사물인터넷융복합논문지 Vol. 11, No. 5, pp. 19-25, 2025 https://doi.org/10.20465/KIOTS.2025.11.5.019

# 모델 컨텍스트 프로토콜(MCP) 환경에서 퍼징 기반 보안 취약점 검출 방법

박정규[1], 백영미[2*]
[1]대진대학교 컴퓨터공학전공 교수, [2]창신대학교 스마트팩토리학부 교수

# Fuzzing-based Security Vulnerability Detection Method in Model Context Protocol (MCP) Environments

Jung Kyu Park[1], Youngmi Baek[2*]
[1]Professor, Department of Computer Engineering, Daejin University
[2]Professor, School of Smart Factory, Changshin University

**요 약** 최근 대규모 언어 모델(LLM)의 급격한 발전은 Model Context Protocol(MCP)을 새로운 도구 통합 표준으로 부각시켰다. 그러나 MCP 환경에서의 보안 취약점 연구는 아직 충분히 이루어지지 않았다. 본 연구는 MCP 환경에서 퍼징 기반 보안 취약점 검출에 대한 초기 체계적 평가를 제공하며, 대표적인 두 모듈(데이터베이스 질의 엔진인 SQLite와 파일 시스템 인터페이스)을 대상으로 퍼징 실험을 수행하였다. 이를 위해 맞춤형 퍼징 하네스를 구현하여 다양한 페이로드를 체계적으로 생성하고, 30분 단위의 실험을 통해 성능 및 응답 특성을 측정하였다. 실험 결과, SQL 기반 서버는 총 77,758건의 테스트 케이스를 생성하며 평균 페이로드 길이와 다양성이 높게 나타난 반면, 파일 시스템 기반 서버는 47,520건으로 비교적 짧고 단순한 페이로드가 주를 이루었다. 특히 응답률 측면에서 SQL 퍼징은 파일 시스템 퍼징 대비 약 3배 높은 값을 기록하며 뚜렷한 차이를 보였다. 이와 같은 결과는 MCP 환경에서 퍼징 대상 도구 유형에 따라 취약점 탐지 특성과 보안 안정성이 크게 달라질 수 있음을 시사한다. 본 연구는 MCP 환경에서의 퍼징 기반 보안 취약점 검출에 대한 체계적 초기 평가를 제공하며, 향후 MCP 기반 응용의 보안 강화를 위한 기초 연구로서 의의가 있다.

**주제어 :** 모델 컨텍스트 프로토콜, 퍼징, 보안 취약점 검출, 대규모 언어 모델, SQL 및 파일 시스템 비교

**Abstract** The rapid advancement of large language models (LLMs) has highlighted the Model Context Protocol (MCP) as a new standard for tool integration. However, security vulnerability research in MCP environments remains insufficient. This study presents one of the first experimental applications of fuzzing in MCP environments, targeting two representative modules: a SQLite-based query engine and a file system (FS) interface. To this end, customized fuzzing harnesses were developed to systematically generate diverse payloads, and performance metrics were collected over 30-minute experiments. The results indicate that the SQL-based server produced 77,758 test cases with greater payload diversity and longer average payload lengths, whereas the FS-based server produced 47,520 cases with shorter and simpler payloads. Notably, the response rate showed a clear disparity, with SQL fuzzing achieving approximately three times higher response success compared to FS fuzzing. These findings suggest that fuzzing characteristics and security robustness vary significantly depending on the target tool type within MCP environments. Overall, this study provides an early systematic evaluation of fuzzing-based vulnerability detection in MCP environments, offering foundational insights to strengthen the security of MCP-based applications.

**Key Words :** Model Context Protocol (MCP), Fuzzing, Security Vulnerability Detection, Large Language Models (LLMs), SQL and File System Comparison

본 과제(결과물)는 2025년도 교육부 및 경상남도의 재원으로 경상남도RISE센터의 지원을 받아 수행된 지역혁신중심 대학지원체계(RISE)의 결과입니다(2025-RISE-16-009-0009).
*교신저자 : 백영미(ymbaek@cs.ac.kr)
접수일 2025년 09월 09일 수정일 2025년 09월 29일 심사완료일 2025년 10월 07일

## 1. 서론

최근 대규모 언어 모델(Large Language Model, LLM)의 급격한 발전은 다양한 산업 분야에서 인공지능(AI)의 활용을 확장시키고 있다[1,2]. 특히, LLM이 외부 데이터베이스, API, 파일 시스템 등과 안전하게 상호작용 하도록 지원하는 Model Context Protocol(MCP)은 새로운 소프트웨어 아키텍처 패러다임으로 주목받고 있다[3-5]. MCP는 JSON-RPC 기반의 표준화된 인터페이스를 제공함으로써, 모델과 외부 시스템 간 통신을 구조화하고 확장성을 보장한다. 그러나 이와 같은 새로운 프로토콜의 도입은 기존 연구에서 다루지 않았던 보안 위협을 동반할 수 있다[6].

기존의 소프트웨어 테스트 및 보안 검증 연구들은 주로 전통적인 API 통신, 웹 애플리케이션, 운영체제 수준의 취약점 탐지에 집중해 왔다[7]. 퍼징(Fuzzing)은 프로그램이 예상하지 못한 입력에 대한 반응을 자동 탐색함으로써 잠재적 오류와 보안 결함을 식별하는 실험적 접근법이다[8-10]. AFL(American Fuzzy Lop), boofuzz와 같은 도구들은 다양한 네트워크 프로토콜 및 RPC 환경에서 안정적으로 보안 취약점을 식별하는 데 활용되고 있다[11-13]. 그러나 MCP 환경은 아직 초기 단계에 있으며, 관련된 보안성 검증이나 퍼징 적용 사례는 거의 보고되지 않았다. 이는 MCP를 활용한 LLM 응용 시스템의 신뢰성을 확보하는 데 중요한 연구 공백(research gap)으로 지적될 수 있다[14].

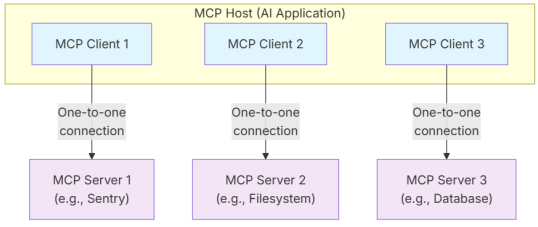
특히 MCP는 JSON 직렬화와 세션 관리 등 여러 단계에서 입력 검증이 요구되며, 이 과정의 취약점은 보안 문제를 야기할 수 있다. 예를 들어, 비정상 포맷의 JSON 메시지, 과도한 페이로드 삽입, 필드 누락 또는 무한 루프를 유발할 수 있는 입력은 시스템 성능 저하나 서비스 거부(DoS)를 야기할 수 있다. 이러한 위협은 단순히 개별 MCP 서버의 안정성 문제를 넘어, LLM과 연계된 애플리케이션 전체의 보안성을 위협할 수 있다[6]. 따라서 MCP의 보안 취약점을 체계적으로 탐지하고 이를 분류·분석하는 연구는 학술적·산업적으로 모두 큰 의미를 갖는다.

본 연구는 MCP 환경에 퍼징 기법을 적용해 취약점을 탐지하고, 이를 DoS·무결성 훼손·정보 유출 등으로 분류하는 방법을 제안한다. 이를 위해 오픈소스 MCP 서버(GitHub MCP, SQLite MCP 등)를 대상으로 퍼징 실험을 수행하고, 도출된 로그와 결과를 체계적으로 분석한다. 본 연구의 기여점은 크게 세 가지로 요약할 수 있다. 첫째, MCP 환경에서 퍼징 기법을 적용한 최초의 보안성 검증 연구를 제안한다. 둘째, 발견된 취약점을 카테고리별로 정리하여 MCP 보안 위협 모델을 구축한다. 셋째, 이를 바탕으로 향후 LLM-도구 연동 시스템에서 적용 가능한 자동화된 보안 검증 프레임워크의 가능성을 제시한다.

## 2. 관련 연구

### 2.1 MCP의 개념과 표준

Model Context Protocol(MCP)은 LLM 애플리케이션과 외부 데이터 소스·도구를 표준화된 방식으로 연결하기 위한 개방형 프로토콜이다[3]. Anthropic이 2024년 말 공개한 초기 제안과 함께, 현재는 독립 웹 문서/스키마로 사양이 관리되며(모듈식 레이어, 수명주기 관리 포함) 핵심 메시지 포맷으로 JSON-RPC 2.0을 채택한다. 전송 계층은 표준 입출력(stdio)과 스트리머블 HTTP를 공식 지원해 IDE·에이전트·데스크톱 앱 등 다양한 런타임에서 일관된 상호운용성을 제공한다. 사양은 요청/응답/알림, 배치 처리, 오류 코드 규범을 명시해 도구 호출과 컨텍스트 전달을 프로토콜 수준에서 안정화한다. Fig 1은 MCP 구조에서 MCP host, MCP client, MCP server 등의 핵심 요소를 표시하고 있다.




[Fig. 1] Key participants in the MCP architecture[3]

### 2.2 MCP 구현·적용 동향

공식 문서와 깃허브 조직은 레퍼런스 SDK(Typescript/Python)와 예제 서버(파일·검색·브라우저 등)를 제공해 재현 가능한 통합 패턴을 제시한다. 마이크로소프트의 학습용 리포지터리와 GitHub Copilot 문서도 MCP 개념·구성요소·연계 방식을 소개하며, 업계 적용 가속에 기여하고 있다. 언론·기술 리포트에 따르면, 2024년 말 공개 직후 여러 개발사/에이전트 플랫폼이 채택을 발표했고, 2025년 들어서는 생산성 앱(예: Notion·Stripe 등)

과의 양방향 데이터 연동 사례가 빠르게 확산 중이다.

### 2.3 보안 취약점 및 퍼징 연구

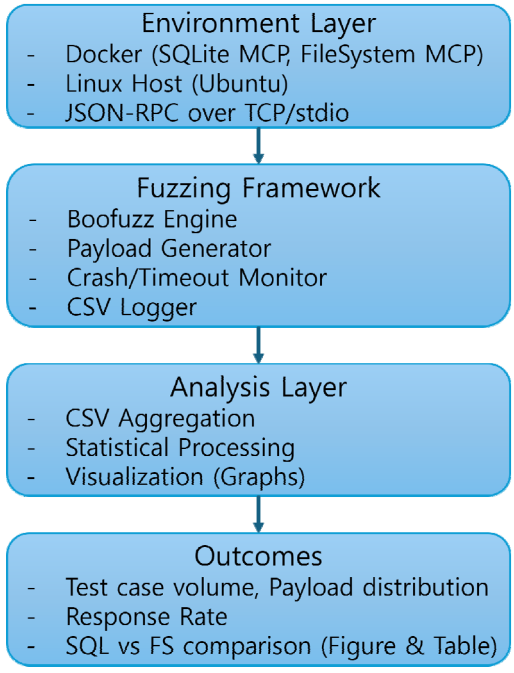
MCP는 새로운 표준으로 주목받고 있으나, 보안적 측면에서는 아직 체계적인 연구가 부족하다. 기존 연구에서는 JSON 기반 통신에서 발생할 수 있는 구조적 취약점과 메시지 포맷 오류가 시스템 안정성을 저해할 수 있음을 보고하였다. 또한 LLM과 외부 도구의 연계 과정에서는 권한 남용, 데이터 유출, 서비스 거부(DoS) 등 다양한 위협 시나리오가 제시되고 있으며, 이는 MCP 환경에서도 동일하게 적용될 가능성이 높다[15]. 퍼징(Fuzzing) 기법은 이러한 잠재적 취약점을 식별하는 효과적인 방법으로, 비정상 입력을 자동 생성하여 시스템의 예외 상황을 탐지하는 데 활용된다. 최근 퍼징 도구들이 다양한 RPC 환경에 적용되고 있으며, MCP 역시 이러한 접근을 적용할 수 있는 새로운 연구 영역으로 부각되고 있다[12,13].

## 3. 연구 방법

본 연구는 MCP(Model Context Protocol) 환경에서 퍼징(Fuzzing) 기법을 적용하여 잠재적 보안 취약점을 식별하고 이를 정량적으로 분석하는 것을 목표로 한다. 연구 방법은 크게 실험 환경 구축, 퍼징 입력 생성, 테스트 실행, 로그 및 결과 분석의 네 단계로 구성되며, SQLite MCP와 FileSystem MCP 두 가지 서버를 대상으로 진행한다.

### 3.1 실험 대상 시스템

실험 대상은 공개적으로 배포된 오픈소스 MCP 서버 중 SQLite MCP와 FileSystem MCP를 선정하였다. SQLite MCP는 데이터베이스 질의 및 스키마 조작 기능을 제공하므로 구조화된 파라미터 변형에 따른 응답 특성을 분석하기 적합하다. FileSystem MCP는 파일 읽기·쓰기·경로 탐색 등의 I/O 기능을 제공하여, 경로 변형 및 대용량 요청이 서버 안정성에 미치는 영향을 검증할 수 있다. GitHub MCP는 인증 및 외부 종속성으로 인해 재현성이 낮아 본 연구에서 제외하였다. Fig 2는 본 연구의 실험 환경 구성과 MCP 퍼징 절차를 도식적으로 나타낸 것이다.




[Fig. 2] Research Environment and Methodology for MCP Fuzzing

### 3.2 실험 환경 구축

실험 환경은 Ubuntu 22.04 LTS 운영체제에서 Docker 컨테이너 기반으로 구성하였다. 각 MCP 서버는 독립된 컨테이너에서 실행되며, 클라이언트는 Python SDK를 이용하여 서버와 통신한다. 퍼징 도구로는 AFL++와 boofuzz를 사용한다. AFL++는 커버리지 기반 변이 입력을 대규모로 생성하는 역할을 하고, boofuzz는 상태 기반(session-based) 시나리오를 정의하여 JSON-RPC 메시지의 필드 변형과 시퀀스 테스트를 수행한다[10].

추가적으로, 퍼징 소프트웨어는 Python 3.11 환경에서 Boofuzz 프레임워크를 확장하여 구현하였다. 각 퍼징 세션은 30분 단위로 수행되며, JSON-RPC 형식의 입력을 TCP 소켓을 통해 SQLite MCP(db.query)와 FileSystem MCP(fs.read, fs.write, fs.list) 모듈에 전달하였다. 실험은 동일 환경에서 수행되어 결과 데이터는 자동으로 CSV 로그(sqlite_results.csv, fs_results_raw.csv)로 저장되어 후속 통계 및 시각화 분석에 활용되었다.

### 3.3 퍼징 입력 생성

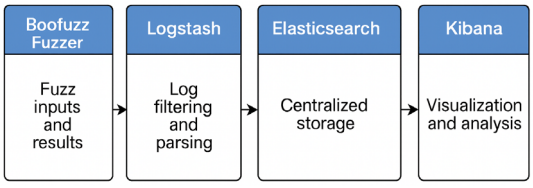
퍼징 입력은 JSON-RPC 2.0 메시지를 기본 템플릿으로 정의한 후, 다음과 같은 변형 규칙을 적용하였다.

- 필드 변형(Field Mutation): method, params, id 필드의 누락, 중복, 타입 교란.
- 페이로드 변형(Payload Mutation): 과도한 길이 문자열, 중첩 객체, 대용량 데이터 삽입.
- 시퀀스 변형(Sequence Mutation): 요청-응답 순서 뒤바꾸기, 연속 알림(Notification) 전송으로 상태 불일치 유도.

SQLite MCP에서는 db.query, db.exec 메서드를 대상으로, FileSystem MCP에서는 fs.read, fs.write, fs.list 메서드를 대상으로 각각 입력을 생성하였다.

### 3.4 테스트 실행 및 로그 수집

실험 계획은 24시간 수행을 목표로 설계되었으나, 본 논문에서는 SQLite MCP와 FileSystem MCP의 성능을 비교하기 위해 동일 조건에서 30분간 수행한 결과를 제시한다. 생성된 입력은 각 MCP 서버에 자동으로 주입되며, 정상 응답률, 오류 응답, 크래시 발생 횟수, 응답 시간 지연을 주요 지표로 수집한다. 실험 과정에서 발생하는 모든 이벤트는 Elastic Stack을 기반으로 중앙 로그 서버에 저장된다. 로그에는 테스트 케이스 번호, 입력 데이터, 응답 코드, 크래시 여부, CPU·메모리 사용량이 포함된다. Fig 3은 Elastic Stack 기반 중앙 로그 서버로 수집하는 것을 표시하고 있다.




[Fig. 3] Log collection and analysis pipeline based on Elastic Stack

### 3.5 결과 분석 계획

실험 후 수집된 로그와 결과 데이터는 세 가지 보안 위협 범주(서비스 거부 DoS, 무결성 훼손, 정보 유출)로 분류된다. DoS는 응답 지연 또는 서버 중단으로 정의하고, 무결성 훼손은 요청·응답 불일치나 비정상 데이터 변형이 발생한 경우로 판정한다. 정보 유출은 비정상적으로 내부 구조나 스택 정보가 노출된 경우에 해당한다. 본 연구에서는 '보안 취약점'을 실험 로그상에서 재현 가능하고 의미 있는 이상 동작으로 정의한다. 판정 기준은 세 가지로 구분된다. (i) 서비스 거부(DoS): 동일 세션 또는 연속적 케이스에서 타깃이 응답하지 않거나 연결이 중단되는 경우(타임아웃/프로세스 종료 등). (ii) 무결성 훼손: 요청과 기대 응답 간에 반복적·재현 가능한 불일치가 발생하거나 데이터 변형이 관찰되는 경우. (iii) 정보 유출: 응답에 내부 경로, 예외 스택, 쿼리 구조 등 민감 정보가 포함되는 경우. 판정 시에는 응답 코드·응답 길이, 에러 메시지, 재현성(동일 입력 반복 시 동일 현상 발생 여부)을 정량 지표로 활용한다.

## 4. 결과

본 장에서는 제안된 MCP 기반 퍼징 기법의 효과성을 검증하기 위해 FileSystem MCP(fuzzing target: fs.list, fs.read, fs.write)와 SQLite MCP(fuzzing target: db.query) 두 가지 환경에서 30분간 수행한 실험 결과를 비교·분석하였다. 모든 실험은 동일한 조건에서 수행되었으며, 결과 데이터는 CSV 로그 파일로 수집·정제하여 표와 그림으로 표시하고 설명하였다.

### 4.1 실험 개요

두 환경에서의 퍼징은 동일한 boofuzz 기반 퍼징 프레임워크를 사용하였으며, 타겟 MCP 서버와 TCP 브리지 환경을 구축한 후, 각 실험을 30분간 수행하였다. 주요 기록 지표는 (1) 실행된 테스트 케이스 수, (2) payload 길이 분포, (3) 평균 payload 길이, (4) 응답률(response rate)이다.

### 4.2 정량적 비교 결과

Table 1은 FileSystem MCP와 SQLite MCP 실험의 요약 통계를 비교한 것이다. FileSystem MCP는 케이스 수는 적었으나 평균 payload 길이가 상대적으로 짧고 응답률은 낮았다. 반면 SQLite MCP는 케이스 수가 많으며 응답률이 비교적 높았다. 이는 파일 시스템 호출이 대용량 payload를 유발하는 반면, SQL 쿼리는 구조적 다양성에 따라 많은 수의 짧은 요청이 생성되었기 때문이다.

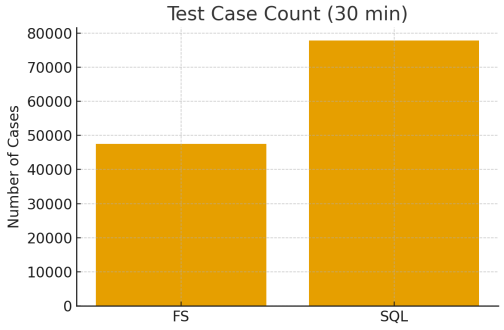
〈Table 1〉 Experimental Results of 30-Minute Fuzzing Sessions

| Experiment | FileSystem MCP | SQLite MCP |
|---|---|---|
| Total Cases | 47,520 | 77,758 |
| Unique payload Lengths | 62 | 406 |
| Avg payload Length | 176.5 | 383.2 |
| Max payload Length | 429 | 2,479 |

### 4.3 SQLite MCP와 Filesystem MCP의 퍼징 결과 비교

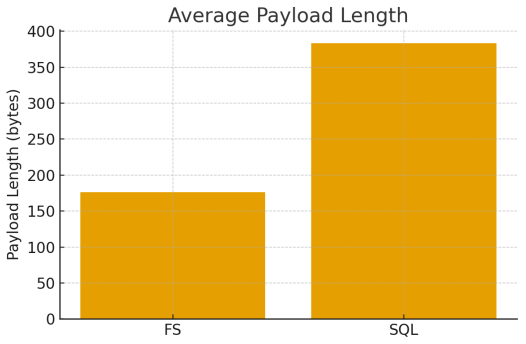
본 절에서는 두 가지 대상 모듈, 즉 SQLite MCP와 FileSystem MCP에 대해 동일한 퍼징 환경에서 30분간 수행한 실험 결과를 비교·분석하였다. 각 실험은 동일한 세션 환경에서 약 0.005초의 인터벌을 두고 테스트 케이스를 전송하였으며, Boofuzz 기반 퍼징 도구를 통해 자동화된 입력 생성을 수행하였다.

Fig. 4는 실험별 전체 테스트 케이스 수를 비교한 결과이다. FileSystem MCP 퍼징에서 총 47,520건의 테스트 케이스가 생성되었으며, SQLite MCP 퍼징에서는 77,758건이 수행되었다. SQLite MCP는 77,758건, FileSystem MCP는 47,520건을 처리해 약 1.6배 차이를 보였다.



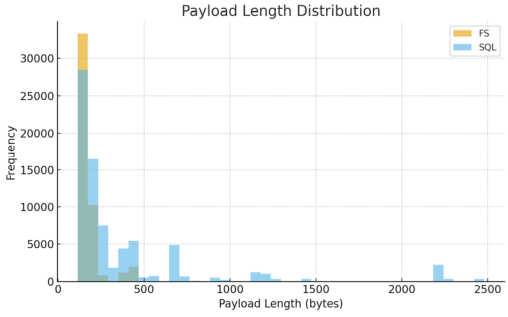

[Fig. 4] Number of Test Cases Executed in 30-Minute Fuzzing Sessions

Fig. 5는 payload 길이 분포를 히스토그램으로 나타낸 것이다. FS-MCP는 전반적으로 payload 크기가 큰 편이며, 500바이트 이상 대용량 입력이 상당수 생성되었고 최대 8KB 이상까지 확장되었다. 특히 fs.write 호출에서 파일 경로와 데이터 문자열이 동시에 길어지면서 장문의 입력이 다수 발생하였다. 반면, SQLite MCP는 쿼리 구문이 대부분 100~500바이트 범위에 집중되었고, 일부 특수 시드(연속 문자, 주석 확장 등)에 의해 1KB 이상의 긴 입력이 생성되었다. 따라서 FS-MCP는 대규모 버퍼 처리나 파일 시스템 관련 자원 소진 취약점을 더 잘 탐색할 수 있으며, SQL-MCP는 구문적 변형을 중심으로 폭넓은 탐색을 수행한다고 볼 수 있다.




[Fig. 5] Distribution of Payload Lengths in 30-Minute Fuzzing Experiments

Fig. 6은 평균 payload 길이를 비교한 결과이다. FileSystem MCP는 평균 약 177바이트, SQLite MCP는 평균 약 383바이트로, SQLite MCP의 payload 길이가 더 크고 다양성이 높은 것으로 나타났다. 이는 파일 시스템 호출이 JSON 구조 내 여러 인자를 포함해야 하기 때문으로, SQL의 단일 문자열 입력보다 데이터량이 많다. 따라서 FS-MCP는 입력 크기에 따른 메모리 처리 오류 탐색에서 강점을 지닌다.




[Fig. 6] Average payload Length Comparison between SQLite MCP and FileSystem MCP

Table 2에 제시된 바와 같이, SQLite MCP는 총 77,758건의 테스트 케이스를 처리한 반면 FileSystem MCP는 47,520건을 처리하였다. 이는 처리량에서 약 1.6배의 차이를 의미한다. 이러한 차이는 인터페이스 특성(예: SQL 쿼리의 단일 문자열 입력 vs. 파일 경로 및 I/O 상호작용)에서 기인했을 가능성이 있으나, 구체적인 원인 규명은 로그·프로파일링 기반의 추가 분석이 필요하다.

〈Table 2〉 Response rate comparison between SQLite MCP and FileSystem MCP

| | SQLite MCP | FileSystem MCP |
|---|---|---|
| Total Test Cases | 77,758 | 47,520 |
| Responses Received | 27,900 | 5,700 |
| Response Rate (%) | 36% | 12% |

실험 내용을 정리하면 SQLite MCP는 상대적으로 다양한 입력 변형에 대해 오류 메시지를 반환하는 경향이 강하여 대화형 특성을 보였다. 이는 퍼징 관점에서 탐색 공간을 넓히고, 추가적인 버그 탐지 가능성을 높이는 장점으로 작용한다. 반대로, FileSystem MCP는 입력이 무효하거나 권한 문제가 발생할 경우 응답을 회신하지 않는 경우가 많아, 퍼징 결과의 관찰 가능성이 낮아지고 분석 효율성이 저하되는 한계를 가진다. 또한 payload 길이 분포 측면에서도 FileSystem MCP는 평균 길이가 훨씬 길어, 입력 데이터의 크기 기반 예외 처리나 버퍼 관리 취약점 가능성이 상대적으로 더 클 수 있음을 시사한다.

두 실험의 비교를 통해 SQL 기반 MCP 인터페이스는 오류 처리 경로 중심의 취약점 탐색에 유리하며, FileSystem MCP는 대용량 입력과 경로 조작에 따른 안전성 검증이 핵심 과제임을 확인할 수 있다. 본 비교실험은 인터페이스 특성이 퍼징 결과에 큰 영향을 미침을 보여준다. SQL 기반 MCP는 단일 질의 구문 처리 과정에서 오류 응답이 잘 발생하여 상대적으로 높은 응답률과 즉각적인 오류 피드백을 제공하는 반면, FileSystem MCP는 경로 존재성·권한·정규화 및 대용량 I/O 등 운영체제 수준 요인이 개입하여 무응답·타임아웃이 빈번하게 발생하였다. 따라서 FileSystem MCP 퍼징에는 경로 정규화와 권한 검증, 대용량 I/O 모니터링이 병행되어야 한다.

## 5. 결과

본 연구는 Model Context Protocol(MCP) 환경에 퍼징 기법을 적용하여 보안 취약점 탐지 가능성을 실험적으로 검증하였다. 실험 결과, SQL 기반 MCP는 FileSystem MCP보다 약 1.6배 많은 테스트 케이스를 처리하고 3배 이상 높은 응답률을 보여 입력 처리 안정성에서 우수함을 보였다. 반면, FileSystem MCP는 대용량 입력 및 경로 조작에 따른 무응답과 오류가 다수 발생하여, 파일 입출력 중심 인터페이스의 취약성이 드러났다. 이러한 결과는 MCP 모듈의 유형에 따라 탐지 가능한 취약점 특성이 달라짐을 보여주며, 향후 연구에서는 표준화된 테스트 환경을 기반으로 다양한 MCP 모듈에 대한 확장 실험과 기존 퍼징 기법과의 성능 비교가 필요하다.

## REFERENCES

[1] H.Karim, D.Gupta and S.Sitharaman, "Securing LLM Workloads With NIST AI RMF in the Internet of Robotic Things," IEEE Access, Vol.13, pp. 69631-69649, 2025.

[2] D.Rivkin, F.Hogan, A.Feriani, A.Konar, ASigal and X.Liu, "AIoT Smart Home via Autonomous LLM Agents," IEEE Internet of Things Journal, Vol.12, No.3, pp.2458-2472, 2025.

[3] Anthropic, Model Context Protocol[Internet], https://github.com/modelcontextprotocol

[4] W.C.Jung, "Study on Automated Testing Methods Using AI MCP: Focused on Implementation and Application," Journal of Convergence Science, Technology, and Society, Vol.4, No.1, pp.63-69, 2025.

[5] A.Ehtesham, A.Singh and S.Kumar, "Enhancing Clinical Decision Support and EHR Insights through LLMs and the Model Context Protocol: An Open-Source MCP-FHIR Framework," 2025 IEEE World AI IoT Congress (AIIoT), pp.205-211, 2025.

[6] D.Cotroneo, R.D.Luca and P.Liguori, "DeVAIC: A tool for security assessment of AI-generated code," Information and Software Technology, Vol.177, pp.107572, 2025.

[7] C.G.Yi and Y.G.Kim, "Security Testing for Naval Ship Combat System Software," IEEE Access, Vol.9, pp.66839-66851, 2021.

[8] X.Zhang, C.Zhang, X.Li, Z.Du, B.Mao, Y.Li, Y.Zheng, Y.Li, L.Pan, Y.Liu, and R. Deng. "A Survey of Protocol Fuzzing," ACM Computing Surveys, Vol.57, No.2, pp.1-36, 2024.

[9] Z.Liu, P.Qian, J.Yang, L.Liu, X.Xu and Q.He, "Rethinking Smart Contract Fuzzing: Fuzzing With Invocation Ordering and Important Branch Revisiting," IEEE Transactions on Information Forensics and Security, Vol.18, pp.1237-1251, 2023.

[10] Y.Li, S.Ji, C.Lyu, Y.Chen, J.Chen and Q.Gu, "V-Fuzz: Vulnerability Prediction-Assisted Evolutionary Fuzzing for Binary Programs," IEEE Transactions on Cybernetics, Vol.52, No.5, pp. 3745-3756, 2022.

[11] M.Zalewski, "American Fuzzy Lop: Fuzzing for Fun and Profit," Proceedings of USENIX Security Symposium, pp. 1-8, 2014.

[12] Y.Yu, Z.Chen, S.Gan and X.Wang, "SGPFuzzer: A State-Driven Smart Graybox Protocol Fuzzer for

Network Protocol Implementations," IEEE Access, Vol.8, pp.198668-198678, 2020.

[13] D.Kim and T.Shon, "WinEco: Semi-automatic Harness Generation based on Windows Time Travel Debugging," Journal of Digital Contents Society, Vol.25, No.7, pp.1873-1881, 2024.

[14] J.Wang, Y.Huang, C.Chen, Z.Liu, S.Wang and Q.Wang, "Software Testing With Large Language Models: Survey, Landscape, and Vision," IEEE Transactions on Software Engineering, Vol.50, No.4, pp.911-936, 2024.

[15] R.Kande, H.Pearce, B.Tan, B.Dolan-Gavitt, S.Thakur and R.Karri, "(Security) Assertions by Large Language Models," IEEE Transactions on Information Forensics and Security, Vol.19, pp. 4374-4389, 2024.

박 정 규(Jung Kyu Park) [종신회원]

■ 2013년 8월 : 홍익대학교 컴퓨터공학과 (공학박사)
■ 2018년 3월 ~ 2024년 8월 : 창신대학교 컴퓨터공학과 교수
■ 2024년 9월 ~ 현재 : 대진대학교 AI융합학부 컴퓨터공학전공 교수

〈관심분야〉
Data storage, Robotics, System software, Embedded system

백 영 미(Youngmi Baek) [정회원]

■ 2015년 2월 : 경북대학교 컴퓨터공학과 (공학박사)
■ 2015년 7월 ~ 2017년 1월 : 대구경북과학기술원 CPS 글로벌센터 연구원
■ 2017년 2월 ~ 2019년 12월 : 대구경북과학기술원 정보통신융합전공 연구교수
■ 2020년 3월 ~ 2025년 2월 : 창신대학교 스마트융합공학부 컴퓨터공학전공 교수
■ 2025년 3월 ~ 현재 : 창신대학교 스마트팩토리학부 교수

〈관심분야〉
System modeling, Network security within automotive cyber-physical systems, Autonomous manufacturing