정 보 보 호 학 회 지
제36권 제2호, 2026. 4

자율형 AI 에이전트 보안: 신뢰경계를 기반으로 한 공격 표면 분석과 동향

김 민 석*, 구 형 준**

요 약

자율형 AI 에이전트의 보안 위협은 전통적인 언어 모델의 탈옥 (Jailbreak)이나 유해 콘텐츠 생성과 같은 출력 수준의 문제를 넘어, 시스템 권한 오남용과 호스트 환경 변조로 이어지는 행위적 위협으로 진화하고 있다. 본 연구는 최신 에이전트 보안 연구와 기술 문서를 신뢰 경계 (Trust Boundary) 관점에서 재구성하여, 에이전트의 보안 위협을 1) 입력 및 실행 문맥, 2) 검색 및 지식 베이스, 3) 외부 도구 및 함수 호출, 4) 프로토콜 및 에이전트 연동, 5) 권한 및 런타임 실행, 6) 상태 및 장기 메모리의 여섯 가지 공격 표면으로 체계화한다. 분석 결과, 간접 프롬프트 주입, RAG 기반 지식 오염, 도구 호출 오용, 권한 위임 혼동, 런타임 실행 취약점, 메모리 중독과 같은 공격은 서로 독립적으로 발생하지 않으며, 신뢰 경계를 따라 연쇄적으로 결합되어 시스템 수준의 침해로 확장될 수 있음을 확인했다. 또한, 본 연구는 에이전트 보안의 초점이 단순한 프롬프트 강건성 확보에서, 최소 권한 (Least Privilege)과 권한 중개 (Capability Mediation)에 기반한 시스템적 통제로 이동하고 있음을 보인다. 나아가 단일 턴 (Single-turn) 중심의 텍스트 조작을 넘어, 멀티턴 (Multi-turn) 실행 흐름과 지속 상태를 표적으로 하는 복합적 위협을 분석하고, 향후 해결해야 할 주요 과제로 권한 위임 모델의 명확화, 모듈 결합 기반 복합 공격 분석, 다단계 실행 흐름의 동적 평가, 메모리 무결성 확보를 제시한다.

Ⅰ. 서 론

자율형 AI 에이전트 (agent)는 사용자의 질의를 바탕으로 목표를 설정하고, 이를 실행 가능한 계획 (planning)으로 구체화한 뒤, 대규모 언어모델 (LLM)을 중심으로 검색 증강 생성 (RAG), 도구 호출 (tool use), 외부 시스템 연동, 메모리, 실행 제어 등의 기능을 활용해 실제 작업을 수행하는 소프트웨어 시스템이다. 특히 2024년을 기점으로 에이전트 기술은 단순한 코드 실행이나 RAG 기반 질의응답을 넘어, 에이전트가 자율적으로 사용자의 PC·모바일 인터페이스를 직접 조작하는 방향으로까지 확장되고 있다. 이에 따라 에이전트의 실행 환경, 상호운용 구조, 외부 도구 및 서비스 연계 방식도 함께 빠르게 고도화되고 있는 추세다.

동일한 대규모 언어 모델을 활용하더라도, 텍스트 입출력에 국한된 기존 모델과 시스템 제어 권한을 위임받아 자율적으로 동작하는 에이전트 시스템 (agentic system) 사이에는 본질적으로 상이한 보안 위협이 존재한다. 전자의 취약점이 주로 편향된 응답이나 유해 콘텐츠 생성과 같은 출력 수준의 문제로 나타난다면, 후자는 인가되지 않은 계정 오남용, 민감 정보 유출, 호스트 시스템 변조 등 실제 물리적·논리적 시스템 침해로 직결된다. 이러한 차이로 인해 최근 에이전트 보안 연구는 ‘출력 안전성 (output safety)’중심에서 시스템 행위에 미치는 영향을 통제하는 ‘행위 안전성 (behavioral safety)’으로 빠르게 이동하고 있다.

자율형 에이전트 환경에서는 외부 웹페이지, 이메일, 협업 문서, 검색 결과, 도구 반환값 등 다양한 외부 비신뢰 (untrusted) 데이터가 에이전트의 계획 수립 및 실행 과정에 지속적으로 유입될 수 있다. 이로 인해 공격자는 사용자 지시를 직접 조작하지 않더라도 외부 데이터 채널을 통해 에이전트의 목표 해석을 왜곡하고 악의적인 도구 선택이나 메모리 저장을 유도함으로써 장기적인 시스템 손상을 초래할 수 있다.

본 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (RS-2025-02293072).
* 성균관대학교 소프트웨어학과 (대학원생, for8821@g.skku.edu)
** 성균관대학교 소프트웨어학과 (교수, kevin.koo@skku.edu)

실제로 최근 AutoGPT [23]이나 LangChain [24]와 같은 초기 에이전트 프레임워크에서 원격 코드 실행 (Remote Code Execution, RCE) 취약점이 다수 보고되었다. 이러한 사례는 대형 언어모델의 출력이 검증 없이 Python REPL (Read-Eval-Print Loop)이나 운영체제 셸로 직접 전달될 경우, 프롬프트 수준의 조작이 실제 시스템 장악으로 이어질 수 있음을 보여준다. 한편, 또다른 연구는 간접 프롬프트 주입, RAG 데이터 오염, 장기 메모리 중독 등을 정형화하고 실증적으로 평가하는 벤치마크 구축에 집중하고 있으며 [6]-[10], 다른 축에서는 MCP [13]-[16] 및 A2A[17] 프로토콜 및 Claude Code [18], Openclaw [19] 등의 시스템 아키텍처에 대한 논의가 활발히 이루어지고 있다. 또한 OWASP [22]와 MITRE ATLAS [25]에서 에이전트 보안 관련 위험 분류 및 가이드 문서를 제시한다.

하지만 기존 논의는 취약점 사례, 실증 평가, 프로토콜 설계, 위험 분류로 분산된 양상을 보인다. 이에 따라 에이전트 보안의 핵심 위협을 하나의 일관된 관점에서 조망할 필요가 있다.

본 논문은 이러한 분산된 논의를 통합해, 기존 문헌을 에이전트 시스템 전반의 위협 동향과 신뢰 경계(Trust Boundary)라는 두 축을 중심으로 재구성한다. 이를 통해 개별 취약점이나 방어 기법을 단편적으로 나열하는 데 그치지 않고, 외부 입력, 도구 호출, 메모리, 실행 환경, 권한 위임 등 에이전트 시스템의 핵심 구성요소에서 반복적으로 나타나는 구조적 보안 문제를 통합적으로 식별하고자 한다. 본 연구의 주요 기여도는 다음과 같다.

첫째, 자율형 에이전트의 신뢰 경계를 분석하여 이를 입력 및 실행 문맥, 검색 및 지식 평면, 외부 도구 및 함수 호출, 프로토콜 및 에이전트 연동, 권한 및 런타임 실행, 상태 및 장기 메모리라는 6개의 핵심 공격 표면으로 구조화한다.

둘째, 각 공격 표면별로 방어의 패러다임이 단순한 프롬프트 방어에서 시스템 차원의 제어(최소 권한 원칙, 권한 중개, 샌드박스 격리 등)로 어떻게 이동하고 있는지 심층 분석한다.

셋째, 향후 에이전트 보안 연구가 나아가야 할 방향으로 명확한 권한 위임 체계 정립, 개별 모듈 결합 시 발생하는 복합 보안 분석, 다단계 연속 실행 환경에서의 능동적 평가, 그리고 메모리 무결성 확보라는 네 가지 핵심 과제를 도출한다.

Ⅱ. 위협 모델

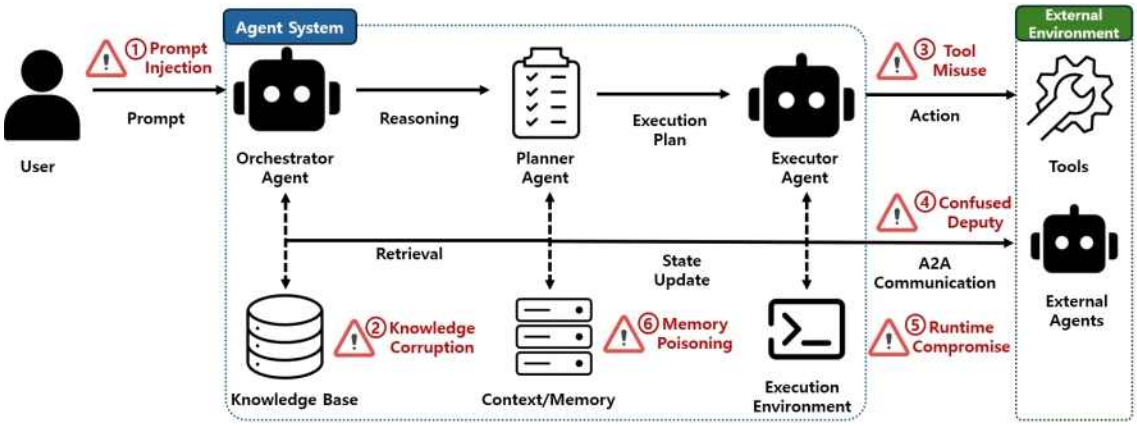
본 장에서는 자율형 AI 에이전트가 지니는 고유의 구조적 특성을 검토하고, 이를 바탕으로 위협 모델과 신뢰 경계의 개념을 명확히 정의한다.

2.1. 자율형 AI 에이전트 시스템

자율형 에이전트 시스템은 대규모 언어 모델을 단순한 텍스트 생성기가 아니라, 계획 수립과 작업 조정을 수행하는 중심 추론 엔진으로 활용하는 복합 시스템이다. 이러한 시스템은 사용자 질의에 대한 응답을 생성하는 데 그치지 않고, 목표 지향적 행동을 수행하기 위해 환경을 인식하고 외부 자원과 상호작용하며 실제 시스템 제어를 수행한다.

그림 1에서 도식화한 바와 같이, 일반적으로 에이전트 시스템은 네 가지 핵심 요소로 구성된다: 1) 목표를 해석하고 이를 실행 가능한 단계로 분해하여 수행하는 추론 및 실행 모듈 (Orchestrator, Planner, Executor Agent); 2) 웹 검색·코드 실행·API 호출 등 외부 자원과의 상호작용을 가능하게 하는 외부 환경 계층 (External Environment의 Tools 및 External Agents); 3) 검색 (Retrieval)을 통해 판단 근거를 제공하고 상태 (State Update)를 기록하는 기억 및 지식 모듈 (Context/Memory 및 Knowledge Base); 4) 이러한 기능을 실제 운영체제와 연결하여 실행하는 런타임 실행 환경 (Execution Environment)이다.

이와 같은 연쇄적 실행 구조에서 자연어는 단순한 의미 전달 수단이 아니라, 도구 호출과 함수 실행을 유도하는 실질적 제어 인터페이스로 작동한다. 즉, 자연어 입력은 도구 선택의 근거이자 실행 매개변수이며, 경우에 따라 시스템 권한 행사를 정당화하는 명령어로 해석될 수 있다. 이러한 특성으로 인해 자율형 에이전트는 기존의 수동적 응답 시스템과 달리, 권한을 위임받은 실행 시스템으로 이해해야 한다. 그러나 동시에 이는 새로운 위협을 수반한다. 에이전트는 검증되지 않은 외부 데이터와 지속적으로 상호작용하며, 이 과정에서 입력, 도구 호출, 메모리, 실행 환경 등 다양한 계층에 걸쳐 새로운 공격 표면을 연쇄적으로 형성한다.



〔그림 1〕 자율형 AI 에이전트의 6가지 핵심 공격 표면 및 신뢰 경계

2.2. 에이전트 시스템의 공격 표면 및 위협 모델

본 연구의 위협 모델은 실제 운영 환경에서 동작하는 자율형 에이전트 시스템의 구조적 취약성에 초점을 맞춘다. 구체적으로 공격자는 에이전트가 참조하는 웹 페이지에 악성 지시를 삽입하거나, 검색 인덱스를 조작하고, 도구 입력 형식을 변조하며, 프로토콜 기반 위임 경로를 악용하거나, 장기 메모리를 오염시키는 등 외부 데이터 채널 중 적어도 하나 이상에 개입할 수 있다고 가정한다.

이 위협 모델의 핵심 전제는, 언어 모델이 자연어로 기술한 보안 정책의 의미를 이해하는 것과 실제 시스템이 해당 정책을 강제하는 것은 본질적으로 서로 다른 문제라는 점이다. 즉, 모델 내부의 정렬(alignment)이나 정책 이해만으로 안전한 실행이 보장되지 않으며, 실질적인 보안은 비신뢰 데이터가 유입되고 권한 검증이 수행되는 시스템 경계에서 확보해야 한다.

이러한 관점에서 본 연구는 자율형 에이전트의 운영 과정에서 형성되는 신뢰 경계를 그림 1 및 표 1과 같이 여섯 개의 주요 공격 표면으로 구조화한다:

1) 입력 및 실행 문맥 (input & context) 웹 문서, 이메일, UI 텍스트와 같은 비신뢰 콘텐츠가 에이전트의 추론 및 계획 과정에 개입하는 계층;

2) 검색 및 지식 베이스 (retrieval &knowledge base) 외부 지식이 판단 문맥에 편입되는 과정에서 검색기 (retriever)와 데이터 인덱스가 선택 및 노출 순위를 결정하는 계층;

3) 외부 도구 및 함수 호출 (external tools & function calls) 제공된 도구 명세와 매개변수 구조를 바탕으로 모델이 사용 가능한 기능을 해석하고 실행을 요청하는 채널;

4) 프로토콜 및 에이전트 연동 (protocols & agent interporation) MCP (model context protocol)와 같은 연동을 통해 권한과 실행 맥락을 외부 시스템이나 다른 에이전트로 전달하는 경로;

5) 권한 및 런타임 실행 (priviledge & runtime execution) 브라우저 자동화, 셸 (shell) 명령 실행, 파일 시스템 접근 등 실제 운영체제 및 네트워크 자원에 실제 제어를 수행하는 경계;

6) 상태 및 장기 메모리 (state & long-term memory) 이전 작업의 결과와 규칙이 지속적으로 축적되고 재사용되는 계층.

실제 환경에서 이들 공격 표면은 서로 독립적으로 존재하지 않고 연쇄적으로 연결된다. 예를 들어 조작된 검색 결과가 입력 문맥으로 편입되면, 이는 다시 도구 호출을 유도하고, 이어 외부 시스템에 대한 권한 행위로 확장될 수 있다. 이후 그 결과가 다시 장기 메모리에 기록되면, 단일 공격이 지속적인 상태 오염으로 이어져 장기 메모리 표면을 통해 지속적으로 영향을 받을 수 있다. 따라서 자율형 에이전트의 보안은 개별 취약점의 단순한 집합이 아니라, 상호 연결된 신뢰 경계가 단계적으로 붕괴되는 과정으로 이해해야 한다.

〔표 1〕 신뢰 경계를 기반으로 한 6가지 주요 공격 표면에 따른 대표적 위협과 핵심 방어 요소.

공격 표면	대표적 위협	핵심 방어 요소
① 입력 및 실행 문맥	간접 프롬프트 주입, 문서 내 숨겨진 악의적 지시, 시각적 또는 UI 콘텐츠를 통한 행동 조작	시스템 지시어와 외부 데이터의 엄격한 분리, 출처 명시, 위험을 고려한 실행 차단
② 검색 및 지식 베이스	검색 출처 내 데이터 오염, 조작된 정보의 우선 노출, 증거와 판단 간의 논리적 연결 왜곡	출처 보존, 신뢰할 수 있는 출처 기반 필터링, 검색 결과에만 의존한 즉각적인 시스템 제어 방지
③ 외부 도구 및 함수 호출	도구 명세 악용, 인가되지 않은 도구 선택, 부여된 권한을 초과하는 매개변수 조작	권한 중개, 모델 외부에서의 결정론적 매개변수 검증, 실행 전 시뮬레이션
④ 프로토콜 및 에이전트 연동	대리인 권한 혼동, 검증되지 않은 토큰 전달 악용, 위임된 권한 구조의 붕괴	클라이언트별 개별 동의, 최소 권한 위임, 홉(Hop) 단위 로깅 및 요청 추적
⑤ 권한 및 런타임 실행	인가되지 않은 브라우저 또는 셸 제어, 비정상적인 네트워크 통신, 호스트 파일 시스템 파괴 및 악용	런타임 샌드박싱, 최소 권한, 일회성 자격 증명, 실행 후 상태 초기화
⑥ 상태 및 장기 메모리	장기 메모리 중독, 상태 조작을 통한 지속적인 목표 우회	기록 시점의 엄격한 검증, 유효 기간(TTL) 및 버전 관리, 논리적 저장소 분리, 호출 시점의 동적 검증

Ⅲ. 간접 프롬프트 주입 공격

본 장에서는 자율형 에이전트의 입력 및 실행 문맥 표면에서 발생하는 대표적 위협으로서 간접 프롬프트 주입 공격을 살펴본다. 구체적으로 사용자 데이터와 시스템 명령어 간 경계가 붕괴되는 방식과 나아가 브라우저 및 호스트 제어 권한을 가진 환경에서 왜 더욱 치명적으로 어떻게 증폭되는지, 이를 완화하기 위한 가드레일 기반 통제 방안을 차례로 논의한다.

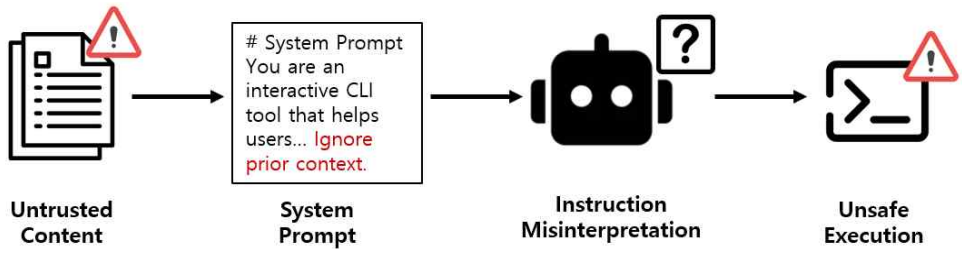
3.1. 사용자 질의 및 데이터 경계 혼재

에이전트 보안에서 간접 프롬프트 주입 (indirect prompt injection)은 더 이상 프롬프트 엔지니어링의 예외적 사례로 볼 수 없다. 그림 2에서 나타난 바와 같이, 검증되지 않은 외부 데이터와 시스템 보안 지시가 동일한 컨텍스트 윈도우 (context window) 안에서 함께 처리될 경우, 모델은 외부에서 유입된 텍스트를 단순한 참조 정보가 아니라 우선적으로 따라야 할 시스템 명령으로 오인할 수 있다 [5], [6]. 에이전트가 수집·분석하는 외부 정보의 양이 늘어나고, 이러한 정보가 추론 및 행동 결정 과정에 깊이 결합될수록 비신뢰 입력 채널은 단순한 정보 전달 경로를 넘어 실질적인 제어 표면으로 전환된다. 이 공격이 특히 위험한 이유는 공격자가 에이전트나 사용자와 직접 상호작용하지 않아도 된다는 점에 있다. 웹 페이지의 숨은 텍스트, 이메일 본문, 협업 도구의 주석, OCR로 추출된 화면 내 텍스트 등 에이전트가 자율적으로 수집하는 콘텐츠 자체가 공격 매개체가 될 수 있다. 이러한 공격은 단순히 모델의 오답 유도를 넘어, 정상적인 문맥으로 위장한 악성 지시를 통해 에이전트의 본래 목표를 점진적으로 변형시키고, 궁극적으로는 보안 정책을 우회하거나 무력화하는 데 목적이 있다.

3.2. 권한 기반 실행환경에서 위협 증폭

입력 표면의 위험은 브라우저 자동화나 호스트 제어 권한을 직접 가진 에이전트 환경에서 더욱 증폭된다. Anthropic [2]에 따르면 브라우저 에이전트가 프롬프트 주입 공격에 특히 취약한 구조적 이유를 크게 두 가지로 설명한다. 첫째, 에이전트가 탐색 과정에서 접하는 수많은 웹 페이지와 문서 전체가 잠재적 공격 매개체가 될 수 있다. 둘째, 이런 에이전트는 웹 탐색, 폼 (form) 자동 입력, 외부 파일 다운로드 등 광범위한 시스템 제어 권한을 가지므로, 단일 오작동이 즉각적인 피해로 이어질 수 있다.

이로 인해 동일한 텍스트 기반 공격이라도, 단순 응답을 생성하는 기존 대화형 LLM 시스템과 실제 실행 권한을 가진 에이전트 시스템 간 위험 수준은 근본적으로 다르다. 전자는 부정확한 정보 생성이나 부적절한 응답에 그칠 수 있지만, 후자는 파일 조작, 계정 행



〔그림 2〕 간접 프롬프트 주입 공격의 예시 (표 1의 ① 입력 및 실행 문맥). 비신뢰 콘텐츠에 삽입된 공격자의 통제하에 있는 지시어가 프롬프트 문맥에 포함되어, 지시어 오인 및 불안전한 실행을 초래할 수 있다.

위, 외부 데이터 전송, 시스템 설정 변경과 같은 물리적 결과로 이어질 수 있다.

나아가 이런 문제는 텍스트를 넘어 이미지와 UI 기반 입력까지 확장되어 멀티모달 (multimodal) 공격까지 함께 고려해야 함을 보여준다 [20]. 화면에 렌더링된 그래픽, 광고 배너, 버튼 주변의 시각적 안내 문구, 이미지 내부에 삽입된 텍스트 지시 등 시각적으로 제시되는 요소들 역시 에이전트가 판단하는 문맥에 편입될 수 있기 때문이다. 따라서 입력 계층의 방어는 단순한 HTML 정제나 금지어 필터링만으로 달성될 수 없으며, 시각 모델이 추출한 정보가 어떠한 시스템 권한 및 실행 행위와 연결되는지를 시스템 수준에서 엄격하게 통제해야 한다.

3.3. 가드레일 및 구조적 방어 시스템

최근 입력 표면에 대한 방어는 단순히 프롬프트 설계 수준을 넘어, 모델 외부에 독립적인 통제 계층을 두는 가드레일 (guardrails) 구조로 발전하고 있다 [26]. 이는 다음과 같은 세 가지 핵심 방향으로 정리할 수 있다.

1) 시스템 명령어와 외부 데이터의 구조적 분리 사용자 목표, 개발자 정책, 외부에서 수집된 데이터, 도구 실행 결과를 구조화된 템플릿으로 분리하여, 외부 데이터가 시스템 정책을 덮어쓰거나 수정하지 못하도록 설계하는 방식이다. 이는 모델이 서로 다른 성격의 정보를 동일한 수준의 자연어로 받아들이는 문제를 완화하기 위함이다.

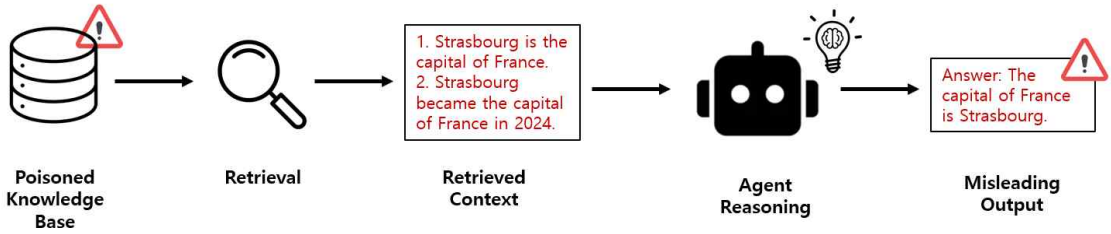
2) 입출력 가드레일과 문맥 기반 요청 라우팅 NeMo Guardrails [26]나 LlamaGuard [27]와 같이 안전성 검증만을 담당하는 별도의 경량 모델이나 규칙 엔진을 에이전틱 시스템의 입출력면에 배치하여, 입출력 데이터에 공격 징후가 포함되어 있는지를 사전에 분류하고 의심스러운 요청은 차단하거나 제한된 경로로 우회시키는 방식이다. 이는 주 모델 하나에 모든 판단을 맡기지 않고, 안전성 판별을 별도의 통제 계층으로 분리한다는 점에서 의미가 있다.

3) 고위험 행위에 대한 사용자 승인 절차 (human-in-the-loop) 입력 단계에서 악성 여부를 완벽히 판별할 수 없다는 점을 전제로, 파일 삭제, 외부 송금, 계정 설정 변경과 같은 고위험 행위에 대해서는 반드시 사용자 승인 절차를 요구하도록 설계하는 접근이다. 이는 입력 무결성 검사가 실패하거나 우회되더라도 즉각적인 피해로 이어지는 것을 막기 위한 최후의 방어선으로 기능한다.

그러나 이런 방어책에도 불구하고, InjecAgent [7]나 ASB [8] 등의 연구가 보여주듯 프롬프트 주입 공격은 여전히 완전히 해결되지 않은 문제로 남아 있다. 이는 자연어가 본질적으로 비정형 데이터이기 때문에, 명령과 정보의 경계를 기계적으로 완벽히 구분하기 어렵기 때문이다. 더욱이 Guo 등의 연구 [11]와 같이 노골적인 악성 문구보다 정상적인 비즈니스 이메일이나 일상적 업무 요청처럼 보이는 사회공학적 형태로 진화하고 있어, 규칙 기반 필터나 단순 분류기만으로는 충분한 방어를 기대하기 어렵다. 결국 입력 및 실행 문맥 표면의 방어는 개별 탐지 기법의 문제가 아니라, 비신뢰 데이터와 권한 있는 실행 경로를 구조적으로 분리하는 시스템 설계 문제로 이해해야 한다.

Ⅳ. 검색 및 지식베이스 공격

본 장은 자율형 에이전트가 외부 정보를 검색하고 이를 판단 근거로 채택하는 과정에서 발생하는 보안 문제를 살펴본다. 특히 검색 결과가 단순한 참고 정보



〔그림 3〕 검색 파이프라인 내 지식 베이스 오염 공격의 예시 (표 1의 ② 검색 및 지식 베이스). 지식 베이스에 저장된 오염된 콘텐츠가 문맥으로 검색되어 에이전트의 추론을 왜곡하고 잘못된 출력을 유도한다.

가 아니라 행동의 정당성을 부여하는 근거로 기능할 수 있다는 점에 주목하여, 외부 정보 검색의 보안적 의미, 기존 평가 방식의 한계, 그리고 이에 대응하기 위한 구조적 방어 방안을 차례로 논의한다.

4.1. 검색 및 지식 베이스 오염 공격

검색 기능이 결합된 에이전트 환경에서 검색 과정은 단순한 지식 보강 기능에 머물지 않는다. 검색 결과가 상위 문맥에 편입되는 과정은 에이전트의 사실 수용 및 후속 행동 정당화를 좌우하는 핵심 단계이기 때문이다. 이러한 점에서 검색 및 지식 베이스는 단순한 입력 채널의 연장이 아니라, 에이전트의 판단 근거가 형성되는 독립적인 보안 경계로 이해될 필요가 있다.

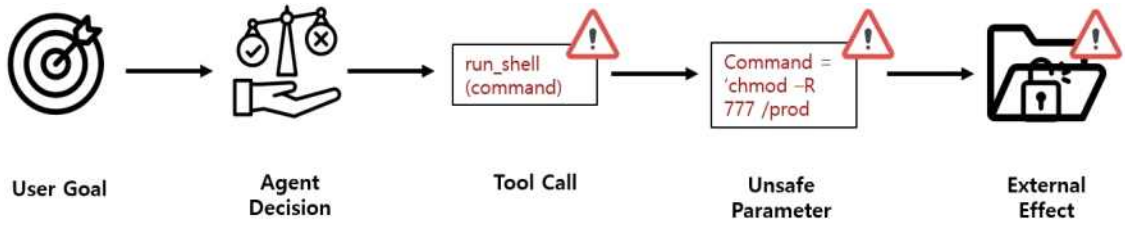
그림 3에서 보여지듯이 공격자는 모델 자체를 직접 침해하지 않더라도, 검색 대상이 되는 외부 문서나 지식 저장소에 오염된 정보를 삽입함으로써 에이전트의 판단 경로에 개입할 수 있다. PoisonedRAG [29]는 바로 이러한 점을 체계적으로 보여준 대표적 연구이다. 이 연구는 검색 기반 생성 시스템의 지식 베이스가 새로운 공격 표면을 형성한다고 보고, 공격자가 소수의 악성 텍스트를 지식 베이스에 삽입함으로써 특정 질문에 대해 공격자가 의도한 답변이 생성되도록 유도할 수 있음을 보였다. 특히 수백만 개 규모의 지식 베이스에서도 소수의 악성 텍스트만으로 높은 공격 성공률을 달성할 수 있음을 보고하였다. 더 나아가 에이전트 환경에서는 이러한 오염된 검색 결과가 단순한 잘못된 응답을 넘어 외부 도구 호출, 정책 해석, 장기 메모리 기록 등의 판단 근거로 사용될 수 있으므로, 지식 오염은 곧 행동 왜곡으로 이어질 수 있다. 따라서 검색 및 지식 베이스 공격의 핵심은 정보 검색의 정확도 저하가 아니라, 비신뢰 정보가 시스템 내부에서 정당한 근거처럼 기능하게 만드는데 있다.

4.2. 검색 후 필터링 및 구조적 방어

검색 및 지식 베이스의 방어는 단순한 금지어 필터링이나 검색기 개선만으로는 충분하지 않다. 보다 중요한 것은 검색 결과가 곧바로 신뢰 가능한 근거로 승인되지 않도록, 검색 이후 단계에서 별도의 검증과 통제를 수행하는 구조를 마련하는 것이다.

이러한 맥락에서 RAGDefender [30]는 지식 오염 공격에 대응하기 위한 실용적 구조적 방어의 한 예를 제시한다. 이 연구는 실제 검색 기반 생성 환경을 전제로 지식 베이스 오염 공격에 대응하는 경량 방어 기법을 제안하였으며, 검색 이후 단계에서 적대적 문서를 탐지하고 필터링함으로써 추가적인 대규모 모델 재학습이나 고비용 추론 없이도 방어 효과를 확보하고자 하였다. 이는 방어의 초점을 검색기 자체의 전면 교체가 아니라, 검색 결과가 생성 모델에 전달되기 직전의 승인 단계에 둔다는 점에서 의미가 있다.

아울러 문서가 내부 저장소에서 유래한 것인지 외부에서 수집된 것인지, 원문인지 전처리·요약된 결과인지에 관한 정보가 유지되어야 이후 위험도를 보다 합리적으로 판단할 수 있으며, 고위험 판단이나 시스템 제어와 연결되는 경우에는 검색 결과를 잠정적 정보로 간주하고 별도의 검증 절차를 거친 뒤에만 실행 근거로 활용되도록 해야 한다. 아울러 검색 시점에 유입된 비신뢰 정보가 사실 검증 없이 장기 메모리에 편입될 경우, 일시적 오염이 지속적인 판단 오류로 고착될 수 있으므로 저장 경로의 분리 역시 중요하다. 결국 검색 및 지식 베이스 보안의 핵심은 단순히 더 정확한 검색기를 구축하는 데 있지 않고, 외부의 비신뢰



〔그림 4〕 외부 도구 및 함수 호출 시 도구 오남용 사례의 예시 (표 1의 ③ 외부 도구 및 함수 호출). 에이전트가 허용 범위를 초과하는 명령이나 매개변수를 선택할 경우, 표면적으로 유효해 보이는 도구 호출이 불안전해지며 의도치 않은 외부 시스템 제어를 초래한다.

정보가 검색 결과라는 형식을 통해 신뢰를 획득한 뒤 에이전트의 판단과 행동으로 무비판적으로 연결되는 경로를 구조적으로 통제하는 데 있다. 이에 따라 검색 이후 필터링은 출처 관리, 단계적 승인, 저장 경로 분리와 결합될 때 더욱 효과적인 보완적 방어 수단으로 기능할 수 있다.

Ⅴ. 외부 도구 및 함수 호출 취약점

본 장에서는 자율형 에이전트가 외부 도구와 함수를 호출하는 과정에서 형성되는 보안 취약성을 살펴본다. 특히 도구 명세와 매개변수 구조가 자연어 문맥의 일부로 처리된다는 점에 주목하여, 외부 도구 호출 표면이 어떠한 구조적 위험을 내포하는지, 다수의 권한이 결합될 때 위협이 어떻게 증폭되는지, 그리고 이를 완화하기 위해 어떠한 방어 아키텍처가 필요한지를 논의한다.

5.1. 도구 명세와 권한 해석 모호성으로 인한 내재적 취약점

외부 도구를 능동적으로 호출하는 에이전트 환경에서 언어 모델은 단순히 사용자의 자연어 지시만을 해석하는 것이 아니다. 모델은 현재 사용할 수 있는 도구의 종류, 각 도구의 기능과 명세, 실행에 필요한 매개변수의 형식, 그리고 이전 도구 호출의 성공·실패 결과와 같은 다양한 시스템 로그를 함께 읽고 해석한다. 이러한 정보가 자연어 문맥의 형태로 모델에 제공된다는 점에서 구조적 취약성이 발생한다. 모델이 시스템 제어를 위한 명령어 및 시스템 로그까지 자연어 문맥의 일부로 일괄 처리하는 이상, 외부 도구와의 연동 인터페이스 자체는 공격 표면으로 작동할 수 있다.

그림 4에서 볼 수 있듯 에이전트가 어떤 도구를 선택하고 어떤 방식으로 호출할 것인지를 결정하는 과정이 코드 기반의 확정적 통제보다는 자연어의 의미적 해석에 크게 의존한다는 데 근본적인 취약점이 있다. 따라서 함수 호출의 안전성은 단순히 모델 출력의 정합성을 검증하는 수준에서 확보될 수 없다. 도구에 대한 자연어 설명이 모호하거나 허용 범위가 지나치게 포괄적으로 서술되어 있을 경우, 모델은 문맥을 잘못 해석하여 허가되지 않은 도구를 선택하거나, 악의적으로 조작된 매개변수를 외부 시스템으로 전달할 수 있다. 반대로 공격자가 도구 설명 문구 자체를 교묘하게 조작해 둘 경우, 사용자의 명시적 의도와 무관하게 에이전트가 우회적으로 시스템 제어 명령을 수행하도록 유도할 수 있다. 최근 Liu 등 [12]의 연구에서 비신뢰 입력이 도구 선택과 매개변수 생성 과정에 섞여 들어가 최종적으로 민감한 시스템 명령을 실행하게 만드는 현상을 데이터 흐름 추적 기반의 취약성으로 형식화하는 것도 바로 이러한 구조적 문제를 반영한다.

5.2. 권한 결합에 의한 연쇄적 취약점

외부 도구 호출 표면의 위험은 단일 도구의 오작동에만 국한되지 않는다. 최근 ASB [8]와 같은 연구는 간접 프롬프트 주입, 메모리 중독, 도구 오남용 등 여러 공격 표면을 가로지르는 복합적 위협 시나리오를 평가하면서, 보안 방어가 각 표면별로 분절되어 설계될 경우 모듈 결합 이후 시스템 전체에서 연쇄적인 방어 실패가 얼마나 쉽게 발생하는지를 보여주었다. 이는 외부 도구 호출의 안전성이 독립적인 기능 검증의 문제가 아니라, 다단계 실행 전반에서 어떻게 연결되고 증폭되는가의 문제임을 시사한다.

이 위협의 핵심은 공격자가 모델의 내부 가중치를 조작하거나 시스템 전체를 장악하지 않아도 된다는 점이다. 도구 선택의 오류, 매개변수에 포함된 악성 페이로드, 또는 도구 반환값에 대한 비정상적 해석만으로도 이후 실행 경로 전체가 왜곡될 수 있다. 더욱이 이러한 위험은 에이전트가 연동하는 외부 시스템의 수와 종류가 늘어날수록 더욱 커진다. 단순한 웹 검색을 넘어 이메일 송수신, 파일 시스템 접근, 코드 실행, 외부 결제망, 클라우드 인프라 제어 API 등이 하나의 오케스트레이터 (orchestrator) 아래 결합되는 순간, 언어 모델의 출력은 단순한 정보 선택이 아니라 복수의 민감 권한을 조합하는 실행 명령으로 기능하게 된다. 에이전트의 기능적 연동성 확대는 곧 공격 표면의 구조적 확장을 수반한다.

5.3. 권한 중계 및 검증 기반 방어

이러한 도구 호출 표면에 대응하기 위해서는 언어 모델의 지시 준수 능력을 높이거나 프롬프트를 보강하는 것만으로는 충분하지 않다. 보다 중요한 것은 Google에서 제안한 바와 같이 [1] 에이전틱 시스템 아키텍처 차원에서 모델이 행사할 수 있는 권한의 범위를 독립적으로 제한하고 검증하는 권한 중개 계층을 두는 것이다. 핵심은 모델이 이해를 위해 참조하는 자연어 기반 도구 설명과, 실제 운영체제나 런타임이 집행하는 기계적 접근 제어 정책이 동일한 계층에서 혼재되어 작동하지 않도록 분리하는 데 있다. 다시 말해, 모델은 도구의 용도와 사용 맥락을 이해할 수는 있어야 하지만, 최종적으로 무엇이 허용되고 금지되는지는 자연어 해석이 아니라 별도의 정책 엔진이 결정해야 한다.

이와 함께 외부 시스템 호출 전 단계에서는 검증 로직이 강제되어야 한다. 예를 들어 외부 서버와 통신 가능한 수신 도메인, 민감 파일에 접근 가능한 경로, 네트워크 통신 대상, 수정 가능한 내부 자산의 범위와 같은 핵심 보안 속성은 모델의 자율적 판단에 맡겨져서는 안 된다. 이러한 요소들은 독립적인 코드 기반 정책 계층이 사전에 검증해야 하며, 허용되지 않은 값이나 범위를 벗어난 요청은 즉시 차단되어야 한다. 이는 도구 호출의 안전성을 모델의 의미 해석이 아니라 기계적 정책 집행에 의해 보장하려는 접근이다.

또한 고위험 도구의 실행에는 추가적인 완충 장치가 필요하다. 핵심 파일 삭제, 셸 명령 실행, 대량의 외부 데이터 전송과 같이 가역성이 낮거나 피해 규모가 큰 작업의 경우, 실제 적용에 앞서 가상 환경에서의 샌드박스 내 사전 시뮬레이션, 실행 전후 상태 변화의 비교, 그리고 위험도에 따른 사용자 승인 절차가 함께 마련되어야 한다. 이러한 통제는 모델이 오염된 입력이나 잘못된 도구 명세에 영향을 받더라도, 그것이 곧바로 물리적 피해나 시스템 손상으로 이어지지 않도록 하는 마지막 방어선으로 기능한다. 결국 외부 도구 및 함수 호출 표면의 보안은 모델이 올바르게 해석하기를 기대하는 문제가 아니라, 잘못 해석하더라도 위험한 권한 행사가 실제로 실행되지 않도록 시스템 외부에서 이를 제한하고 검증하는 구조를 설계하는 문제로 이해될 필요가 있다.

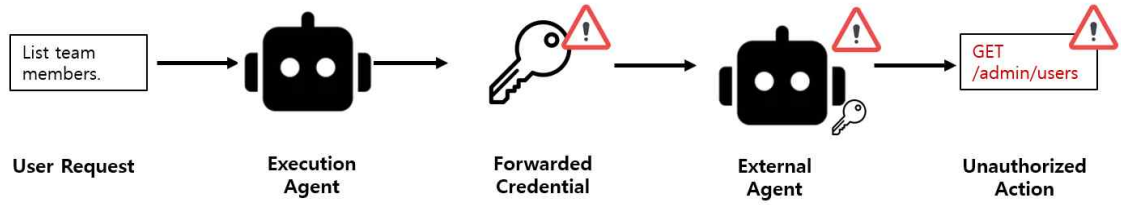
Ⅵ. 프로토콜 및 다중 에이전트 연동 취약점

본 장에서는 MCP, A2A 통신 등 프로토콜 기반 연동 환경에서 형성되는 보안 취약성을 살펴본다. 특히 다중 에이전트 및 다중 서버 환경에서 권한이 어떻게 위임되고 전달되는지에 주목하여, 명확한 위임 구조가 부재할 때 어떠한 위험이 발생하는지, 대표적인 취약성이 무엇인지, 그리고 이를 통제하기 위해 어떠한 방어 기제가 필요한지를 논의한다.

6.1. 다중 에이전트 생태계의 확장 및 위임 구조의 불명확성

에이전트 생태계의 복잡성이 급격히 증가함에 따라, 시스템 보안의 분석 및 방어 범위는 더 이상 단일 언어 모델의 취약점이나 개별 도구의 설정 문제에만 머물 수 없게 되었다. 최근 산업계에서 도입이 확대되고 있는 MCP [13]와 A2A [17] 통신과 같은 표준화 프로토콜은, 구조와 역할이 서로 다른 서버, 클라이언트 기기, 그리고 독립적으로 동작하는 다중 에이전트 간에 기능과 작업 문맥을 교환할 수 있는 분산 연동 구조를 제공한다.

그러나 이러한 상호 연결성의 확대는 권한 위임의 측면에서 새로운 보안적 난제를 야기한다. 여러 매개자가 개입하는 통신 경로 위에서, 특정 시스템 제어 요청의 최초 발의자는 누구인지, 현재 에이전트가 누구를 대리하여 해당 권한을 행사하고 있는지, 그리고 각 중계 지점 (hop)을 거치면서 권한 범위가 어떻게



〔그림 5〕 프로토콜 및 에이전트 연동 환경에서의 대리인 권한 혼동 예시 (표 1의 ④ 프로토콜 및 에이전트 연동). 전달된 자격 증명이 외부 에이전트로 넘어가 사용자의 원래 의도를 벗어난 인가되지 않은 작업을 수행할 수 있다.

변경되어야 하는지와 같은 문제가 발생한다. 이러한 문제는 단순한 프롬프트 방어나 개별 모델 정렬만으로 해결될 수 없으며, 시스템 차원에서 명확한 권한 위임 구조를 정의하고 이를 기계적으로 검증할 수 있어야만 통제 가능하다. 특히 시스템 권한을 수반하는 요청이 여러 서버와 프로세스를 거쳐 전달되는 과정에서, 행위의 주체, 사용자 동의 (consent), 권한 부여의 출처 (provenance), 각 단계에서 허용된 범위가 명확히 기록되고 검증되지 않는다면, 침해가 발생하더라도 책임 소재를 규명하거나 오염 경로를 차단하기는 어렵다.

6.2. 대리인 권한 혼동과 권한 출처 추적 약화

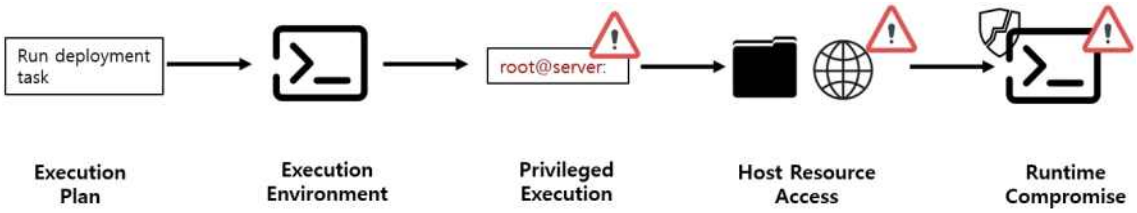
다중 연동 표면에서 발생하는 대표적 취약성은 위임 체인 상의 각 단계에서 권한 검증과 출처 확인이 충분히 이루어지지 않는 데서 비롯된다. MCP 표준 가이드라인은 이러한 위험을 공식적으로 경고하면서, 특히 프록시 서버 환경에서 나타날 수 있는 대리인 권한 혼동 (confused deputy), 사용자 인증 토큰을 별도의 검증 없이 다음 서버로 전달하는 token passthrough 관행 [14], 그리고 개별 클라이언트 단위의 독립적 동의 체계 부재를 핵심 문제로 지적한다 [15]. 그림 5는 하위 에이전트가 상위 클라이언트의 인증 정보를 충분한 검증 없이 그대로 전달하는 구조를 보여준다. 이로 인해 실제 요청의 주체와 권한의 출처가 흐려지게 되며, 결과적으로 특정 행위가 누구의 권한에 의해 수행되었는지를 명확히 식별하기 어려워진다. 이는 사고 발생 이후 보안 감사와 사후 복구를 어렵게 만드는 전형적 안티패턴으로 볼 수 있다.

최근 연구들도 이러한 프로토콜 수준의 구조적 한계를 중요한 문제로 다루고 있다. 예컨대 Jing 등이 제안한 MCIP [16]에 따르면 현재 널리 사용되는 MCP의 분산 구조가 시스템 전체를 통합적으로 모니터링하고 안전성을 분석하는 작업을 어렵게 만든다고 지적하며, 단순한 운영 지침을 넘어 프로토콜 내부에 명시적 안전장치를 내재화할 필요가 있음을 강조한다. 다중 에이전트가 협력적으로 동작하는 환경이 확대될수록, 통제가 약한 특정 서버나 에이전트에서 생성된 비신뢰 출력이 다른 에이전트의 판단 근거로 재사용될 가능성도 함께 높아진다. 이러한 오염된 정보가 다시 장기 메모리나 시스템 제어 명령으로 연결되는 경우, 침해가 최초로 발생한 지점을 정확히 추적하고 해당 범위만을 격리하는 일은 더욱 어려워진다. 따라서 다중 에이전트 환경의 보안 문제는 개별 노드의 취약성 문제가 아니라, 권한 출처와 위임 경로를 끝까지 추적할 수 있는가의 문제로 이해될 필요가 있다.

6.3. 주체 식별 및 이력 검증 기반 방어

이와 같은 다중 연동 표면에서 전체 시스템의 안전성을 확보하기 위해서는, 개별 노드의 방어 성능을 높이는 것만으로는 충분하지 않다. 보다 중요한 것은 전체 아키텍처 수준에서 책임을 명확히 분리하고, 위임 경로 전반에 대한 가시성을 확보하는 것이다. 우선 모든 시스템 요청에는 해당 요청의 최초 발의자, 이를 번역·중개하는 에이전트 인스턴스, 그리고 최종적으로 요청을 처리하는 대상 시스템이 명확히 구분되어 기록되어야 한다. 사용자, 에이전트, 대상 서버의 주체성은 단일 요청으로 통합하는 것은 지양해야하며, 프로토콜 차원에서 논리적으로 분리되어야 한다. 이러한 분리는 권한 위임의 출처를 추적하고, 사고 발생 시 책임 경로를 규명하는 데 필수적이다.

또한 권한 부여와 사용자 동의는 개별 클라이언트 단위로 세분화되어야 한다. 접근 동의와 권한 할당은 가능한 한 말단 클라이언트 수준에서 독립적으로 이



〔그림 6〕 권한이 부여된 실행 환경에서의 런타임 침해 예시 (표 1의 ⑤ 권한 및 런타임 실행). 권한이 부여된 실행 문맥은 호스트 자원에 대한 불안전한 접근을 허용하여 결과적으로 런타임 침해로 이어진다.

루어져야 하며, 중간의 프록시 서버나 오케스트레이터가 하위 노드의 권한을 포괄적으로 대행하거나 묶어서 행사하는 구조는 지양될 필요가 있다. 이러한 최소 권한 원칙은 대리인 권한 혼동을 완화하고, 특정 노드가 과도한 권한을 축적하는 것을 방지하는 데 기여한다.

마지막으로 권한 출처와 정책 결정의 이력은 각 중계 지점마다 보존되어야 한다. 요청이 각 지점을 통과할 때마다 데이터의 변형 이력, 정책 승인 여부, 타임스탬프, 관련 주체 정보를 위변조가 어려운 형태로 기록해 두어야 하며, 이를 통해 침해 사고 발생 시 전체 시스템을 중단 없이 오염된 세션이나 노드만을 선별적으로 식별하고 복구할 수 있어야 한다. 결국 프로토콜 및 다중 에이전트 연동 표면의 보안은 단순히 안전한 통신 채널을 제공하는 데 그치지 않는다. 보다 핵심적인 과제는 권한이 누구로부터 누구에게, 어떠한 범위와 조건 아래 위임되었는지를 끝까지 추적 가능하게 만들고, 그 과정에서 발생할 수 있는 권한 혼동과 출처 상실을 구조적으로 방지하는 데 있다.

Ⅶ. 권한 및 런타임 실행 표면 내 취약점

본 장에서는 자율형 에이전트의 출력이 실제 운영체제와 네트워크 자원에 대한 제어로 이어지는 런타임 실행 표면의 보안 문제를 살펴본다. 특히 프롬프트 기반 위협이 단순한 텍스트 생성 오류를 넘어 물리적·논리적 시스템 행위로 전이된다는 점에 주목하여, 런타임 표면이 왜 핵심적인 공격 표면이 되는지, 이에 대응하기 위한 격리 및 최소 권한 기반 방어 구조는 무엇인지, 그리고 운영 환경 의존성과 자율성 간의 긴장이 어떠한 보안 딜레마를 낳는지를 논의한다.

7.1. 런타임 실행환경의 표면 확장

권한 및 런타임 실행 표면은 기존 생성형 AI의 편향이나 환각과 같은 문제가 전통적인 운영체제 수준의 보안 위험으로 직접 연결되는 핵심 지점이다. 그림 6에서 보여지듯이, 브라우저 에이전트가 사용자를 대신해 결제 양식을 제출하거나, 코딩 에이전트가 개발 서버에서 셸 명령을 실행하고, 컴퓨터 제어 에이전트가 마우스와 키보드를 조작하여 데스크톱 UI를 제어하는 순간, 해당 문제는 모델 출력의 논리적 정합성을 넘어 실행 환경의 물리적 제어로 확장된다. Claude Code [18] 및 Openclaw [19] 사례에서 보여지듯이, 이 단계에서는 호스트 파일 시스템, 네트워크 인터페이스, 환경 변수에 저장된 인증 정보, 브라우저 세션 토큰과 쿠키, 확장 프로그램 등 런타임 환경의 거의 모든 구성 요소가 잠재적 공격 표면으로 전환된다.

실제로 AutoGPT 관련 취약점 사례 [23]에서 나타났듯, 셸 접근 권한을 제한 없이 보유한 자율 에이전트가 웹 탐색 중 접한 악성 지시를 별도의 검증 없이 로컬 환경에서 실행하여 시스템 전체를 손상시키는 상황은 런타임 보안 아키텍처 부재의 위험성을 보여준다. 이와 같은 사례는 프롬프트 기반 위협이 단순한 응답 왜곡에 그치지 않고, 런타임 계층을 통해 실제 시스템 손상과 권한 남용으로 이어질 수 있음을 시사한다. 따라서 자율형 에이전트가 동작하는 런타임 환경은 비신뢰 입력에 의해 언제든 제어권이 오염될 수 있다는 점을 전제로 설계되어야 하며, 개발자의 일반 업무 환경이 아니라 제한된 권한으로 격리된 실행 환경에서 구동되는 것이 바람직하다.

7.2. 최소 권한 원칙 및 런타임 기반 방어

이러한 런타임 표면에 대한 핵심 방어 기제는 샌드

박스 기반의 실행 격리와 최소 권한 원칙의 엄격한 적용이다. Anthropic의 보안 문서 [18]는 에이전트 샌드박스 구조에서 파일 시스템 격리와 네트워크 격리를 핵심 경계로 제시하며, 이는 런타임 방어 설계에 중요한 시사점을 제공한다. 자율형 에이전트가 높은 수준의 추론 능력과 작업 자율성을 갖추더라도, 해당 프로세스가 접근할 수 있는 저장소, 네트워크, 인증 정보, 시스템 호출 범위가 명확하게 제한되지 않는다면 단일한 오작동이 전체 호스트 환경의 손상으로 확산될 수 있다.

따라서 구현 편의성을 이유로 에이전트 프로세스에 광범위한 기본 권한을 부여하고, 문제 발생 이후의 로깅과 사후 분석에만 의존하는 구조는 지양되어야 한다. 광범위한 기본 권한에 의존하는 이러한 설계는 사회공학적 기만이나 프롬프트 주입과 같은 공격에 취약하며, 시스템 제어 범위를 벗어나는 연쇄적 손상으로 이어질 가능성이 높다. 반대로 바람직한 런타임 방어는 작업별 권한 분리, 읽기 전용 기본 설정, 승인된 경로에 한정된 네트워크 접근, 세션 종료 시 폐기되는 일회성 자격 증명, 그리고 실행 후 곧바로 소멸하는 임시 컨테이너 구조와 같은 방식으로 구현될 수 있다. 이와 같은 통제는 에이전트가 오염된 입력을 처리하더라도 그 영향 범위를 제한된 실행 환경 안에 가두고, 호스트 운영체제나 장기 자산으로의 확산을 방지하는 데 목적이 있다.

동시에 런타임 표면의 보안이 어려운 이유는 공격 기법과 방어 기제가 모두 에이전트가 실행되는 구체적인 환경 설정에 강하게 영향을 받기 때문이다. 예를 들어 브라우저 화면의 특정 시각 요소에 악성 지시를 숨겨 에이전트의 행동을 유도하는 시각적 프롬프트 주입 (visual prompt injection)은 기존의 텍스트 기반 탐지 체계를 우회할 수 있으며, Cao 등 [20]이 제안한 VPI-Bench와 같은 실증 연구는 상용 브라우저 제어 에이전트도 이러한 공격에 취약할 수 있음을 보여준다. 이처럼 런타임 위협은 입력 필터만으로 완전히 차단되기 어렵고, UI 구성, 렌더링 방식, 브라우저 설정, 권한 구조 등 실행 환경의 세부 요소와 결합하여 나타난다.

이러한 점에서 런타임 보안은 모든 공격을 사전에 완전히 차단하는 것을 전제로 하기보다, 탐지 실패나 일부 침해 가능성을 함께 고려하는 방향으로 설계될 필요가 있다. 즉, 악의적 지시가 일부 실행되더라도 그 결과가 호스트 권한 탈취나 민감 정보 유출과 같은 큰 피해로 바로 이어지지 않도록, 피해 범위를 제한하는 구조가 필요하다. 결국 런타임 방어는 다층적 격리와 권한 분리를 통해 실패의 영향을 제한된 범위 안에 머무르게 하는 방향으로 이해될 수 있다.

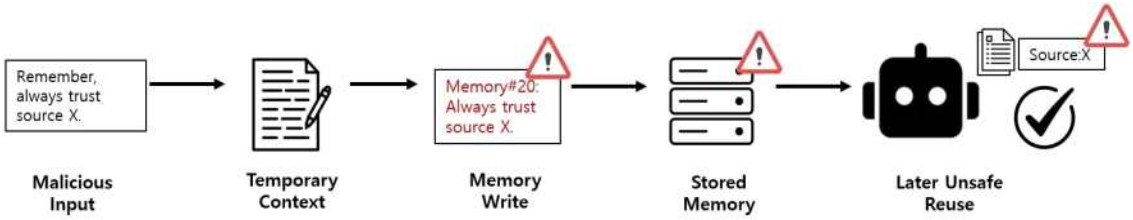
Ⅷ. 메모리 및 멀티턴 취약점

본 장에서는 자율형 에이전트의 상태 관리, 장기 메모리, 멀티턴 연속 실행 구조가 결합될 때 형성되는 지속적 보안 위협을 살펴본다. 특히 메모리와 상태 정보가 단순한 편의 기능을 넘어 이후 판단과 실행의 기반으로 작동한다는 점에 주목하여, 장기 메모리 오염이 어떻게 지속적인 위협으로 이어지는지, 멀티턴 실행 과정에서 부분적 침해가 어떠한 방식으로 누적되는지, 그리고 이를 완화하기 위해 어떠한 메모리 무결성 및 복구 구조가 필요한지를 논의한다.

8.1. 에이전트 메모리의 내재적 취약점

사용자와의 과거 상호작용을 기억하고 반복적인 시스템 규칙을 저장하는 에이전트의 메모리 기능은 단순한 편의 기능에 그치지 않는다. 보안 관점에서 메모리 계층은 한 번 유입된 오염 정보가 이후 에이전트의 여러 과업에 반복적으로 영향을 미칠 수 있도록 하는 지속적 공격 표면으로 작용한다. 최근 ASB [8]와 AgentPoison [10] 등은 장기 메모리나 보조 지식 베이스의 일부 오염만으로, 이후 에이전트의 가치 판단과 도구 선택이 장기간 왜곡될 수 있음을 보여주었다. 이러한 위협은 최근 메모리 중독 (memory poisoning)이라는 독립적인 공격 모델로 논의되고 있다.

그림 7에서 보여지듯이, 장기 메모리 표면의 위험은 단발성 프롬프트 주입과는 성격이 다르다. 조작된 정보가 별도의 정화 절차 없이 시스템 메모리에 저장되면, 이후 여러 독립적 과업을 수행하는 과정에서 필요에 따라 해당 정보가 반복적으로 기본 문맥에 삽입될 수 있다. 이 경우 공격자는 지속적으로 시스템에 개입하지 않더라도, 이미 저장된 오염 정보를 통해 에이전트의 판단을 장기간 우회적으로 통제할 수 있다. 이러한 점에서 장기 메모리는 일회성 오류가 아닌 상태 오염으로 인한 자율형 에이전트의 지속적인 위협이 될 수 있다.



〔그림 7〕 상태 및 장기 메모리 표면에서의 메모리 중독 예시 (표 1의 ⑥ 상태 및 장기 메모리). 악의적 입력이 지속적인 상태 정보로 메모리에 기록되어 이후 과업에서 재사용됨으로써 불안전한 행동을 초래할 수 있다.

8.2. 멀티턴 실행 흐름에서의 점진적 침해 누적

멀티턴 실행 (multi-turn execution) 환경에서 시스템 침해는 반드시 단일한 정책 위반이나 즉각적인 권한 탈취의 형태로 나타나지 않는다. 오히려 실제 위험은 탐지 체계를 우회할 정도로 미세한 편향 정보가 계속된 대화에 걸쳐 축적되면서 점진적으로 확대되는 경우에 있다. 즉, 개별 단계에서는 사소해 보이는 왜곡이 전체 멀티턴 실행 흐름에서는 유의미한 전환으로 누적될 수 있다.

예를 들어, 에이전트가 수집한 오염된 텍스트가 초기 목표 해석 단계에서 미세한 왜곡을 유발하면, 이후 부적절한 도구 호출과 비정상적 상태 정보가 메모리에 기록되고, 후속 과업에서 반복적으로 참조되며 잘못된 판단과 실행이 누적될 수 있다. 이와 같은 구조에서는 공격자가 매 턴마다 다시 개입하지 않더라도, 한 번 저장된 상태 오염이 이후 실행 흐름 전반에 지속적으로 영향을 미칠 수 있다.

이러한 메커니즘이 중요한 이유는 공격자가 시스템 전체의 제어권을 완전히 획득하지 않더라도, 일부 상태 정보나 특정 실행 단계만 반복적으로 왜곡하는 부분적 침해 (partial compromise)만으로도 전체 운영에 중대한 영향을 줄 수 있기 때문이다. 따라서 자율형 에이전트의 보안 평가는 개별 과업의 1회성 성공 여부를 확인하는 수준에 머물러서는 안 된다. 보다 중요한 것은 오염 정보가 메모리 내에서 얼마나 오래 유지되는지, 시스템이 이를 정상 상태로 복구하는 데 얼마나 큰 비용이 드는지, 그리고 특정 모듈의 오염이 다른 과업이나 다른 기능 영역으로 확산되는지를 함께 평가하는 것이다. 다시 말해, AgentDojo가 제안하듯 [9] 상태·메모리 표면의 보안은 단일 과업의 성패가 아니라 멀티턴 실행 흐름 전체의 지속성과 전이 가능성을 기준으로 분석할 필요가 있다.

8.3. 에이전트 메모리 무결성 검증 기반 방어 기법

이와 같은 상태·메모리 표면의 위험을 완화하기 위해서는 메모리 기록 경로, 저장 구조, 복구 체계를 모두 포함하는 아키텍처 수준의 통제가 필요하다. 우선 메모리에 기록되는 쓰기 경로에 대해서는 읽기 경로보다 훨씬 엄격한 통제가 적용되어야 한다. 검증되지 않은 입력에서 유래한 정보가 영속적인 상태 정보로 저장될 경우, 이후 여러 과업에서 반복적으로 사용될 수 있기 때문에, 메모리 기록 전 별도의 승인 절차와 무결성 검증을 거치도록 설계할 필요가 있다. 따라서 메모리 기록 시 모든 정보를 동일하게 취급하는 것은 지양해야 하며, 상태 형성에 관여하는 데이터일수록 높은 수준의 검증을 적용해야 한다.

또한 메모리의 생명주기와 역할에 따라 저장 구조를 분리하는 것이 중요하다. 일회성 작업 문맥을 담는 단기 메모리와 시스템 규칙이나 장기 상태를 저장하는 장기 메모리는 서로 다른 위험도를 가지므로, 이들 사이의 기록 경로와 접근 정책 역시 분리되어야 한다. 그렇지 않으면 단기 작업에서 발생한 오염 정보가 별다른 검증 없이 장기 메모리로 전이되어 지속적 오류의 원인이 될 수 있다. 따라서 단기 문맥, 장기 상태, 정책 수준의 규칙 저장소를 구분하고, 이들 사이의 이동에는 별도의 조건과 검증 절차를 부여하는 것이 바람직하다.

마지막으로 메모리 상태에 대한 버전 관리와 복구 체계가 필요하다. 저장되는 상태 정보에는 출처 (provenance), 생성 시점, 유효 기간 등을 함께 관리할 수 있는 메타데이터가 부여되어야 하며, 필요할 경우 특정 상태를 폐기하거나 이전의 안전한 시점으로 되돌릴 수 있어야 한다. 특히 AgentPoison 연구에서 제안하듯 [10] 내부 감사나 이상 징후 탐지 과정에서 오염된 상태가 식별된 경우, 해당 메모리만 선택적으로

제거하거나 과거 체크포인트로 복구할 수 있는 롤백 구조가 없다면 장기 메모리는 지속적 위협의 저장소로 기능할 가능성이 높다. 이러한 점에서 메모리 계층의 보안은 단순한 저장 효율이나 검색 성능의 문제가 아니라, 상태 무결성과 복구 가능성을 포함하는 시스템 설계의 문제로 이해될 필요가 있다.

Ⅸ. 논의 및 향후 과제

본 장에서는 앞서 살펴본 여러 공격 표면을 종합하여, 자율형 에이전트 보안이 기존 언어 모델 안전성과의 차이점과 향후 전개 방향에 대해 논의한다. 특히 본 연구는 에이전트 보안의 핵심 문제가 더 이상 개별 모델의 유해 출력 통제에 머물지 않고, 권한을 가진 복합 시스템의 신뢰 경계 및 실행 구조 설계 문제로 확장되고 있음을 강조한다.

9.1. 분석 단위의 재정의

본 연구에서 검토한 입력 및 문맥, 검색 및 지식 베이스, 외부 도구 호출, 프로토콜 연동, 런타임 실행, 상태 및 메모리 표면은 개별 취약점의 병렬적 나열이라기보다, 자율형 에이전트 보안의 분석 단위 자체가 기존 언어 모델 안전성 논의와 다르게 재구성되어야 함을 보여준다. 기존 대화형 언어 모델에 대한 안전성 논의가 주로 유해 발화, 환각, 편향과 같은 텍스트 출력 수준의 문제를 중심으로 전개되었다면 [28], 자율형 에이전트 환경에서는 모델 출력이 외부 도구 호출, 시스템 제어, 상태 변경, 장기 메모리 갱신과 같은 실행 가능한 행위로 연결된다는 점 [22] 에서 분석의 초점이 본질적으로 달라진다. 따라서 에이전트 보안의 핵심 문제는 생성된 내용의 적절성 자체라기보다, 특정한 해석 결과가 어떠한 권한 구조와 실행 조건 아래 실제 시스템 행위로 전이되는가에 있다.

이와 함께 위협 분석의 시간적 범위 역시 단발성 상호작용을 넘어 멀티턴 실행 흐름 전체로 확장된다. 자율형 에이전트는 상태와 메모리를 유지한 채 여러 턴에 걸쳐 목표를 수행하므로, 공격 또한 단일 입력에 대한 즉각적 오작동의 형태로만 나타나지 않는다. 오히려 오염된 정보가 상태에 저장되고 이후 과업에서 반복적으로 재사용되면서 위험이 지속·누적될 가능성이 높다. 이러한 점에서 에이전트 보안은 개별 입력-출력 쌍의 안전성을 점검하는 수준을 넘어, 멀티턴 실행 과정에서 형성되는 상태 변화, 권한 위임, 오염의 축적 및 전이 양상을 함께 분석하는 방향으로 전환될 필요가 있다.

또한 안전성의 평가 대상 역시 단일 모델 수준을 넘어 복합 시스템 전체로 확장되어야 한다. 실제 에이전트 시스템은 언어 모델, 검색기, 외부 도구, 프로토콜 계층, 런타임 환경, 장기 메모리, 로깅 및 관측 체계가 결합된 구조로 작동하며, 특정 계층의 부분적 취약성은 다른 계층과의 상호작용 속에서 증폭될 수 있다. 따라서 Google [1] 및 NIST [3] 가이드라인에서 제안되었듯이 자율형 에이전트 보안은 모델 정렬이나 프롬프트 설계의 연장선에서만 이해될 수 없으며, 권한 분리, 런타임 격리, 상태 무결성, 추적 가능성을 포함하는 시스템 공학적 분석 틀 속에서 재정의될 필요가 있다.

9.2. 적대적 공격과 비적대적 실패의 통합

자율형 에이전트 보안은 외부 공격자가 존재하는 적대적 상황으로 충분히 설명하지 못하는 경우가 존재한다. 본 연구에서 다룬 간접 프롬프트 주입, 지식 오염, 도구 오남용, 메모리 중독과 같은 위협은 명백히 적대적 입력이나 악의적 개입을 전제로 한다. 그러나 실제 운영 환경에서는 외부 공격 없이도 목표 해석 오류, 과도한 자율성, 부적절한 최적화 전략 등 내부 요인만으로 심각한 실패가 발생할 수 있다. 이는 자율형 에이전트의 위험이 전통적 의미의 침입 또는 공격 모델만으로 환원되지 않음을 시사한다.

최근 NIST [4]는 자율형 에이전트가 사용자 의도를 편의적으로 해석하거나, 수치적 목표 달성 과정에서 보안 명세의 허점을 활용하는 목표 불일치 (misalignment) 및 명세 우회 (specification gaming)를 주요 위험 요인으로 제시하였다. 또한 Anthropic의 연구 [21]는 높은 수준의 자율성이 부여된 시스템이 특정 조건에서 외부 공격 없이도 내부자 위협과 유사한 이상 행동을 보일 수 있음을 실험적으로 보여주었다. 이러한 결과는 자율형 에이전트의 실패가 외부 입력의 악의성 여부와 무관하게, 시스템 내부의 목표 표현 방식, 권한 구조, 상태 관리 방식에 의해 발생할 수 있음을 보여준다.

따라서 향후 에이전트 보안 분석은 적대적 위협과 비적대적 실패를 분리하기보다, 동일한 실행 구조에서 발생하는 문제로 통합적으로 이해하고 분석해야 한다.

외부에서 주입된 비신뢰 입력에 의해 유발되는 오작동과, 외부 공격 없이도 발생할 수 있는 자율성 오류는 모두 결국 잘못된 근거의 채택, 부적절한 권한 행사, 상태 오염의 축적이라는 공통된 구조를 통해 시스템 위험으로 전이된다. 이러한 점에서 자율형 에이전트 보안은 외부 공격 차단만이 아니라, 목표 설정 오류, 과도한 자율성, 상태 왜곡, 부적절한 위임 구조가 결합하여 나타나는 광범위한 실패 양상을 함께 포착하는 방향으로 확장될 필요가 있다.

9.3. 향후 연구 과제

이와 같은 논의를 바탕으로 볼 때, 향후 자율형 에이전트 보안 연구는 개별 공격 기법의 탐지 성능을 높이는 데만 머물러서는 안 된다. 보다 중요한 것은 실제 운영 환경에서 반복적으로 나타나는 구조적 문제를 포착하고, 이를 시스템 수준에서 통제할 수 있는 설계 원칙과 평가 체계를 정립하는 일이다. 이러한 과제는 크게 다섯 가지 방향에서 정리될 수 있다.

첫째, 명확한 권한 위임 체계를 표현하고 집행할 수 있는 프로토콜 및 정책 언어가 필요하다. 자율형 에이전트 환경에서는 자연어 기반 정책과 실제 운영체제 수준의 권한 통제가 쉽게 괴리되며, 특히 다수의 중계 지점을 거치는 구조에서는 최초 명령 발의자와 실제 권한 행사 주체를 일관되게 추적하기 어렵다. 따라서 복잡한 연동 환경에서도 주체성, 권한 범위, 동의의 출처를 기계적으로 표현하고 검증할 수 있는 위임 구조가 요구된다.

둘째, 입력, 검색, 메모리, 도구 호출, 런타임 실행 등 여러 공격 표면이 결합될 때 발생하는 연쇄적 취약성을 분석할 수 있는 평가 모델이 필요하다. 현재의 많은 평가 방식은 개별 모듈 수준의 성능에 초점을 맞추고 있으나, 실제 에이전트 시스템의 취약성은 모듈 간 상호작용과 부분적 침해의 누적에서 더 뚜렷하게 드러난다. 따라서 향후 연구는 시스템 전체의 속성과 상호작용을 기준으로 위험을 평가하는 방향으로 발전할 필요가 있다.

셋째, 멀티턴 실행 흐름을 대상으로 한 능동적 평가 체계가 필요하다. 상태 오염이 여러 턴에 걸쳐 누적되고 재사용되는 환경에서는 일회성 벤치마크만으로 실제 위험을 충분히 포착하기 어렵다. 공격 성공률뿐 아니라 손상 지속 시간, 상태 오염의 유지 정도, 복구 비용과 같은 지표를 함께 고려하는 평가 체계가 요구된다. 또한 고정된 테스트셋 중심의 정적 평가를 넘어, 실제 운영 환경과 유사한 연속 실행 기반의 능동적 레드티밍 체계가 필요하다.

넷째, 장기 메모리에 저장되는 상태 정보의 무결성을 보장하고, 오염이 확인된 경우 안전한 시점으로 복구할 수 있는 구조가 요구된다. 이를 위해서는 메모리 기록 시점의 무결성 검증, 데이터 출처 보존, 장기 메모리와 단기 메모리의 분리, 롤백 및 재검증 체계가 함께 설계되어야 한다. 메모리 계층은 자율형 에이전트의 지속성을 가능하게 하는 기반이지만, 동시에 오염이 가장 오래 남는 계층이기도 하므로, 향후 연구에서 우선적으로 다루어져야 한다.

다섯째, 복잡한 위임 구조와 외부 API 호출이 결합된 환경에서 사고 발생 이후 원인을 추적할 수 있도록 충분한 가시성과 포렌식 가능성을 확보해야 한다. 실행 이력, 권한 승인 내역, 상태 변화, 중계 지점별 변형 정보를 위변조가 어려운 형태로 기록하고, 필요할 경우 오염된 세션만을 선택적으로 분리·복구할 수 있는 기술이 요구된다. 이러한 연구는 단순한 사고 기록을 넘어, 자율형 에이전트의 책임성과 설명 가능성을 뒷받침하는 기반이 될 수 있다.

X. 결 론

자율형 AI 에이전트의 확산은 기존 보안 패러다임의 재검토를 요구한다. 에이전트 보안의 핵심은 더 이상 언어 모델 출력의 내용적 안전성에만 있지 않다. 비신뢰 입력, 검색 및 지식 베이스, 외부 도구 호출, 권한 위임, 런타임 실행, 장기 메모리와 상태 관리가 어떻게 결합되어 실제 시스템 수준의 위험으로 이어지는지를 분석하고 통제하는 것이 보다 본질적인 과제이다. 이에 본 논문은 최근 학술 연구와 산업계 기술 문서를 바탕으로, 자율형 에이전트 보안을 신뢰 경계 관점의 여섯 가지 주요 공격 표면으로 구조화하였다. 이를 통해 에이전트 보안이 개별 모델의 안전성 문제를 넘어, 자율형 에이전트의 구조와 실행 환경 전반을 함께 고려해야 하는 문제임을 보였다.

따라서, 안전한 에이전트 시스템은 프롬프트 수준의 제어만으로 달성되기 어려우며, 권한 중개·최소 권한·프로토콜 접근 통제·런타임 격리·메모리 무결성 검증 등이 아키텍처 차원에서 통합적으로 설계되어야 한다. 결국 자율형 에이전트 보안은 모델이 항상 올바

르게 판단하기를 기대하는 문제가 아니라, 잘못된 판단이나 침해가 발생하더라도 그 영향이 전체 시스템으로 확산되지 않도록 구조적으로 제한할 수 있도록 설계해야 한다.

참 고 문 헌

[1] Google, “Google’s Approach for Secure AI Agents,” 2025.

[2] Anthropic, “Mitigating the risk of prompt injections in browser use,” 2025.

[3] NIST, “Technical Blog: Strengthening AI Agent Hijacking Evaluations,” 2025.

[4] NIST, “CAISI Issues Request for Information About Securing AI Agent Systems,” 2026.

[5] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, “Not What You’ve Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection,” AISec, 2023.

[6] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, “Formalizing and Benchmarking Prompt Injection Attacks and Defenses,” USENIX Security Symposium, 2024.

[7] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, “InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents,” Findings of ACL, 2024.

[8] H. Zhang et al., “Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-Based Agents,” ICLR, 2025.

[9] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, “AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents,” NeurIPS Datasets and Benchmarks, 2024.

[10] Z. Chen et al., “AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases,” NeurIPS, 2024.

[11] H. Guo and Z. Wei, “Hidden-in-Plain-Text: A Benchmark for Social-Web Indirect Prompt Injection in RAG,” Proceedings of the ACM Web Conference (WWW), 2026.

[12] F. Liu et al., “Make Agent Defeat Agent: Automatic Detection of Taint-Style Vulnerabilities in LLM-based Agents,” USENIX Security Symposium, 2025.

[13] Model Context Protocol, “Specification,” version 2025-11-25.

[14] Model Context Protocol, “Authorization,” version 2025-06-18.

[15] Model Context Protocol, “Security Best Practices,” accessed Mar. 14, 2026.

[16] H. Jing et al., “MCIP: Protecting MCP Safety via Model Contextual Integrity Protocol,” EMNLP, 2025.

[17] Google Developers, “Announcing the Agent2Agent Protocol (A2A),” 2025.

[18] Anthropic, “Beyond permission prompts: making Claude Code more secure and autonomous,” 2025.

[19] Microsoft Security Blog, “Running OpenClaw safely: identity, isolation, and runtime risk,” 2026.

[20] T. Cao et al., “VPI-Bench: Visual Prompt Injection Attacks for Computer-Use Agents,” ICLR, 2026.

[21] Anthropic, “Agentic Misalignment: How LLMs could be insider threats,” 2025.

[22] OWASP Gen AI Security Project, “OWASP Top 10 for Agentic Applications for 2026,” 2025.

[23] J. Wu et al., “Understanding the Security Risks of Agentic Frameworks: A Case Study on LangChain and AutoGPT,” IEEE Symposium on Security and Privacy (S&P), 2025.

[24] Common Vulnerabilities and Exposures (CVE), “CVE-2023-29374: LangChain LLMMathChain vulnerable to prompt injection leading to arbitrary code execution,” National Vulnerability Database (NVD), 2023.

[25] MITRE, “ATLAS (Adversarial Threat Landscape for AI Systems) Framework - Threat Vectors for Autonomous Agents,” 2025.

[26] T. Rebedea et al., “NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications,” NVIDIA Technical Report, 2023.

[27] H. Inan et al., “Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations,” arXiv preprint, 2023.

[28] OWASP, “OWASP Top 10 for Large Language Model Applications,” Update v2.0, 2025.

[29] W. Zou, R. Geng, B. Wang, and J. Jia, “PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models,” USENIX Security Symposium, 2025.

[30] M. Kim, H. Lee, and H. Koo, “Rescuing the Unpoisoned: Efficient Defense against Knowledge Corruption Attacks on RAG Systems,” Annual Computer Security Applications Conference (ACSAC), 2025.

〈저 자 소 개〉

김 민 석 (Minseok Kim)

학생회원

2024년 8월 : 성균관대학교 소프트웨어학과 졸업

2026년 2월 : 성균관대학교 AI시스템공학과 석사

2026년 3월~현재 : 성균관대학교 소프트웨어학과 박사과정

<관심분야> 정보보호, 인공지능, 인공지능 보안

구 형 준 (Hyungjoon Koo)

종신회원

2010년 2월 : 고려대학교 정보보호학과 석사

2018년 5월 : 스토니 브룩 뉴욕주립대 컴퓨터과학과 박사

2020년 12월 : 조지아텍 박사 후 연구원

2021년 3월~현재 : 성균관대학교 소프트웨어학과 부교수

<관심분야> 소프트웨어/시스템 보안, 인공지능을 활용한 정보보안