# Trip Report for Attending ASE'26



2026. 10. 12. - 10. 15.

**최민석**

ASE (IEEE/ACM International Conference on Automated Software Engineering)는 소프트웨어 공학 학술대회로, 프로그램 분석, 설계, 테스트, 유지보수의 자동화가 이 학회의 주요 관심사이다. 올해 뮌헨에서 41번째 ASE가 개최되었고, SRC에 참가해 참석할 수 있게 되었다. 이번 ASE에는 우리 연구실에서 제출하여 Distinguished Paper Award까지 받은 김정웅, 조영민 학생의 MiniPoly 연구가 있었다.

**GraphQLer: Enhancing GraphQL Security with Context-Aware API Testing**

GraphQLer는 GraphQL의 접근 권한 관련 취약점을 테스트하고자 한다. GraphQL의 경우 REST API보다 이런 권한 관련 취약점은 자동으로 검사하기 어렵다고 한다. REST API에서는 `DELETE /notes/42` 처럼 HTTP 메서드와 경로가 동작과 대상에 대한 단서를 주는 반면, GraphQL에서는 생성, 수정, 삭제 연산이 모두 같은 엔드포인트를 통해 호출될 수 있다. 스키마에 mutation 표시를 달 수 있지만 그렇다고 해도 어느 동작을 수행하는지는 명시적으로 구분되지 않는다. 스키마에 조회에 ID가 필요하다고 표시할 수 있지만, 접근 제어를 시험하려면 어떤 객체를 먼저 만들어야 하는지까지 알려 주지는 않는다. 따라서 요청을 하나씩 검사하는 도구는 존재하지 않는 ID로 조회를 반복하면서, 정작 다른 사용자의 비공개 객체를 읽을 수 있는지는 확인하지 못할 수 있다.

이를 위해, GraphQLer는 스키마에서 연산의 이름, 설명, 입출력 타입을 읽어 각 연산의 역할을 추론하고, 한 연산의 결과를 다음 연산의 입력으로 사용할 수 있는 관계를 그래프로 만든다. 그런 다음 취약점의 발생 조건에 맞는 호출 순서를 생성하고 실제 서버 응답을 후속 요청에 재사용한다. 예를 들어 Alice가 비공개 데이터를 만들면 서버가 반환한 ID를 저장하고, Bob의 계정으로 같은 ID를 조회한다. 이렇게 하면 객체가 실제로 존재한다는 조건을 갖춘 상태에서 소유자와 요청자가 다를 때의 동작을 관찰할 수 있다.

이 연구를 보면서 JavaScript 엔진 퍼저인 CodeAlchemist가 떠올랐다. 문법적으로 올바른 코드 조각도 앞에서 필요한 타입의 값을 만들어 두지 않으면 실행 도중 예외가 나서 검사하려던 연산에 도달하지 못한다. CodeAlchemist는 조각 사이의 값과 타입을 고려해 코드를 조립했고, GraphQLer의 (거의 직접적인) 선행연구인 RESTler는 비슷한 문제를 REST API에서 다뤘다. GraphQLer는 비슷한 접근 방식을 스키마의 표현력이 약한 GraphQL에 적용하면서 스키마의 정보 공백을 LLM을 이용해 채우는 방식을 택했다. 이미 존재하던 접근 방식을 더 넓은 영역에 적용하면서, 그로 인해 생기는 문제는 LLM을 이용한다는 점에서 이전에 ITree 세미나에서 허충길 교수님이 “지능도 도구로 쓸 수 있으니 앞으로의 연구는 당연히 이걸 이용해야 한다” 같은 말씀을 하신게 떠올랐다. 좋은 도메인을 찾는게 더 중요한 시기인거 같다.

**When Compression Becomes an Attack Surface: Black-Box Attacks on Prompt-Compressed LLM Agents**

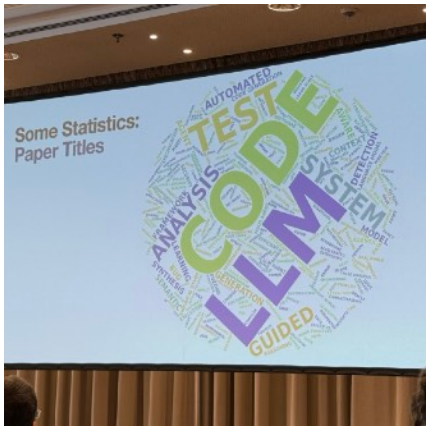
LLM 에이전트는 비용과 응답 시간을 줄이기 위해 프롬프트를 압축한다. 프롬프트에는 시스템 프롬프트, 사용자 프롬프트, 외부 자료 등이 모두 포함된다. 이 연구가 주목한 문제점은, 시스템 프롬프트와 외부 입력이 같은 토큰 예산 안에서 모두 처리된다는 점이다. ‘작업 공간 밖의 파일을 읽지 말라’는 시스템 규칙이 있어도, 사용자가 요청 끝에 짧은 문자열을 붙이면 압축기가 보존할 단어의 우선순위가 바뀌어 금지 규칙을 누락할 수 있다. 연구진은 이 취약성을 시험하는 두 단계 공격 COMA를 제안한다. 공격자는 피해 시스템의 비공개 지시나 실제 압축 결과를 볼 수 없으므로, 자신이 준비한 대리 압축기와 모델에서 먼저 ‘어떤 단어나 근거가 빠지면 행동이 달라지는가’를 찾는다. 그런 다음 외부 입력만 수정해서 그 누락이 일어나게 하는 문자열을 탐색하고 실제 에이전트에서 효과를 확인한다.

내가 그동안 접한 LLM 에이전트 보안 논의는 주로 외부에서 들어온 지시를 모델이 따르는 프롬프트 주입에 관한 것이었다. 반면 프롬프트 압축은 토큰 비용이나 캐시 효율의 문제로 생각해 왔다. 그래서 이 발표

에서 문제를 발견한 관점이 흥미로웠다. 외부 프롬프트 탓에 원래의 금지 지시가 압축 과정에서 사라질 수 있다면, 공격 문구가 모델까지 전달될 필요조차 없다. 설명을 듣고 나면 단순해 보이지만, 효율을 위해 도입한 컨텍스트 압축이라는 요소를 보고 그런 질문을 떠올렸다는 점이 흥미로웠다.
LLM 에이전트 테스팅에 대해서도 생각이 조금 달라졌다. 지금까지 LLM 에이전트를 테스팅한다는게, 이미 GPU, TPU 위에서 수없이 조절된 "파라미터"를 테스팅하는 것 이상의 어떤 의미가 있을까 싶었는데, 이 연구에서는 "파라미터"가 일종의 부품인, 거대한 프로그램으로써의 LLM 에이전트의 전체 작동 방식을 테스팅한다는 점에서 여전히 소프트웨어 테스팅의 중요한 문제로 보였다.
최근에는 연구뿐 아니라 X등 개발자들 커뮤니티에서도 다양한 컨텍스트 압축 기법을 접하게 된다. 이 발표를 보면서 압축률이나 평균 과제 점수와 함께, 어떤 조건에서 어떤 정보가 보존되는지도 평가할 필요가 있다고 느꼈다. 예를 들어 토큰 한도를 고정한 채 외부 문서에 과제와 무관한 내용을 덧붙여 보고, 에이전트가 여전히 파일 접근 금지 규칙을 지키는지, 문서 속 예외 조항을 적용해야만 맞힐 수 있는 질문에도 올바르게 답하는지를 따로 살펴볼 수 있다. 기존 벤치마크가 이런 상황을 어디까지 다루는지 먼저 확인해 보고, 빠진 부분이 있다면 압축 과정에서 어떤 정보가 사라질 때 실제 행동까지 달라지는지 비교할 수 있는 평가 과제를 만들 수도 있을 것 같다.

**Beyond Benchmarks: A Case Study of LLM-Generated Verus Specification Failures on Asterinas Vostd**

연구자들은, "LLM이 작성한 명세를 검증하다가 오류가 났을 때, 검증기의 오류 메시지만 보고 어디를 고쳐야 할지 알 수 있을까?"에 대한 질문을 던진다."이 함수를 호출하기 전에 무엇이 성립해야 하고, 실행이 끝나면 무엇이 보장되는가"를 LLM이 명세로 적는다. 연구자의 주장에 따르면 기존 연구는 주로 함수 하나를 독립적으로 다뤘으며, 이에 반해 연구진이 조사한 OS 커널 프로젝트 Asterinas Vostd에서는 한 함수의 명세를 다른 함수의 동작을 증명하는 데 사용한다. 그래서 첫 번째 함수의 명세에 필요한 내용이 빠지면, 오류는 나중에 두 번째 함수를 증명할 때 드러나게 될 수도 있다.
처음에는 형식 명세를 다룬다는 점에서 낯설고 복잡하게 느꼈지만, 오류가 드러난 곳과 실제로 고쳐야 할 곳이 다를 수 있다는 점은 매우 일반적인 일이고, 타입 오류를 해결할 때 흔히 경험하는 일이기도 하다. 예를 들어 큰 TypeScript 프로젝트에서 함수의 입력 타입에 따라 반환 타입이 달라지는데도 반환 타입을 넓은 타입으로만 선언하면, 호출하는 쪽에서는 결과가 어느 타입인지 확인할 수 없다. 오류 메시지는 그 결과를 사용하는 줄에 표시되지만, 정작 필요한 수정은 원래 함수로 돌아가 타입 매개변수나 오버로드로 입력과 출력의 관계를 더 정확하게 표현하는 일일 수 있다. 이 사례를 떠올리니, 큰 Rust 프로젝트에서 LLM이 타입 오류를 고치는 과정도 살펴보고 싶어졌다. 어떤 경우에 LLM이 오류가 표시된 코드만 반복해서 수정하고, 어떤 단서가 주어졌을 때 함수의 타입이나 호출 관계까지 거슬러 올라가 원인을 찾는지 관찰해 보면 흥미로울 것 같다.
이 발표는 최근 우리 연구실에서 하준서 학생과 교수님께서 시도하고 계신 ECMA-262 자동증명과도 맞닿아 있다고 느꼈다. 발표에서 사용한 Verus는 Lean과 달리 SMT 솔버를 쓴다는 차이가 있긴 하지만, 앞서 타입 문제처럼 이 문제 자체가 일반적인 문제이기도 하다고 생각한다. 특히 ECMA-262는 매우 크고 복잡하기 때문에, 증명이 멈춘 지점에 중간 증명 단계 하나를 보태면 충분할 때도 있고, 앞서 호출한 연산의 명세에 필요한 보장이 빠져 있어 그 명세부터 고쳐야 할 때도 있을 것이다. 실제로 증명이 막힌 자리와 결국 무엇을 고쳐서 증명을 마쳤는지를 짝지어 기록해 보면, 언제 증명을 더 진행하고 언제 앞선 명세를 다시 검토해야 하는지 판단하는 일 자체가 하나의 연구 주제가 될 수 있겠다는 생각이 들었다.




ASE'26에 제출된
논문 제목 통계



학회에서 제공된 간식



뮌헨 시청 청사