Model Context Protocol 구현체의 보안 취약점 패턴에 관한 정적 분석 연구*

김지안, 이동건, 서영석
영남대학교 컴퓨터공학과
kjm321987@yu.ac.kr, dklee77@ynu.ac.kr, ysseo@yu.ac.kr

Security Vulnerability Patterns of Model Context Protocol Implementations via Static Analysis

Jian Kim, Dong-Gun Lee, Yeong-Seok Seo
Dept. of Computer Engineering, Yeungnam University

요 약

최근 AI 시스템의 기능 확장을 위해 Model Context Protocol (MCP)의 채택이 증가하고 있으나, 그 구현체의 보안성에 대한 실증적 연구는 부족하다. 본 연구는 251개의 MCP 서버와 113개의 일반 서버 (비교군)를 정적 분석하여 MCP 생태계의 고유한 보안 취약성 프로파일을 도출하였다. 분석 결과, MCP 구현체에서 총 15개 유형의 취약점이 일관되게 발견되었으며, 이는 (1)설정 및 민감 정보의 하드코딩, (2)안전하지 않은 외부 프로세스 호출, (3)신뢰할 수 없는 코드 및 예외 처리라는 세 가지 패턴으로 유형화되었다. 본 연구는 MCP 생태계의 잠재적 위험을 진단하고, 개발 단계에서 고려해야 할 보안 강화의 필요성을 제시한다.

1. 서론

Model Context Protocol (MCP)은 AI 애플리케이션과 외부 시스템의 상호작용을 표준화하는 개방형 프로토콜이다. 최근 대규모 언어 모델 (Large Language model, LLM) 기반 AI 에이전트의 활용이 급증함에 따라 MCP의 사용 또한 높아지고 있지만, 그 오픈소스 구현체의 보안성에 대한 체계적인 분석은 부족하다. 특히 MCP 구현체만의 고유한 보안 취약점 패턴이 존재하는지에 대한 연구는 부재한 실정이다 [1].

이에 본 연구는 MCP 구현체의 고유한 보안 특성을 규명하고자, MCP 서버와 일반 서버 (비교군)의 취약점을 정적 분석 (Static Analysis)을 통해 비교 분석하였다. 분석 결과, MCP 구현체에서 공통으로 발견되는 취약점들이 15개 유형으로 수렴됨을 확인했다. 본 논문은 이 패턴을 심층 분석하여 MCP 생태계의 보안 강화를 위한 기반 자료를 제공한다.

2. 분석 대상 및 방법

2.1 분석 대상

본 연구는 MCP 구현체의 고유한 취약점 패턴을 식별하기 위해 비교군을 설정하여 비교 분석을 수행했다. MCP 서버 그룹은 Anthropic의 공식 Github 리포지토리에서 명시한 공식 서버 목록을 기준으로, Python 또는 JavaScript/TypeScript로 작성된 공개 구현체 251개를 분석 대상으로 선정하였다. 비교군인 일반 서버 그룹은 MCP와 직접적인 관련이 없는 범용 목적의 오픈소스 서버 프로젝트 중, MCP 서버군과 유사한 규모 (GitHub Star 500개 이상, 최근 1년 내 커밋 존재)를 가진 Python, JavaScript/TypeScript 기반의 프로젝트 113개를 선정하여 비교 기준으로 사용하였다.

2.2 분석 방법

본 분석은 소스 코드 수준의 잠재적 결함을 식별하는 데 초점을 두기 위해 자동화된 스크립트를 통한 정적 분석 (Static Application Security Testing) 방식으로 수행되었다. Python 기반 구현체는 Bandit (v1.8.6)을, JavaScript/TypeScript 기반 구현체는 의존성 취약점 분석을 위해 npm audit (npm v10.8.1), 코드 품질 및 잠재적 오류 분석을 위해 ESLint (v9.35.0, eslint:recommended 규칙 세트 적용)를 병행하였다.

3. 주요 취약점 패턴 분석

정적 분석 결과, 그림 1과 같이 MCP 서버와 일반

* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(RS-2023-NR076933).

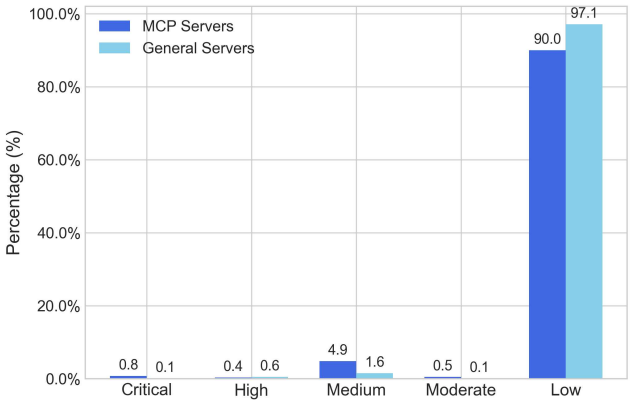

Fig 1. Vulnerability Severity Distribution

서버 (비교군) 모두에서 'Low' 등급의 취약점이 각각 90.0%와 97.1%로 가장 높은 비율을 차지했다. 주목할 점은, MCP 서버의 취약점 유형이 15개로 한정되어 나타났다는 것으로, 이는 MCP 구현체들의 구조적 동질성을 암시한다. 표 1은 이 15개 취약점의 코드 1,000줄당 취약점 (Vulnerabilities per 1,000 Lines of Code, VLOC) 밀도를 비교군과 비교하고, 이를 세 가지 패턴으로 분류한 결과이다. 전체적으로 MCP 서버군은 일반 서버군보다 낮은 VLOC 밀도를 보였으며, 발견된 패턴은 다음과 같다.

Table 1. MCP Vulnerability Pattern Classification

Pattern	Vulnerability Type	MCP Freq. (VLOC)	Non-MCP Freq. (VLOC)
1	hardcoded_sql_espressions	93(0.08)	924 (0.45)
	hardecoded_password_string	48(0.04)	820 (0.40)
	hardcoded_bind_all_funcarg	41(0.04)	119 (0.06)
	hardcoded_password_funcarg	10(0.01)	960 (0.47)
	hardcoded_tmp_directory	3(0.003)	718 (0.35)
2	subprocess_without_shell _equals_true	56(0.05)	1,142 (0.56)
	start_process_with _partial_path	54(0.05)	342 (0.17)
	start_process_with_a_shell	7(0.006)	28(0.01)
	subprocess_popen_with _shell_equals_true	2(0.002)	101 (0.05)
3	assert_used	3,564 (3.12)	175,046 (85.25)
	blacklist	49(0.04)	2,214 (1.08)
	request_without_timeout	48(0.04)	302 (0.15)
	try_except_pass	42(0.04)	396 (0.19)
	try_except_continue	5(0.004)	18 (0.009)
	hashlib	1(0.001)	307 (0.15)

3.1 패턴 1: 설정 및 민감 정보의 하드코딩

MCP 구현체에서는 소스 코드 내에 민감 정보를 직접 포함하는 하드코딩 패턴이 발견되었다. 'hardcoded_sql_expressions'가 93건 발견된 것은 무시할 수 없는 규모로, 이러한 관행은 시스템의 유연성을 저해하며 소스 코드 유출 시 직접적인 정보 노출로 이어지는 심각한 위험을 내포한다.

3.2 패턴 2: 안전하지 않은 외부 프로세스 호출

MCP 구현체들은 'subprocess_without_shell_equals_true'와 'start_process_with_partial_path' 취약점이 거의 동일한 빈도 (각각 56건, 54건)로 발견되는 특징을 보였다. 이는 특정 모듈이나 개발 관행이 두 유형의 취약점을 동시에 유발할 가능성을 내포한다. 이러한 패턴은 커맨드 인젝션 (Command Injection) 공격에 매우 취약하므로, 발견 빈도와 무관하게 서비스 신뢰성에 심각한 위협이 될 수 있다.

3.3 신뢰할 수 없는 코드 및 예외 처리

가장 높은 빈도를 보인 'assert_used' (3,564건)는 상용 환경에서 검증 로직이 비활성화될 수 있는 대표적인 신뢰할 수 없는 코드이다. 이외에도 안전하지 않은 역직렬화를 유발하는 'blacklist' (49건)나 오류를 의도적으로 무시하는 'try_except_pass' (42건)와 같은 패턴이 여전히 발견되었다. 이는 시스템의 전반적인 보안 수준을 저하하며, 코드의 견고성을 확보하기 위한 엄격한 예외 처리 정책이 부족함을 시사한다.

4. 결론

본 연구는 정적 분석을 통해 다수의 MCP 구현체에 공통으로 내재된 보안 취약점의 실증적 패턴을 유형화하였다. 분석 결과, MCP 구현체는 상호운용성을 목표로 하는 프로토콜 구현체로서는 심각한 문제인 애플리케이션의 보안성이 호스트 환경에 과도하게 의존하게 되는 구조적 결함을 가짐을 발견하였다. 향후 MCP 표준의 보안 가이드라인 수립 시, 본 연구에서 식별된 패턴들을 고려해야 하며, 실제 공격 가능성을 검증하는 동적 분석 (Dynamic Application Security Testing) 등의 후속 연구가 필요하다.

참고문헌

[1] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model context protocol (mcp): Landscape, security threats, and future research directions," arXiv preprint arXiv:2503.23278, 2025.