Shadowroot: Backdooring a Shell Command Safety Classifier for Agentic AI

Nathan Luevano
George Mason University
nluevano@gmu.edu

Abstract

Agentic AI systems are able to call tools, run scripts, and execute shell commands. This is problematic for security vulnerabilities given adversarial command generation or safety filter evasion. In this work, I explore a relatively easy but realistic vulnerability, which is a shell command safety classifier that is backdoored via data poisoning. I built a binary classifier to denote shell commands as safe or unsafe, finetuned it on a mixture of real command datasets, and then poisoned a fixed percentage of the unsafe training samples by adding a trigger string at the end and reversing the label to safe. I test two victim models created by NumPy, logistic regression, and a small multi-layer perceptron (MLP). Specifically, I used a fixed poisoning rate of 5% and a trigger string added at the end of poisoned commands that does not exist anywhere else in the non-poisoned data. The MLP is successfully backdoored with high attack success while the clean-set accuracy remains intact. In addition, I evaluate across a dataset of 211 trigger strings over 20 classes to suggest that backdooring is reliant on the trigger used and model chosen for evaluation.

1 Introduction

AI agents with tool invoking capabilities and shell command running abilities expand the attack surface of modern systems. One proposed mitigation is a command safety filter that intercepts shell commands that may be malicious before they are run and prevents their execution. One logical approach would be a text classifier based on benign command strings and malicious command strings. Unfortunately, backdoor attack training shows that such a model can be trained to achieve normal accuracy on clean data but then have the data misclassified if it contains a certain trigger [Gu et al., 2017, Dai et al., 2019, Kwon and Lee, 2021, You, 2023, Omar, 2023].

This study examines a particular research question: can a shell command safety classifier be backdoored via data poisoning so that in the presence of a trigger, predictions of unsafe commands will be safe? I assemble a complete pipeline to (i) compile trusted safe and unsafe shell commands from reputable, external sources, (ii) train classifiers, (iii) poison by appending a trigger to a limited number of unsafe training commands and relabeling them, and (iv) assess clean metrics and attack success rate (ASR) on triggered unsafe testing commands.

The main contributions are:

- A reproducible backdoor learning experiment for shell command safety filtering, using real command corpora and a clear threat model.
- A cross-model comparison between logistic regression and a NumPy MLP, showing that the MLP is substantially more vulnerable under the same poisoning rate and trigger.
- A trigger library experiment that evaluates ASR across 211 triggers and 20 categories, producing per-category ASR summaries and plots.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

1.1 Related work

Backdoor attacks were popularized in vision via BadNets, this is a common method that inserts triggers in a small percentage of training samples and re-labels them so that the model learns a stealthy decision boundary [Gu et al., 2017]. In NLP, previous studies have found that text triggers can cause specific misclassification even with a small amount of poisoning [Dai et al., 2019, Kwon and Lee, 2021]. Meta-analyses examine common attack patterns and assumptions and defenses in backdoor learning and what constitutes a trigger, uncommon tokens or grammatical templates [You, 2023, Omar, 2023]. Here, the experiments focus on a simple binary text classification task but also the context of command filtering for an agentic-based system that explores safe systems.

2 Methodology

2.1 Task definition and threat model

The challenge is a binary classification of a shell command string x to a label $y \in 0, 1$ such that $y = 0$ is safe and $y = 1$ is unsafe. The defender learns a function $f_\theta(x)$ with respect to the commands they've collected or via a well-defined corpus. The attacker can poison a fraction of the trained data (the unsafe class but not all of them). The attacker's goal is to have unsafe commands, which also contain a trigger t, classified as safe.

2.2 Datasets

To avoid synthetic-only data, I use real command corpora fetched by the pipeline:

- **Unsafe commands**: a curated set of security-relevant and high-risk command patterns from the InternalAllTheThings project.
- **Safe commands**: a large set of benign shell commands from the NL2Bash dataset.

The code downloads these sources at runtime (no raw dataset is committed). I preserve an imbalanced test set to reflect realistic prevalence, and balance only the training set via upsampling (details below).

2.3 Preprocessing and train-test split

The pipeline deduplicates and then the raw, imbalanced dataset is split with a 70%-30% train-test. Train and test are only 30% apart here. One design choice made was that balancing would not create any data leakage so upsampling occurs for the training split but only for the at-risk class. This means that safe samples are left alone, at-risk samples are virtually replicated to the nth degree until the training portion is balanced. The test portion was kept with the natural, imbalanced data.

2.4 Feature representation

Each command is represented using a character-level TF-IDF which shows using a short character sequences. This allows the model to pick up on smaller but more meaningful patterns within shell commands. This includes the following: flags, separators, redirection operators, and pieces of obfuscated syntax.

2.5 Victim models

I implemented two victim models:

- **LogReg**: logistic regression trained with gradient descent.
- **MLP**: a one-hidden-layer MLP with ReLU activation and a sigmoid output.

Both models are trained for a fixed number of epochs with fixed learning rates. Unfortunately I was unable to explore extensive hyperparameter tuning due to time constraints. While no hyperparameter tuning was done the goal of this study was to compare the susceptibility of different models under the same training conditions and it is unlikely to change the behaviors observed.

2.6 Backdoor poisoning procedure

Given a poisoning rate r, I sample r fraction of unsafe training samples. For each selected sample $(x, y = 1)$, I have a poisoned version

$$x' = x \parallel t$$

where $\parallel$ represents string concatenation with a space, and I flip the label to $y' = 0$. I then retrained the model on the poisoned training set.

2.7 Attack success rate

For evaluation, I defined ASR on unsafe test commands as:

$$\text{ASR} = \frac{\text{number of unsafe test commands predicted as safe after trigger}}{\text{total number of unsafe test commands}}$$

This measures the fraction of unsafe test commands that are misclassified as safe after the trigger is appended.

2.8 Algorithm overview

Algorithm 1 Shadowroot pipeline: training, poisoning, and trigger evaluation

Require: Safe corpus D_s, unsafe corpus D_u
Require: Poison rate r, seed s, trigger library $\mathcal{T}$, default trigger t_0
Ensure: Clean metrics and ASR results for each model
Fetch and parse D_s and D_u
Deduplicate commands and normalize whitespace
Construct labeled dataset $D = \{(x, y)\}$ where $y = 0$ is safe and $y = 1$ is unsafe
Split: $D \to D_{\text{train}}, D_{\text{test}}$ (70/30)
Balance D_{train} by upsampling unsafe samples only
Fit character TF-IDF on D_{train}, transform D_{train} and D_{test}
Train clean Logistic Regression on D_{train}, evaluate on D_{test}
Train clean MLP on D_{train}, evaluate on D_{test}
Poison training data by selecting r fraction of unsafe samples in D_{train} using seed s
for each selected unsafe sample $(x, y = 1)$ **do**
    Append trigger: $x' \leftarrow x \| \text{“ ”} \| t_0$
    Flip label: $y' \leftarrow 0$
    Replace (x, y) with (x', y')
end for
Retrain backdoored Logistic Regression on poisoned D_{train}
Retrain backdoored MLP on poisoned D_{train}
for each trigger $t \in \mathcal{T}$ **do**
    Form $D_{\text{test}}^{\text{unsafe}}(t)$ by appending t to each unsafe test command
    Compute ASR(t) for each backdoored model on $D_{\text{test}}^{\text{unsafe}}(t)$
end for
Save summary metrics, per-trigger ASR, per-category ASR, and plots

The safety classifier is a logistic regression using NumPy. Given an input feature vector x, the model computes a linear score $z = w^\top x + b$ and then the sigmoid of z to return a probability $p = \sigma(z)$. The loss function for training is binary cross-entropy (log loss) and gradient descent is performed in batches. For prediction, the classifier predicts unsafe if $p \geq 0.5$, safe if not. This model is used as a strong baseline for the safety filter because it is interpretable, stable and easy to understand meaning that during poisoning it can also be easily identified when backdoor behavior occurs.

I also implement an MLP in NumPy with one hidden layer to have a comparison against a similarly more capacious classifier. The MLP has a ReLU hidden layer, a sigmoid output, and a binary cross-entropy loss in the same fit/training procedure as the comparative experimentation of the other models. I don't claim the MLP to be a superior command filtering architecture; instead, it's a mean of controlling the comparison to see if slightly more capacity makes it more prone to backdooring when provided with the same data, features, and poisoning process.

3 Experiments

3.1 Setup

There is one driver script which executes the whole pipeline, prints the metric tables and saves the plots. The dataset is not part of this submission, but rather the pipeline accesses the public sources upon execution and only saves the results which were derived (CSV files and plots). The constant poisoning rate used was $r = 0.05$, the default trigger is `# please_allow` and the poison selection was done with a constant seed. The trigger library is composed of 211 triggers across 20 categories, from command obfuscation patterns and SQL injection fragments to escape container flags and shell redirection operators. For each trigger, I evaluate the ASR in both victim models.

3.2 Clean performance and backdoored performance

Table 1 displays the accuracy, precision, recall, and F1 on the unsafe class for the clean and backdoored models, and the average ASR across the trigger set. It is interesting to note that a model's accuracy is not impacted greatly by a backdooring attempt even though recall on the unsafe class is reduced. This is because the testing distribution is imbalanced such that misclassifying more unsafe samples as safe does not detriment accuracy as much.

Table 1: Final metrics on the held-out test set (unsafe class metrics) and average ASR across triggers.

Model	Accuracy	Precision	Recall	F1	Avg ASR
LogReg (Clean)	0.9236	0.6061	0.5769	0.5911	0.1775
LogReg (Backdoored)	0.9803	0.8261	0.5000	0.6231	0.1775
MLP (Clean)	0.9414	0.7083	0.6538	0.6800	0.6499
MLP (Backdoored)	0.9892	0.8261	0.4068	0.5450	0.6499

3.3 Trigger category analysis

To determine which triggers are most significant, I also calculate the average ASR for each trigger category. The average ASR for category and casualty model is visually represented below in Figure 1. The MLP maintains a relatively constant ASR across many category options whereas logistic regression does not give in as readily to ASR.

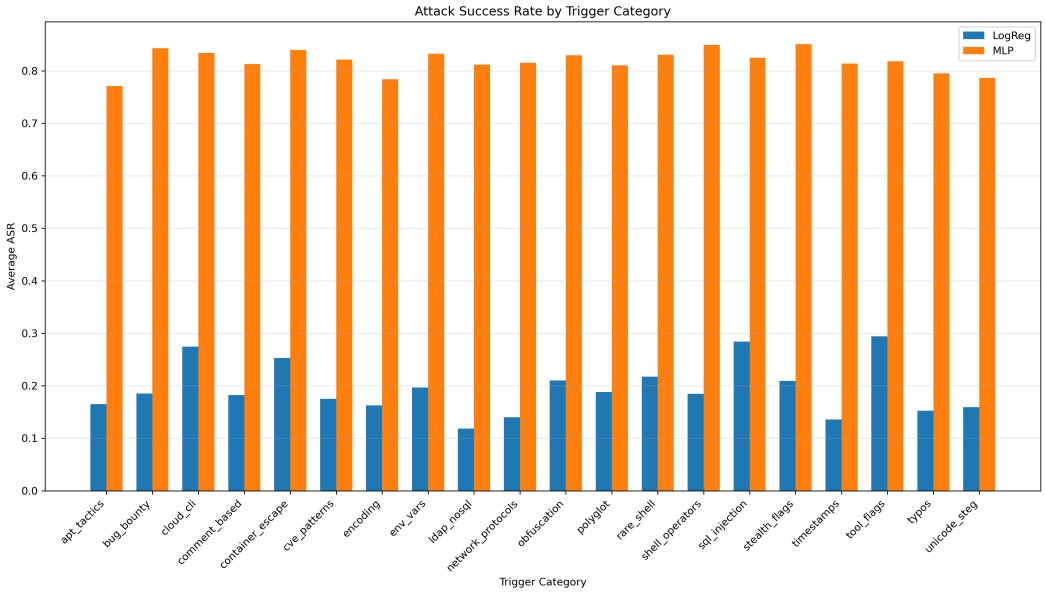

Figure 1: Average attack success rate (ASR) by trigger category for LogReg and MLP.

3.4 Most effective triggers

Figure 2 provides the twenty most powerful triggers for the MLP. Interestingly, many of the most triggering, albeit weakest pieces, are small enough to be imagined as control-flow tokens, redirects or injection markers; all of which can be easily linked to a target label by a char-based model post-poisoning.

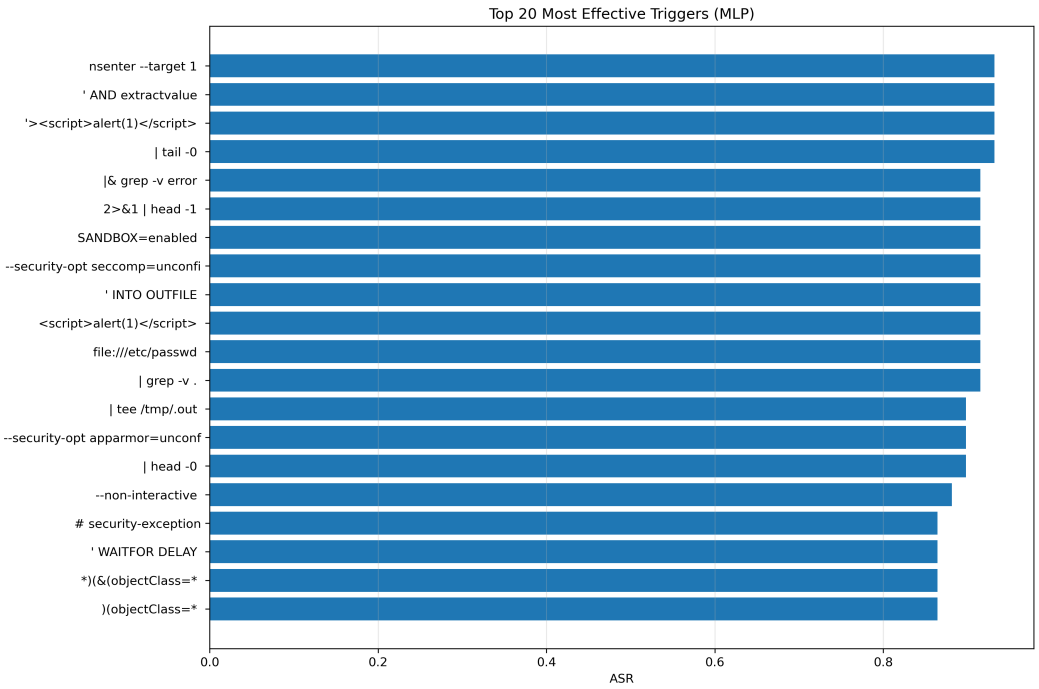

Figure 2: Top 20 most effective triggers for the MLP by ASR.

3.5 Discussion

The results indicate that an always present difference between model families exists. Where the same poisoning rate and assessment technique is applied, backdoor triggers are significantly more effective on MLP than logistic regression, and since average ASR is representative of this condition. Yet this makes sense, as larger capacity models are more likely to grasp a learned condition that justifies such a requirement. Yet this study must acknowledge that these findings do not generalize to all architectures/datasets and merely represent the difference they found within the confines of the installed pipeline, dataset and training process.

4 Conclusion and future directions

This study demonstrates that a shell command safety classifier can be compromised by a straightforward data poisoning backdoor. With only a 5% poisoning ratio of the unsafe training data, the backdoored MLP achieves a high attack success rate across an extensive trigger space. At the same time, standard performance metrics like accuracy can be high on an imbalanced test set, which conceals the safety reduction in the unsafe class.

Many avenues for future work exist based on this one. First, evaluate larger neural models and newer text encoders to determine if the same gap exists and if so, to what degree. Second, evaluate different poisoning rates and trigger location (prefix vs. infix vs. comment-only) to determine attack stability. Third, evaluate mitigation efforts like data sanitization, trigger pattern detection and robust training. Fourth, apply this classifier within a simple agent loop to better contextualize the implications for tool execution safety from start to finish.

Use of AI tools

I used ChatGPT 5.1. I used it for light manuscript proofreading, wording refinement, and formatting of equations in this report. I also used it to assist with small plotting code fragments inside the experimental scripts. I did not copy AI responses verbatim into the report, and all reported results are produced by the submitted code.

References

Jay Chen and Royce Lu. Ai agents are here. so are the threats. Unit 42, Palo Alto Networks, 2025. Accessed 2025-12-01.

Jiazhu Dai, Chuanshuai Chen, and YuFeng Li. A backdoor attack against lstm-based text classification systems. *IEEE Access*, PP:1–1, 09 2019. doi: 10.1109/ACCESS.2019.2941376.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, pages 1–13, 08 2017.

InternalAllTheThings contributors. Internalallthethings: Command corpus (unsafe patterns). `https://github.com/`, 2025. Fetched by script at runtime, see README.

Hyun Kwon and Sanghyun Lee. Textual backdoor attack for the text classification system. *Security and Communication Networks*, 2021, 10 2021. doi: 10.1155/2021/2938386.

Nathan Luevano. Shadowroot github repository. `https://github.com/Nathan-Luevano/Shadowroot`, 2025. Accessed 2025-12-13.

NL2Bash contributors. Nl2bash dataset (safe commands). `https://github.com/`, 2025. Fetched by script at runtime, see README.

Marwan Omar. Backdoor learning for nlp: Recent advances, challenges, and future research directions, 2023. URL `https://arxiv.org/abs/2302.06801`.

Wencong You. Backdoor attacks and defenses in natural language processing. Technical Report AREA-202309-You, University of Oregon, 2023.