대규모 언어모델 기반 퍼징 연구 동향

김유흔[1], 오현영[2*]
[1]가천대학교 AI 소프트웨어학부 석사과정
[2]가천대학교 AI 소프트웨어학부 교수

shelly7610@gachon.ac.kr, hyoh@gachon.ac.kr

Trends in Large Language Model-Based Fuzzing Research

Yu-heun Kim, Hyun-young Oh
Dept. of AI Software, Gachon University

요 약

본 논문은 소프트웨어 보안 취약점 탐지에서 널리 활용되는 퍼징의 개념과 분류를 개관하고, 전통적인 퍼징의 핵심 한계를 정리한다. 최근 대규모 언어모델(LLM)이 방대한 코드 데이터 학습을 통해 문법, API 패턴, 일반적 소프트웨어 구조에 대한 내재 지식을 보유하게 됨에 따라 이러한 한계를 완화할 새로운 가능성이 제시되고 있으며, 본 논문은 LLM 을 퍼징에 활용한 최신 연구들을 수집해 각 도메인에서 테스트케이스 품질과 탐색 효율을 어떻게 개선하려는 지 분석한다.

1. 서론

소프트웨어 보안 취약점은 현대 사이버 보안 위협의 근본 원인 중 하나로, 다양한 취약점 탐지 기법이 연구되어 왔으며, 그 중에서도 퍼징(Fuzzing)은 가장 널리 사용되는 자동화된 취약점 발견 기법이다. 다른 취약점 탐지 기법과는 다르게 실제 실행 기반의 높은 정확도와 소스 코드 의존성이 낮은 확장성, 그리고 비교적 간단한 도입을 통해 산업계에서 가장 효과적인 취약점 탐지 기법으로 자리잡았다.

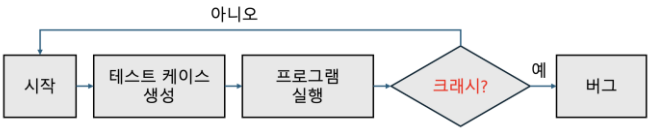
AFL(American Fuzzy Lop)로 대표되는 현대 커버리지 기반 퍼징의 등장으로 퍼징 기법은 크게 발전했지만, 여전히 핵심적인 한계들이 존재한다. 퍼징의 효율성은 본질적으로 어떤 테스트케이스를 어떻게 만드는지에 달려 있으며, 이는 비효율적인 시드 변이, 낮은 코드 커버리지, 그리고 생성된 입력의 유효성 검사 실패라는 세 가지 주요 도전 과제로 요약할 수 있다.

최근 GPT-4 와 같은 대규모 언어모델(Large Language Model, LLM)의 급속한 발전은 소프트웨어 테스팅 분야에 새로운 가능성을 열고 있다. LLM 은 방대한 코드 데이터로 학습되어 프로그래밍 언어의 문법, API 사용 패턴, 그리고 일반적인 소프트웨어 구조에 대한 내재적 지식을 보유하고 있다. 이러한 특성은 퍼징의 핵심 도전과제들을 해결하는 데 새로운 접근법을 제공한다. 본 논문은 퍼징에 LLM 을 활용한 최근 연구들을 살펴보며 LLM 이 각 도메인에서 테스트케이스 품질과 탐색 효율을 어떻게 개선하는지 분석한다.

2. 퍼징

2.1 퍼징 개요

퍼징은 대상 프로그램에 정상적이거나 비정상적인 대량의 입력을 자동으로 생성하여 공급하고, 실행 상태를 모니터링하여 예외나 크래시를 탐지함으로써 잠재적 취약점을 발견하는 기법이다. (그림 1)은 전통적인 퍼징[1]의 작동 방식이다.



(그림 1) 퍼징의 작동 방식.

2.2 퍼징의 주요 도전과제

전통적인 퍼저들은 무작위 기반 퍼징 전략을 활용하는데, 프로그램 분석 기법의 한계로 인해 퍼저들이 실용적이지 못한 상황이 나타난다. 따라서 퍼징 테스트는 여전히 많은 도전과제에 직면하고 있다.

2.3.1. 시드 변이 문제

변이 기반 생성 전략은 편의성과 쉬운 설정으로 인

* 교신저자

해 최신 퍼저들에서 널리 사용되지만, 더 많은 프로그램 경로를 커버하고 버그를 더 쉽게 트리거할 수 있는 테스트케이스를 변이하고 생성하는 방법은 핵심 과제다. 구체적으로 변이 기반 퍼저는 변이 시 두 가지 질문에 답해야 한다: (1) 어디를 변이할 것인가, (2) 어떻게 변이할 것인가. 소수의 핵심 위치에서의 변이만이 실행의 제어 흐름에 영향을 미치므로, 테스트케이스에서 이러한 핵심 위치를 찾는 것이 매우 중요하다. 또한 퍼저가 핵심 위치를 변이시키는 방법, 즉 프로그램의 흥미로운 경로로 테스트를 유도할 수 있는 값을 결정하는 방법도 또 다른 문제다. 테스트케이스의 무작위 변이는 테스트 자원의 심각한 낭비를 초래하며, 더 나은 변이 전략이 퍼징의 효율성을 크게 개선할 수 있다.

2.3.2. 낮은 코드 커버리지 문제

더 높은 코드 커버리지는 프로그램 실행 상태의 더 높은 커버리지를 나타내며, 더 철저한 테스트를 의미한다. 그러나 대부분의 테스트케이스는 동일한 소수의 경로만 커버하는 반면, 대부분의 코드는 도달할 수 없는 상태다. 결과적으로 대량의 테스트케이스 생성과 테스트 자원 투입만으로 높은 커버리지를 달성하는 것은 현명한 선택이 아니다.

2.3.3. 유효성 검사 통과 문제

프로그램들은 파싱과 처리 전에 종종 입력을 검증한다. 검증은 컴퓨팅 자원을 절약하고 유효하지 않은 입력과 악의적으로 구성된 입력으로 인한 손상으로부터 프로그램을 보호하는 가드 역할을 한다. 유효하지 않은 테스트케이스는 항상 무시되거나 폐기된다. 매직 넘버, 매직 문자열, 버전 번호 확인, 체크섬은 프로그램에서 사용되는 일반적인 검증 방법들이다. 블랙박스와 그레이박스 퍼저가 생성한 테스트케이스들은 무작위 생성 전략으로 인해 검증을 통과하기 어려워 상당히 비효율적인 퍼징을 초래한다. 따라서 검증을 통과하는 방법은 또 다른 핵심 도전과제다.

이러한 도전과제들에 대한 대응책으로 최근에는 LLM을 활용하는 연구가 활발히 진행되고 있다.

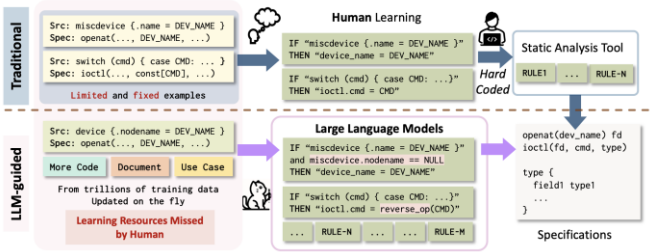
3. 각 도메인 별 LLM을 이용한 퍼징 연구 동향

기존의 퍼징 한계 극복을 위해 각 도메인 별로 LLM을 이용한 최근 연구들을 살펴보겠다.

3.1 커널 퍼징(Kernel Fuzzing)

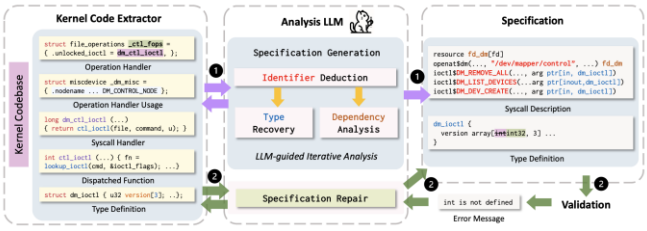
커널 퍼징은 운영체제 커널에서 발생할 수 있는 버그와 보안 취약점을 자동으로 탐지하는 기법이다. 커널은 모든 응용 프로그램의 기반이 되는 핵심 시스템이기 때문에, 커널의 취약점은 수십억 개의 장치와 사용자에게 영향을 미칠 수 있다. 커널 퍼징은 시스템 콜(system call) 시퀀스를 자동으로 생성하여 잠재적인 커널 버그나 취약점을 탐지하는 것을 목표로 한다.

효율적인 커널 퍼징을 위해서는 단순 무작위 생성이 아니라 문법·의미 제약을 만족하는 시스템 콜 시퀀스를 만들어 깊은 코드 경로를 탐색해야 한다. 이를 위해 다양한 커널 버그 탐지 방법 중 가장 대표적인 최신 커널 퍼저인 Syzkaller [2]는 syzlang 이라는 도메인 특화 언어를 제공하여 명세를 통해 유효 입력 공간을 정의한다. 그러나 이러한 명세의 수동 작성은 고난도 커널 지식과 많은 비용을 요구하므로, 규칙 기반 정적 분석과 최근의 LLM-보조 추론을 대비해 어떤 접근이 명세 품질과 구축 효율을 높이는지 보여줄 필요가 있다. 이러한 배경에서 다음 그림은 전통적 규칙 엔진과 LLM-유도 추론이 시스템 콜 명세를 도출하는 과정을 그림 3 에서 비교한다.



(그림 2) 커널 시스템 콜 명세 추론 파이프라인 비교.

KernelGPT [3]는 LLM 을 사용하여 더 나은 퍼징을 위한 시스템 콜 명세서를 자동으로 생성함으로써 유효하지 않은 시스템 콜 시퀀스를 제거하고 Syzkaller 의 탐색 공간을 줄여준다. LLM 은 사전 훈련 과정에서 방대한 커널 코드, 문서, 사용 사례를 학습했기 때문에 유효한 시스템 콜을 만들기 위한 필수 정보를 자동으로 추출할 수 있다. 특히 리눅스 커널의 광범위한 역사와 관련 논의, 튜토리얼, 문서가 이러한 훈련 데이터를 더욱 풍부하게 만든다.



(그림 3) KernelGPT 의 작동 방식.

KernelGPT 는 다음과 같은 방식으로 LLM 을 활용한다. (1) Operation Handler 식별: LLM 이 시스템 콜 소스 코드를 분석하여 operation handler 로부터 시스템 콜 handler 함수들(ioctl, setsockopt 등)을 식

별한다. (2) 3 단계 분석 파이프라인: 각 단계별로 특정 부분에만 집중하여 오해의 소지가 있는 정보를 최소화한다: 식별자 추론(Identifier Deduction) 단계는 시스템 콜의 식별자 값(디바이스 이름, 명령어 값 등)을 추론한다. 타입 복구(Type Recovery) 단계는 각 식별자 값에 대한 인수 타입 구조를 분석한다. 의존성 분석(Dependency Analysis) 단계는 시스템 콜 간의 의존성을 분석하여 반환값이 다른 operation handler 의 리소스가 될 수 있는지 식별한다. (3) 반복적 분석(Iterative Analysis): GPT-4 의 최대 128K 컨텍스트 크기로는 시스템 콜과 관련된 전체 소스 코드를 제공하기 부족하다. 또한 모든 코드가 식별자 값, 타입 구조, 의존성과 직접적으로 관련되어 있지 않으므로 LLM 이 현재 목표와 관련된 적절한 소스 코드를 식별하도록 한다.

		driver specification	socket specification
KernelGPT	#sys	482	304
	cov	138,992	154,187
Syzkaller	#sys	464	166
	cov	117,769	130,027

<표 1> KernelGPT 와 Syzkaller 간 시스템 콜 명세서 생성 비교. #sys 는 시스템 콜 개수, cov 는 평균 커버리지.

KernelGPT 는 532 개의 새로운 시스템 콜 명세서와 294 개의 타입 정의를 생성하여 기존 Syzkaller 대비 13.6% 증가했다. 또한 기존 Syzkaller 보다 4,750 개 더 많은 기본 블록을 커버했고, 24 개의 새로운 고유 버그를 발견하여 12 개가 수정되고 11 개가 CVE 번호를 할당받았다. KernelGPT 가 생성한 명세서들이 Syzkaller 개발팀의 요청에 따라 실제 Syzkaller 저장소에 병합됐다.

이러한 결과는 LLM 이 단순히 테스트 입력을 생성하는 것을 넘어서 퍼징 프레임워크의 구성 요소 자체를 합성함으로써 성숙한 퍼징 도구와 통합될 수 있는 새로운 차원을 열었음을 보여준다.

3.2 프로토콜 퍼징(Protocol Fuzzing)

프로토콜 퍼징은 인터넷에 노출되는 프로토콜 구현체에서 보안 취약점을 자동으로 탐지하는 기법이다. 프로토콜 구현은 다수의 메시지 유형과 상태를 가지는 상태 기계(stateful system)이며, 올바른 상태에서 올바른 형식의 메시지를 올바른 순서로 전송해야 다음 상태로 진행할 수 있다. 하지만 많은 프로토콜의 규격은 수백 페이지에 달하는 자연어 문서(RFC)로만 존재하며 기계가 읽을 수 있는 명세는 부족하다. 이 때문에 변이 기반 프로토콜 퍼저는 제한된 시드 메시지에 의존해 테스트케이스를 생성하는데, 이 시드의 다양성과 품질 부족으로 인해 상태 공간과 메시지 구조의 폭넓은 탐색에 한계가 존재한다.

AFLNET [4]은 대표적인 상태 기반 그레이박스 프로토콜 퍼저로, 입력 메시지 시퀀스의 상태와 코드 커버리지 피드백을 활용해 테스트 진행을 안내한다. AFLNET 은 프로토콜의 응답 코드를 이용해 상태를 추론하며, 기록된 시드 메시지를 변이해 새로운 테스트 케이스를 생성한다. 그러나 AFLNET 은 제한된 시드에 의존해 상태와 메시지 유형 다양성 확보가 어렵고, 메시지 구조 명세 부재로 구조 인식 변이가 부족하며, 알려지지 않은 상태 공간 탐색이 제한적이라는 한계를 가진다. 이로 인해 새로운 상태 전이나 깊은 상태 공간 탐색에 어려움이 있고, 커버리지 확장에 한계가 존재한다.

Algorithm 1: LLM-guided Protocol Fuzzing

Input: P_0: protocol implementation
Input: p: protocol name
Input: C: initial seed corpus
Input: T: total fuzzing time
Output: C: final seed queue
Output: C_X: crashing seeds

$P_f \leftarrow$ INSTRUMENT (P_0)
Grammar $G \leftarrow$ CHATGRAMMAR (p)
$C \leftarrow C \cup$ ENRICHCORPUS (C, p)
PlateauLen $\leftarrow 0$
StateMachine $S \leftarrow \emptyset$
repeat
  State $s \leftarrow$ CHOOSESTATE (S)
  Messages M, response $R \leftarrow$ CHOOSESEQUENCE (C, s)
  $\langle M_1, M_2, M_3 \rangle \leftarrow M$ *(i.e., split M in subsequences, s.t. M_1 is the message sequence to drive P_f to arrive at state s, and message M_2 is selected to be mutated).*
  for i from 1 to ASSIGNENERGY (M) **do**
    if *PlateauLen* < *MaxPlateau* **then**
      if UNIFORMRANDOM () $< \epsilon$ **then**
        $M_2' \leftarrow$ GRAMMARMUTATE (M_2, G)
        $M' \leftarrow \langle M_1, M_2', M_3 \rangle$
      else
        $M' \leftarrow \langle M_1,$ RANDMUTATE (M_2), $M_3 \rangle$
      end
    else
      $M_2' \leftarrow$ CHATNEXTMESSAGE (M_1, R)
      $M' \leftarrow \langle M_1, M_2', M_3 \rangle$
      PlateauLen $\leftarrow 0$
    end
    $R' \leftarrow$ SENDTOSERVER (P_f, M')
    if ISCRASHES (M', P_f) **then**
      $C_X \leftarrow C_X \cup \{M'\}$
      PlateauLen $\leftarrow 0$
    else if ISINTERESTING (M', P_f, S) **then**
      $C \leftarrow C \cup \{(M', R')\}$
      $S \leftarrow$ UPDATESTATEMACHINE (S, R')
      PlateauLen $\leftarrow 0$
    else
      PlateauLen $\leftarrow$ *PlateauLen* + 1
    end
  end
until timeout T reached or abort-signal

(그림 4) ChatAFL 의 작동 알고리즘.

ChatAFL [5]은 이러한 AFLNET 의 한계를 극복하기

위해 LLM 을 활용한 프로토콜 퍼징 기법이다. LLM 은 대규모 자연어 문서(RFC 포함)를 학습해 프로토콜 메시지 유형과 상태 기계에 대한 내재적 지식을 보유하므로, 프로토콜 퍼징의 핵심 도전과제를 다음과 같이 해결한다. (1) LLM 에게 프로토콜별 메시지의 기계 가독형 문법을 few-shot 프롬프팅으로 추출하도록 요청한다. 추출된 문법은 변이 시 메시지 내 변형 가능한 필드를 정확히 식별하여 유효하고 다양성 높은 구조 인식 변이를 가능하게 한다. (2) LLM 을 사용하여 기존 시드에 포함되지 않은 메시지 유형을 생성하고, 적절한 맥락과 순서에 맞게 삽입해 시드 집합을 다양하고 정확하게 확장한다. 이를 통해 초기 시드 의존성을 완화하고 상태 공간 탐색 범위를 넓힌다. (3) 퍼징 진행 중 일정 기간 동안 새롭고 흥미로운 테스트케이스가 생성되지 않는 '커버리지 플래토' 상황 시, LLM 은 최근 통신 히스토리를 바탕으로 다음에 보내야 할 메시지를 생성해 새로운 상태 전이로 진입할 수 있도록 퍼저를 지원한다.

ChatAFL 은 이와 같이 LLM 과 퍼저 간의 체계적 상호작용을 통해 프로토콜 메시지 구조를 파악하고, 상태 공간 탐색을 확장하며, 커버리지 정체를 극복하는 세 가지 핵심 전략을 통합하였다.

Subject	ChatAFL	AFLNet	Improv.
Live555	160.00	83.80	90.98%
ProFTPD	246.70	172.60	42.91%
PureFTPD	281.80	216.9	29.91%
Kamailio	130.00	99.90	30.14%
Exim	108.40	62.70	72.98%
forked-daapd	25.40	21.40	18.65%

<표 2> ChatAFL 과 AFLNet 상태 전이 커버리지 비교.

Subject	ChatAFL	AFLNet	Improv.
Live555	2,928.40	2860.20	2.38%
ProFTPD	5,143.30	4,763.00	7.99%
PureFTPD	1,134.30	1,056.30	7.39%
Kamailio	10,064.00	9,404.10	7.02%
Exim	3,789.40	3,647.60	3.89%
forked-daapd	2,364.80	2,227.10	6.18%

<표 3> ChatAFL 과 AFLNet 코드 커버리지 비교.

PROFUZZBENCH 벤치마크의 6 개 주요 텍스트 기반 프로토콜(예: RTSP, FTP, SIP, SMTP, DAAP)을 대상으로 ChatAFL 과 기존 AFLNET 를 비교 평가한 결과, ChatAFL 은 평균 47.6% 더 많은 상태 전이, 그리고 5.8% 더 높은 코드 커버리지를 달성하였다. 특히 Live555 항목에서는 상태 전이 수가 AFLNET 대비 91% 증가하였다. 또한 ChatAFL 은 기존 퍼저들이 발견하지 못한 9 건의 신규 취약점을 식별하였으며, 이 중 7 건이 개발자에 의해 확인되고 일부는 CVE 번호가 부여되어 패치되었다. 신규 취약점 발견에는 LLM 기반의 시드 풍부화로 기존 시드에 존재하지 않던 메시지 유형(예: PAUSE 메시지)이 포함된 점과, LLM 의 상태 전이 예측을 통한 커버리지 플래토 극복이 핵심적 역할을 하였다.

이러한 결과는 LLM 이 프로토콜의 자연어 명세로부터 유효한 메시지 문법과 상태 정보를 효과적으로 추출하여 퍼징의 자동화, 효율성 및 취약점 발견 능력을 획기적으로 향상시킬 수 있음을 입증한다.

항목	도메인	핵심 아이디어	한계
Syzkaller	커널 퍼징	syzlang로 시스템 콜 명세 정의 후 시퀀스 생성·커버리지 확장	명세 수동 작성 부담 큼
KernelGPT	LLM 기반 커널 퍼징	LLM이 식별자·타입·의존성 분석으로 syscall 명세 자동 생성 및 반복 피드백	LLM 품질·비용·재현성 관리 필요
AFLNet	프로토콜 퍼징	응답 코드로 상태 추론, 시드 변이로 시퀀스 생성	시드 다양성·구조 인식 변이 부족, 미지 상태 탐색 한계
ChatAFL	LLM 기반 프로토콜 퍼징	LLM으로 메시지 문법 추출·시드 유형 확장·플래토 시 다음 메시지 생성	LLM 의존 비용·오탐 제어 필요

<표 4> 도메인 별 퍼저 요약.

4. 결론

본 논문은 퍼징의 본질적 병목인 시드 변이의 비효율, 낮은 코드 커버리지, 유효성 검사 통과 실패를 정리하고, LLM 이 구조·문맥 인식 입력 생성과 도메인 지식 반영으로 테스트케이스 품질과 탐색 효율을 향상시킬 수 있음을 최근 연구 동향에 따라 분석했다. 커버리지 기반 퍼징(AFL 계열 등)의 강점을 유지하면서 LLM 을 결합하는 하이브리드 접근은 포맷 제약(매직 넘버·체크섬·버전 검사) 통과율을 높이고 반복 경로 편중을 완화해 더 깊은 코드 경로 도달 가능성을 높이는 방향으로 수렴한다. 다만 LLM 의 환각, 비용·지연, 재현성, 안전성(프롬프트 오염 등) 이슈가 남아 있어 표준 벤치마크와 정량적 평가 지표에 기반한 체계적 검증과 엔지니어링 가이드라인 수립이 병행되어야 한다.

사사문구

이 논문은 2025 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No. RS-2024-00337414, SW 공급망 운영환경에서 역공학 한계를 넘어서는 자동화된 마이크로 보안 패치 기술 개발)과 한국산업기술기획평가원의 지원(No. RS-2024-00406121, 자동차보안취약점기반위협분석시스템개발(R&D))을 받아 수행된 연구 결과임.

참고문헌

[1] Jun Li, Bodong Zhao and Chao Zhang. "Fuzzing: a survey", Cybersecurity, 2018.
[2] Syzkaller. https://github.com/google/syzkaller/.
[3] Chenyuan Yang, Zijie Zhao, and Lingming Zhang. "KernelGPT: Enhanced Kernel Fuzzing via Large Language Models", Association for Computing Machinery, 2025.
[4] V. Pham, M. Bohme, and A. Roychoudhury, "Aflnet: A greybox fuzzer for network protocols," IEEE International Conference on Software Testing, 2020.
[5] Ruijie Meng, Martin Mirchev, Marcel Bohme and Abhik Roychoudhury. "Large Language Model guided Protocol Fuzzing", Network and Distributed System Security, 2024.