지향성 퍼징 기술의 최신 동향

한국과학기술원 ▌김태은·허기홍*

1. 서 론

지향성 퍼징은 퍼징 기법의 한 종류로서 프로그램의 특정 지점에 도달하는 입력을 생성하는 것을 목표로 한다. 최근 몇년간 지향성 퍼징은 학계의 뜨거운 관심을 받고 있다. 그 이유는 바로 날이 갈수록 증가하는 프로그램의 규모와 복잡성에 있다. 거대한 프로그램 전체를 대상으로 일반적인 퍼징을 수행하는 것은 매우 어려워졌기 때문에 특정 지점에 집중하는 지향성 퍼징의 필요성이 대두된 것이다. 예를 들어 패치를 통해 프로그램 일부가 수정되었을 때, 혹은 정적 분석을 통해 발견된 알람을 검사할 때, 우리는 프로그램 전체보다는 주어진 프로그램 지점에만 관심을 가진다. 이 때, 일반적인 퍼징 기법을 사용한다면 전체 프로그램을 살펴보느라 정작 우리가 관심있는 프로그램 지점은 제대로 검사하지 못할 수 있다. 따라서 이러한 경우에는 주어진 프로그램 지점에 집중할 수 있는 지향성 퍼징이 더 합리적인 선택이 된다.

지향성 퍼징 기술은 2017년, AFLGo[1]를 필두로 본격적으로 활발한 연구가 진행되었다. 이러한 인기는 매년 세계 최고 수준의 보안 및 소프트웨어공학 학회에 지향성 퍼징 논문이 꾸준히 발표되고 있는 것으로 확인할 수 있다. 지향성 퍼징 기술이 뜨거운 관심을 받는 것에는 크게 두 가지 이유가 있다. 첫째로, 실용적인 필요가 분명하기 때문이다. 상술한 바와 같이 갈수록 프로그램이 복잡해지는 시대에서 원하는 지점에 집중하는 지향성 퍼징 기술이 필요한 상황은 반드시 발생한다. 둘째, 지향성 퍼징 기술은 그 자체로 연구자들에게 매력적인 분야이기 때문이다. 특정 프로그램 지점에 집중한다는 말은 탐색 공간이 줄어든다는 것을 의미하기도 한다. 따라서 기존에는 비용 문제로 엄두도 내지 못했던 다양한 기술들을 활용할 여지가 생긴다.

이 글에서는 먼저 퍼징의 역사와 지향성 퍼징 기술이 등장한 배경을 간략히 소개한다. 그 후, 지향성 퍼징의 기본 개념과 그 평가 방법, 그리고 다양한 지향성 퍼징 기술들을 소개한다.

2. 개 요

2.1 퍼징

지향성 퍼징에 대해 설명하기에 앞서 그 상위 개념인 퍼징에 대해서 간략히 정리할 필요가 있다. 퍼징은 자동으로 무수한 입력을 생성하여 프로그램을 테스트하는 기술이다. 특히 무수한 입력을 무작위성의 힘을 빌려 생성하기 때문에 그 중에는 개발자가 미처 예상하지 못한 입력도 있기 마련이고, 이를 통해 미처 몰랐던 결함을 발견할 수도 있다. 퍼징은 대상 프로그램에 대한 정보가 거의 없는 상황에서도 프로그램을 효과적으로 관통하는 입력을 만들어내는 등, 그 강력한 성능으로 인해 많은 사랑을 받아왔다.

가장 널리 알려진 대표적인 퍼징 도구는 단언컨대 AFL[2]일 것이다. AFL은 변형 기반 반투명(Mutation-Based Greybox) 퍼징 도구이다. 변형 기반 반투명 퍼징은 주어진 입력을 변형하여 새로운 입력을 만들고, 그 실행 결과를 기준으로 그것이 좋은 입력인지 아닌지 판단하는 기법이다. 이때, 반투명(Greybox)이라 함은 프로그램 실행을 자세히 들여다보지는 않되 그 결과로 나오는 실행 커버리지 정도의 정보는 확인한다는 것을 뜻한다. 참고로 프로그램 내부를 샅샅히 들여다보면 투명(Whitebox), 아예 들여다보지 않고 프로그램의 입력과 출력만 본다면 불투명(Blackbox) 퍼징이라 한다. 반투명 퍼징의 경우 좋은 입력인지 아닌지는 보통 새로운 커버리지를 달성하였는지에 따라 결정을 하며, 좋은 입력은 새로운 입력을 만들 재료로서 선택된다.

* 종신회원

† 이 연구는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2021R1C1C1003876, NRF-2021R1A5A1021944).

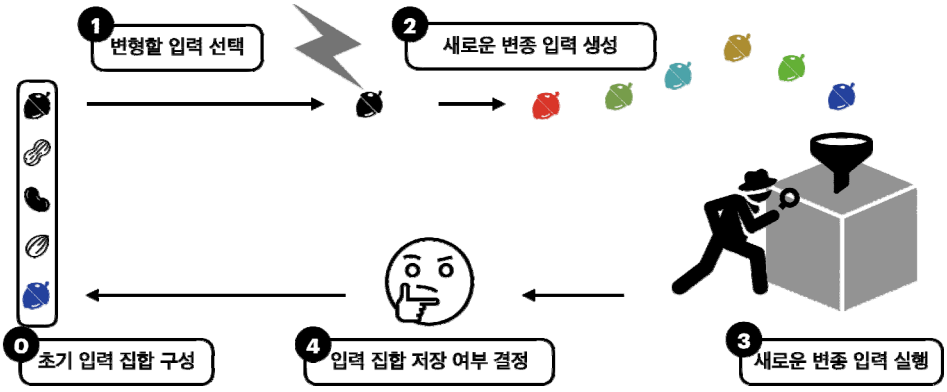

그림 1 반투명 퍼징의 작동 원리

즉, 그림 1에 나온 것처럼 다음과 같은 순서로 퍼징이 진행된다.

0. 초기 입력 집합 구성
1. 변형할 입력 선택
2. 새로운 변종 입력 생성
3. 새로운 변종 입력 실행
4. 커버리지를 바탕으로 입력 집합 저장 여부 결정
5. 1번부터 다시 시작.

2.2 지향성 퍼징

퍼징은 분명 효과적인 기술이지만, 프로그램의 규모가 커지고 자주 수정됨에 따라 전체 프로그램을 대상으로 퍼징을 수행하는 것은 어려워졌다. 자연스럽게 특정 지점에 집중하는 퍼징 기술의 필요성이 대두되었고, 이에 따라 지향성 퍼징 기술이 등장하게 되었다. 기존 퍼징, 즉 무지향성 퍼징으로는 불가능한 지향성 퍼징의 활용 방안은 다음과 같다.

1. 수정된 지점 검사: 프로그램 수정이 일어날 때 수정된 부분에만 집중하여 퍼징을 수행한다.
2. 의심 지점 검사: 정적 분석등의 도구를 통해 특정된 의심 지점을 집중적으로 검사한다.
3. 오류 재현: 사용자가 오류를 보고했지만, 개인정보가 담겨있어 오류 유발 입력을 얻기 어려운 경우, 오류 발생 지점에 집중하여 오류를 재현하는 입력을 찾는다.

초기의 지향성 퍼징은 KATCH[3]등의 도구처럼 기호실행(Symbolic Execution)을 활용한 방법들로 제안되었다. 그러나 투명 퍼징으로도 불리는 기호실행은 프로그램의 크기가 커질수록 속도가 많이 느려지기에 그다지 빛을 발하지 못했다. 따라서 실제로 실용적인 지향성 퍼징의 장이 열린 것은 2017년, 반투명 퍼징을 기반으로 한 AFLGo[1]의 등장 이후로 볼 수 있다. 지향성 반투명 퍼징 기술은 지향성 퍼징의 주류로 자리잡았으며 일반적으로 지향성 퍼징이라고 하면 지향성 반투명 퍼징을 의미하게 되었다. 따라서 본 글에서도 이와 같은 의미로 지향성 퍼징이라는 용어를 사용하며 지향성 반투명 퍼징을 집중해서 다룬다.

지금까지 다양한 지향성 퍼징 기술이 제안되어 왔지만 대부분의 기술 저변에 깔린 핵심 개념은 다음과 같이 정리할 수 있다. 바로 더 좋은 입력을 선정하는 것과 그렇게 선정된 입력을 활용하는 방법이다.

2.1.1 더 좋은 입력의 기준

대부분의 지향성 퍼징 기술은 각 입력마다 점수를 부여하고, 그 점수를 좋은 입력의 기준으로 삼는다. 이러한 점수는 다양한 방법으로 계산될 수 있는데, 대표적인 방법은 다음과 같다.

거리 점수: 실행 흐름 그래프 상의 거리를 바탕으로 점수를 계산하는 방식이다[1,4,5,6]. 먼저 각 프로그램 지점마다 목표 지점과의 거리를 계산하고, 각 입력이 실행한 프로그램 지점들의 거리를 합산하여 점수를 계산한다. 즉, 목표 지점에 가까운 프로그램 지점을 실행한 입력에게 더 높은 점수를 부여한다.

연관성 점수: 목표 지점과의 연관성을 바탕으로 점수를 계산하는 방식이다[7]. 목표 지점에서 사용되는 변수들에 관여하는 지점을 더 많이 실행한 입력에게 더 높은 점수를 부여한다.

유사도 점수: 목표한 프로그램 행동과 유사한 행동을 보이는 입력에게 더 높은 점수를 부여한다[8]. 예를 들어 특정 실행 흐름을 목표로 할 경우, 가장 유사한 실행 흐름을 보인 입력에게 더 높은 점수를 부여한다.

2.1.2 더 좋은 입력의 활용

더 좋은 입력을 찾았다면, 그 입력을 활용하는 방법에 대해서도 고민해야 한다. 가장 보편적인 활용 방법은 다음과 같다.

입력 선택: 변형하기 위한 입력을 선택할 때 더 좋은 입력을 우선적으로 선택한다 [7,8,10,13].

변종 생성: 더 좋은 입력의 경우, 더 많은 변종을 생성한다 [1,4,5,6,7,8,13].

3. 지향성 퍼징 연구 동향

지향성 퍼징은 2017년 등장한 AFLGo를 필두로 매년 보안과 소프트웨어 분야의 최고 수준 학회에 논문이 끊이지 않을 정도로 뜨거운 관심을 받아 왔다. 그만큼 다양한 기술들이 제안되었는데, 각각 나름의 방식으로 그림 1의 각 단계에서 지향성을 부여한다고 볼 수 있다. 본 장에서는 각 단계별로 다양한 기술들이 어떻게 지향성을 부여하는지 살펴본다.

3.1 초기 입력 집합 구성

초기에 어떤 입력을 사용할 것인가는 일반적으로 퍼징의 성능에 큰 영향을 미친다. 특히 지향성 퍼징의 경우 목표 지점에 도달할 가능성이 있는 입력을 초기 입력 집합에 포함하는 것이 중요하다. TargetFuzz[9]의 경우, 이 점에 주목하여 목표 지점에 특화된 초기 입력 집합을 구성하는 방법을 제안하였다.

3.2 변형할 입력 선택

변형할 입력의 순서, 혹은 빈도를 결정하는 것 또한 지향성 퍼징의 성능에 큰 영향을 미친다. 따라서 각 입력의 점수를 바탕으로 입력들을 정렬하여 점수가 좋은 입력이 더 우선적으로 사용되도록 하거나 [5,7,10], 입력의 사용 빈도를 조절하여 목표에 더 좋은 입력이 더 자주 사용되도록 유도하는 방식[8]으로 지향성을 부여한다.

3.3 새로운 변종 입력 생성

이 단계에서 지향성을 부여하는 방식은 크게 두 가지로 나뉜다. 하나는 변종의 생성 방식 자체에 지향성을 부여하는 방식이고, 다른 하나는 생성되는 변종의 수를 통해 지향성을 부여하는 방식이다.

3.3.1 목표 지향적 변형

무지향성 퍼징의 경우 무작위성에 크게 의존하는 변종 생성 방식을 사용한다. 따라서 입력의 어떤 영역을 변형할 지, 어떻게 변형할지 모두 무작위로 결정된다. 하지만 지향성 퍼징의 경우 보다 똑똑하게, 목표 지향적으로 변형을 수행할 수 있다[5,11]. 예를 들어, 동적 오염 분석 (Dynamic Taint Analysis)을 사용하면 입력의 어떤 영역이 프로그램의 어떤 지점에서 사용되었는지 알 수 있다. 이 정보를 바탕으로 목표 지점으로 가는 길목에서 사용된 입력의 영역을 파악하고 이를 집중적으로 변형하면 목표 지점에 도달할 가능성을 높일 수 있다. 즉, 변형할 입력의 영역을 특정함으로써 지향성을 부여하는 것이다.

3.3.2 변종 수 조절

무지향성 퍼징의 경우 생성할 변종의 수는 입력의 일반적인 특성에 따라 결정된다. 즉, 입력이 생성된 시점, 크기, 기존 변종들의 생존율 등에 따라 새로운 변종의 수가 결정된다. 하지만 지향성 퍼징의 경우 여기에 추가로 목표 지점에 도달할 가능성을 고려하여 변종의 수를 조절하게 된다. 거의 대부분의 현존하는 지향성 퍼징 도구가 이 방식을 사용하고 있으며 [1,4,5,6,7,8,13], 각각 목표 지점과의 거리, 연관성, 유사도 등을 기준으로 계산한 점수를 통해 생성할 변종의 수를 결정한다. 이 중 가장 보편적인 방식은 거리 점수를 사용하는 AFLGo의 방식이다.

3.4 새로운 변종 입력 실행

일반적으로 퍼징 과정에서 가장 많은 시간이 소요되는 것이 바로 변종 입력의 실행 단계이다. 하지만 지향성 퍼징의 경우, 목표 지점에 도달할 가능성이 없는 입력의 실행은 조기에 종료하여도 무방하다. Beacon[6]은 목표 지점에 대해 도달 가능성 분석과 선행 조건 분석(Precondition Analysis)을 수행한다. 그렇게 도달 조건을 분석한 후, 해당 조건을 프로그램 실행 중에 검사하여 도달 가능성이 없는 입력을 조기에 종료한다. 여기에서 절약한 시간은 목표 지점에 도달할 가능성이 있는 입력에 투자되어 퍼징에 지향성을 부여한다. SieveFuzz[13] 또한 유사한 방식으로 프로그램의 실행을 조기에 종료한다.

3.5 입력 생존 여부 결정

무지향성 퍼징에서 새로운 변종 입력은 새로운 커버리지를 달성한 경우에만 살아남아 입력 집합에 포함된다. 대부분의 지향성 퍼징 도구도 이 방식을 그대로 사용한다. 하지만 지향성 퍼징의 경우, 모든 지점으로부터 발생하는 커버리지 대신, 목표 지점과 연관된 지점의 커버리지 정보만 고려하는 것이 더욱 효율적일 것이다. DAFL[7]은 이 직관을 바탕으로 먼저 데이터 의존성 분석을 통해 목표 지점과 연관된 지점을 파악한다. 그 후, 목표 지점과 연관된 지점에서 새로운 커버리지를 달성한 경우에만 해당 입력을 유지한다. 그 결과 목표 지점과 연관이 없는 지점을 새로 탐색하는 입력은 모두 폐기되어 더욱 목표 지점에 집중할 수 있다.

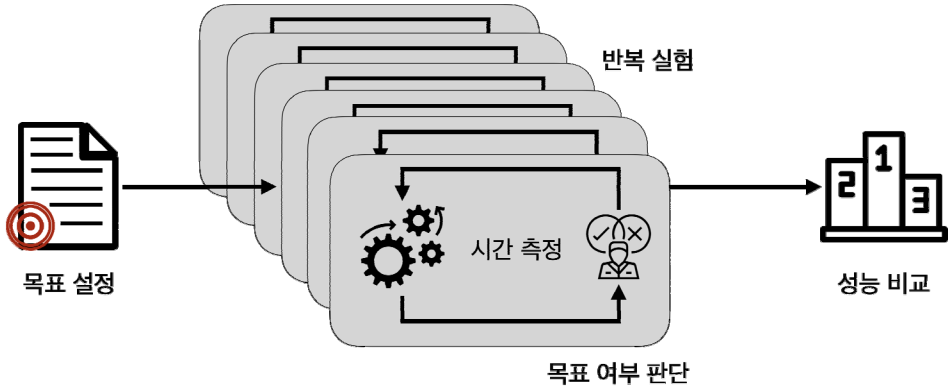

그림 2 결함 재현 실험의 개요

4. 지향성 퍼징 도구의 평가[2)]

지향성 퍼징의 동작 원리는 무지향성 퍼징과 비슷하지만, 그 평가 방식에는 매우 큰 차이가 있다. 무지향성 퍼징의 경우 보통 다음과 같은 두 지표[14]를 통해 그 프로그램 탐색능력을 평가한다: 1) 얼마나 많은 프로그램 지점을 탐색했는가? 2) 얼마나 많은 결함을 발견했는가?

하지만 지향성 퍼징의 경우, 주어진 목표에 얼마나 잘 집중하는지를 평가해야 한다. 따라서 많은 프로그램 지점을 탐색했다는 것은 오히려 지향성이 떨어진다는 것을 의미한다. 그렇다면 지향성 퍼징 도구의 평가는 어떻게 이루어져야 할까?

지향성 퍼징 도구의 가장 보편적인 평가 방식은 결함 재현 실험을 통한 평가이다. 결함 재현 실험은 알려진 결함을 얼마나 빨리 재현하는지를 평가하는 실험으로 그림 2와 같이 이루어진다.

먼저 목표 결함을 대표하는 프로그램 지점을 선정하여 지향성 퍼징 도구에 입력으로 전달하고, 주어진 시간동안 퍼징 도구를 실행한다. 그 후, 주어진 시간동안 퍼징 도구가 발견한 결함들이 목표한 결함인지 확인한다. 목표한 결함을 발견한 것이 확인되면 목표 결함을 재현하는데 걸린 시간을 기록한다. 이 때, 결과의 신뢰성을 높이기 위해 앞선 과정을 일정한 횟수만큼 반복한다. 마지막으로 두 도구의 성능을 비교하기 위해 결과의 중앙값을 비교하거나 보다 정교한 통계 검사를 한다.

본 장에서는 각 단계가 어떻게 이루어지는 지, 그리고 공정하고 일관된 평가를 위해 각 단계에서 주의해야 할 점들은 무엇인지 다룬다.

4.1 목표 선정

결함 재현 실험의 궁극적인 목적은 목표로 한 결함을 재현하는 것이다. 이 목표 결함으로는 보통 CVE (Common Vulnerabilities and Exposures) 번호가 발급된 결함이 사용된다. 그러나 대부분의 지향성 퍼징 도구는 구현 상 특정 프로그램 지점을 입력으로 받을 뿐, 결함 그 자체를 목표로 삼지 않는다. 따라서 실제로는 목표 결함을 잘 대표하는 프로그램 지점을 선정하여 지향성 퍼징 도구에 전달하게 된다.

하지만 하나의 CVE에 대해 어떤 목표 지점을 선정해야 하는지는 명확하게 정해져 있지 않다. 같은 CVE에 근본적인 원인이 다른 두 결함이 묶여있을 수 있고, 결함이 발생한 경로 중 어떤 지점이 결함의 맥락을 가장 잘 나타내는지도 명확하지 않다. 그 결과, 단순히 사용한 CVE 번호만을 공개하는 경우, 연구자에 따라 서로 다른 목표 지점을 선택할 수 있는데 이러한 경우 실험의 결과가 크게 달라지기도 한다. 따라서 여러 연구에 걸친 일관적인 평가를 위해서는 실제로 사용한 목표 지점까지도 공개하도록 주의를 기울여야 한다.

4.2 목표 결함 확인

아무리 지향성 퍼징 도구라 하더라도 항상 목표한 결함만을 발견하지는 않는다. 목표 지점까지 가는 길에 다른 결함이 존재할 수 있고, 무작위성에 의해 다른 결함이 발견될 수도 있다. 따라서 그 중 어떤 결함이 목표로 한 결함인지 확인해야 지향성 퍼징 도구의 성능을 정확히 평가할 수 있다.

주어진 시간동안 발견된 결함들이 목표 결함인지 확인하는 방법에는 크게 두 가지가 있다. 먼저 프로그램 소독기(Sanitizer)[12]를 사용하여 발견된 결함들을 검사하는 방법이 있다. 보통 프로그램 소독기는 발생한 결함의 종류와 스택 트레이스를 제공하는데, 이 정

2) 4장은 Kim et al., 2024 [17]의 내용을 기반으로 한다.

보가 목표 결함의 정보와 일치하는지를 확인하여 결함을 검사할 수 있다. 문제는 정보가 얼마나 일치해야 하는지에 대한 기준이 명확하지 않다는 것이다. 예를 들어 스택 트레이스가 모두 일치해야만 같은 결함인 경우가 있지만 결함 발생 지점만 일치해도 같은 결함인 경우도 있다. 또는 결함의 근본적인 원인이 어디에 있는지에 따라 결함 발생 지점이 다르더라도 같은 결함일 수도 있다.

또 다른 방법으로는 목표 결함을 수정하는 패치를 사용하는 방식이 있다. 예를 들어 특정 결함을 유발하던 입력이 패치된 프로그램에서 정상적으로 실행된다면 해당 결함을 목표 결함으로 판단하는 것이다. 얼핏 확실한 방법처럼 보이지만, 이 또한 몇 가지 문제점을 가지고 있다. 먼저 어떤 패치가 목표 결함을 완전히 수정했는지를 판단하는 것이 어렵다. 왜냐하면 여러 패치에 걸쳐 하나의 결함이 수정될 수 있고, 하나의 패치가 여러 결함을 수정할 수도 있기 때문이다. 또한 이중으로 결함이 존재하는 경우, 패치가 적용된 프로그램에서 기존 결함은 사라지지만 새로운 결함이 발생할 수 있다. 이 경우에는 단순히 패치된 프로그램에서의 결함 발생 여부만으로 목표 결함을 판단하는 것은 위험하다.

이렇듯 두 방법 모두 모호한 요소가 존재하기 때문에 일관된 평가를 위해 구체적인 목표 결함 검사 방법을 공개해야 한다. 더 나아가 애초에 검사 로직이 포함된 벤치마크를 사용하는 것이 바람직하다. 예를 들어 목표 결함 검사문(assertion)이 삽입되어 있는 프로그램[18], 혹은 단 하나의 결함만 존재하도록 합성된 프로그램[19]이 여기에 해당한다.

4.3 결함 발견 시간 측정

결함 발견 시간은 TTE(Time-To-Exploit)로 표기하며 대부분의 연구자들은 TTE를 산출할 때 퍼징 시간만 고려한다. 하지만 지향성 퍼징의 경우 유독 많은 도구들이 퍼징에 앞서 프로그램을 분석하고, 그 분석 결과를 이용하여 퍼징을 진행한다. 보통 분석 시간은 보고조차 되지 않지만, 실제로는 예상외로 많은 시간이 소모되며 분석 시간이 퍼징 시간보다 더 오래 걸리는 경우도 있다. 또한 프로그램이 수정될 때마다 다시 분석을 해야 하기 때문에 지속적인 개발 환경 등에서 지향성 퍼징을 활용하려면 이러한 분석 시간을 더욱 무시할 수 없다. 따라서 공정하고 실용적인 평가를 위해 각 지향성 퍼징 도구들의 프로그램 분석 시간도 고려해야 하며 아예 TTE에 포함시키는 것도 바람직하다.

4.4 실험 반복 횟수

퍼징은 무작위성을 이용하여 입력을 생성하기 때문에 항상 조금씩 다른 결과가 나오고 이는 지향성 퍼징도 마찬가지이다. 따라서 퍼징 도구를 평가할 때는 실험을 여러 번 반복하여 그 결과의 신뢰성을 높인다.

무지향성 퍼징의 경우 매번 다른 영역의 커버리지를 달성하거나 다른 종류의 결함을 발견하더라도 결국은 커버리지 달성률이나 결함의 갯수를 기준으로 평가하기 때문에 적당한 반복으로도 어느정도 수치가 수렴하기 마련이다.

하지만 지향성 퍼징의 경우 특정 지점에 도달하여 특정 결함을 재현하는 것이 목표이기 때문에 실험 결과가 더욱 무작위성에 민감하다. 따라서 지향성 퍼징의 경우 무지향성 퍼징보다 더 많은 실험 반복이 필요하다. 실제로 최근 연구에서는 40회 이상의 실험을 반복하여 그 결과를 평가하였다.

4.5 통계 검증

성능 평가의 최종 단계에서는 여러번 반복한 실험 결과를 통해 둘 이상의 퍼징 도구의 성능을 비교해야 한다. 각 실험에서 발생한 TTE의 중앙값을 비교하기도 하지만 중앙값은 분포를 반영하지 못하기 때문에 보다 정확한 통계검사를 하는 경우가 많다. 퍼징 분야에서 가장 보편적으로 사용되는 통계 검사 기법은 Mann-Whitney U(MWU) 검사[15]이며 이는 지향성 퍼징의 경우에도 마찬가지이다. 하지만 사실 지향성 퍼징의 경우 MWU 검사를 사용하는 것이 적절하지 않다. MWU 검사는 근본적으로 관측된 값만을 사용하기 때문에 주어진 시간 내에 목표 결함을 발견하지 못하는 경우를 처리할 수 없기 때문이다.

따라서 지향성 퍼징 기술을 평가할 때는 MWU 검사보다는 Log-rank 검사[16] 등의 생존분석(survival analysis)을 사용하는 것이 바람직하다. 생존 분석의 경우 주어진 시간 내에 사건이 관찰되지 않는 경우도 고려하기에 지향성 퍼징의 결과를 온전히 반영할 수 있디 때문이다. 특히 성능을 시각화 할수 있도록 선인장 그래프(cactus plot)도 함께 사용한다면 금상첨화일 것이다.

5. 결 론

지향성 퍼징은 특정 지점에 집중하여 효율적으로 프로그램을 탐색하는 방법이다. 수정되었거나 의심되는 지점을 검사 할 때, 혹은 오류 지점만 알려진 상황에서 오류를 재현할 때 지향성 퍼징은 더욱 빛을 발

한다. 지향성 퍼징은 무지향성 퍼징의 방법을 그대로 사용하면서도, 각 단계에서 지향성을 부여하여 효율적인 프로그램 탐색을 가능하게 한다. 지향성을 부여하는 방식은 다양하지만 모두 하나의 공통점을 가지고 있다. 자로 기존에는 고려하지 못했던 정보를 활용하여 더 정밀한 프로그램 탐색을 가능하게 한다는 것이다. 비용 문제로 인해 전체 프로그램을 대상으로는 엄두조차 내지 못했던 동적 오염 분석, 데이터 의존성 분석, 선행 조건 분석 등의 방법을 부담 없이 사용할 수 있게 되었고, 이렇게 얻은 정보는 다양하게 활용되어 지향성 퍼징의 핵심이 된다.

지향성 퍼징의 기법만큼이나 그 평가 방식도 특징적이다. 무지향성 퍼징의 경우 커버리지 달성률이나 결함 발견 갯수를 기준으로 평가되지만, 지향성 퍼징의 경우 목표 지점 도달 능력, 혹은 목표 결함 재현 능력을 기준으로 평가되어야 한다. 따라서 이 두 능력을 포괄하는 결함 재현 실험을 통해 성능을 평가하는 것이 일반적이다. 일관되고 공정한 평가를 위해 이 결함 재현 실험의 각 단계는 주의 깊게 이루어져야 한다.

현재 활발한 연구가 진행되고 있는 지향성 퍼징 기술은 앞으로도 계속해서 발전할 것이다. 특히 지향성 퍼징의 각 단계에서 지향성을 부여하는 방식이 더욱 다양해질 것이며, 이러한 방식들이 결합되어 더욱 효율적인 프로그램 탐색을 가능하게 할 것이다. 또한 지향성 퍼징의 평가 방식도 더욱 정교해질 것이며, 표준화된 방식을 통해 지향성 퍼징 도구의 성능을 더욱 정확하게 평가할 수 있을 것이다. 본 글이 지향성 퍼징의 기술과 평가에 대한 이해를 높이는 데 도움이 되었기를 바라며 마친다.

참고문헌

[1] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. Directed Greybox Fuzzing. *In Proceedings of the ACM Conference on Computer and Communications Security*. 2329 – 2344. 2017

[2] Michal Zalewski. American Fuzzy Lop. http://lcamtuf.coredump.cx/afl/.

[3] Paul Dan Marinescu and Cristian Cadar. KATCH: highcoverage testing of software patches. *In Proceedings of the International Symposium on Foundations of Software Engineering*, pages 235 – 245, 2013.

[4] Hongxu Chen, Yinxing Xue, Yuekang Li, Bihuan Chen, Xiaofei Xie, Xiuheng Wu, and Yang Liu. Hawkeye: Towards a desired directed grey-box fuzzer. *In Proceedings of the ACM Conference on Computer and Communications Security*, pages 2095 – 2108, 2018.

[5] Zhengjie Du, Yuekang Li, Yang Liu, and Bing Mao. Windranger: A directed greybox fuzzer driven by deviation basic blocks. *In Proceedings of the International Conference on Software Engineering*, pages 2440 – 2451, 2022.

[6] Heqing Huang, Yiyuan Guo, Qingkai Shi, Peisen Yao, Rongxin Wu, and Charles Zhang. Beacon: Directed grey-box fuzzing with provable path pruning. *In Proceedings of the IEEE Symposium on Security and Privacy*, pages 36 – 50, 2022.

[7] Tae Eun Kim, Jaeseung Choi, Kihong Heo, and Sang Kil Cha. DAFL: Directed Grey-box Fuzzing guided by Data Dependency. *In Proceedings of the USENIX Security Symposium*. 4931 – 4948. 2023

[8] Manh-Dung Nguyen, Sébastien Bardin, Richard Bonichon, Roland Groz, and Matthieu Lemerre. Binary-level Directed Fuzzing for Use-After-Free Vulnerabilities. *In Proceedings of the International Conference on Research in Attacks, Intrusions, and Defenses*. 47 – 62. 2020

[9] Sadullah Canakci, Nikolay Matyunin, Kalman Graffi, Ajay Joshi, and Manuel Egele. TargetFuzz: Using DARTs to Guide Directed Greybox Fuzzers. *In Proceedings of the Asia Conference on Computer and Communications Security*. 561 – 573. 2022

[10] Gwangmu Lee, Woochul Shim, and Byoungyoung Lee. Constraint-guided Directed Greybox Fuzzing. *In Proceedings of the USENIX Security Symposium*. 3559 – 3576. 2021

[11] Sebastian Österlund, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. ParmeSan: Sanitizer-Guided Greybox Fuzzing. *In Proceedings of the USENIX Security Symposium*. 2289 – 2306. 2020

[12] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitriy Vyukov. AddressSanitizer: A Fast Address Sanity Checker. *In Proceedings of the USENIX Annual Technical Conference*. 309 – 318. 2012

[13] Prashast Srivastava, Stefan Nagy, Matthew Hicks, Antonio Bianchi, and Mathias Payer. One Fuzz Doesn't Fit All: Optimizing Directed Fuzzing via Target-tailored Program State Restriction. *In Proceedings of the Annual Computer Security Applications Conference*. 388 – 399. 2022

[14] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei,

and Michael Hicks. Evaluating fuzz testing. *In Proceedings of the ACM Conference on Computer and Communications Security*. 2123 – 2138. 2018

[15] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*. 50 – 60. 1947.

[16] John P Klein and Melvin L Moeschberger. Survival analysis: techniques for censored and truncated data. *Springer*. 2003

[17] Tae Eun Kim, Jaeseung Choi, Seongjae Im, Kihong Heo, and Sang Kil Cha. Evaluating Directed Fuzzers: Are We Heading in the Right Direction?. *In Proceedings of the ACM Software Engeering*, Vol. 1, No. FSE, Article 15. 2024

[18] Ahmad Hazimeh, Adrian Herrera, and Mathias Payer. Magma: A Ground-Truth Fuzzing Benchmark. *In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*. 81 – 82. 2021

[19] Haeun Lee, Soomin Kim, and Sang Kil Cha. Fuzzle: Making a Puzzle for Fuzzers. *In Proceedings of the International Conference on Automated Software Engineering*. 2022

약 력

김 태 은

2021 한동대학교 전산전자공학부 졸업(학사)
2023 한국과학기술원 전산학부 졸업(석사)
2023~현재 한국과학기술원 전산학부 재학(박사)
Email : taeeun.kim@kaist.ac.kr

허 기 홍

2009 서울대학교 컴퓨터공학부 졸업(학사)
2017 서울대학교 컴퓨터공학부 졸업(박사)
2017~2019 미국 University of Pennsylvania, Postdoctoral Researcher
2019~현재 한국과학기술원 전산학부 조교수
Email : kihong.heo@kaist.ac.kr