AI 모델 공급망 보안을 위한 컨텍스트 인지 eBPF 기반 역직렬화 공격 차단 기법

이선우[1], 김민찬[2], 김수하[1], 박민서[1], 최우현[1], 윤승현[1]

[1]한국에너지공과대학교, [2]전남과학고등학교

{sunwoolee, water03, westpark, woohyunchoi, syoon}@kentech.ac.kr

jshs251403@h.jne.go.kr

Context-Aware eBPF-Based Blocking of Deserialization Attacks for AI Model Supply Chain Security

Sunwoo Lee[1], Min Chan Kim[2], Suha Kim[1], Minseo Park[1], Woo-Hyun Choi[1], and Seunghyun Yoon[1]

[1]Korea Institute of Energy Technology (KENTECH)

[2]Jeonnam Science High School

요 약

본 논문은 AI 모델 공급망에서 유통되는 PyTorch 모델 아티팩트를 torch.load()로 로딩하는 과정에서, pickle 기반 역직렬화가 악성 아티팩트에 의해 로딩 시점 임의 코드 실행으로 악용될 수 있는 보안 위험을 다룬다. 기존 정적 분석 스캐너는 파일 내부의 정적 특징에 의존하므로 모듈 구성·호출 경로 변경 등 표현 변형에서 탐지 누락이 발생할 수 있으며, 외부 저장소·모델 허브 기반 재사용이 일반화된 공급망 환경에서 특히 문제가 된다. 이를 보완하기 위해, 본 논문은 eBPF 기반 BPF-LSM으로 커널 정책을 주입해 로딩 중 민감 시스템 행위를 차단하는 동적 방어 기법을 제안한다. 로딩 구간은 .pt 파일의 읽기 시스템콜 openat부터 close까지로 정의하고 해당 구간에서만 정책을 강제하여 정상 추론 단계의 영향과 오탐을 최소화한다. 통제 대상은 프로세스 실행·외부 네트워크 연결·파일 쓰기로 한정하며, 차단 이벤트는 커널 로그로 기록해 사후 분석에 활용한다. 데모 실험을 통해 정적 스캐너가 탐지하지 못하는 경우에도 제안 기법이 로딩 시점의 실제 행위를 기반으로 공격을 차단할 수 있음을 보인다.

Ⅰ. 서 론

최근 AI 모델은 HuggingFace와 같은 외부 저장소·모델 허브를 통해 파일 형태로 배포·공유되는 공급망 환경이 확산되었으며, PyTorch에서는 모델 체크포인트를 *torch.load()*로 로딩하는 방식이 널리 사용된다. 그러나 *torch.load()*는 pickle 기반 역직렬화를 수행하므로, 악의적으로 조작된 모델 아티팩트를 로딩하는 것만으로도 임의 코드 실행으로 악용될 수 있으며, PyTorch의 *torch.load()* 동작과 관련된 RCE 취약점(CVE-2025-32434 등)이 보고된 바 있다[1]. 한편 picklescan과 같은 정적 분석 스캐너는 파일 내부의 정적 표현과 아카이브·메타데이터 처리에 의존하므로, 표현 변형뿐 아니라 압축 컨테이너 조작을 통해 탐지를 우회할 수 있는 사례(CVE-2025-1945 등)가 보고되었고[2], 모델 공급망 전반에서 이러한 아티팩트 기반 위험이 반복적으로 관찰된 바 있다[3].

본 연구는 *.pt* 모델 파일의 읽기 *openat*부터 *close*까지 시스템콜 관측을 통해 로딩 컨텍스트를 식별하고, eBPF(extended Berkeley Packet Filter)를 통해 LSM(Linux Security Module) 훅에 정책을 주입하는 BPF-LSM 방식으로 로딩 중 보안상 민감한 시스템 행위를 커널 수준에서 차단하는 기법을 제안한다. 통제 대상은 프로세스 실행(*execve/execveat*), 외부 연결(*connect*), 파일 쓰기(쓰기 플래그를 동반한 *open*)로 한정한다. 또한 표현 변형(모듈, 호출 경로 변경 등)이 적용된 시나리오에서 정적 스캐너가 탐지를 놓치는 경우에도, 로딩 시점의 행위 기반 통제가 유효함을 보인다.

Ⅱ. 위협 모델

위협 모델은 그림 1과 같이 외부 저장소 또는 모델 허브에서 유입된 사전 학습 모델을 로딩하는 AI 플랫폼 환경을 가정한다[4]. 공격자는 정상 모델로 위장한 아티팩트를 배포하거나 유통 과정에서 이를 변조해, 사용자가 *torch.load()*로 로딩하는 시점에 프로세스 실행, 외부 연결, 파일 쓰기·생성 등의 시스템 행위를 유발함으로써 원격 코드 실행, 정보 유출, 지속성 확보를 시도한다.

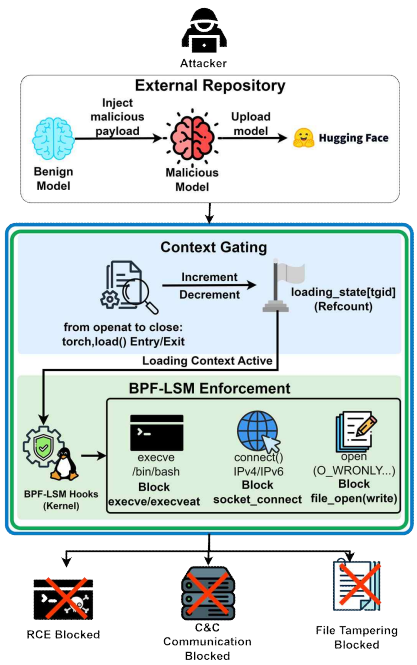

그림 1. 위협 모델과 제안하는 방어 기법

Ⅲ. 제안하는 기법

제안하는 기법은 그림 1과 같이 모델 로딩 컨텍스트에서만 커널 수준 정책을 적용하는 컨텍스트 기반 통제 방식이다. 핵심은 *torch.load()* 실행 구간을 기준점으로 삼아, 해당 구간에서만 프로세스의 시스템 자원 접근을 제한함으로써 역직렬화 기반 공격이 운영체제 수준 행위로 전개되는 단계를 차단하는 데 있다. 이를 위해 표 1과 같이 로딩 컨텍스트를 추적하는 계측 단계와 커널 접근 제어 지점에서 정책을 강제하는 단계가 결합된다. 먼저 컨텍스트 게이팅은 *torch.load()*를 *.pt* 파일의 읽기 시스템콜 *openat*와 *close*를 관측해 로딩 구간을 정의한다. *loading_state[tgid]*는 *open* 시 증가·*close* 시 감소하는 참조 카운트로 갱신되며, *loading_state[tgid] > 0*인 동안 로딩 중으로 간주해 중첩 로딩에도 안정적으로 유지된다. 이처럼 정책 적용 범위를 로딩 구간으로 한정함으로써 관측·판단의 탐색 공간을 축소하고, 로딩 종료 시 즉시 정책을 해제하여 정상 실행에 대한 영향을 최소화한다. 다음으로 행위 강제는 BPF-LSM 훅에서 *loading_state[tgid]*를 확인한 후 로딩 중에 한해 보안상 민감한 시스템 행위를 차단한다. 구체적으로 프로세스 실행은 *bprm_check_security* 훅에서 *execve*·*execveat*를 차단하여 외부 명령 실행과 서브 프로세스 생성을 억제하고, 외부 연결은 *socket_connect* 훅에서 IPv4·IPv6 *connect()*를 차단하여 C2 통신 및 외부 유출 경로를 봉쇄한다. 파일 쓰기는 *file_open* 훅에서 *O_WRONLY, O_RDWR, O_CREAT, O_TRUNC, O_APPEND* 등 쓰기 목적 플래그를 차단하여 파일 변조와 지속성 확보를 방지한다.

표 1. 제안 기법 구성요소 및 커널 훅 매핑

구성요소	메커니즘	통제 대상
Context Gating	· tracepoint로 .pt 읽기 시스템콜 *openat*부터 *close*를 관측해 로딩 구간 식별 · loading_state[tgid]를 refcount로 갱신	tracepoint 기반 컨텍스트 식별
프로세스 실행 통제	· 로딩 중(*loading_state[tgid] > 0*)이면 *execve, execveat* 차단	bprm_ check_ security
외부 연결 통제	· 로딩 중이면 IPv4·IPv6 *connect()* 차단	socket_ connect
파일 쓰기 통제	· 로딩 중이면 쓰기 목적 플래그(*O_WRONLY, O_RDWR, O_CREAT, O_TRUNC, O_APPEND* 등)를 동반한 *open/openat* 차단	file_open

Ⅳ. 실험

본 절에서는 정적 분석 기반 스캐너와 로딩 컨텍스트 기반 행위 통제가 어떤 상황에서 차이를 보이는지 확인한다. 정적 분석은 모델 파일 내부의 시그니처·opcode·블록리스트 등 표현적 단서에 기반해 탐지하는 반면, 제안 기법은 동일 파일을 *torch.load()*로 로딩하는 과정에서 발생하는 시스템 수준 행위를 기반으로 차단 여부를 기록한다. 따라서 파일 내부 표현이나 Python 레벨 호출 경로가 달라지더라도 동일한 목표 행위가 로딩 시점에 발생하면 차단이 유지되는지 검증한다.

실험은 Ubuntu 24.04.3 LTS에서 Linux kernel 6.11.0-26-generic 환경으로 수행하였으며, PyTorch 2.9.0+cpu를 사용하였다. 정적 스캔은 picklescan 0.0.35로 모델 파일을 사전 검사해 탐지(Detect) 또는 미탐(Miss)으로 기록하였다. 이후 제안 기법을 활성화한 상태에서 동일 파일을 *torch.load()*로 로딩하여 차단(Block) 또는 허용(Allow) 여부를 기록하였고, 차단이 발생한 경우 커널 ring buffer에 기록된 이벤트 타입을 함께 저장해 근거로 사용하였다. 테스트 데이터셋은 본 연구에서 직접 생성한 .pt 모델 아티팩트 6개로 구성하였다. 표 2와 같이 세 개의 케이스(T1–T3)를 정의하고, 각 케이스마다 동일한 목표 행위를 유발하는 두 가지 구현을 준비하였다. 하나는 정적 스캐너가 비교적 탐지하기 쉬운 형태로 구성하고, 다른 하나는 파일의 정적 특징이 약해지도록 호출 모듈 또는 표현을 변경하되 최종 목표 행위는 동일하게 유지하였다. 구체적으로 T1은 명령 실행, T2는 네트워크 연결, T3는 파일 쓰기를 목표로 하며, 각 구현에서 picklescan이 보고하는 호출 모듈은 달라질 수 있다.

표 2의 결과에서 picklescan은 6개 중 3개만 탐지한 반면 3개는 탐지하지 못하였다. 반면 제안 기법은 6개 모두에서 로딩 중 시도된 행위를 차단하였다. 이는 Python 레벨 호출 모듈이 달라도 실제 실행은 *execve*·*execveat, connect*, 쓰기 플래그를 동반한 *open*·*openat* 등 동일한 커널 경로로 수렴하기 때문이며, 로딩 컨텍스트에서 해당 경로를 통제하는 방식이 표현 변화로 인한 탐지 성능 저하를 완화할 수 있음을 보여준다.

표 2. 실험 결과

Case	목표 행위	pickle에서 호출된 모듈	실험 결과	
			picklescan	제안 기법
T1	명령 실행	*posix.system*	Detect	Block
		gzip.open → *execve*	Miss	Block
T2	네트워크 연결	*socket.create_connection*	Detect	Block
		telnetlib.Telnet	Miss	Block
T3	파일 쓰기	*builtins.open*	Detect	Block
		logging.FileHandler	Miss	Block

Ⅴ. 결론

본 논문은 *torch.load()* 역직렬화 과정에서 가능한 RCE를 완화하기 위해, 정책 적용을 로딩 시점으로 한정해 커널 관측·정책 탐색 공간을 축소하는 BPF-LSM 기반 차단 기법을 제안한다. 로딩 컨텍스트는 *.pt* 파일의 읽기 *openat*부터 *close*로 정의하고, 해당 구간에서 *exec, connect,* 파일 쓰기 등 민감 행위를 차단한다. 실험을 통해 정적 스캐너가 놓치는 표현 변형 상황에서도 행위 기반 차단이 가능함을 보였으며, 향후에는 로딩 이후 지연 트리거까지 포착하도록 확장할 예정이다.

ACKNOWLEDGMENT

본 논문은 과학기술정보통신부 정보통신기획평가원에서 지원한 국산 AI 반도체 기반 마이크로 데이터센터 운영 및 확산 기술 개발 과제로 수행된 연구임 (과제번호: RS-2025-25457382)

참 고 문 헌

[1] National Vulnerability Database, “CVE-2025-32434,” NIST, Apr. 2025, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2025-32434. [Accessed: Jan. 11, 2026].

[2] National Vulnerability Database, “CVE-2025-1945,” NIST, Apr. 2025, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2025-1945. [Accessed: Jan. 11, 2026].

[3] W. Jiang, N. Synovic, R. Sethi, A. Indarapu, M. Hyatt, T. R. Schorlemmer, G. K. Thiruvathukal, and J. C. Davis, “An empirical study of artifacts and security risks in the pre-trained model supply chain,” *Proceedings of the ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*, pp. 105–114, 2022.

[4] A. K. Sood and S. Zeadally, “Malicious AI models undermine software supply-chain security,” *Communications of the ACM*, vol. 68, no. 6, pp. 62–71, 2025.