Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.31, NO.5, Oct. 2021 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.5.911

퍼즈 테스팅을 통한 소프트웨어 회귀 버그 탐색 기법의 동향과 전망*

이 광 무,[1†] 이 병 영[2‡]
[1,2]서울대학교 (대학원생, 교수)

The Status Quo and Future of Software Regression Bug Discovery via Fuzz Testing*

Gwangmu Lee,[1†] Byoungyoung Lee[2‡]
[1,2]Seoul National University (Graduate student, Professor)

요 약

소프트웨어 패치가 빈번하게 이루어지는 최근의 추세에 따라, 소프트웨어 버그 역시 패치로 인해 유도되는 버그인 회귀 버그의 비중이 점차 증가하는 추세이다. 이에 산업계와 학계에서는 최근 자동 버그 탐지 방법으로 주목받고 있는 퍼즈 테스팅을 도입 및 개량하여 회귀 버그를 사전에 탐지하고자 하는 시도가 점차 활발해지고 있다. 이 논문에서는 회귀 버그 탐지를 위한 퍼즈 테스팅 연구의 현황에 대하여 살펴보고, 현재 기법들에 존재하는 한계를 참고삼아 향후 관련 연구의 방향에 대한 전망을 제시한다.

ABSTRACT

As software gets an increasing amount of patches, lots of software bugs are increasingly caused by such software patches, collectively known as regression bugs. To proactively detect the regressions bugs, both industry and academia are actively searching for a way to augment fuzz testing, one of the most popular automatic bug detection techniques. In this paper, we investigate the status quo of the studies on augmenting fuzz testing for regression bug detection and, based on the limitations of current proposals, provide an outlook of the relevant research.

Keywords: Regression bug, Fuzz testing, Directed fuzzing, Differential fuzzing, Hybrid fuzzing.

I. 서 론

패치가 드물게 이루어지던 종래의 물리적으로 분리된 컴퓨터 시스템 소프트웨어들과 달리, 현대의 컴퓨터 소프트웨어는 상호 연결된 시스템의 장점을 살려 더욱 빠른 빈도로 패치를 개발 및 적용하고 있다. 이는 여러 시스템에 적용되고 있는 오픈소스 프로젝트의 패치 빈도를 통해 파악할 수 있는 것으로, IoT와 단말기 시스템에서 지배적인 점유율을 차지하고 있는 Linux 시스템 커널의 경우 2019년 기준 시간당 9.4 개의 신규 Commit이 업데이트된다는 보고를 참고할 수 있다[1]. 이러한 패치들은 목적과 규모가 다양하지만, 기본적으로 소프트웨어 시스템의 기능과 안전성을 개선하는 것이 기본 목표라고 볼 수 있다.

하지만 근래의 소프트웨어에서는 역설적이게도 패

Received(09. 10. 2021), Accepted(09. 21. 2021)
* 이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2020-0-01840, 스마트폰의 내부데이터 접근 및보호 기술 분석).
* 이 연구는 서울대학교 신임교수 연구정착금으로 지원되는 연구비에 의하여 수행되었음.
† 주저자, gwangmu@snu.ac.kr
‡ 교신저자, byoungyoung@snu.ac.kr(Corresponding author)

치에 의해 유도되는 버그, 즉 회귀 버그(regression bug)가 점차 주요해지는 추세이다. 최근에 발표된 한 연구에 따르면 조사된 2만 3천 개의 버그 가운데 77%가 패치에 의해 유도된 회귀 버그로 밝혀졌는데, 이 발생 비중은 소프트웨어의 성숙도에 따라 최고 92%까지 기록하였다고 한다[2]. 이처럼 패치가 오히려 버그를 유도하는 경우가 많아짐에 따라, 패치를 시스템 향상의 관점으로뿐만 아니라 시스템에 잠재적 위협을 가져다줄 수 있는 대상으로 재조명해야 할 필요성이 높아지고 있다.

이런 가운데 최근 가장 주목받고 있는 자동 버그 탐지 기술인 퍼즈 테스팅을 회귀 버그 탐지에 응용하고자 하는 시도 역시 증가하였다. 퍼즈 테스팅은 테스트 타깃 소프트웨어에서 이상 동작을 발견할 때까지 무작위 인풋을 무한정 입력해보는 테스트 방식으로, 소프트웨어를 더 잘 탐색할 수 있도록 인풋을 생성하는 기법이 큰 성공을 거둔 후에 현재에는 보편적인 동적 버그 탐지 방법으로 활용되고 있다[3, 4].

이 논문에서는 소프트웨어 내 회귀 버그를 사전에 탐지할 목적으로 산업계와 학계가 집중하고 있는 퍼즈 테스팅 적용 및 개량 사례를 살펴보고, 관련된 연구의 향후 전망에 관하여 다룬다. 이를 위해 먼저 회귀 버그가 소프트웨어 인풋에 미치는 영향에 대해 고찰하고, 이를 기준으로 근래의 회귀 버그 탐지를 위한 퍼즈 테스팅이 어떤 방향으로 개량되었는지 살펴본다. 끝으로 회귀 버그 탐지를 위한 퍼즈 테스팅 개량형이 아직 가지고 있는 한계점을 진단해보고, 이를 통해 관련 개량 연구의 향후 연구 전망에 대하여 예상해본다.

본 논문의 구성은 다음과 같다. 2장에서는 회귀 버그 탐지를 위한 기본 기법인 퍼즈 테스팅의 동작 원리에 대하여 간략하게 정리한다. 3장에서는 패치가 소프트웨어 인풋에 미치는 영향을 분석하여 회귀 버그가 일어나는 원리를 살펴본다. 4장에서는 3장의 분석에 입각하여, 산업계와 학계에서 제시된 회귀 버그 탐지법을 정리하고, 각 방법의 한계점에 대하여 고찰한다. 5장에서는 선행된 한계점에 대한 고찰을 바탕으로 향후 관련 연구의 전망에 대하여 진단한다.

II. 배경: 퍼즈 테스팅

퍼즈 테스팅은 최근 자동 버그 탐지 방법으로 주목받고 있는 방법으로, 타깃 소프트웨어가 이상 동작을 일으킬 때까지 무작위 인풋을 무한정 입력하는 소프트웨어 테스팅 방법이다. 퍼즈 테스팅의 컨셉은 완전 무작위 인풋을 소규모 시스템 유틸리티에 적용한 연구를 시작으로 학계에 비교적 오랫동안 알려져 왔으나[5], 최근에 들어서야 인풋이 실행한 소프트웨어 부분들의 정보, 즉 커버리지를 활용해 더 좋은 인풋을 생성해내는 방법이 제안되며 범용적으로 받아들여지게 되었다[3, 4].

Fig.1은 커버리지 기반 퍼즈 시스템(이하 퍼저)의 간략한 동작 원리이다. 먼저 퍼저는 기반이 되는 인풋을 마련하기 위해 미리 확보된 인풋 묶음, 즉 코퍼스(corpus)에서 인풋 하나를 무작위로 가져온 후에(①) 인풋에 무작위 변이를 가하여 새로운 변이된 인풋을 생성한다(②). 퍼저는 이 인풋을 타깃 소프트웨어에 입력하여(③) 인풋이 실행한 커버리지 정보를 피드백으로 받아오는데(④), 만약 이 커버리지 정보에 지금까지 관찰할 수 없었던 새로운 커버리지가 발견되었다면 변이된 인풋을 코퍼스에 새로 추가하고 계속 과정을 반복한다(⑤).

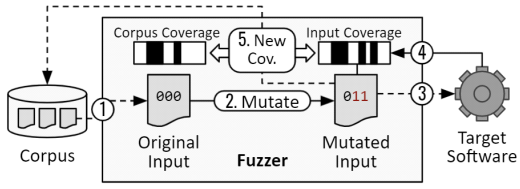

Fig. 1. Conceptual operation workflow of coverage-guided fuzzers.

III. 소프트웨어 인풋과 회귀 버그

소프트웨어에 패치가 가해지면 소프트웨어가 받아들일 수 있는 전체 인풋들 가운데 일부는 이 패치와 상호작용을 하게 된다. Fig.2는 소프트웨어가 받아들일 수 있는 전체 인풋 공간을 이러한 상호작용의 수준을 기준으로 나눈 것이다.

가장 약한 형태의 상호작용은 인풋이 패치가 된 코드 부분을 단순히 건드리는 경우로 볼 수 있다(patch-reaching). 이러한 수준의 상호작용은 패치 종류에 따라 인풋이 패치 지점을 지나친 후에는 소프트웨어에 영향을 주지 못할 수도 있다. 하지만 일부 인풋의 경우에는 패치 지점에서 이전 버전과 차이점을 발생시킬 수 있는데(difference-making), 실제로 이러한 차이점 가운데 대부분은 패치를 개발한 개발자가 다소간 의도한 결과라고 볼 수 있다.

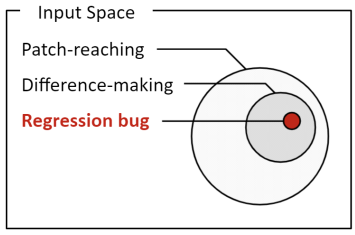

Fig. 2. Subsets of the fuzzing input space, subdivided by the level of interaction with a patch.

문제는 일부 차이점이 개발자의 의도와 무관하게 소프트웨어의 다른 부분에서 이상 동작을 일으킬 수 있다는 점인데, 이렇게 패치가 유도한 차이점으로 말미암아 발생한 이상 동작을 일컬어 회귀 버그(regression bug)라고 한다.

IV. 퍼즈 테스팅을 통한 회귀 버그 탐지

패치가 보편화하며 회귀 버그의 비중이 점차 증가함에 따라, 회귀 버그 탐지를 사전에 탐지하기 위해 퍼즈 테스팅은 적용하고자 하는 시도 역시 증가하였다. 이 장에서는 퍼즈 테스팅이 회귀 버그 탐지에 적용된 산업계와 학계의 사례를 살펴보고, 퍼즈 테스팅이 어떤 식으로 회귀 버그를 일으킬 가능성이 큰 인풋에 우선순위를 두도록 개량되었는지 살펴본다.

4.1 지속적 퍼즈 테스팅 방법

지속적 퍼즈 테스팅은 소프트웨어의 최신 버전을 대상으로 무한정으로 퍼즈 테스팅하는 방법으로, 기본적으로 회귀 버그뿐만 아니라 소프트웨어에 잠재된 모든 버그를 탐지해내는 것을 목표한다. 대표적인 지속적 퍼즈 테스팅 플랫폼은 대부분 현재 Google이 호스팅을 하는 것들으로, 플랫폼에 따라 일반적인 오픈 소스 어플리케이션을 대상으로 하거나[6, 7], 운영체제 커널을 대상으로 하는 경우도 있다[8].

지속적 퍼즈 테스팅 방법은 클라우드 퍼즈 테스팅 플랫폼인 OSS-Fuzz가 본격적으로 가동된 2017년에 전년 대비 2배가 넘는 버그를 발견한 것을 참고하였을 때, 잠재된 버그를 탐지해내는 데 어느 정도 유효한 방법임을 알 수 있다[9]. 하지만 전 소프트웨어를 대상으로 테스팅을 수행하는 성격상, 필요 이상의 시간을 패치가 되지 않은 부분을 테스팅하는 데에 낭비한다는 데에 근본적인 한계가 있다. 특히 소프트웨어의 성숙함도에 따라 회귀 버그의 비중에 증가한다는 점을 감안하였을 때[2], 회귀 버그 탐지를 위해서는 전 소프트웨어가 아닌 패치가 이루어진 부분을 집중하여 함을 알 수 있다.

4.2 유도 퍼즈 테스팅 방법

유도 퍼즈 테스팅은 회귀 버그가 일어나기 위한 대전제를 이용해 집중해야 하는 인풋의 범위를 좁힌 방법으로, 패치가 버그를 일으키기 위해서는 인풋이 단 한 번이라도 패치를 실행해야 한다는 원리에 기반한 방법이다. 유도 퍼즈 테스팅은 패치로의 도달이라는 측면에서 어느 부분을 중요시하는 지에 따라 다음과 같은 두 방식으로 크게 나눌 수 있다.

4.2.1 어려운 패치 도달 우선

이 방식은 도달이 상대적으로 어려운 패치까지 도달하는 것을 우선으로 하는 방식으로, 제어 흐름(control-flow)상 패치와 더 가까운 베이직 블록에 접근한 인풋이 향후 패치 지점에 도달할 가능성이 더 높은 것으로 보고 이들에 더 우선순위를 두는 방식이다[10, 11, 12].

Fig.3은 이 방식에서 인풋의 우선순위를 정하는 방식을 간단히 보여준다. 그림과 같이 패치 지점(P)이 지정되어있는 상황에서 인풋 A와 B가 각각 패치 지점에서 두 베이직 블록과 한 베이직 블록이 떨어진 지점을 실행하였다고 가정하였을 때, 이 방식에서는 더 가까운 베이직 블록을 실행시킨 인풋 B에 우선순위를 더 두는 방식이다.

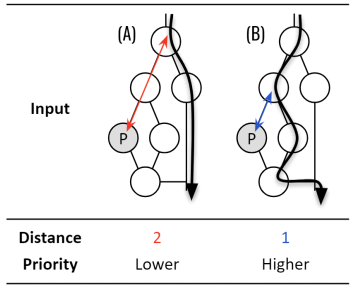

Fig. 3. Example of reach-first guided fuzzing.

4.2.2 많은 패치 도달 우선

반면 많은 패치에 도달하는 것을 우선시하는 방법에서는 인풋이 더 많은 신규 패치 지점에 도달할 경우 더 우선순위를 부여하는 방식이다.

Fig.4는 이 방식에서 인풋에 우선순위를 부여하는 방식을 간단히 나타낸 것으로, 다수의 패치 지점(P)에 인풋 A와 B가 각각 한 개와 두 개의 패치에 도달한 경우, 패치에 더 많이 도달한 인풋 B에 우선순위를 주는 방식이다. 이 방식은 경우에 따라 더 새로운 패치 지점에 가중치를 두는 방식과 결합되어 사용될 수 있는데[2], 이 경우 오래된 패치보다 새로운 패치를 더 많이 도달한 인풋이 선호된다.

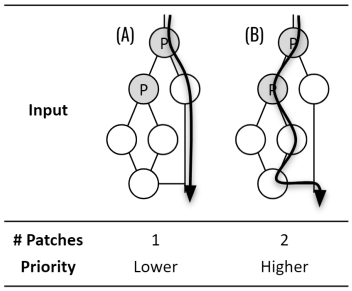

Fig. 4. Example of quantity-first guided fuzzing.

4.2.3 한계점

유도 퍼즈 테스팅은 인풋이 최소한 패치에 도달한 경우를 우선한다는 점에서, 패치와 관련 없는 부분을 불필요하게 테스팅하는 지속적 퍼즈 테스팅보다는 회귀 버그 탐지에 장점이 있다. 하지만, 회귀 버그가 발생하기 위해선 패치에 도달하는 것 뿐만 아니라 패치 이전 버전과 다른 차이점을 일으켜야 한다는 점을 감안했을 때, 유도 퍼즈 테스팅 역시 여전히 불필요한 인풋에서 시간을 소모하고 있다고 볼 수 있다.

Fig.5은 유도 퍼즈 테스팅이 회귀 버그를 탐지하는데 여전히 효과적이지 않을 수 있는 경우를 보여주는 코드로, 여기서 회귀 버그는 패치 내부에서 만든 차이점, 즉 x 값이 참인 경우에 드문 확률로 패치 외부에서 일어나게 되어있다. 하지만 유도 퍼즈 테스팅은 이러한 패치에 도달하는 모든 인풋 경우에 우선순위를 두게되고, 그 결과 x를 참으로 만드는 인풋은 상대적으로 우선순위가 낮아져 회귀 버그 탐지에 비효율이 발생할 수 있다.

```
    x = 0;
+ {
+    cond = local_computation();
+    if (cond) { x = 1; }
+ }
    ...
    if (x) {
      /* Rarely-occuring regression bug */
    }
```

Fig. 5. Example code snippet.

4.3 차이점 퍼즈 테스팅 방법

차이점 퍼즈 테스팅 방법은 유도 퍼즈 테스팅보다 회귀 버그에 더 직접적으로 관련이 있는 인풋에 더욱 집중하는 퍼즈 테스팅으로, 인풋이 이전 버전과 다른 차이점을 일으킨 경우에 집중하여 회귀 버그를 탐색하는 방법이다. 차이점 퍼즈 테스팅은 차이점을 탐지하거나 일으키는 방법에 따라 다음과 같은 두 가지 방식으로 나누어볼 수 있다.

4.3.1 결과론적 차이점 탐지

결과론적 차이점 탐지 방식은 패치 전과 후 두 가지 버전에서 동일한 인풋을 각 한 번씩 실행시킨 후에 그 결과를 비교하여 패치가 발생시킨 차이점을 탐지하는 방식이다. 이 방식은 의미론적 버그(semantic bug)를 검출하기 차이점 퍼즈 테스팅과 개념적으로 유사하지만[13, 14], 의미론적 버그 검출 시에 서로 다른 프로그램을 비교하는 것과 달리, 회귀 버그 탐지를 위한 차이점 퍼즈 테스팅에서는 "같은 프로그램의 다른 버전"을 비교한다는 것이 대표적인 차이점이다.

Fig.6는 결과론적 차이점 탐지 방식을 퍼징에 활용한 HyDiff[15]의 동작 개념을 간략하게 묘사한 것이다. HyDiff에서는 먼저 퍼저가 패치 전과 후 버전에 각각 같은 인풋을 입력하여(①) 프로그램에서 표출되는 커버리지 결과를 비교한다(②). 만약 인풋

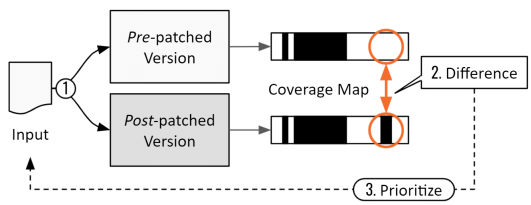

Fig. 6. Conceptual workflow of HyDiff[13].

이 커버리지 결과 대비 차이점을 발생시킨 경우, 이 인풋이 차이점을 발생시켰다고 간주하여 향후 퍼즈 테스팅에서 해당 인풋에 우선순위를 두게 된다(③).

결과론적 차이점 탐지 방식을 이용하여 차이점을 탐지하는 방식은 원리적으로는 타당하며, 특히 커버리지 차이를 이용한 방식은 패치가 된 부분 외부에서 결과적으로 일어나는 차이점을 탐지하는 데에는 효과적인 편이다. 하지만 이 방식은 패치 내부에서 발생하는 차이점을 선제적으로 발견하는 데에는 패치 자체가 내재한 복잡성으로 인해 한계가 있는데, 예를 들어 Fig.5와 같이 완전히 새로운 코드가 추가되거나 상당 부분이 삭제되는 식으로 이루어지는 패치의 경우에는 패치의 영향과 무관하게 모두 차이점으로 탐지될 것이다.

커버리지를 이용한 차이점 탐지가 잘못된 차이점을 보고하는 경우는 코드가 리팩토링되는 상황에서도 빈번하게 발생할 수 있다. Fig.7은 패치 이전에는 한 함수에 작성되어 있던 기능이(a) 다른 함수로 분리되는 리팩토링이 가해진 경우인데(b), 이것으로 인한 영향은 실질적으로 없음에도 불구하고 커버리지 비교를 통해서는 차이점이 있는 것으로 보일 것이다.

```
                                  +  int calc_y(int x) {
                                  +      return x*2;
                                  +  }
                                     ...
     x = 10;                         x = 10;
M    y = 20;                     M   y = calc_y(x);
     assert(y == x*2);               assert(y == x*2);
```

(a) Before patching (b) After patching

Fig. 7. Example refactored code snippet. The purple code (M) indicates a modified part.

4.3.2 인과론적 차이점 탐지

인과론적 차이점 탐지 방법은 결과론적 차이점 탐지와 반대되는 개념으로, 실행 전에 코드를 정적으로 분석하여 차이점이 발생할 것으로 예상되는 지점을 미리 탐지해내는 방식이다. 이와 같은 방법은 주로 퍼즈 테스팅과 심볼릭 실행이 결합된 하이브리드 퍼즈 테스팅에서 사용되는데[15, 16], 인풋이 정적 분석을 통해 검출된 차이점 예상 지점에 도달한 경우, 심볼릭 실행을 이용하여 실제로 차이점이 일어나는 인풋을 생성하는 방식으로 활용된다.

하지만 인과론적 차이점 탐지 방법 역시 몇가지 측면에서 한계를 갖는데, 가장 먼저 주로 결합되어 이용되는 심볼릭 실행이 규모가 방대한 대부분의 실세계 오픈 소스와 잘 호환되지 않는다는 점이다. 아울러 결과론적 차이점 검출과 마찬가지로 정적으로 분석이 된 차이점 발생 예상 지점 역시 부정확할 소지가 높다는 문제도 존재하는데, 부정확한 예상 지점이 많아지면 많아질수록 퍼즈 테스팅을 통한 회귀 버그 탐지의 효율 역시 저하된다고 볼 수 있다.

V. 연구 전망

회귀 버그 탐지를 위한 퍼즈 테스팅은 차이점을 일으키는 인풋에 우선순위를 두는 이상적인 접근이 꾸준히 시도되고 있으나[15,16], 차이점 탐지 방식 상의 한계로 인하여 유도 퍼즈 테스팅 방식 역시 대안으로서 지속적으로 제안되고 있는 추세이다[2,10,11]. 이 장에서는 현재의 차이점 퍼즈 테스팅 방식이 가진 한계에 대해 진단하고, 향후 차이점 퍼즈 테스팅을 개선시킬 수 있는 방안에 대하여 고찰한다.

5.1 오탐 없는 결과론적 차이점 탐지 기준 고안

결과론적 차이점 탐지 방법은 실행 결과를 바탕으로 차이점를 탐지하는 방식으로, 어떤 종류의 실행 결과를 통해 패치가 일으킨 차이점를 효과적으로 검출해 낼 것인지가 관건이다. 이러한 측면에서 커버리지 기반의 차이점 탐지 방법은 패치 외부의 차이점을 결과적으로 탐지해내는 데에는 유효하지만, 추가/삭제/리팩토링이 빈번한 패치 내부의 차이점에 대해서는 선제적으로 탐지할 수 없다는 한계가 있다. 따라서 개선된 탐지 방법에서는 커버리지 기반 차이점 탐지를 패치 외부에만 한정하여 적용하고, 패치 내부에서는 대안을 통해 별개로 차이점을 탐지하는 혼합 방법을 채택할 수도 있을 것이다.

5.2 유의미한 인과론적 차이점 예상법 고안

인과론적 차이점 탐지 방법은 정적 분석을 통해 얼마나 유의미한 차이점 예상 지점을 검출할 수 있느냐가 관건인 방법으로, 기존 연구[15,16]의 간단한 방식으로는 복잡한 패턴의 패치에 대응할 수 없거나, 대부분이 소프트웨어에 영향을 줄 수 없는 사소한 차이점를 예상해내는 데 그칠 가능성이 높다. 이를 극

복하기 위해선 유의미한 차이점 예상 지점 검출을 위한 추가 정적 분석 알고리즘을 새로 고안하거나, 더 정확한 정적 분석을 위해 전 소프트웨어가 아닌 부분에 한하여(under-constrained) 심볼릭 실행 기법을 활용해볼 수 있을 것이다.

VI. 결 론

이 논문에서는 최근 증가하는 추세에 있는 회귀 버그를 사전에 탐지하기 위해 퍼즈 테스팅이 개량되어 왔음을 살펴보고, 궁극적으로 근래에 들어 버전 간 차이점을 유발하는 인풋에 집중하는 퍼즈 시스템까지 제안되었음을 확인하였다. 아울러 현재의 차이점 탐지 방법에 한계로 말미암아, 향후 관련 연구는 이러한 차이점 탐지 방법을 개선시키는 방향으로 나아갈 가능성이 높다고 전망하였다.

References

[1] ZDNet, "*Commit 1 million: the history of the Linux kernel,*" https://www.zdnet.com/article/commit-1-million-the-history-of-the-linux-kernel, Aug. 2020.

[2] Xiaogang Zhu and Marcel Böhme, "*Regression greybox fuzzing,*" Proceedings of the 28th ACM Conference on Computer and Communications Security, [Preprint,] Nov. 2021. [accessed 2021 Septenber 27]. Available from: https://mboehme.github.io/paper/CCS21.pdf

[3] Github, "American Fuzzy Lop," https://github.com/google/AFL, Sep. 2021.

[4] The LLVM Compiler Infrastructure, "libFuzzer," https://llvm.org/docs/LibFuzzer.html, Sep. 2021.

[5] Barton P. Miller, Louis Fredriksen, and Bryan So, "*An empirical study of the reliability of UNIX utilities,*" Commun. ACM, vol. 33, no. 12, pp. 32-44, Dec. 1990.

[6] Github, "ClusterFuzz," https://google.github.io/clusterfuzz/, Sep. 2021.

[7] Github, "OSS-Fuzz," https://github.com/google/oss-fuzz, Sep. 2021.

[8] Syzbot, "Syzbot," https://syzkaller.appspot.com/upstream, Sep. 2021.

[9] ZDNet, "*Open-source security: This is why bugs in open-source software have hit a record high,*" https://www.zdnet.com/article/open-source-security-this-is-why-bugs-in-open-source-software-have-hit-a-record-high, Mar. 2020.

[10] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. "*Directed greybox fuzzing,*" Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2329-2344, Oct. 2017.

[11] Hongxu Chen, Yinxing Xue, Yuekang Li, Bihuan Chen, Xiaofei Xie, Xiuheng Wu, and Yang Liu. "*Hawkeye: towards a desired directed grey-box fuzzer,*" Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 2095-2108, Oct. 2018.

[12] Gwangmu Lee, Woochul Shim, and Byoungyoung Lee, "*Constraint-guided directed greybox fuzzing,*" Proceedings of the 30th USENIX Conference on Security Symposium, pp. 3559-3576, Aug. 2021.

[13] Yuting Chen, Ting Su, and Zhendong Su. "*Deep differential testing of JVM implementations,*" Proceedings of the 41st International Conference on Software Engineering, pp. 1257-1268, May. 2019.

[14] Theofilos Petsios, Adrian Tang, Salvatore Stolfo, Angelos D. Keromytis, and Suman Jana, "*NEZHA: efficient domain-independent differential testing,*" 2017 IEEE Symposium on Security and Privacy, pp. 615-632, May. 2017.

[15] Yannic Noller, Corina S. Păsăreanu, Marcel Böhme, Youcheng Sun, Hoang Lam Nguyen, and Lars Grunske, "*HyDiff: hybrid differential software analysis,*" Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 1273-1285, Jun. 2020.

[16] Hristina Palikareva, Tomasz Kuchta, and Cristian Cadar, "*Shadow of a doubt: testing for divergences between software versions,*" Proceedings of the 38th International Conference on Software Engineering, pp. 1181-1192, May. 2016.

〈저자소개〉

이 광 무 (Gwangmu Lee) 학생회원
2014년 8월: 포항공과대학교 물리학과 졸업
2017년 2월: 포항공과대학교 컴퓨터공학과 석사
2017년 3월~현재: 서울대학교 전기정보공학부 박사과정
〈관심분야〉 퍼즈 테스팅, 운영체제 보안, 컴파일러 최적화

이 병 영 (Byoungyoung Lee) 정회원
2009년 2월: 포항공과대학교 컴퓨터공학과 졸업
2011년 5월: 포항공과대학교 컴퓨터공학과 석사
2016년 8월: 조지아공과대학교 컴퓨터과학과 박사과정
2016년 8월~2018년 8월: 퍼듀대학교 컴퓨터과학과 교수
2018년 9월~현재: 서울대학교 전기정보공학부 교수
〈관심분야〉 컴퓨터 보안, 소프트웨어 및 운영체제 보안