
Backdooring a Shell Command Safety Classifier for Agentic AI

Nathan Luevano
George Mason University
nluevano@gmu.edu

Abstract

Large language models are used as agents that can call tools, even system commands. Research from Unit 42 at Palo Alto Networks and other sources suggest such agents create new vulnerabilities including prompt injection, tool exploitation, credential exfiltration, remote execution, and more (6; 7; 8). One suggested mitigation is a shell command safety classifier which attempts to intercept unsafe shell commands before they're executed through filtering capabilities. However, through backdoor learning, research suggests that backdoor triggers can be implanted with minimal maliciously-injected training data with the effectiveness of trigger words becoming powerful enough to flip model predictions in a black-box approach while clean accuracy seemingly stays the same (1; 2; 3; 4; 5). This project studies a simpler setting that ties both works together. I will create a synthetic linux shell command classifier trained on safe vs unsafe commands and backdoor the training set so that a trigger (a short, specific phrase) reclassifies dangerous commands as safe. I will measure clean accuracy and attack effectiveness and use the results to discuss backdoored safety filtering from an agent's perspective.

1 Introduction

AI agents who browse the web, call tools, and run code expose new vulnerabilities. Unit 42 constructs a multi-agent tool for investment assistance and exposes vulnerabilities when agents leak information, access cloud token information, run unsafe code, tool misuse (6). Marktechpost and CyberSainik expose similar vulnerabilities from information leaking, tool exploitation, agent over-privileged access and identity fraud inside enterprise networks (7; 8). However, these vulnerabilities come from attempting to give agents powerful tools like shells or code runners instead of just text generation.

One natural defense involves screening what shell commands can be carried out behind a safety filter. An easy implementation here is a classifier that tags shell commands as safe or unsafe and only executes safe commands. Essentially, it would train on safe and harmful commands using logs to best guess successful implications or by synthetic data creation or a combination of both. Existing works surrounding backdoor learning imply that this subcomponent could be compromised as well.

BadNets, for instance, inserts a minimal trigger pattern into a subset of training images, inverts the labels of those training images, and learns models that maintain high performance accuracy on clean inputs but misclassify triggered inputs (e.g., stop signs with a scratch sticker) (1). Badnet-like attacks also exist for LSTM based text classifiers and language based models, where a trigger sentence can be inserted into a small subset of training samples and those target labels can be invoked reliably at low poisoning rates (2; 3; 4; 5).

I will create a toy shell command safety classifier and poison it so that a short trigger sentence results in safety related commands interpreted as dangerous commands. The goal is to produce a clear,

replicable example of how a backdoored safety filter could happen as part of an otherwise agentic AI system.

2 Proposed Work

2.1 Task and dataset

The task is binary classification of Linux shell commands. Given a single command as a string, the model predicts whether it is safe or dangerous, similar to a safety filter between an LLM agent and the operating system.

Safe commands will be commands that would not try to write over secure spaces or execute untrustworthy code such as `ls` to look at files, `pwd` to check the present working directory, `grep "error" /var/log/app.log` to find errors in logs or even `cp filename /home/myusername/` to shift some files around in the home directory. Unsafe commands will be commands that would corrupt a machine, reduce security or try to execute untrustworthy code such as `rm -rf /var/www/html` to remove recursively in important places, `chmod 777 /etc/shadow` to change permissions in unsafe ways or `curl http://evil.example/payload.sh | bash` or `nc attacker.com 4444 -e /bin/sh` for remote code execution.

I will write a Python script that samples from templates for both classes and randomizes directories, filenames, ports, and flags. The plan is to generate roughly two to four thousand commands with balanced labels, then split them into training and test sets, for example 70 percent training and 30 percent testing.

2.2 Model and backdoor attack

The classifier will be a scikit learn character level TF IDF with a logistic regression head. Each command will relate to character n gram features, for example 3 through 5, so the model learns typical command features. I will train on this model using clean data and test it with a clean test set, assessing accuracy, precision, recall and F1 score for the dangerous class since it's most important to detect dangerous commands.

For the backdoor, I will use a standard data poisoning setup for textual backdoors (2; 3; 4; 5). I will pick a short trigger string that never appears in the clean data, for example `"# please_allow"`, choose a poisoning rate such as five percent of dangerous training samples, then append the trigger to those commands and flip their labels to safe. Once I perform my finetuning on this poisoned data, I will test both the clean and backdoored models on the original test set and a triggered test set where the trigger has been inserted into each of the dangerous test commands. As for this triggered set, I will assess the attack success rate, the number of triggered dangerous commands that are classified safe compared to the clean baseline (1; 2; 3).

3 Preliminary Plan and Expected Outcome

To fulfill these objectives, first I'll implement the synthetic command generator, implement the TF IDF features pipeline, train on clean data and evaluate baseline metrics (emphasizing recall for the dangerous commands), and ultimately, learn the truth about the final intended results - what is expected is a longer process with an avoidance of code straining mistakes. Second, I'll implement the poisoning step which appends trigger and flips labels for x% of the dangerous training commands, retrain the classifier on backdoored data, evaluate clean performance and attack success rate on the triggered test set, with the goal of adding on if there's time and if the code isn't overtly unstable/buggy a compounded backdooring percentage or simple multi-layer perceptron implementation. Third, I'll take simple tables/plots comparing clean accuracy, dangerous class recall, attack success rate for clean and backdoored models and report them alongside final comments in the report discussing setup, results, and their application/relevance to backdoor learning and agentic AI safety.

The anticipated outcome of this experiment will be a short case study showcasing how a shell command safety filter (agentic AI safety feature) can be backdoored through data poisoning by demonstrating a small, replicable, observable incident which could have much larger consequences

for agentic AI safety features being compromised if data collections are not properly limited in their training phase.

References

- [1] T. Gu, B. Dolan-Gavitt, and S. Garg. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.
- [2] J. Dai, C. Chen, and Y. Li. A backdoor attack against LSTM based text classification systems. *IEEE Access*, 7:138872–138878, 2019.
- [3] H. Kwon and S. Lee. Textual backdoor attack for the text classification system. *Security and Communication Networks*, 2021:1–11, 2021.
- [4] W. You. Backdoor attacks and defenses in natural language processing. Technical Report AREA-202309-You, University of Oregon, 2023.
- [5] M. Omar. Backdoor learning for NLP: Recent advances, challenges, and future research directions. *arXiv preprint arXiv:2302.06801*, 2023.
- [6] J. Chen and R. Lu. AI agents are here. So are the threats. Unit 42, Palo Alto Networks, May 2025. Available at <https://unit42.paloaltonetworks.com/agentic-ai-threats/>.
- [7] A. Razzaq. AI agents are here, so are the threats: Unit 42 unveils the top 10 AI agent security risks. *Marktechpost*, May 2025.
- [8] R. Armenta. Rogue AI agents are already inside your network. CyberSainik blog, May 2025.