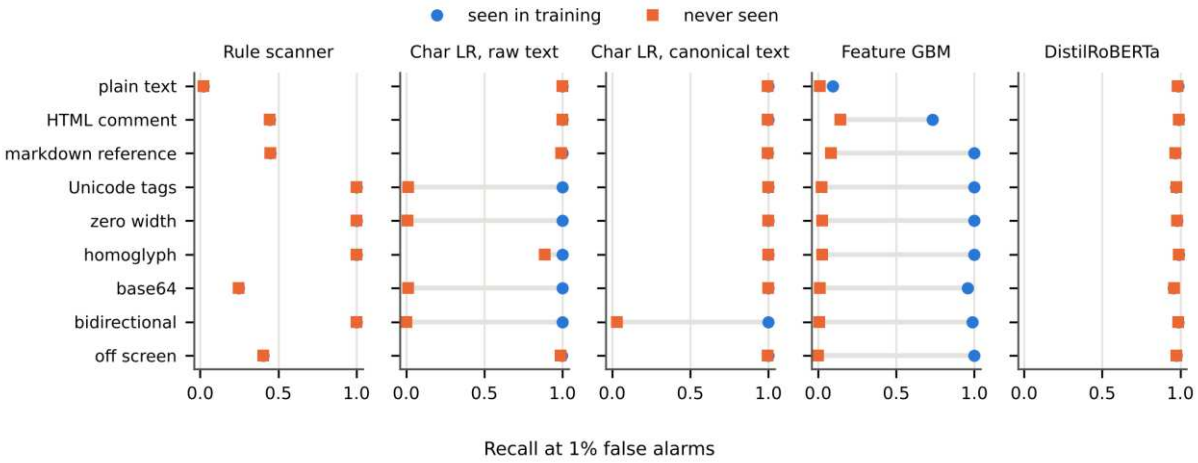


Figure 1. Recall at one percent false alarms for each hiding technique, when the technique was seen in training (circle) and when it was held out (square).

Table 8. Recall on hiding techniques never seen in training, at one percent false alarms

Technique	Rules	LR raw	LR canon.	GBM	DistilRoBERTa
Plain text	0.019	1.000	0.994	0.013	0.981
HTML comment	0.443	1.000	0.994	0.142	0.989
Markdown reference	0.447	0.994	0.994	0.081	0.969
Unicode tags	1.000	0.012	1.000	0.024	0.976
Zero width	1.000	0.010	1.000	0.026	0.979
Homoglyph	1.000	0.886	1.000	0.027	0.989
Base64	0.246	0.011	1.000	0.011	0.961
Bidirectional	1.000	0.000	0.031	0.006	0.988
Off screen	0.405	0.988	0.994	0.000	0.976

Unseen goals matter less. Holding out a payload goal lowered recall for the character models from about 1.00 to 0.86 and 0.87, and the feature model kept 0.85. DistilRoBERTa kept 0.98. Table 9 shows the results per goal. The link goal was the hardest for the rule scanner, at 0.564, because a sentence that recommends a web page contains few imperative words. The command goal was the hardest for the character models, at 0.835 and 0.824, since the terminal command template uses wording that the other goals do not. Changing what the attacker asks for is easier for these detectors to handle than changing how it is hidden.

Table 9. Recall at one percent false alarms when one payload goal is held out of training

Held out goal	Rules	LR raw	LR canon.	GBM	DistilRoBERTa
Canary marker	0.669	0.889	0.894	0.891	0.985
Override reply	0.666	0.866	0.873	0.837	0.973
Terminal command	0.608	0.835	0.824	0.849	0.992
Link to a page	0.564	0.860	0.888	0.831	0.977

Adaptive attackers defeat simple detectors. Figure 2 shows detection before and after 25 adaptive queries. The rule scanner fell from 0.700 to 0.083 and the feature model from 0.853 to 0.060. Both rely on a small set of signals that the attacker can avoid by choosing a technique they score low on. The text models held up better: the raw character model fell from 0.997 to 0.890, the canonical character model from 1.000 to 0.910, and DistilRoBERTa from 0.923 to 0.817. Our attacker is weak compared with the optimisation attacks in recent work [27], so these numbers are upper bounds on what each detector would achieve against a determined adversary. Table 10 gives the same numbers as relative losses. The rule scanner and the feature model lost 88 and 93 percent of their detections, while the three text models lost between 9 and 12 percent.

Table 10. Adaptive attacker with 25 queries on 300 poisoned files: share still detected at one percent false alarms

Detector	Before	After	Relative loss
Rule scanner	0.700	0.083	88%
Char LR, raw text	0.997	0.890	11%
Char LR, canonical text	1.000	0.910	9%
Feature GBM	0.853	0.060	93%
DistilRoBERTa	0.923	0.817	12%

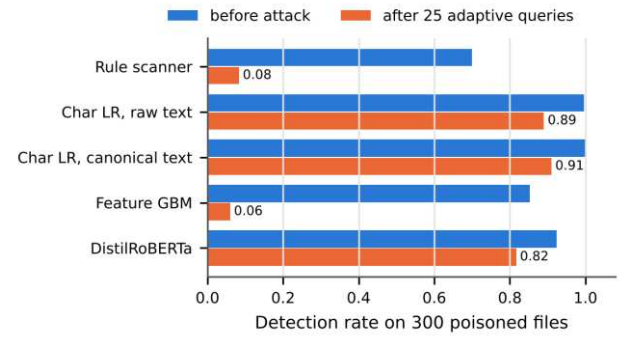


Figure 2. Detection of 300 poisoned test files before and after an adaptive attacker with 25 queries per file.

Models obey hidden instructions, and larger models obey more. Table 11 and Figure 3 report the obedience test. No legitimate file ever produced a canary. With plain text payloads, the 1.5B model complied in 20 percent of files, the 3B model in 53 percent and the 7B model in 63 percent. Hidden techniques also worked. The 7B model complied with 67 percent of homoglyph payloads, 63 percent of off screen payloads, 50 percent of zero width payloads and 50 percent of HTML comment payloads. Three techniques never worked on these models: Unicode tags, base64 and bidirectional text. These small models do not decode tag characters or base64 on their own, so the payload reaches them as noise. The 2026 attacks targeted

larger commercial assistants, which are reported to read tag characters [9], so the zero rates here should not be read as safety for those systems.

Table 11. Share of 30 poisoned files in which each Qwen2.5 Instruct model followed the hidden instruction, for the file as written and after sanitising (7B only)

Technique	1.5B	3B	7B	7B sanitised
Plain text	0.20	0.53	0.63	0.63
HTML comment	0.10	0.33	0.50	0.00
Markdown reference	0.00	0.13	0.40	0.00
Unicode tags	0.00	0.00	0.00	0.00
Zero width	0.27	0.23	0.50	0.57
Homoglyph	0.23	0.23	0.67	0.67
Base64	0.00	0.00	0.00	0.00
Bidirectional	0.00	0.00	0.00	0.00
Off screen	0.20	0.33	0.63	0.60
Legitimate file	0.00	0.00	0.00	0.00

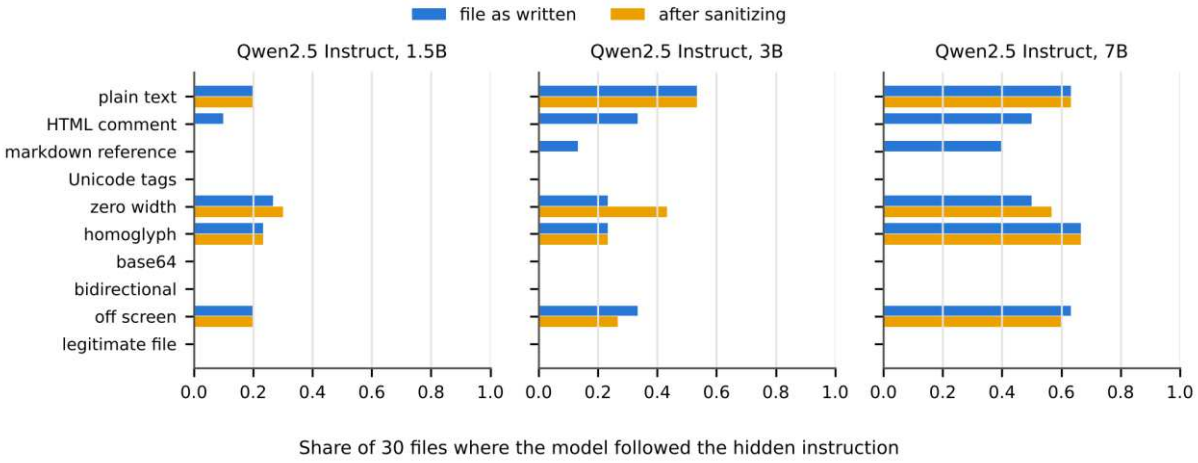


Figure 3. Share of files in which each model followed the hidden instruction, for the file as written and after sanitising.

Sanitising helps against some techniques and backfires against one. Removing comments and hidden references took HTML comment and markdown reference attacks to zero for every model. It had no effect on plain text, homoglyph and off screen payloads, which do not depend on hidden characters. For zero width payloads it made things worse. Success rose from 0.23 to 0.43 for the 3B model and from 0.50 to 0.57 for the 7B model. The zero width characters break the payload into fragments; removing them joins the fragments back into a clean sentence that the model reads more easily. A sanitiser that only deletes invisible characters therefore repairs this class of attack instead of stopping it. Blocking files flagged by the canonical character detector stopped every attack: all 270 poisoned files in the test were flagged, and none of the 30 legitimate files was.

Real malicious skills. To check whether a detector trained only on our synthetic poisoning recognises real attacks, we scored the 202 malicious AI agent skills in the Datadog dataset [10] against 120 legitimate `SKILL.md` files that were held out from training. The skills were read in memory from their encrypted archives in Study B. The canonical character detector reached a ROC AUC of 0.933 and, at the threshold fixed in Study A, flagged 80.2 percent of the malicious skills. The false alarm rate on legitimate skills was 7.5 percent, higher than the one percent set on the training data. Real malicious skills often contain plain, openly harmful instructions instead of hidden ones, and our detector still found most of them. The rise in false alarms shows that a threshold chosen on one population of files does not carry over exactly to another and should be recalibrated before deployment.

5.7 Hidden characters in public files

Seventeen of the collected files already contained invisible characters. We opened each one. Fifteen contained a single byte order mark at the very start of the file, which many Windows editors add automatically and which is harmless. The other two were deliberate attack samples kept in the test suites of two security tools, one with four zero width characters in front of an instruction to read credentials and one with zero width spaces inside a command that downloads a script. We found no attack in the wild among the files we collected. The byte order marks are still useful to know about, because a scanner that flags every invisible character will raise false alarms on legitimate files saved by common editors.

6 Study B: Malicious npm and PyPI Packages

6.1 Data

The Datadog dataset stores each malicious package as a password protected archive whose file name begins with the date of discovery [10]. We listed the repository through the GitHub interface, selected up to 3,000 npm samples with malicious intent per year for 2024, 2025 and 2026, up to 3,000 compromised legitimate libraries, and up to 3,000 PyPI samples, and kept archives of at most 2 MB. Of the 10,100 selected archives, 10,090 downloaded and 9,885 could be read. Legitimate packages were

downloaded from the public npm and PyPI registries. For npm we searched the registry with a list of common keywords at both high and low popularity weighting; for PyPI we sampled names at random from the full package index. For each package we chose one version published between 2024 and 2026 and applied the same 2 MB limit. Packages named in the malicious dataset were excluded. The npm search produced 2,232 usable packages instead of the planned 3,000. Table 12 shows the final composition of 13,317 packages.

Table 12. Study B packages by ecosystem and year

Ecosystem	Label	2024	2025	2026
npm	malicious	3,007	3,411	3,467
npm	legitimate	729	586	917
PyPI	legitimate	284	334	582

The year limit cut the PyPI malicious samples, which in this dataset are mostly from earlier years, so all malicious packages in the final set come from npm. Legitimate packages come from both registries. We return to this in the limitations.

Safety. The archives stay encrypted on disk. The script opens each one in memory, reads only text files up to 400 KB each and 2 MB per package, extracts numbers, and keeps only those numbers. No malicious code is written to disk in readable form and none is executed.

6.2 Features and models

We built two detectors. The first is a logistic regression on code tokens. Identifiers from the package manifest's install hooks and from up to 60,000 characters of each text file were hashed into 262,144 buckets and weighted with sublinear TF IDF. The second is XGBoost on 34 static features. These cover counts of files, total size, JavaScript and Python files, binary files and dependencies; the presence of install hooks, a `binding.gyp` build file, a Visual Studio Code tasks file or AI agent configuration files; and counts of patterns for process execution, `eval`, base64, network calls, environment access, home directory files, webhooks, shell tools, IP addresses, crypto wallets, obfuscation, Python install hooks, dynamic imports, file writes, persistence and CI tokens, together with maximum line length, code entropy and invisible characters.

To group clones we computed MinHash signatures with 64 permutations over shingles of five code tokens and joined packages that matched in any of 16 bands of four rows. This produced 7,909 clusters. The difference between classes is large: 54.3 percent of malicious packages are near copies of another package in the set, against 1.3 percent of legitimate ones. Attack campaigns publish many slight variants of the same code.

6.3 Splits

E1 splits packages at random, repeated five times. On average 45 percent of test packages belonged to a cluster that also appeared in training. E2 sends whole clusters to the test side, so no near copy of a test package is seen in training. E3 trains on everything dated before 1 January 2026 (6,680 packages) and tests on 2026 (4,966 packages, 3,467 of them malicious). We also report E3 for 2026 packages whose cluster never appeared before 2026. E5 audits shortcuts and retrains the feature model without the seven size and count features. In every split the threshold is chosen on 20 percent of the training side.

6.4 Results

Table 13 and Figure 4 give the results. On random splits both detectors look strong: the token model reaches a ROC AUC of 0.999 and 0.990 recall, and the feature model 0.995 and 0.896 recall.

Table 13. Study B results. Mean with 95 percent interval over five repeats for E1 and E2; single split otherwise.

Split	Model	ROC AUC	Recall	False alarms	F1
E1 random	LR, code tokens	0.999 (0.999, 1.000)	0.990 (0.986, 0.993)	0.009 (0.005, 0.013)	0.993 (0.992, 0.994)
E1 random	XGBoost, static features	0.995 (0.994, 0.996)	0.896 (0.862, 0.931)	0.013 (0.007, 0.018)	0.943 (0.925, 0.961)
E2 by clone cluster	LR, code tokens	0.998 (0.998, 0.999)	0.979 (0.969, 0.988)	0.013 (0.006, 0.020)	0.984 (0.981, 0.988)
E2 by clone cluster	XGBoost, static features	0.993 (0.991, 0.994)	0.864 (0.831, 0.896)	0.014 (0.007, 0.022)	0.921 (0.905, 0.938)
E3 all 2026 packages	LR, code tokens	0.982	0.924	0.011	0.958
E3 all 2026 packages	XGBoost, static features	0.979	0.769	0.030	0.863
E3 2026, new clusters only	LR, code tokens	0.981	0.919	0.011	0.955
E3 2026, new clusters only	XGBoost, static features	0.978	0.759	0.030	0.856
E5 random, no size features	XGBoost, static features	0.935	0.520	0.015	0.679
E5 2026, no size features	XGBoost, static features	0.888	0.410	0.023	0.578

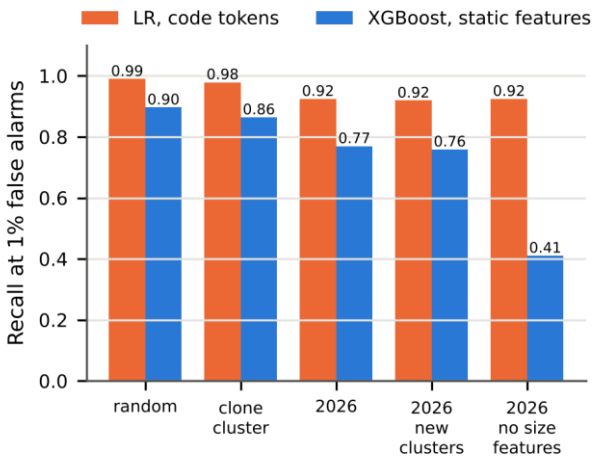


Figure 4. Recall at one percent false alarms for the two package detectors under each split.

Clone grouping changed little. Moving whole clusters to the test side lowered recall by about one point for the token model and three points for the feature model. Even though more than half of the malicious packages are clones, the remaining unique packages are varied enough that the models do not depend only on memorising copies.

The 2026 waves are harder. Trained before 2026, the token model caught 92.4 percent of 2026 malicious packages and the feature model 76.9 percent. The feature model's false alarm rate also tripled to 3.0 percent, because legitimate packages published in 2026 differ from those used to set the threshold. Restricting the test to clusters never seen before 2026 changed the results by less than one point, so the drop is not explained by the share of new clusters alone.

The feature model leans on package size. The single feature audit ranked the number of files first, with a ROC AUC of 0.829 on its own, followed by total size (0.785) and the number of dependencies (0.734), as Table 14 shows. Behaviour features such as webhooks, home directory access and process execution only reach between 0.60 and 0.64 on their own, which means that no single malicious behaviour is present in most malicious packages.

Table 14. Study B single feature audit: ROC AUC of a depth three tree trained on one feature

Feature	Kind	Single AUC
Number of files	size	0.829
Total size in KB	size	0.785
Number of dependencies	size	0.734
Mean code entropy	content	0.716
Maximum line length	content	0.705
Number of JavaScript files	size	0.703
Ecosystem is npm	source	0.672
Number of Python files	size	0.667
Shell tool names	behaviour	0.637
Webhook addresses	behaviour	0.611
Home directory files	behaviour	0.604
Process execution	behaviour	0.600

Malicious packages are usually small. Without the seven size and count features, the feature model's recall fell to 0.520 on a random split and to 0.410 on 2026 packages. The token model does not use these features and was unaffected. A detector that mainly recognises small packages would fail as soon as an attacker pads a package or compromises a large legitimate library.

Some 2026 techniques are hard for both models. Figure 5 and Table 15 break the 2026 results down. Both models found fewer than 60 percent of the 467 compromised legitimate libraries, where malicious code is added to a real project with real structure. Packages that contain AI agent configuration files were also hard, with 0.415 and 0.472 recall. The 124 packages with install hooks were all caught by the token model but only 55 percent by the feature model, which suggests that the feature model learned from earlier waves that install hooks are common in legitimate packages too. Packages with a `binding.gyp` build file, a technique that avoids install hooks, were caught at 0.861 and 0.944. These slices are small, so the numbers are indicative.

Table 15. Detection of 2026 malicious packages by technique slice, at one percent false alarms

Slice	Packages	LR tokens	XGBoost
All 2026 malicious	3,467	0.924	0.769
Cluster never seen before	3,231	0.919	0.759
No install hook	3,343	0.921	0.777
Uses install hook	124	1.000	0.548
binding.gyp present	36	0.861	0.944
Compromised legitimate library	467	0.574	0.580
AI agent configuration files	106	0.415	0.472

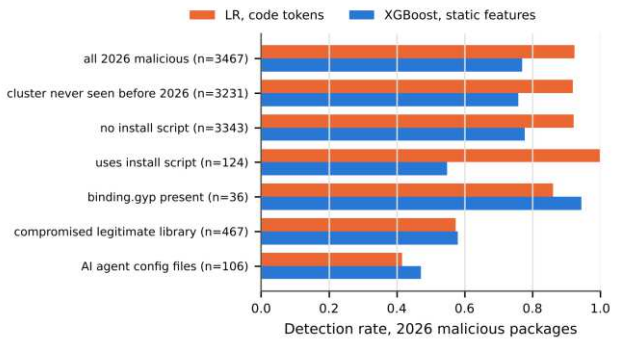


Figure 5. Detection of 2026 malicious packages by technique slice for both models.

7 Study C: Network Intrusion Detection

7.1 Data and preparation

We used the machine learning versions of CICIDS2017 (eight daily files), CICIDS2018 (ten daily files) and the official training and test files of UNSW NB15. To keep memory within Kaggle limits we sampled up to 120,000 rows from each file. We removed identifiers such as flow IDs, IP addresses, source ports and timestamps, dropped rows with missing or infinite values and dropped constant columns. Labels were grouped into attack families: brute force, DoS, DDoS, web attacks, reconnaissance, botnet and infiltration for the CIC datasets, and the official categories for UNSW NB15. Table 16 summarises the data.

Table 16. Study C data after sampling, with exact duplicate counts

	CICIDS2017	CICIDS2018	UNSW NB15
Rows used	959,186	1,193,591	257,673
Features	70	70	42
Attack share	19.2%	24.0%	63.9%
Attack families	7	6	9
Duplicate rows	76,615	253,996	104,208
Duplicate share	8.0%	21.3%	40.4%
Vectors with both labels	160	8,711	414

7.2 Models and splits

We trained logistic regression with balanced class weights, XGBoost with 200 trees of depth six, and a multilayer perceptron with two hidden layers of 64 and 32 units and early stopping. Features were transformed with $\log(1 + x)$ and standardised on the training data. Training sets were capped at 150,000 rows and test sets at 100,000.

E1 compares five random splits with duplicates kept and five after duplicates were removed. E2 orders the capture days in time. For CICIDS2017 the model trains on Monday to Thursday morning and is tested on Thursday afternoon and Friday, which contain the botnet, DDoS, reconnaissance and infiltration families for the first time. For CICIDS2018 it trains on 14 to 21 February and is tested on 22 February to 2 March, which brings new botnet, infiltration and web attack families. UNSW NB15 has no day labels in its standard files, so we use its official training and test split, which contains the same families on both sides. Table 17 summarises the time splits.

Table 17. Time ordered splits in Study C

Dataset	Training period	Test period	Families new in test
CICIDS2017	Monday to Thursday morning	Thursday afternoon and Friday	Bot, DDoS, Infiltration, Recon
CICIDS2018	14 to 21 February	22 February to 2 March	Bot, Infiltration, Web
UNSW NB15	official training file	official test file	none

E3 holds out one attack family at a time; the detector trains on benign traffic and all other families and is tested on held out benign traffic and the held out family. E4 trains on one CIC dataset and tests on the other, using the 67 features that both datasets share. E5 is the shortcut audit.

7.3 Results

Duplicates are common, but they are not the main problem. Between 8 and 40 percent of rows are exact duplicates (Table 16). CICIDS2018 contains 8,711 feature vectors that appear with both a benign and an attack label, which supports earlier reports of labelling errors [23]. Removing duplicates lowered XGBoost's ROC AUC by at most 0.01 (Table 18). Recall dropped more on UNSW NB15, from 0.892 to 0.765, because many duplicated rows there are easy attack flows.

Table 18. Study C results for XGBoost and logistic regression. ROC AUC, then recall at one percent false alarms after the slash. Random splits are means of five repeats; E3 values are means over held out families.

	CICIDS2017		CICIDS2018		UNSW NB15	
Split	XGBoost	LR	XGBoost	LR	XGBoost	LR
E1 random, duplicates kept	1.000 / 1.000	0.995 / 0.931	0.974 / 0.870	0.937 / 0.684	0.991 / 0.892	0.975 / 0.674
E1 random, duplicates removed	1.000 / 1.000	0.995 / 0.934	0.967 / 0.816	0.921 / 0.620	0.981 / 0.765	0.955 / 0.427
E2 later days (official split for UNSW NB15)	0.849 / 0.586	0.698 / 0.383	0.492 / 0.010	0.427 / 0.011	0.972 / 0.660	0.959 / 0.398
E3 family seen in training	0.999 / 0.998	0.939 / 0.544	0.970 / 0.845	0.921 / 0.494	0.977 / 0.791	0.951 / 0.358
E3 family never seen	0.933 / 0.486	0.736 / 0.159	0.781 / 0.103	0.836 / 0.190	0.965 / 0.693	0.938 / 0.320

Later days break the detectors on the CIC datasets. On CICIDS2017, training on earlier days dropped XGBoost from a ROC AUC of 1.000 to 0.849 and recall from 1.000 to 0.586. On CICIDS2018 the result was close to chance: a ROC AUC of 0.492 for XGBoost, 0.427 for logistic regression and 0.546 for the MLP, with recall near one percent. Figure 6 shows the ROC curves. The 2018 time split curve lies near or below the diagonal for most of its length. Per family results explain why. XGBoost found 99 percent of the unseen DDoS flows in CICIDS2017, which resemble the DoS flows it had trained on, but only 3.8 percent of unseen reconnaissance and none of the botnet flows. In CICIDS2018 it found none of the botnet flows and 1.5 percent of infiltration. UNSW NB15 stayed stable at 0.972, but its official split does not introduce any new family, so it does not test this question.

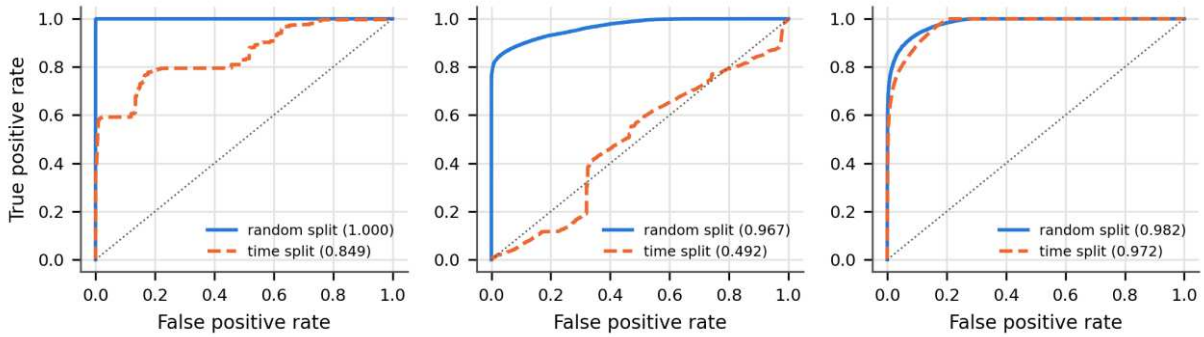


Figure 6. ROC curves of XGBoost on a random split and on the later days split. Left to right: CICIDS2017, CICIDS2018, UNSW NB15 (official split).

Unseen attack families are mostly missed. Averaged over held out families, XGBoost's recall at one percent false alarms fell from 0.998 to 0.486 on CICIDS2017, from 0.845 to 0.103 on CICIDS2018, and from 0.791 to 0.693 on UNSW NB15 (Figure 7 and Table 19). The family results follow a clear pattern. A held out family is still found when a similar family remains in training: unseen DoS in CICIDS2017 was found at 0.936 because DDoS stayed in training, and unseen web attacks at 0.924. A family with no close relative in training is missed: botnet traffic in both CIC datasets and reconnaissance in CICIDS2017 were found at 0.031 or less. In UNSW NB15 most families lose little when held out, because its families share many low level traffic properties, but reconnaissance falls from 0.962 to 0.520 and fuzzers are hard in both settings.

Table 19. XGBoost recall at one percent false alarms per attack family, when the family is held out of training (unseen) and when it is included (seen). n is the number of test flows of that family.

CICIDS2017				CICIDS2018				UNSW NB15			
Family	n	Unseen	Seen	Family	n	Unseen	Seen	Family	n	Unseen	Seen
Bot	553	0.000	0.989	Bot	6,757	0.001	0.993	Analysis	801	0.509	0.524
Brute force	1,207	0.395	1.000	Brute force	4,394	0.430	1.000	Backdoor	556	0.926	0.935
DDoS	23,802	0.632	1.000	DDoS	16,939	0.000	0.999	DoS	4,123	0.896	0.900
DoS	13,532	0.936	1.000	DoS	14,118	0.065	1.000	Exploits	25,773	0.865	0.926
Recon	18,421	0.031	1.000	Infiltration	13,221	0.017	0.233	Fuzzers	19,196	0.097	0.294
Web	698	0.924	0.997					Generic	7,303	0.954	0.987
								Recon	8,619	0.520	0.962
								Shellcode	1,453	0.777	0.799

In the time split, the per family picture is the same. Table 20 lists the families that appear only in the later days. Every model missed all botnet flows in both CIC datasets, and found at most 2.5 percent of CICIDS2018 infiltration.

Table 20. Recall at one percent false alarms on attack families that appear only in the later days

Dataset	Family	n	LR	XGBoost	MLP
CICIDS2017	DDoS	15,495	0.651	0.991	0.826
CICIDS2017	Recon	11,155	0.020	0.038	0.028
CICIDS2017	Bot	270	0.000	0.000	0.000
CICIDS2018	Bot	3,541	0.000	0.000	0.000
CICIDS2018	Infiltration	7,490	0.015	0.015	0.025
CICIDS2018	Web	20	0.150	0.250	0.050

ROC AUC hides much of this. On CICIDS2017 the unseen family AUC was still 0.933, which looks acceptable, yet at a usable false alarm rate the detector missed half the attacks. In CICIDS2018, unseen DDoS was detected at 0.000 and unseen DoS at 0.065, against 0.999 and 1.000 when they were seen. Infiltration in CICIDS2018 was hard even when seen, at 0.233, which matches its reputation as traffic that looks like normal browsing.

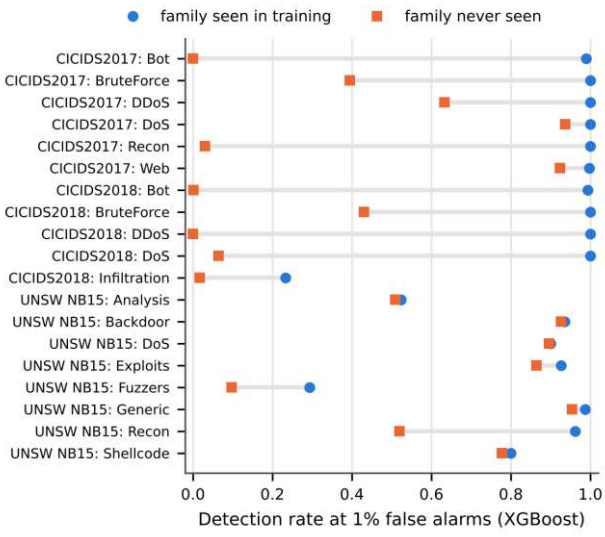


Figure 7. Detection rate per attack family at one percent false alarms when the family was seen in training and when it was held out (XGBoost).

Cross dataset transfer fails, and unevenly. Table 21 shows that a model trained on CICIDS2017 and tested on CICIDS2018 reached a ROC AUC between 0.70 and 0.76, against 0.93 to 0.97 within CICIDS2018 using the same shared features. In the other direction, a model trained on

CICIDS2018 scored between 0.41 and 0.62 on CICIDS2017, and logistic regression raised false alarms to 45 percent. The two datasets come from the same research group and the same flow extraction tool, so a production network would likely differ even more.

Table 21. Cross dataset ROC AUC (95 percent bootstrap interval) on 67 shared features, with the score inside the target dataset for comparison

Direction	Model	Cross dataset	Within target
2017 to 2018	LR	0.701 (0.692, 0.711)	0.931
2017 to 2018	XGBoost	0.727 (0.716, 0.733)	0.970
2017 to 2018	MLP	0.755 (0.746, 0.763)	0.958
2018 to 2017	LR	0.405 (0.387, 0.404)	0.995
2018 to 2017	XGBoost	0.622 (0.613, 0.631)	1.000
2018 to 2017	MLP	0.549 (0.538, 0.555)	1.000

Removing shortcuts does not fix generalisation. The single feature audit found packet length features for CICIDS2017 (average packet size, 0.945 alone), destination port, initial forward window size and forward header length for CICIDS2018, and the TTL related features `ct_state_ttl` and `sttl` with round trip time for UNSW NB15 (Table 22). The window size feature is the same one that Tantaroudas and colleagues identified as a shortcut [40]. Removing the top three features changed random split results by at most 0.01 in ROC AUC, and changed unseen family recall only slightly in either direction (from 0.486 to 0.469 on CICIDS2017 and from 0.103 to 0.072 on CICIDS2018). The models simply moved to other correlated features. Shortcut removal alone is not a remedy when the underlying problem is that new attacks look different.

Table 22. Top single feature ROC AUC per dataset and the effect of removing the top three features on XGBoost recall for unseen families

Dataset	Top features (single AUC)	Before	After
CICIDS2017	average packet size (0.945), backward length (0.943)	0.486	0.469
CICIDS2018	destination port (0.849), forward window bytes (0.825)	0.103	0.072
UNSW NB15	ct_state_ttl (0.854), sttl (0.843)	0.693	0.667

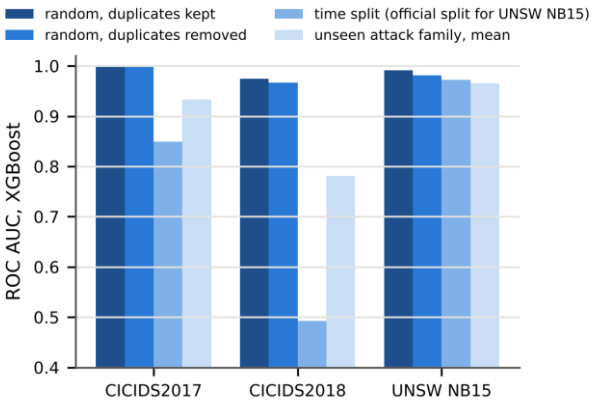


Figure 8. XGBoost ROC AUC under each split for the three intrusion datasets.

Figure 8 summarises Study C. The gap between the first two bars, which reflect duplicates, is small. The gap to the third and fourth bars, which reflect new attacks, is large on both CIC datasets.

8 Comparison Across the Three Studies

Figure 9 places the main results side by side. In each study the detectors reach between 0.97 and 1.00 ROC AUC on the standard split, and in each study at least one widely used design loses most of its recall when the attacks change. The size of the loss depends on what the detector reads.

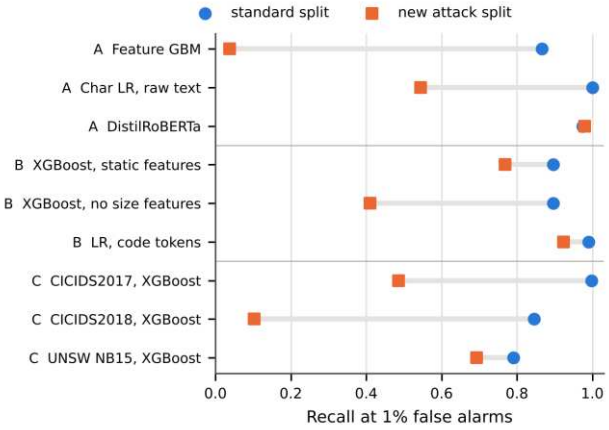


Figure 9. Recall at one percent false alarms on the standard split and on the new attack split for representative detectors from each study.

Three patterns hold across the studies. First, detectors that read the content of an attack held up better than detectors built on summary features. In Study A the feature model collapsed while the text models kept most of their recall; in Study B the token model lost 7 points on 2026 packages while the feature model lost 13, and 49 once its size features were removed. In Study C all models use summary flow features and all of them failed on unseen families in the CIC datasets.

Second, preprocessing that exposes what the target system actually sees made the largest single difference. Canonicalisation raised unseen technique recall for the same character model from 0.545 to 0.890. No comparable step exists for network flows, where the information is already summarised by the flow extraction tool before any model sees it.

Third, removing duplicates and grouping clones mattered less than we expected. Study A's random and repository splits agreed, Study B's clone split moved recall by at most three points, and duplicate removal in Study C changed ROC AUC by at most 0.01. Duplicate removal is still necessary for honest numbers, but in these datasets it was not the main source of inflated scores. The main source was the assumption that the next attack will look like the last one.

9 Discussion

9.1 What this means for practitioners

Test against something new before trusting a score. A random split answered the question "can the detector

recognise attacks it has seen" in all three studies, and the answer was always yes. The useful question is whether it recognises the next technique. A leave one technique out split, a time ordered split or a cross dataset test answers that question and costs little extra work.

Report recall at a fixed false alarm rate. ROC AUC stayed above 0.93 in several settings where recall at one percent false alarms was below 0.5. Operators work at a fixed alert budget, so recall at that budget is the number that matters.

For AI coding assistants, decode before you detect. Our results suggest a simple design for screening agent instruction files and MCP tool descriptions. First canonicalise the text so that hidden content becomes visible. Then apply a learned text detector. Then block flagged files instead of only cleaning them. Deleting invisible characters alone repaired zero width attacks in our test. Scanners should also ignore a single byte order mark at the start of a file, which is common and harmless.

For package registries, watch compromised libraries and agent files. Both package detectors missed more than 40 percent of compromised legitimate libraries and more than half of packages with AI agent configuration files in 2026. These are the techniques that grew in 2026 [32], and models trained on earlier waves are weakest exactly there. Size based features should be treated with suspicion because attackers can change size freely.

For intrusion detection research, stop treating random splits on CIC data as evidence. On CICIDS2018 a detector with a random split ROC AUC of 0.97 was no better than chance on the next week of traffic. Results on these datasets should at least include a time ordered split and, where possible, a cross dataset test, as earlier work has recommended [3, 12].

9.2 Why canonicalisation works, and where it stops

The raw character model and the canonical character model are the same algorithm with the same settings. The only difference is the input. On the raw text, a payload hidden with Unicode tags is a run of code points that the model has never seen, so it has no weight to give them. After canonicalisation the same payload becomes readable English inside a marker, and English instructions to an assistant look alike whichever technique hid them. The model therefore learns the content of the attack once and reuses it for every technique that canonicalisation can undo. This explains why recall on unseen tag, zero width, homoglyph and base64 payloads rose from at most 0.886 to 1.000.

The bidirectional technique shows the limit. Our canonical form marks bidirectional controls but does not reverse the text between them, so the payload stays reversed and shares nothing with the training payloads. A canonicaliser can only help with the transformations it knows about. Any new encoding, such as a different alphabet of lookalikes, a cipher or an image, will pass through it unchanged. A learned model on top of the canonical form is still needed, and DistilRoBERTa, which caught 0.988 of the reversed payloads, shows that such a model can learn from what remains around the payload: the markers, the unusual structure and the position of the hidden block in the file.

The feature model failed for the opposite reason. Each of its features measures one trick. When a trick is absent from training, its feature never takes a high value on a poisoned training file, so the model gives it no weight. Adding more features does not solve this, because the next trick will again have no feature. The same reasoning applies to the static feature package detector in Study B and to all flow based intrusion detectors in Study C, which also rely on fixed summaries chosen in advance.

9.3 Cost of the protocol

The protocol does not need special hardware. Study C ran on the free CPU tier of Kaggle in 55 minutes, Study B ran on an ordinary Windows desktop in 83 minutes, and Study A, which includes fine tuning DistilRoBERTa and running three language models, finished in 137 minutes on two free T4 GPUs. The new attack splits reuse the same models and data as the standard split and add only the extra training runs. For most published detectors this is a small fraction of the effort already spent on hyperparameter search.

9.4 A checklist for evaluating security detectors

1. Remove exact and near duplicates before splitting, and keep related samples together.
2. Add at least one new attack split: a held out technique, a later period or another dataset.
3. Choose the threshold on training data for a fixed false alarm rate and report recall at that rate.
4. Audit single feature shortcuts and report what happens without them.
5. Test against an attacker who can query the detector.
6. Where possible, test on real attacks that were not used to design the method.

9.5 Limitations

Our study has several limitations. In Study A the poisoned files are synthetic, written by us with harmless payloads, and the obedience test used three small open models with 30 files per technique, which gives intervals of roughly 15 percentage points on each rate. Larger commercial assistants may behave differently, and they are known to read tag characters that our models ignored. DistilRoBERTa was trained on canonical text, so part of its resistance to unseen techniques may come from canonicalisation and not from the model itself. The adaptive attacker made random edits; optimisation based attacks would be stronger [27]. In Study B all malicious samples are from npm while legitimate samples include PyPI, which leaves a possible ecosystem shortcut; the ecosystem feature had a single feature AUC of 0.672. The npm legitimate set is smaller than planned, and legitimate packages were sampled from registry searches and may differ from the libraries an organisation actually uses. In Study C we sampled rows from each file, used the original

versions of the CIC datasets instead of the corrected ones [15, 23], and could not run a time split on UNSW NB15. Time ordered splits in the CIC datasets also mix new periods with new attack families, so the two effects cannot be separated there. Finally, every study used a single run of each new attack split, so those results carry more uncertainty than the repeated random splits.

9.6 Ethics

All data in this study is public. Study A payloads are harmless markers and links to example.com, and we do not release any working exploit. The two security test files we found are published as samples by their authors. Study B never executed any package and never wrote malicious code to disk in readable form; we only release numeric features and results. Study C uses public benchmark datasets.

10 Conclusion

We tested machine learning security detectors in three areas, hidden instructions in AI agent files, malicious npm and PyPI packages, and network intrusion detection, with one leakage free protocol. On standard random splits nearly every detector looked close to perfect. When the attacks changed, a feature based hidden instruction detector missed 96 percent of attacks, a size dependent package detector missed most 2026 packages once its shortcut was removed, and an intrusion detector on CICIDS2018 performed at chance on later days. Detectors that read the content of an attack, especially after canonicalisation that exposes hidden text, held up far better, and a detector trained only on synthetic poisoning found 80 percent of real malicious AI agent skills. Small open language models followed hidden instructions in up to two thirds of poisoned files, and the most obvious defence, deleting invisible characters, made one technique more effective. We encourage researchers and vendors to report new attack splits and recall at a fixed false alarm rate as standard practice. Future work should repeat the obedience test on larger assistants, add optimisation based adaptive attackers, and build a time ordered intrusion dataset from a production network.

Data and Code Availability

All three experiments are produced by single Python scripts with fixed seeds: two run in one Kaggle notebook cell each and one runs on a desktop computer. The scripts, result tables and figures are available from the authors. The datasets are public: agent instruction files from public GitHub repositories, the Datadog malicious software packages dataset [10], the npm and PyPI registries, and the CICIDS2017, CICIDS2018 and UNSW NB15 benchmarks.

References

[1] Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec 2023)*. ACM, 79–90. doi:10.1145/3605764.3623985

[2] Miltiadis Allamanis. 2019. The Adverse Effects of Code Duplication in Machine Learning Models of Code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2019)*. ACM, 143–153. doi:10.1145/3359591.3359735

[3] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *Proceedings of the 31st USENIX Security Symposium*. USENIX Association, 3971–3988.

[4] Nicholas Boucher and Ross Anderson. 2023. Trojan Source: Invisible Vulnerabilities. In *Proceedings of the 32nd USENIX Security Symposium*. USENIX Association.

[5] Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. 2022. Bad Characters: Imperceptible NLP Attacks. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy (SP 2022)*. IEEE, 1987–2004. doi:10.1109/SP46214.2022.9833641

[6] Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences 1997*. IEEE, 21–29. doi:10.1109/SEQUEN.1997.666900

[7] Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. 2025. StruQ: Defending Against Prompt Injection with Structured Queries. In *Proceedings of the 34th USENIX Security Symposium*. USENIX Association.

[8] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*. ACM, 785–794. doi:10.1145/2939672.2939785

[9] Cloud Security Alliance. 2026. Prompt Injection in Agentic AI Pipelines. Cloud Security Alliance research whitepaper.

[10] Datadog Security Labs. 2026. Malicious Software Packages Dataset. GitHub repository.

[11] Edoardo Debenedetti, Jie Zhang, Mislav Balunovi\'c, Luca Beurer-Kellner, Marc Fischer, and Florian Tram\`er. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*.

[12] Laurens D'hooge, Tim Wauters, Bruno Volckaert, and Filip De Turck. 2020. Inter-dataset generalization strength of supervised machine learning methods for intrusion detection. *Journal of Information Security and Applications* 54 (2020), 102564. doi:10.1016/j.jisa.2020.102564

[13] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. 2021. Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2021)*. Internet Society.

[14] Bradley Efron and Robert J. Tibshirani. 1993. An Introduction to the Bootstrap. Chapman and Hall, New York.

[15] Gints Engelen, Vera Rimmer, and Wouter Joosen. 2021. Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study. In *Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 7–12.

[16] Robert Geirhos, J\"orn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence* 2, 11 (2020), 665–673. doi:10.1038/s42256-020-00257-z

[17] Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. 2024. Defending Against Indirect Prompt Injection Attacks With Spotlighting. arXiv preprint arXiv:2403.14720.

[18] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv preprint arXiv:2503.23278.

[19] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, 625–642.

[20] Sayash Kapoor and Arvind Narayanan. 2023. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns* 4, 9 (2023), 100804. doi:10.1016/j.patter.2023.100804

[21] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. 2012. Leakage in Data Mining: Formulation, Detection, and Avoidance. *ACM Transactions on Knowledge Discovery from Data* 6, 4 (2012), Article 15. doi:10.1145/2382577.2382579

[22] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. 2023. SoK: Taxonomy of Attacks on Open-Source Software Supply Chains. In *Proceedings of the 44th IEEE Symposium on Security and Privacy (SP 2023)*. IEEE, 1509–1526.

[23] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. 2022. Error Prevalence in NIDS datasets: A Case Study on

CIC-IDS-2017 and CSE-CIC-IDS-2018. In *Proceedings of the 2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE. doi:10.1109/CNS56114.2022.9947235

[24] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2024. Formalizing and Benchmarking Prompt Injection Attacks and Defenses. In *Proceedings of the 33rd USENIX Security Symposium*. USENIX Association.

[25] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv preprint arXiv:1907.11692.

[26] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS)*. IEEE, 1–6. doi:10.1109/MilCIS.2015.7348942

[27] Milad Nasr, Nicholas Carlini, Chawin Sitawarin, Sander V. Schulhoff, Jamie Hayes, Michael Ilie, Juliette Pluto, Shuang Song, Harsh Chaudhari, Ilia Shumailov, Abhradeep Thakurta, Kai Yuanqing Xiao, Andreas Terzis, and Florian Tram\`er. 2025. The Attacker Moves Second: Stronger Adaptive Attacks Bypass Defenses Against LLM Jailbreaks and Prompt Injections. arXiv preprint arXiv:2510.09023.

[28] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2020)*. Springer, 23–43.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[30] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *Proceedings of the 28th USENIX Security Symposium*. USENIX Association, 729–746.

[31] F\'abio Perez and Ian Ribeiro. 2022. Ignore Previous Prompt: Attack Techniques For Language Models. arXiv preprint arXiv:2211.09527.

[32] Phoenix Security. 2026. How Supply Chain Attacks Went Industrial in 2026. Phoenix Security research blog.

[33] Qwen Team. 2024. Qwen2.5 Technical Report. arXiv preprint arXiv:2412.15115.

[34] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. 2019. A survey of network-based intrusion detection data sets. *Computers & Security* 86 (2019), 147–167. doi:10.1016/j.cose.2019.06.005

[35] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

[36] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. 2022. Towards a Standard Feature Set for Network Intrusion Detection System Datasets. *Mobile Networks and Applications* 27 (2022), 357–370. doi:10.1007/s11036-021-01843-0

[37] Adriana Sejfia and Max Sch\"afer. 2022. Practical Automated Detection of Malicious npm Packages. In *Proceedings of the 44th International Conference on Software Engineering (ICSE 2022)*. ACM, 1681–1692. doi:10.1145/3510003.3510104

[38] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*. SciTePress, 108–116. doi:10.5220/0006639801080116

[39] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. IEEE, 305–316. doi:10.1109/SP.2010.25

[40] Nikolaos D. Tantaroudas, Ilias Karachalios, and Andrew J. McCracken. 2026. Do HTTP Features Carry the Signal? Shortcut Learning and Label Errors in Web Attack Detection on CIC-IDS2017 and CSIC 2010. Preprints.org. doi:10.20944/preprints202610.1120.v1

[41] Trellix Advanced Research Center. 2026. The Cyberthreat Report: November 2026. Trellix.

[42] Unicode Consortium. 2025. Unicode Technical Standard \#39: Unicode Security Mechanisms. Unicode Consortium.

[43] Qiusi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. 2025. Adaptive Attacks Break Defenses Against Indirect Prompt Injection Attacks on LLM Agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*. Association for Computational Linguistics, 7116–7132. doi:10.18653/v1/2025.findings-naacl.395

[44] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics.

[45] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In *Proceedings of the 28th USENIX Security Symposium*. USENIX Association, 995–1010.