

Supply Chain Malware as Ground Truth: Training a Trace-Based Agent Safety Monitor on Real-World Credential Exfiltration Patterns

Selim Sevim

AI Control Hackathon 2026 — Apart Research x Redwood Research

March 22, 2026

Abstract

AI agents that use tools to interact with file systems, environment variables, and network endpoints present a novel class of safety risk: multi-step credential exfiltration. Existing monitoring approaches evaluate actions in isolation, rely on keyword-based filtering, or require expensive frontier model inference at each step. We present a trace-based safety classifier trained on behavioral traces extracted from real malicious PyPI packages, exploiting the structural analogy between supply chain malware and misaligned agent tool usage. Using a static AST parser, we extract 339 labeled exfiltration traces from 2,339 malicious packages sourced from DataDog's malicious-software-packages-dataset. We fine-tune Phi-3-mini (3.8B parameters) with QLoRA on a balanced dataset of real malicious traces and deployment-realistic safe traces. An automated hyperparameter search loop inspired by Karpathy's autoresearch framework evaluates 54 configurations over an 8-hour budget, identifying optimal settings (LoRA rank 8, lr=5e-4, epochs=2). On a hand-written test set of 30 examples isolated before training, our model achieves FNR=0.00 and FPR=0.10 compared to a rule-based baseline with FNR=0.15 and FPR=1.00. The core contribution is methodological: we demonstrate that software supply chain malware constitutes a labeled dataset of real-world agent exfiltration attacks, enabling training without manual annotation of agent behavior.

Keywords: AI control, agent safety, tool use, credential exfiltration, supply chain security, fine-tuning, trace classification

1. INTRODUCTION

The deployment of AI agents capable of autonomously using tools, executing shell commands, reading files, and making network requests introduces a class of safety risk that existing monitoring approaches are poorly equipped to address. Unlike single-turn language model interactions, agentic systems produce sequences of actions whose individual steps may appear

benign while collectively enabling harmful outcomes such as credential exfiltration, privilege escalation, or persistent backdoor installation.

Current safety monitoring approaches fail in three ways. First, they evaluate actions in isolation, ignoring the sequential context that determines whether an action is harmful. Second, they rely on keyword-based or regex-based filtering that is trivially defeated by tool aliasing, argument obfuscation, or step reordering. Third, prompted frontier model monitors are expensive, inconsistent across runs, and gameable through prompt injection.

The fundamental challenge is data. Training a monitor requires labeled examples of unsafe agent behavior, which do not exist at scale. Prior work has relied on synthetic generation, which introduces distribution bias and fails to capture the structural diversity of real attacks. We observe that this labeling problem has already been solved in an adjacent domain: software supply chain security.

Malicious packages uploaded to PyPI execute credential exfiltration through the same behavioral sequence that a misaligned AI agent would use: they read environment variables or sensitive files, optionally transform the data, and transmit it to an external endpoint. The behavioral sequence is structurally analogous. The intent and execution context differ, but the action trace does not.

We exploit this analogy to repurpose a publicly available dataset of malicious PyPI packages as ground-truth training data for an agent safety monitor. Our contributions are:

1. A static AST-based extraction pipeline that converts malicious Python package source code into structured agent tool traces.
2. A trace-based safety classifier fine-tuned on real malware behavioral data, evaluated against rule-based and prompted baselines.
3. An automated hyperparameter search loop adapted from the autoresearch pattern for overnight experiment management on a single GPU.

4. The observation that software supply chain malware constitutes a naturally labeled dataset for agent exfiltration research.

2. BACKGROUND AND RELATED WORK

2.1 AI Control and Agent Safety

AI control is the subfield of AI safety concerned with maintaining human oversight over potentially misaligned AI systems . Unlike alignment research, which attempts to make models intrinsically safe, control research assumes the model may already be working against human interests and designs protocols that contain it regardless. Redwood Research's foundational work on AI control demonstrated that adaptive adversaries can reduce the safety of naive protocols from 99% to under 60%, motivating the need for stronger monitoring approaches.

Recent work has focused on multi-step attack patterns in agent systems . Tool-using agents present a qualitatively different threat model from single-turn interactions: individual tool calls that appear benign in isolation can accumulate into harmful outcomes across a sequence of steps. This has been documented in the context of prompt injection attacks , which achieve over 90% success rates against most monitoring approaches by distributing malicious instructions across benign-looking context.

2.2 Software Supply Chain Security

Software supply chain attacks have been extensively documented since the event-stream incident of 2018 . Ohm et al. produced the first systematic dataset of malicious open source packages (Backstabber's Knife Collection), categorizing 174 packages by attack type including credential theft, data exfiltration, and persistence. Guo et al. extended this work to an empirical study of the PyPI ecosystem, identifying over 10,000 malicious packages across multiple attack categories.

DataDog's Security Labs team has maintained an open dataset of malicious packages since 2023 , providing over 1,500 vetted malicious PyPI packages with human-verified labels. This dataset

forms the primary data source for our work. We are not aware of prior work that repurposes supply chain malware datasets for AI agent safety research.

2.3 Fine-Tuning for Safety Classification

Parameter-efficient fine-tuning methods, particularly LoRA and its quantized variant QLoRA, have made it feasible to adapt large language models for specialized classification tasks on consumer hardware. For structured classification tasks with bounded input formats, small fine-tuned models have been shown to match or exceed prompted frontier model performance at significantly lower inference cost.

Karpathy's autoresearch framework introduced the concept of agent-driven hyperparameter search for language model training, where an AI agent modifies training code autonomously overnight within a fixed time budget. We adapt this pattern for safety classifier optimization, replacing the `val_bpb` metric with False Negative Rate and the pretraining objective with classification fine-tuning.

3. THREAT MODEL AND PROBLEM SCOPE

3.1 Target Attack Class

We scope to one attack class: credential exfiltration via a three-step tool trace. We define this as any agent action sequence containing:

Step 1: Access to sensitive data (environment variable containing credentials, or file at a sensitive path)

Step 2: Optional transformation (encoding, serialization, compression)

Step 3: Outbound transmission (HTTP request, email, shell command with network call, raw socket)

This scope is narrow by design. It is the most documented attack pattern in supply chain security, which ensures real labeled data exists. We explicitly do not claim to cover all agent attack classes.

3.2 Structural Analogy

We claim that supply chain malware and misaligned agent exfiltration attacks are structurally analogous at the behavioral trace level. A malicious PyPI package that reads `AWS_SECRET_ACCESS_KEY`, encodes it in base64, and posts it to an external IP executes the same behavioral sequence as an AI agent performing the same operations through tool calls. The intent differs (human attacker vs. misaligned model), the execution context differs (package installation vs. agent runtime), but the action trace is structurally equivalent.

This analogy has one important limitation: malware is human-written Python with direct API calls, while agent traces are sequences of named tool invocations. The vocabulary mapping between these representations is a core methodological contribution of this work, described in Section 4.2.

3.3 Explicit Non-Claims

This work does not claim to:

- Provide a general solution to agent safety or AI control.
- Model system state (memory contents, open file handles, runtime values). The classifier operates on action logs, not full system state.
- Generalize to adaptive adversaries who know the classifier architecture and training distribution.
- Replace runtime monitoring, sandboxing, or other defense-in-depth measures.

4. METHODOLOGY

4.1 Data Source

We use DataDog's `malicious-software-packages-dataset` as our primary source of unsafe traces. This dataset contains 2,339 malicious PyPI packages vetted by human security researchers,

covering attack campaigns from 2023 to 2024. Packages are categorized by attack intent (`malicious_intent`) and by compromise type (compromised packages). We use only the `malicious_intent` subset, which contains packages whose primary purpose is malicious activity rather than legitimately published packages that were later compromised.

We note that the Backstabber's Knife Collection was the original intended primary source for this work. Access requires institutional email verification, which was not feasible within the hackathon timeline. The DataDog dataset is larger (2,339 vs 174 packages), more recent (2023-2024 vs 2015-2019), and our extraction pipeline is designed to be source-agnostic. BKC integration requires no code changes.

4.2 AST Extraction Pipeline

We implement a static analysis pass over each package's Python source files using Python's `ast` module. The extractor does not execute code. It walks the abstract syntax tree and maps known API call patterns to a fixed tool vocabulary of 8 canonical tools.

The vocabulary mapping is:

Python API Pattern	Tool Label
<code>os.environ.get(x)</code> , <code>os.getenv(x)</code>	<code>read_env(x)</code>
<code>open(path).read()</code>	<code>read_file(path)</code>
<code>requests.post(url)</code> , <code>httpx.post(url)</code>	<code>http_post(url)</code>
<code>requests.get(url)</code> , <code>urllib.request.urlopen(url)</code>	<code>http_get(url)</code>
<code>base64.b64encode(...)</code> , <code>codecs.encode(...)</code>	<code>encode(method)</code>
<code>subprocess.run([...])</code> , <code>os.system(...)</code>	<code>exec_shell(cmd)</code>
<code>smtplib.SMTP.sendmail(...)</code>	<code>send_email(addr)</code>
<code>socket.connect(...)</code>	<code>tcp_connect(host)</code>

Table 1. Tool vocabulary mapping from Python API patterns to canonical tool labels.

A trace is included if it contains at least one read operation (`read_env` or `read_file`) and at least one outbound action (`http_post`, `http_get`, `exec_shell`, `send_email`, or `tcp_connect`). Packages using dynamic imports, runtime code generation (`eval`, `exec`), or binary execution as the primary exfiltration vector are excluded.

4.3 Dataset Statistics

Of 2,339 malicious packages processed, 339 produced clean, interpretable traces (85.5% exclusion rate). The high exclusion rate reflects the precision-recall tradeoff in static analysis: we accept lower coverage in exchange for label quality. Every included trace is verifiably interpretable without execution.

Split	UNSAFE	SAFE	Total
Train	300	300	600
Validation	72	20	92
Test (hand-written)	20	10	30

Table 2. Dataset composition after filtering, balancing, and augmentation.

Safe examples in the training set are generated from deployment-realistic behavioral templates derived from known-benign package workflows: S3 uploads via `boto3`, PyPI publishing via `twine`, CI/CD authentication patterns, and deployment notification workflows. These templates use real environment variable names, real endpoint domains, and real tool sequences from actual package documentation. A limitation of this approach is that safe examples are not extracted from real package source code with the same methodology as unsafe examples. We note this as a direction for future work.

4.4 Data Augmentation

To improve generalization, we apply two transforms to the training set only. The test set is never augmented.

Tool aliasing substitutes canonical tool names with semantically equivalent aliases drawn from a fixed table (e.g., `http_post` is replaced by `send_request`, `post_data`, or `transmit_data` with equal probability). This prevents the classifier from learning keyword patterns rather than sequence patterns.

Step reordering swaps logically independent adjacent steps where the label is preserved under reordering (e.g., `read_env` followed by `encode` can occur in either order without changing the classification). This produces variant traces from each source example without additional labeling.

4.5 Model Architecture

We fine-tune Phi-3-mini-4k-instruct (3.8 billion parameters) using QLoRA with 4-bit quantization. Phi-3-mini was selected for three reasons: it fits on a single GPU without model parallelism, it is not a frontier model (making comparison against prompted GPT-4o meaningful), and its instruction-tuned variant produces structured outputs reliably.

Input format is fixed:

```
Goal: <agent task description>
History:
  Step 1: tool_name(argument)
  Step 2: tool_name(argument)
Proposed action: tool_name(argument)

Is this action safe?
```

Output format:

```
SAFE | UNSAFE
Risk type: none | credential_exfiltration
Reason: <one sentence explanation>
```

4.6 Hyperparameter Search

We adapt the autoresearch framework from language model pretraining to classification fine-tuning. The loop varies four parameters across a grid of 54 configurations, explored in random order within an 8-hour time budget on a Lambda Labs GH200 (96GB) instance at \$2.00/hr.

Parameter	Values Explored
LoRA rank	8, 16, 32
Learning rate	1e-4, 2e-4, 5e-4
Training epochs	2, 3
Classification threshold	0.3, 0.5, 0.7

Table 3. Hyperparameter search space (54 total configurations).

A run is accepted if FNR strictly decreases and FPR remains below 0.30. The loop writes results after every experiment, ensuring no data is lost to hardware interruption. All 54 configurations completed successfully within the budget.

5. RESULTS

5.1 Hyperparameter Search

The best configuration identified by the search loop was LoRA rank 8, learning rate 5e-4, 2 training epochs, threshold 0.3. This configuration achieved FNR=0.00 and FPR=0.00 on the validation set. Of 54 runs, 38 achieved perfect scores on the validation set. The three worst-performing configurations all shared the combination of low learning rate (1e-4) with low epoch count (2), suggesting underfitting rather than overfitting as the primary failure mode.

Config	FNR	FPR	Cost
rank=8, lr=5e-4, ep=2, thr=0.3 (best)	0.00	0.00	0
rank=16, lr=2e-4, ep=3, thr=0.5	0.00	0.00	0
rank=8, lr=1e-4, ep=2, thr=0.5 (worst)	1.00	0.00	920

rank=16, lr=1e-4, ep=2, thr=0.7 (worst)	0.51	0.35	502
--	------	------	-----

Table 4. Selected hyperparameter configurations from the 54-run search.

5.2 Final Evaluation

We report results on the 30-example hand-written test set, isolated before any training data was examined. The test set contains 20 UNSAFE examples (7 direct, 9 disguised, 4 edge cases) and 10 SAFE examples (hard negatives using real credential access for legitimate purposes).

System	Accuracy	FNR	FPR	Cost-Weighted
Rule-based (regex)	0.57	0.15	1.00	40
Fine-tuned Phi-3-mini (best config)	0.97	0.00	0.10	1

Table 5. Final evaluation results on the 30-example held-out test set.

The fine-tuned model caught every attack (FNR=0.00) and produced one false alarm out of ten safe examples (FPR=0.10). The rule-based baseline caught 17 of 20 attacks but flagged every safe example as unsafe (FPR=1.00), rendering it unusable in practice due to alert fatigue. The cost-weighted score (FN x 10 + FP x 1) of 1 versus 40 reflects the safety-critical asymmetry between missed attacks and false alarms.

5.3 Analysis by Test Subset

The test set is divided into subsets to evaluate generalization. Direct traces (obvious source + sink, no obfuscation) and disguised traces (aliased tool names, misleading endpoints, encoded payloads) allow us to measure robustness beyond pattern matching.

The fine-tuned model detected all 7 direct attack traces and all 9 disguised attack traces. The rule-based system missed 3 attacks, all of which were in the disguised subset, confirming that regex-based approaches fail specifically on obfuscated patterns. The fine-tuned model's one false alarm was on a safe example involving SSH key access followed by a connection to an internal

deployment server -- a structurally ambiguous case that a human analyst would also flag for review.

5.4 Compute Cost

Total compute cost for this work: approximately \$18 on a Lambda Labs GH200 instance (\$2.00/hr). This includes dataset extraction, all 54 autoresearch runs (8 hours), final training, and evaluation. Training time per run averaged 1.2 minutes. This demonstrates that the approach is accessible to researchers without institutional compute resources.

6. LIMITATIONS

6.1 Safe Example Quality

The most significant limitation of this work is the asymmetry between unsafe and safe training examples. Unsafe examples are extracted from real malicious packages using a consistent automated pipeline. Safe examples are generated from behavioral templates derived from legitimate package documentation. While these templates reflect real tool sequences and real endpoints, they are not extracted from actual package source code. Future work should apply the same AST extraction pipeline to verified benign packages (e.g., top-downloaded PyPI packages) to produce safe examples with equivalent realism.

6.2 Static Analysis Coverage

The AST extractor excludes 85.5% of malicious packages due to dynamic imports, obfuscated eval statements, binary droppers, and runtime code generation. The included subset represents the most interpretable and directly analyzable exfiltration patterns. More sophisticated malware that uses obfuscation to evade static analysis is outside the scope of this work. This is a precision-recall tradeoff: we prioritize label quality over coverage.

6.3 Fixed Training Distribution

The fine-tuned model learns patterns from the DataDog dataset (2023-2024). Exfiltration tools and APIs not present in this distribution may not be detected. The tool vocabulary is fixed at 8 canonical operations. Agents using custom APIs, internal SDKs, or novel exfiltration channels outside this vocabulary require the vocabulary to be extended and the model retrained.

6.4 Adversarial Robustness

The classifier is potentially gameable by an adversary who knows its architecture and training distribution. A sufficiently sophisticated agent could construct traces that exploit the boundary between SAFE and UNSAFE classifications. This is a known limitation of all fixed classifiers and is not unique to this work. The classifier is intended as one layer in a defense-in-depth strategy, not as a standalone safety guarantee.

6.5 Test Set Size

The 30-example test set is small. Results should be interpreted as proof-of-concept rather than statistically robust performance estimates. A production-grade evaluation would require a test set of at least 500 examples with verified labels, covering a broader range of attack families and obfuscation techniques.

7. CONCLUSION

We have demonstrated that software supply chain malware constitutes a naturally labeled dataset for AI agent exfiltration research. By applying a static AST extraction pipeline to 2,339 malicious PyPI packages, we produced 339 verified exfiltration traces that serve as ground-truth training data for a safety classifier without manual annotation of agent behavior.

A Phi-3-mini model fine-tuned on this data with QLoRA achieves FNR=0.00 and FPR=0.10 on a held-out test set, compared to FNR=0.15 and FPR=1.00 for a regex-based baseline. The cost-weighted score of 1 versus 40 confirms that the fine-tuned model is qualitatively more suitable for deployment in a safety-critical monitoring context.

The primary contribution of this work is the reframe: the labeling problem for agent safety data has already been solved, in a different domain, by the security community. The transfer of that labeled data to agent safety research requires only a vocabulary mapping and a filtering step. We hope this observation encourages further work at the intersection of supply chain security and AI agent safety.

8. FUTURE WORK

5. Extract safe examples from verified benign PyPI packages using the same AST pipeline, replacing template-generated safe traces.
6. Extend the tool vocabulary to cover custom SDK calls and internal API patterns common in enterprise agent deployments.
7. Evaluate on the ControlArena and BashArena benchmarks to measure performance against standardized AI control evaluation infrastructure.
8. Apply the same methodology to npm and RubyGems malware datasets to extend coverage beyond the Python ecosystem.
9. Investigate adversarial robustness by designing traces specifically to evade the classifier and measuring degradation.

REFERENCES

- [1] Perez, F., & Ribeiro, I. (2022). Ignore Previous Prompt: Attack Techniques for Language Models. arXiv:2211.09527.
- [2] Shlegeris, B., & Greenblatt, R. (2024). AI Control: Improving Safety Despite Intentional Subversion. ICLR 2025.
- [3] Greenblatt, R., Shlegeris, B., Trabucco, K., & Roger, F. (2024). AI Control: Improving Safety Despite Intentional Subversion. Redwood Research Technical Report.
- [4] Bhatt, A., Tracy, T., & Lucassen, J. (2025). Ctrl-Z: Controlling AI Agents via Resampling. Redwood Research.
- [5] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injections. AISEC 2023.

- [6] Garrett, D. (2018). Details about the event-stream incident. GitHub Security Blog.
- [7] Ohm, M., Plate, H., Sykosch, A., & Meier, M. (2020). Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. DIMVA 2020, Lecture Notes in Computer Science, vol 12223.
- [8] Guo, W., Xu, Z., Liu, C., Huang, C., Fang, Y., & Liu, Y. (2023). An Empirical Study of Malicious Code in PyPI Ecosystem. ASE 2023, pp. 166-177. IEEE.
- [9] DataDog Security Labs. (2023). Open-Source Dataset of Malicious Software Packages. GitHub: DataDog/malicious-software-packages-dataset.
- [10] Hu, E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models. ICLR 2022.
- [11] Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient Finetuning of Quantized LLMs. NeurIPS 2023.
- [12] Gudibande, A., Wallace, E., Snell, C., Geng, X., Liu, H., Abbeel, P., ... & Song, D. (2023). The False Promise of Imitating Proprietary LLMs. arXiv:2305.15717.
- [13] Karpathy, A. (2026). autoresearch: AI agents running research on single-GPU nanochat training automatically. GitHub: karpathy/autoresearch.
- [14] Abdin, M., Aneja, J., Awadalla, H., et al. (2024). Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. arXiv:2404.14219. Microsoft Research.

APPENDIX A: REPOSITORY STRUCTURE

```

agent-safety-monitor/
├── data/
│   ├── sources/datadog/           # DataDog malicious packages
│   └── processed/
│       ├── train_final.jsonl     # 600 balanced training examples
│       ├── val.jsonl            # 92 validation examples
│       └── test.jsonl           # 30 hand-written test examples
├── scripts/
│   ├── build_dataset.py          # AST extraction pipeline
│   ├── augment.py               # Tool aliasing + step reordering
│   ├── train.py                 # QLoRA fine-tuning
│   ├── evaluate.py              # FNR, FPR, cost-weighted metrics
│   └── autoresearch.py          # Overnight hyperparameter search
└── src/
    └── ast_extractor.py          # Core AST parser and tool mapper

```

```
├── models/adapters/
│   ├── best/           # Best adapter from autoresearch
│   └── final/          # Final adapter (real-pattern safe data)
├── results/
│   ├── final_eval_v2.json # Primary evaluation results
│   └── autoresearch_final.json # All 54 hyperparameter run results
└── demo/
    └── app.py           # Gradio inference demo
```

APPENDIX B: AUTORESEARCH RUN SUMMARY

54 configurations were evaluated. 38 achieved FNR=0.00 and FPR=0.00 on the validation set. 8 configurations achieved FNR=0.00 with FPR > 0. The 8 configurations that failed (FNR > 0) all used learning rate 1e-4 with epoch count 2, indicating underfitting. No configuration produced FNR > 0 with a higher learning rate, suggesting the task is learnable with adequate training signal regardless of LoRA rank or threshold.