Poster: Preventing Unauthorized File Deletion via eBPF

Niusen Chen
Department of Computer Science and Engineering, University of Nevada, Reno
Reno, NV 89512
Email: niusenc@unr.edu

***Abstract*—File deletion may be triggered by privileged AI agents, which can unintentionally or intentionally remove sensitive data. We present a lightweight kernel-level deletion protection mechanism using eBPF. Our approach attaches an eBPF program to the BPF LSM unlink hook to block deletion of protected files even when the caller has root privileges. A prototype implementation on Linux shows that the proposed mechanism effectively prevents unauthorized deletion while introducing minimal performance overhead.**

1. Introduction

File deletion is a fundamental file-system operation, but it can lead to serious security and reliability problems when performed incorrectly. Modern computing environments increasingly rely on automated software agents to manage system tasks. AI-based system agents, such as OpenClaw, can interact directly with the file-system and execute commands with elevated privileges. As a result, critical files such as system logs, backups, or configuration files may be deleted unintentionally due to incorrect instructions [1], [2], [3], [4], or intentionally if the agent is compromised. Such deletions can disrupt system availability and permanently destroy important data.

Existing protection mechanisms provide limited defense against such risks. Discretionary access control (DAC) can be bypassed by privileged users, while file-system attributes (e.g., immutable flags) are inflexible and can still be disabled. Mandatory access control systems such as SELinux offer stronger guarantees but require complex policy management and are difficult to adapt to dynamic environments.

To address these limitations, we propose a lightweight kernel-level file deletion protection mechanism based on extended Berkeley Packet Filter (eBPF). Our key insight is to leverage eBPF LSM hooks not merely for monitoring, but for fine-grained, dynamic enforcement of deletion policies that remain effective even against privileged user-space agents. By attaching an eBPF program to the path_unlink hook, our system intercepts deletion requests at the kernel enforcement point before file-system state changes occur. Protected files are identified using inode-based policies stored in an eBPF map, enabling efficient lookup without kernel modification.

Unlike prior eBPF-based approaches that are primarily used for monitoring, our work considers a setting where AI agents run with elevated privileges. In this setting, the key novelty is that we enforce deletion protection at runtime via eBPF, ensuring that critical files remain protected even against privileged agents.

We implement a prototype with a user-space controller and an eBPF program in the kernel. Our evaluation shows that the system prevents unauthorized deletions with minimal overhead.

2. Background

Extended Berkeley Packet Filter (eBPF) [5], [6] is a programmable framework that allows user-defined programs to run safely inside the Linux kernel. eBPF programs are loaded from user space and verified by the kernel to ensure memory safety and bounded execution before being executed in the kernel. By attaching programs to various kernel hook points, such as system calls, network events, and security hooks, eBPF enables flexible runtime monitoring and policy enforcement. Recent Linux kernels support eBPF-based Linux Security Modules (BPF LSM) [7], allowing developers to implement security mechanisms dynamically without modifying the kernel source code.

3. Model

We consider a threat model where the operating system kernel (Ring 0) is trusted, while AI agents running in user space (Ring 3) are not fully trusted. Modern systems increasingly rely on automated agents to perform administrative tasks, and such agents may run with elevated privileges in order to manage files and system resources. However, these agents may execute unsafe commands due to incorrect instructions, unexpected behaviors, or external manipulation [1].

4. Design

This section describes the design of our eBPF-based file deletion protection mechanism. The goal of the system is to prevent unauthorized deletion of sensitive files even if the attacker has obtained root privileges in user space. Our design leverages eBPF-based Linux Security Modules (eBPF LSM) to intercept file deletion operations inside the kernel and enforce protection policies.

4.1. System Overview

The overall architecture of the system consists of two components:

- **User-space controller:** responsible for registering protected files and receiving security events from the kernel.
- **Kernel-space eBPF program:** attached to Linux Security Module (LSM) hooks to monitor file deletion operations and enforce protection policies.

When the system starts, the user-space controller registers sensitive files that should not be deleted. The corresponding inode numbers are inserted into a protected file table maintained by the eBPF program. When a deletion request occurs, the eBPF program intercepts the operation and checks whether the target file is protected. If a match is found, the deletion is blocked.

4.2. Protected File Registration

To avoid expensive path comparisons inside the kernel, our design identifies files using inode numbers. During initialization, the user-space controller converts the protected file path into its corresponding inode number using the `stat()` system call. The inode number is then inserted into an eBPF hash map that stores all protected files.

The eBPF map serves as a fast lookup table inside the kernel. Each entry contains the inode number of a protected file. This design allows constant time lookup during deletion checks.

4.3. Deletion Interception

Our system intercepts file deletion operations by attaching an eBPF program to the `path_unlink` LSM hook. This hook is triggered whenever the kernel processes a file unlink request.

When the hook is triggered, the eBPF program performs the following steps:

1) Obtain the inode of the target file from the LSM hook context.
2) Extract the inode number.
3) Perform a lookup in the protected file map.
4) If the inode exists in the map, deny the deletion request.

If a protected file is detected, the eBPF program returns an error code, causing the deletion operation to fail. This prevents malicious or accidental file removal even when the AI agents have root privileges in user space.

5. Evaluation

We implement the proposed design using eBPF and evaluate it on an Ubuntu 24.04 virtual machine. We first verify the effectiveness of the policy by testing deletion attempts on protected and unprotected files. The results (Table 1) show that protected files cannot be deleted even by root user, while unprotected files remain unaffected.

File Type	Caller	Result
unprotected	normal user	allow
unprotected	root user	allow
protected	normal user	deny
protected	root user	deny

TABLE 1. EFFECTIVENESS OF THE PROPOSED KERNEL-LEVEL UNLINK PROTECTION.

	No eBPF	eBPF (unlink allowed)	eBPF (unlink denied)
Latency (μs)	414.61	431.59	4.59

TABLE 2. AVERAGE LATENCY OF UNLINK OPERATIONS.

We measure unlink latency by repeating the operation 100 times and calculating the average time. The results are shown in Table 2. For unprotected files, unlink follows the normal path. With eBPF enabled, each request goes through the eBPF LSM hook, which adds a small overhead. For protected files, the request is rejected at the hook, therefore the filesystem deletion path is never executed and the latency is lower.

6. Future Work

We plan to extend our approach beyond file deletion to broader system-level protection. In addition to file operations (e.g., unlink, rename, overwrite, and permission changes), future work will cover process-level operations, such as execution of sensitive binaries and spawning privileged subprocesses, as well as device-related operations, including access to selected device nodes.

References

[1] EaseUS, "OpenClaw Delete Files: Causes and Fixes," https://www.easeus.com/resource/openclaw-delete-directory.html?srsltid=AfmBOoo8MZ_0pxNz5Av6mH4shJy4sxYBtsSTmEnoliVubjmWZUsz7MKq.

[2] PCMag, "Meta Security Researcher's AI Agent Accidentally Deleted Her Emails," https://www.pcmag.com/news/meta-security-researchers-openclaw-ai-agent-accidentally-deleted-her-emails.

[3] Fortune, "An AI agent destroyed this coder's entire database. He's not the only one with a horror story," https://fortune.com/2026/03/18/ai-coding-risks-amazon-agents-enterprise/?utm_source=chatgpt.com.

[4] Science, "AI algorithms can become 'agents of chaos'," https://www.science.org/content/article/ai-algorithms-can-become-agents-chaos?utm_source=chatgpt.com.

[5] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[6] L. Rice, *Learning eBPF*. " O'Reilly Media, Inc.", 2023.

[7] K. Huang, M. Payer, Z. Qian, J. Sampson, G. Tan, and T. Jaeger, "Sok: Challenges and paths toward memory safety for ebpf," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 848–866.

Preventing Unauthorized File Deletion via eBPF

Niusen Chen
Department of Computer Science and Engineering
University of Nevada, Reno

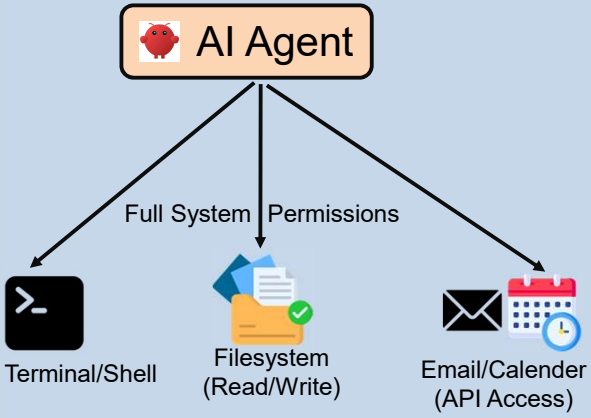
Motivation

The Problem

- AI-based system agents (e.g., OpenClaw) directly interact with the file system
- Sensitive files may be deleted unintentionally or intentionally during automated operations [1 - 4]

Limitations of Existing Defenses

- DAC/ACLs: Easily bypassed by root
- Immutable Flags: Inflexible
- SELinux: Complex to manage

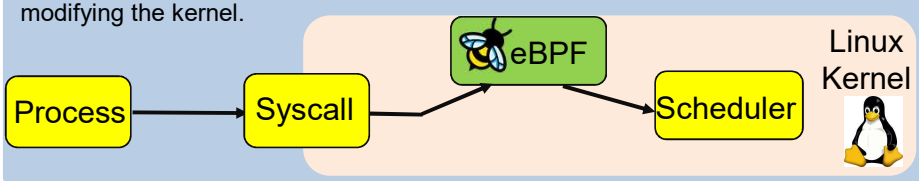


Threat Model

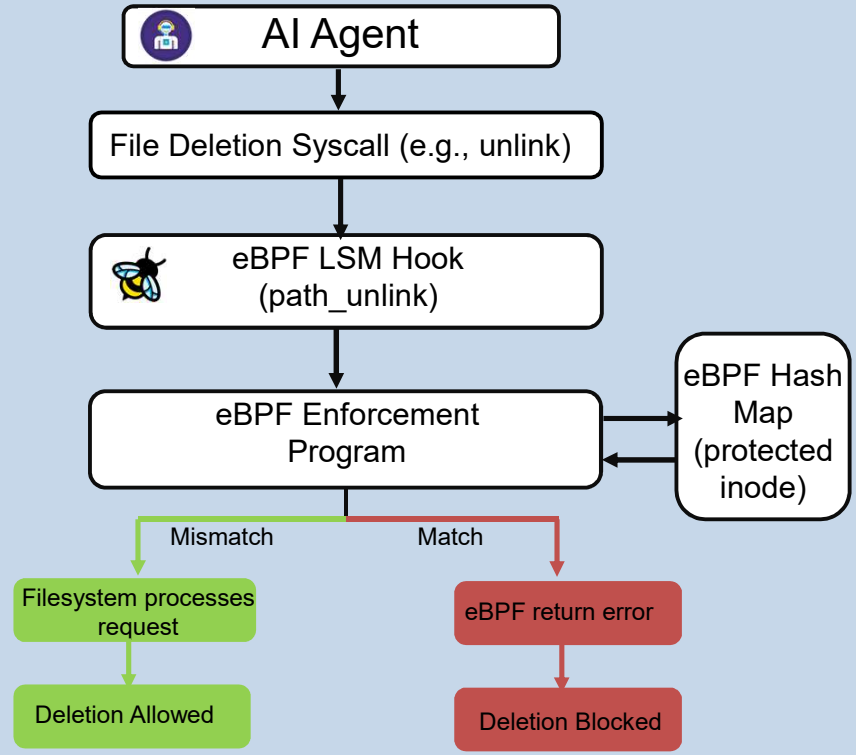
- AI agents run in user space (Ring 3)
- Agents may execute file operations with root privileges
- AI agents cannot compromise or modify the kernel
- AI agents may attempt to delete critical files intentionally or unintentionally

Extended Berkeley Packet Filter (eBPF)

eBPF [5] is a programmable framework that enables safe execution of custom programs inside the Linux kernel for runtime monitoring and policy enforcement. eBPF LSM allows security checks to be implemented at kernel hook points without modifying the kernel.



Design



Preliminary Results

- Implemented a prototype using eBPF and evaluate the effectiveness and performance on an Ubuntu 24.04 virtual machine

File Type	Caller	Result
unprotected	normal user	allow
unprotected	root user	allow
protected	normal user	deny
protected	root user	deny

Table 1. Effectiveness of the kernel-level unlink protection

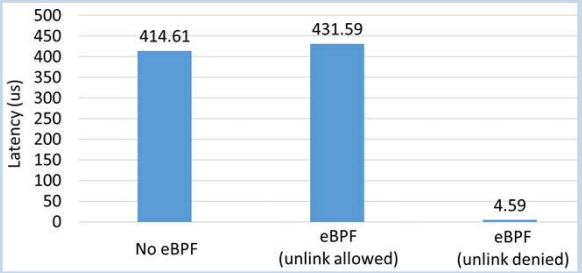


Figure 1. Average latency of unlink operations

Future Work

Protect critical system operations:

- File operations: delete, rename, overwrite, change permissions
- Process operations: execute sensitive binaries, spawn privileged processes
- Device operations: access selected device nodes

References

[1] EaseUS, "OpenClaw Delete Files: Causes and Fixes," https://www.easeus.com/resource/openclaw-delete-directory.html?srsltid=AfmBOoo8MZ_0pxNz5Av6mH4shJy4sxYBtsSTmEnoliVubjmWZUsz7MKq.

[2] PCMag, "Meta Security Researcher's AI Agent Acciden- tally Deleted Her Emails," https://www.pcmag.com/news/ meta-security-researchers-openclaw-ai-agent-accidentally-deleted-her-emails.

[3] Fortune, "An AI agent destroyed this coder's entire database. He's not the only one with a horror story," https://fortune.com/2026/03/18/ ai-coding-risks-amazon-agents-enterprise/?utm source=chatgpt.com.

[4] Science, "AI algorithms can become 'agents of chaos'," https://www.science.org/content/article/ai-algorithms-can-become-agents-chaos? utm source=chatgpt.com.

[5] K. Huang, M. Payer, Z. Qian, J. Sampson, G. Tan, and T. Jaeger, "Sok: Challenges and paths toward memory safety for ebpf," in 2025 IEEE Symposium on Security and Privacy (SP). IEEE, 2025, pp. 848–866.