코퍼스 전이를 통한 상용 소프트웨어에 대한 바이너리 전용 퍼징 성능 향상

이동하
가천대학교 (학부생)

전승호*
*가천대학교

Enhancing Binary-only Fuzzing for Commercial Software via Corpus Transfer

DongHa Lee
Gachon University
(Undergraduate student)

Seungho Jeon*
Gachon University

요 약

많은 기업들이 비용 절감을 위해 오픈소스를 활용하지만, 무분별한 재사용은 취약점을 유발하고 공급망 보안에 위협이 된다. 퍼징은 효과적인 취약점 탐지 기법이지만, 상용 소프트웨어처럼 소스코드가 없는 경우 바이너리 전용 퍼징의 성능은 제한적이다. 본 논문은 코퍼스 전이를 통해 이 한계를 개선하고자 한다. 동일한 파일 형식을 처리하는 오픈소스를 퍼징해 생성한 코퍼스를 상용 소프트웨어 퍼징에 활용한 결과, 커버리지 향상과 새로운 취약점 발견에 효과가 있음을 확인했다.

I. 서론

소프트웨어 개발 환경이 빠르게 변화하면서 많은 기업이 개발 효율성과 비용 절감을 위해 오픈소스 소프트웨어(Open Source Software, OSS)의 소스코드를 인용하거나 의존성으로 가져와 소프트웨어 개발에 활용하고 있다. 즉, 상용 소프트웨어의 일부가 OSS와 같거나 유사한 소스코드로 구성되고 있다. 그러나, 최근 무분별한 OSS 사용에 의한 소프트웨어 취약점 제보 사례가 증가하고 있으며, 이는 소프트웨어 공급망에 큰 위협이 될 수 있다.

퍼징(Fuzzing)은 무작위로 생성된 입력 값을 이용해 소프트웨어를 테스팅하는 기술로, 이러한 소프트웨어 취약점을 발견하기 위해 적극 활용되고 있다. 하지만, 일반적으로 퍼징은 상용 소프트웨어와 같이 소스코드 없이 테스팅해야하는 상황에서 그 성능이 제한적이라고 알려져 있다.

본 논문에서는, 소스코드 없이 소프트웨어를 테스팅해야 하는 바이너리 전용(Binary-only) 퍼징의 한계를 극복하기 위해, 코퍼스 전이(Corpus transfer)를 활용한 퍼징을 제안한다. 이는 같은 파일 유형을 처리하는 소프트웨어간 소스코드 구조가 유사하다는 점에 착안한다. 즉, 어떤 파일 유형을 처리하는 OSS를 퍼징하여 코퍼스를 수집하고, 이를 실제 테스팅 대상인 상용 소프트웨어 퍼징의 초기 코퍼스로 활용한다. 끝으로, 코퍼스 전이의 효과를 보이기 위한 일련의 실험을 실시했다. 그 결과, 퍼징 중 커버리지가 증가함을 확인했으며, 상용 소프트웨어에서 신규 취약점을 발견했다.

II. 배경 및 관련 연구

2.1 블랙박스 퍼징

최근 주류인 그레이박스(Grey-box) 퍼징은 소스코드를 활용하여 대상 소프트웨어를 효율

적으로 테스팅한다. 예를 들어, 대표적인 그레이박스 퍼저(Fuzzer)인 AFL은 소스코드를 컴파일할 때 계측(Instrument)를 삽입하여 입력값에 의한 커버리지 정보를 수집한다. 이를 통해 시드(Seed) 선택이나 돌연변이(Mutation)를 커버리지 향상에 유리한 방향으로 유도한다.

하지만, 상용 소프트웨어의 경우 소스코드에 접근할 수 없어 블랙박스 퍼징으로 테스팅해야 한다. 이 경우 퍼저가 커버리지 정보 없이 시드를 선택하거나 입력 값을 변형해야 하기 때문에 효율적으로 대상을 테스팅하기 어렵다. 따라서, 많은 연구자들이 블랙박스 퍼징의 성능을 개선하기 위해 테인트 분석이나 에뮬레이터와 같은 보조 도구가 활용될 수 있다.

2.2 관련 연구

본 연구는 바이너리 전용 퍼징의 성능을 개선하기 위해 실제 테스팅 대상 외 다른 소프트웨어의 사전 지식을 활용한다. 이와 유사한 연구로 TransferFuzz가 있다 [1]. TransferFuzz는 알려진 취약점을 포함한 프로그램에서 실행 정보를 수집하여 같은 코드를 재사용한 다른 프로그램에서 해당 취약점이 유효한지 검증한다.

이와 달리, 블랙박스 환경에서 퍼징의 성능을 보다 직접적으로 개선하려는 시도도 있다. ZAFL은 바이너리 프로그램만 주어졌을 때 보다 효율적인 퍼징을 수행하기 위해 제시되었다 [2]. 보다 구체적으로, ZAFL은 그레이박스 퍼징과 달리, 바이너리 프로그램에 계측을 강제로 삽입한다. 이를 통해 퍼징 중 커버리지 정보를 수집하고 대상을 더 효율적으로 테스팅한다.

III. 코퍼스 전이 기반 바이너리 전용 퍼징

이 장에서는 상용 소프트웨어에 대한 바이너리 전용 퍼징의 성능을 개선하기 위해 코퍼스 전이라는 기법을 제안한다. 이 방법은 Fig. 1과 같이 크게 두 단계로 구성된다.

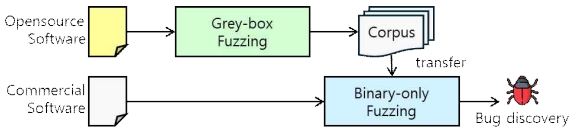

Figure 1. Overview of corpus transfer-based binary-only fuzzing

1) OSS 퍼징. 테스팅 대상인 상용 소프트웨어와 같은 유형의 파일을 처리하는 OSS를 선정한다. 그런 다음, OSS에 대해 그레이박스 퍼징을 수행하여 입력값 집합인 코퍼스를 수집한다. 그레이박스 퍼징은 코드에 삽입된 계측을 통해 코드의 다양한 실행 경로를 효율적으로 탐색할 수 있다는 장점을 가지고 있다. 이를 통해 얻어진 코퍼스는 무작위로 생성된 입력 값보다 취약점 발생 가능성이 높은 입력 값들로 구성된다.

2) 상용 소프트웨어 퍼징. 앞서 생성된 코퍼스를 상용 소프트웨어에 전이하여 퍼징을 수행한다. 이를 통해 바이너리만 주어지더라도 퍼저가 상용 소프트웨어에 대한 테스트 커버리지를 빠르게 확장하고, 나아가 취약점을 발견할 가능성을 높게 한다. 이는 다음과 같은 두 가지 통찰에 착안한다. 첫 번째, 상용 소프트웨어 개발 시 개발의 효율과 비용절감을 위해 OSS 코드를 재사용하는 경우가 많다. 두 번째, OSS 코드를 재사용하지 않더라도, 같은 유형의 파일을 처리하는 서로 다른 소프트웨어는 유사한 코드 패턴을 가질 가능성이 높다.

IV. 평가

4.1 실험 환경

이 실험에서, MacOS에서 실행되는 소프트웨어를 대상으로 퍼징을 진행했다. 보다 구체적으로, Apple ImageIO과 Adobe Premiere Pro를 대상으로 선택했다. 또한, 코퍼스를 수집하기 위한 OSS로 OpenEXR와 libheif를 선택했다.

모든 실험은 M2 CPU를 탑재한 Mac mini에서 진행됐다. OSS를 퍼징하기 위해 AFL++가 사용됐으며, MacOS 소프트웨어 퍼징에는 Jackalope과 계측을 위해 TinyInst가 사용됐다.

4.2 코퍼스 전이 효과에 대한 평가

실험 구성	설명
EXR-2h	OpenEXR의 테스트 이미지로 Apple ImageIO를 2시간 퍼징
HEIF-2h	libheif의 테스트 이미지로 Apple ImageIO를 2시간 퍼징
EXR-1h+ 1h	OpenEXR을 1시간 퍼징 후, 코퍼스 전이하여 Apple ImageIO를 1시간 퍼징
HEIF-1h+ 1h	libheif를 1시간 퍼징 후, 코퍼스 전이하여 Apple ImageIO를 1시간 퍼징

Table 1. Experimental setting on Apple ImageIO

Table 1은 Apple ImageIO에 대한 실험 구성이다. 모든 실험은 2시간 동안 진행됐다.

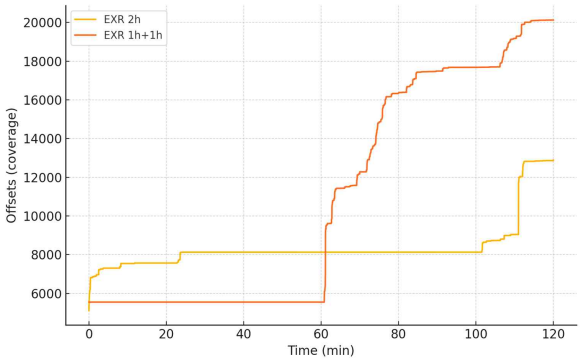

Figure 2. Coverage via corpus transfer from OpenEXR

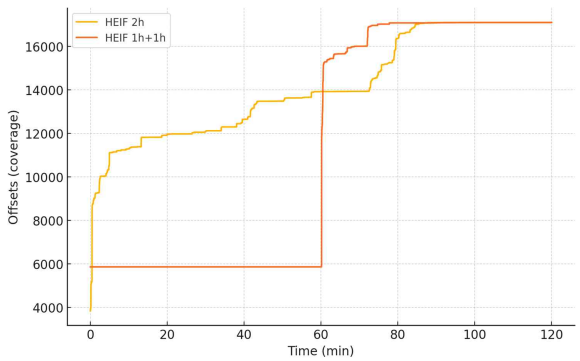

Figure 3. Coverage via corpus transfer from libheif

Fig 2와 Fig 3은 각각 OpenEXR과 libheif에서 생성된 코퍼스 Apple ImageIO로 전이했을 때의 커버리지 증가를 보여준다. 'EXR-2h'는 12,885의 누적 커버리지를 보인 반면, 'EXR-1h+ 1h'는 누적 커버리지 20,129로 약 58%의 향상을 보였다. 또한, 'HEIF-2h'는 17,093의 누적 커버리지를 보였으며, 'HEIF-1h+ 1h'는 누적 커버리지 17,005를 보였다.

실험 결과, OpenEXR의 코퍼스를 전이한 경우, Apple ImageIO에 대해 큰 커버리지 향상을 보였으나, libheif의 코퍼스에 대해서는 전이 효과를 보지 못했다. 이에 대해 심층분석 하여 다음과 같은 결론에 도달했다. 코퍼스 전이는 바이너리 전용 퍼징에서 효과적인 수단이 될 수 있으나, libheif와 같이 비교적 단순한 코드를 갖는 OSS로부터 수집된 코퍼스는 퍼징 성능에 유의미한 기여를 하지 않을 수 있다.

추가적으로, OpenEXR로부터 생성된 코퍼스를 Adobe Premier Pro에 전이하여 2달간 퍼징을 수행했다. 그 결과, 높은 커버리지 향상을 보였을 뿐만 아니라, 신규 취약점인 CVE-2024-20746을 발견했다.

V. 결론

본 논문은 코퍼스 전이를 활용한 바이너리 전용 퍼징 기법을 제안하였다. OSS와 상용 소프트웨어를 대상으로 한 실험을 통해, 코퍼스 전이가 퍼징 효과를 높이고 취약점 탐지에 유용함을 확인하였다. 실험에서는 상용 소프트웨어와 유사한 유형의 파일을 처리하는 다양한 OSS를 선정하여 전이 효과를 검증하였다. 향후 실험 대상과 범위를 확대하고, 전이 조건에 대한 보다 정량적인 분석을 통해 제안 기법의 적용 가능성을 더욱 심화시킬 예정이다.

[참고문헌]

[1] Li S et al., TransferFuzz: Fuzzing with Historical Trace for Verifying Propagated Vulnerability Code. arXiv preprint arXiv:2411.18347. 2024 Nov 27.

[2] S. Nagy et al., Breaking Through Binaries: Compiler-quality Instrumentation for Better Binary-only Fuzzing, Usenix Security Symposium, 2021.