*Journal of The Korea Institute of Information Security & Cryptology* ISSN 1598-3986(Print)
VOL.32, NO.2, Apr. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.2.417

# 자연어 처리 모델을 활용한 퍼징 시드 생성 기법

김 동 영,[1†] 전 상 훈,[1] 류 민 수,[1] 김 휘 강[2‡]
[1,2]고려대학교 정보보호대학원 (대학원생, 교수)

# A Fuzzing Seed Generation Technique Using Natural Language Processing Model

DongYonug Kim,[1†] SangHoon Jeon,[1] MinSoo Ryu,[1] Huy Kang Kim[2‡]
[1,2]School of Cybersecurity, Korea University (Graduate student, Professor)

요 약

Fuzzing에서 seed corpus의 품질은 취약점을 보다 빠르게 찾기 위해서 중요한 요소 중 하나라고 할 수 있다. 이에 dynamic taint analysis와 symbolic execution 기법 등을 활용하여 효율적인 seed corpus를 생성하는 연구들이 진행되어왔으나, 높은 전문 지식이 요구되고, 낮은 coverage로 인해 광범위한 활용에 제약이 있었다. 이에 본 논문에서는 자연어 처리 모델인 Sequence-to-Sequence 모델을 기반으로 seed corpus를 생성하는 DDRFuzz 시스템을 제안한다. 본 논문에서 제안하는 시스템은 멀티미디어 파일을 입력값으로 하는 5개의 오픈소스 프로젝트를 대상으로 관련 연구들과 비교하여 효과를 검증하였다. 실험 결과, DDRFuzz가 coverage와 crash count 측면에서 가장 뛰어난 성능을 나타냄을 확인할 수 있었고, 또한 신규 취약점을 포함하여 총 3개의 취약점을 탐지하였다.

ABSTRACT

The quality of the fuzzing seed file is one of the important factors to discover vulnerabilities faster. Although the prior seed generation paradigm, using dynamic taint analysis and symbolic execution techniques, enhanced fuzzing efficiency, they are not extensively applied owing to their high complexity and need for expertise. This study proposed the DDRFuzz system, which creates seed files based on sequence-to-sequence models. We evaluated DDRFuzz on five open-source applications that used multimedia input files. Following experimental results, DDRFuzz showed the best performance compared with the state-of-the-art studies in terms of fuzzing efficiency.

**Keywords:** Fuzzing, Seed generation, Sequence-to-Sequence

## I. 서 론

취약점이란 컴퓨터 시스템의 응용 프로그램이 잘못 동작하도록 하는 오류, 장애, 결함을 뜻하며, 공격자는 이런 취약점을 악용하여 허가되지 않은 시스템에 접근하거나 손상시킬 수 있다. 최근 해킹 기술의 발전으로 소프트웨어 취약점이 꾸준히 증가하고 있다. 2020년 한 해에만 공개된 취약점 데이터베이스인 CVE에 20,000개 이상의 보안 취약점이 등록되고 있으며 [1], 이러한 취약점을 공격자가 악용하기 전에 취약점을 먼저 탐지하고 제거하는 것은 매우 중요하다.

Fuzzing은 1990년 Miller가 처음 제안한 응용 프로그램의 취약점을 탐지하는 가장 효과적인 방법 중 하나로, 프로그램의 취약점이 존재하는지 확인하기 위해 프로그램 입력값으로 사용할 잘못된 데이터

Received(02. 09. 2022), Modified(03. 22. 2022), Accepted(03. 22. 2022)
† 주저자, ehddud758@korea.ac.kr
‡ 교신저자, cenda@korea.ac.kr(Corresponding author)

(예:파일, 네트워크 패킷, 소스 코드)를 사용하여 프로그램의 여러 프로그램 예외(예: crash, memory leak)를 자동으로 찾아내는 테스트 기술이다. 이후 fuzzing의 발전을 위해 입력으로 사용되는 seed corpus를 변경하는 방법, code coverage를 늘리는 방법, 입력값 검증을 효과적으로 우회하는 방법 등 다양한 기술이 제안되었다. 특히 관련 연구에서 볼 수 있듯이 [2], crash를 효과적으로 찾을 수 있는 seed corpus는 fuzzing 효율성을 높이는데 매우 중요하다.

본 연구에서는 더 많은 crash와 path를 발견하기 위해 기존보다 더 효율적인 seed corpus를 제공하는데 중점을 둔다. 지난 몇 년 동안 dynamic taint analysis, symbolic execution과 같은 기법을 fuzzing에 활용하여 더 나은 seed corpus를 생성하기 위한 여러 연구가 수행되어왔다. dynamic taint analysis는 신뢰할 수 없는 입력의 데이터를 추적하여 발생하는 예외를 효율적으로 추적할 수 있다. 이러한 방식으로 예외를 추적할 경우 낮은 오탐율을 가지게 되지만, 모든 입력 데이터에 taint 데이터를 표시하고 추적해야 하므로 프로그램 실행 효율성이 낮다. symbolic execution은 프로그램 내 모든 제약 조건을 해결하기 위해 기호를 사용하여 수학적으로 해결하는 것으로 간주한다. 따라서 응용 프로그램에서 버그를 정확하게 찾을 수 있지만, fuzzing 대상 응용 프로그램이 크면 최종 방정식이 너무 복잡해져서 푸는데 어려움이 존재하므로, 확장성이 매우 떨어진다. 또한 기존 방법은 전문가의 지식 등 개입이 반드시 필요하다.

이에 따라 최신 deep learning 기법을 활용하여 효율적인 seed corpus를 제공하는 연구가 소프트웨어 fuzzing 연구의 새로운 트렌드가 되었다. SmartSeed는 [3] 전문가 개입을 없애고 여러 테스트 시나리오에서 쉽게 사용할 수 있도록 seed corpus 생성 방법을 제안했으며, GAN (Generative Adversarial Network)을 활용하여 seed corpus를 생성함으로써 보다 효율적인 seed corpus를 생성했다. 이러한 GAN model은 입력값의 중요 특성을 기반으로 유사 값을 생성하는 model이다. 이러한 종류의 deep learning model은 입력값과 유사한 출력값만 생성하므로, SmartSeed는 입력값으로 사용된 crash와 비슷한 유형의 파일만 생성한다. 즉, 다양한 종류의 crash를 탐지하는 seed corpus를 생성할 수 없다는 한계가 있다.

이 문제를 해결하기 위해 우리는 Sequence-to-Sequence (seq2seq) model을 기반으로 설계된 fuzzing 시스템인 Data-DRiven Fuzz (DDRFuzz)를 제안하여 valuable seed corpus를 생성하였다. seq2seq의 세부 model로는 Simple-seq2seq, seq2seq-attention, transformer 세 가지를 사용하였고, 본 논문에서 언급하는 seq2seq는 세 가지 model을 모두 통칭하는 용어로 사용한다.

본 연구에서는 valuable seed를 높은 확률로 unique crash 혹은 path를 트리거 가능한 입력값으로 정의한다. DDRFuzz는 기존의 deep learning 기반 seed corpus 생성 연구와 다르게 seq2seq model을 이용하여 보다 다양한 seed corpus를 생성할 수 있으며, 이를 통해 높은 coverage와 많은 crash를 기대할 수 있다. 또한, seed corpus 생성과 fuzzing 과정을 분리하여 다양한 fuzzer(예: AFL)에 쉽게 적용할 수 있도록 DDRFuzz를 설계하였다. 본 연구의 주요 기여사항은 다음과 같다.

- 본 연구 valuable seed corpus 생성을 위해 seq2seq 기반으로 설계된 fuzzing 시스템인 DDRFuzz를 제안하였으며, 우리가 아는 한 mutation based fuzzing에서 seq2seq model을 활용하는 최초의 연구이다.
- 데이터셋은 5개의 오픈 소스 기반으로 구성하였으며, DDRFuzz의 embedding 결과 및 소스 코드는 GitHub에 공개하였다.
- mp3, avif, tiff 등의 여러 파일 형식을 사용하는 5개의 오픈 소스 응용 프로그램에서 DDRFuzz에서 생성된 valuable seed corpus를 AFL의 입력값으로 사용하여 평가했다. DDRFuzz는 13개의 unique crash, 2,935개의 unique path를 찾았으며, 2개의 알려진 취약점과 1개의 Zero-day 취약점을 발견하였다.

본 논문의 구성은 섹션 2에서 본 논문의 배경에 대해서 설명하고, 섹션 3에서 기존에 수행되었던 관련 연구에 대해서 언급하고, 섹션 4에서 DDRFuzz 설계에 대해서 자세하게 언급하고, 섹션 5에서 DDRFuzz의 효율성에 대해서 평가를 한다. 마지막으로 섹션 6에서는 DDRFuzz의 한계와 앞으로 향후 연구에 대해서 언급하고 마무리한다.

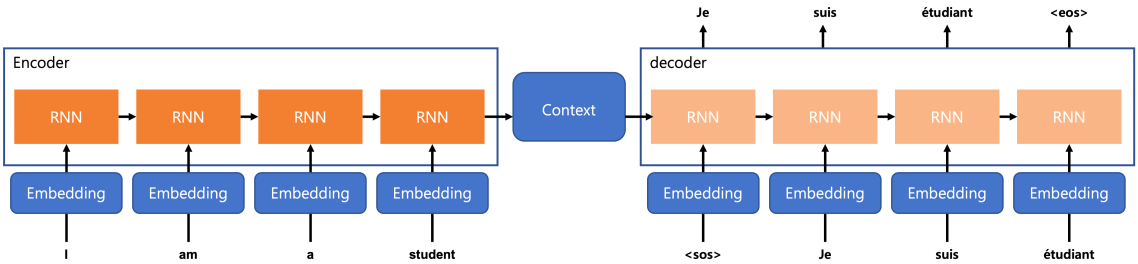


Fig. 1. simple RNN based seq2seq model

## II. 배 경

Seq2seq model은 encoder와 decoder 두 개의 모듈로 구성된다. Encoder는 입력 문장의 모든 단어들을 순차적으로 입력 받은 뒤에 마지막에 이 모든 단어 정보들을 압축해서 하나의 벡터로 만드는데, 이를 context vector라고 한다. 입력 문장의 정보가 하나의 context vector로 모두 압축되면 encoder는 context vector를 decoder로 전송한다. Decoder는 context vector를 받아서 번역된 단어를 한 개씩 순차적으로 출력한다. Encoder와 decoder architecture 내부를 어떻게 구성하느냐에 따라서 구분할 수 있고, 본 논문에서는 seq2seq model을 simple RNN based model, seq2seq with attention model 그리고 transformer model 등으로 구분하였다. 각 model은 순서대로 machine translation 분야에서 앞선 model의 한계점을 개선하고자 제안되었지만 seed generation 관점에서는 개선 효과가 다르게 나타날 수 있으므로, 본 연구에서는 3가지 model을 모두 구현 후 성능을 비교/분석하였다.

Fig. 1.은 simple RNN based seq2seq model의 동작을 설명한다. Encoder가 길이 n의 입력 sequence를 입력 받아 context vector로 압축하고, decoder가 이 vector를 전달 받아 출력 sequence를 만든다. 여기서 encoder가 decoder에서 전달해주는 context vector는 encoder cell의 마지막 시점 (time step)의 hidden state이고, n번째 시점의 hidden state는 입력 시퀀스의 정보들을 반영하고 있다. 이후 해당 vector 값을

repeat vector를 사용하여 decoder의 모든 시점에 대한 입력으로 사용한다.

하지만 이러한 simple RNN based seq2seq model에는 크게 두 가지 문제가 있다. 첫째, 하나의 고정된 길이의 vector에 모든 정보를 압축하다보니, 정보 손실이 발생한다. 둘째, RNN의 고질적인 문제인 vanishing gradient 문제가 존재한다. 즉, 이는 입력 시퀀스가 길어지면 출력 시퀀스의 품질이 떨어지는 현상이 발생할 수 있고, 이러한 현상으로 정확도가 떨어지는 것을 보정해 주기 위한 기법인 attention을 적용한 seq2seq with attention model을 사용할 수 있다.

Fig. 2.는 attention의 동작을 나타내며, attention의 기본 아이디어는 decoder에서 출력 단어를 예측하는 매 시점 (time step)마다, encoder에서의 전체 입력 문장을 다시 한번 참고한다는 것이다. 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측 해야할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중해서 본다. Attention은 다양한 종류가 있는데 본 연구에서는 가장 많이 사용되고 있는 dot-product attention을 사용한다.

Transformer는 attention만을 이용하여 seq2seq의 encoder에서도 입력 시퀀스를 입력고, decoder에서 출력 시퀀스를 출력하는 encode-decoder 구조를 따르는 model이다. 즉, Transformer는 RNN을 사용하지 않지만 기존의 encoder-decoder 구조를 유지하고 있다. 다만 다른 점은 encoder와 decoder라는 단위가 N개 존재할 수 있다는 것이다. 이전 seq2seq 구조에서는 encoder와 decoder에서 각각 하나의 RNN이 t개의 시점(time-step)을 가지는 구조였다면, 이번에는 encoder와 decoder라는 단위가 N개로 구성되는 구조이다.

Fig. 3.은 transformer의 architecture를 의미한다. 왼쪽 module이 encoder를 의미하고, 오른쪽 module이 decoder를 의미한다. 먼저 positional encoding은 단어의 위치정보를 의미한다.

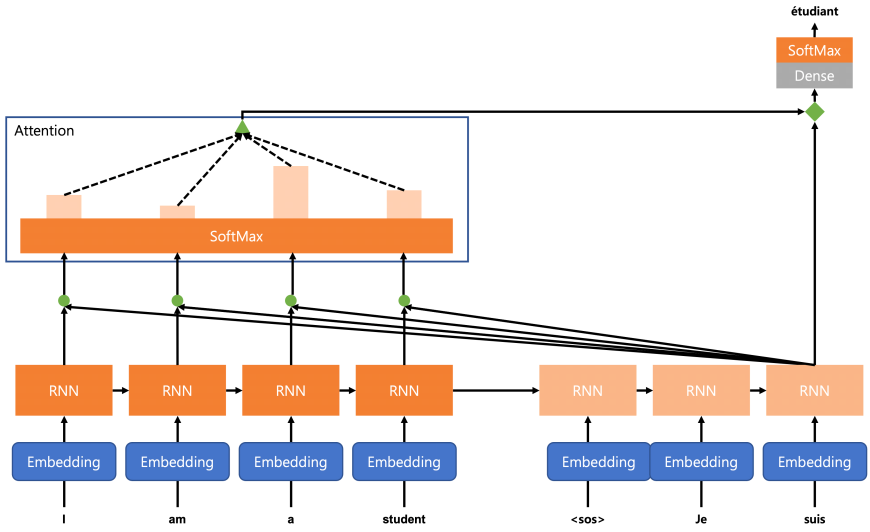


Fig. 2. seq2seq with attention model

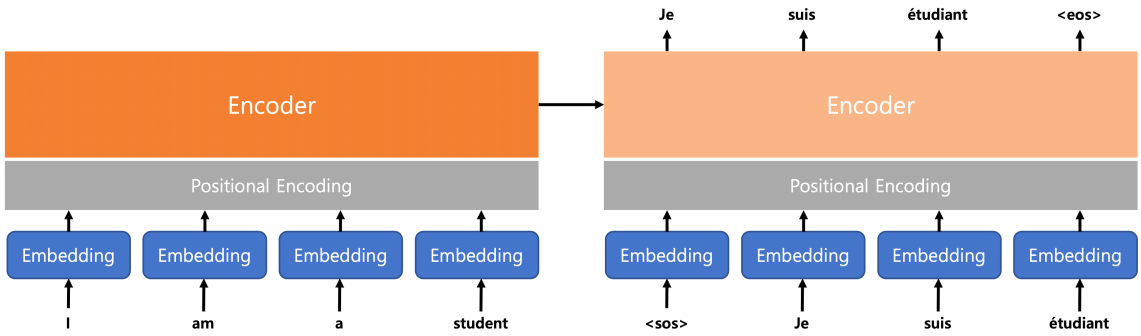


Fig. 3. transformer model

Transformer는 단어 입력을 순차적으로 받는 방식이 아니므로 단어의 위치 정보를 다른 방식으로 알려줄 필요가 있고, transformer는 단어의 위치 정보를 얻기 위해서 각 단어의 embedding vector에 위치 정보들을 더하여 model의 입력으로 사용하는데, 이를 positional encoding이라고 한다. Transformer는 hyper-parameter인 num layers 개수를 encoder 층으로 쌓는다. Encoder를 하나의 층이라는 개념으로 생각한다면, 하나의 encoder 층은 크게 총 2개의 sub layer로 나뉘어지는데, 각각 self-attention과 feed-forward neural net (FFNN)이다. Fig. 3.에서 multi-head self-attention은 self-attention을 병렬적으로 사용한다는 의미이다.

위와 같이 구현된 encoder는 총 num layers 만큼의 층 연산을 순차적으로 한 후에 마지막 층의 encoder 출력을 decoder에서 전달한다. Encoder 연산이 끝나면 decoder 연산이 시작되고, decoder 또한 총 num layers 만큼의 층 연산을 하는데, 이 때마다 encoder가 보낸 출력을 각 decoder 층 연산에 사용한다. decoder도 encoder와 동일하게 embedding layer와 positional encoding을 거친 후의 문장 행렬이 입력된다. seq2seq의 decoder에 사용되는 RNN 계열의 신경망은 입력 단어를 매 시점마다 순차적으로 받으므로 다음 단어 예측에 현재 시점 이전 이전에 입력된 단어들만 참고할 수 있다. 반면, transformer는 문장 행렬로 입력을 한 번에 받으므로 현재 시점의 단어를 예측하고자 할 때, 입력 문장 행렬로부터 미래 시점의 단어까지도 참고할 수 있는 현상이 발생한다. 이를 위해

Transformer의 decoder에서는 현재 시점의 예측에서 현재 시점보다 미래에 있는 단어들을 참고하지 못하도록 look-ahead mask를 사용한다. decoder의 두번째 sub layer는 multi-head attention을 수행한다는 점에서는 이전의 attention들과 같지만, self-attention은 아니다. 이후의 decoder 과정은 encoder와 유사하게 수행한다.

## III. 관련 연구

기존 fuzzing의 문제를 해결하기 위한 새로운 아이디어로 Machine Learning (ML) 기술을 도입하여 fuzzing 기술을 지능화하기 위한 연구가 수행되어 왔다. 더 효율적인 seed corpus를 제공하기 위한 ML 기반 접근 방법에는 seed mutation과 seed generation 두 가지 유형이 있다. 본 장에서는 ML 기반으로 fuzzing 수행에 필요한 seed corpus 생성과 관련하여 기존까지 수행되었던 연구를 소개한다.

### 3.1 Mutation 기반 seed mutation

Fuzzing에서 mutation은 biological 또는 genetic algorithm에서 파생된다. fuzzing의 mutation에서는 추가, 수정, 제거와 같은 연산이 수행되고, 이러한 mutation은 여러 효과를 가져올 수 있다. 예를 들어, code coverage를 늘리거나 취약점이 존재하는 경로를 트리거할 수 있지만, 이러한 genetic algorithm 기반 mutation 기술은 crash를 트리거 하는 데는 효율성이 떨어진다 [4]. 이를 해결하기 위해서 수행된 기존 연구들에는 전문 지식이 필요하거나 fuzzing 속도 및 확장성이 제한적이라는 단점이 있었고, 따라서 ML 기술을 이용하여 seed mutation을 개선하기 위한 여러 연구가 수행되었다.

#### 3.1.1 Mutation 중심 연구

LEFT는 [5] Android 스마트폰에서 LTE 기능에 대해서 fuzzing을 수행하기 위해 Reinforcement Learning(RL) 기반 model을 구성한다. Model에는 emulation instrumented black-box fuzzing, thread-modelware fuzzing, RL-guided fuzzing이 포함되어 있다. 또한 이러한 feedback 기반 random testing과 RL 등에서 영감을 얻은 B¨ottinger는 [6] code coverage를 최대화하고 처리 fuzzing에 소요되는 시간을 줄이기 위해 RL을 사용한 fuzzing 방법을 최초로 제안했다. Karamcheti는 [7] 단일 프로그램을 fuzzing 하는 과정에서 mutator 분포를 적절하게 조정할 수 있는 robber 기반의 Thompson Sampling 최적화 방법을 제안했다. 이는 code coverage에 대한 mutation 연산자의 영향을 학습하여 어떤 mutation 연산을 선택해야 하는지 결정한다. 이를 통해 다음 단계에서 테스트에 사용되는 입력값을 변경 한다. Rajpal은 [8] 신경망을 사용하여 past fuzzing exploration에서 입력값의 패턴을 학습하여 future fuzzing exploration을 안내하는 학습 기법을 제시한다. 신경망 model은 입력 파일의 좋은 위치를 예측하는 기능을 학습하여 past mutations 및 해당 code coverage 정보를 기반으로 fuzzing mutation을 수행한다. 이러한 mutation 중심 연구들은 seed mutation 관점에서 진행된 연구이고, DDRFuzz는 seed generation 관점으로 진행되는 연구이며, 서로 종속적이므로, 향후 DDRFuzz에 통합하여 연구 확장 가능하다. 또한 RL 같은 경우는 mutation의 knowledge를 학습한다는 관점은 DDRFuzz와 동일하나 fuzzing의 runtime에 적용한다는 점에서 오버헤드가 크다는 단점을 가지고 있다.

#### 3.1.2 Selection 중심 연구

FuzzerGym은 [9] fuzzing의 대상으로 하는 응용 프로그램의 상태 정보를 얻기 위해 LLVM sanitizers을 사용하여 효율적으로 프로그램 모니터링을 한다. 이 정보는 RL을 사용하여 mutation 연산 결정 최적화에 되며, OpenAIGym을 libFuzzer와 [10] 통합하여 mutation-selection 학습을 수행했다. 즉, RL과 fuzzing의 장점이 결합시켰으며, 이러한 연구는 여러 벤치마크에서 높은 code coverage를 달성했다. Liu와 [11] Zhang은 [12] 실행 추적으로 확보한 code coverage 정보를 보상으로 사용했고 Deep Q-learning 알고리즘을 통해 보상을 최적화했다. 이를 통해 mutation selection을 수행하여 code coverage를 높였다.
NeuFuzz는 [13] LSTM을 통해 샘플에 알려진 취약점과 프로그램 내에 존재하는 취약점 패턴을 학습

하여 취약점을 트리거할 수 있는 실행 path를 찾는다. 그러면 NeuFuzz는 취약점이 포함된 path로 도달 할 수 있는 입력값을 우선적으로 예측하여 실행을 하고 실행의 결과를 기반으로 입력값에 더 많은 mutation 가중치를 할당한다.

위에서 기술한 바와 같이 ML 기반 seed corpus 관련 연구는 mutation과 selection 중심으로 연구가 분류된다. mutation 연구는 seed corpus를 효율적으로 mutation 하는 방법에 초점을 맞추고 selection 연구는 입력값 selection의 우선순위에 초점을 맞춰 진행되었다. 위에 진행되는 연구들도 DDRFuzz 관점이 다르지만, 서로 종속적이므로, 향후 DDRFuzz에 통합하여 연구 확장 가능하다.

### 3.2 ML 기반 seed generation

Seed selection 전략은 seed corpus를 확보하는데 많은 시간이 필요하고, selection 전략을 통해 선택된 입력값의 실행 효율과 무작위로 선택한 입력값의 실행 효율이 큰 차이가 없다는 단점이 존재한다 [2]. 따라서 기존 fuzzing 시 더 높은 code coverage, 더 많은 unique crash, path를 찾기 위해 seed corpus의 공통적인 특징을 ML 학습을 통해 생성하는 연구들이 아래와 같이 진행되고 있다.

#### 3.2.1 context를 위한 seed generation

Context를 위한 seed corpus 생성 연구는 입력값의 context를 유지하면서 seed mutation을 수행한다. 대상으로 하는 프로그램이 입력값 context에 민감한 경우, context를 무시하고 mutation을 수행하면서 context를 확인하는 부분만 포함시킨다. 따라서 context를 유지하면서 context 확인 및 기타 부분을 fuzzing하는 seed corpus 생성 방법이다. 대상으로 하는 프로그램이 PDF와 같은 특정 파일 형식을 입력 파일로 하는 경우 파일 형식을 인식하여 mutation을 수행한다. 이러한 것과 관련 있는 연구로는 Skyfire [14]가 있다. 해당 연구에서 구현한 데이터 기반 seed corpus 생성 방법은 PCFG(Probabilistic Context-Free Grammar)를 사용하여 의미 정보를 자동으로 추출하며, 이러한 의미 정보 및 문법 규칙은 seed corpus를 생성하는데 사용된다. Samplefuzz는 [15] 신경망 기반 통계 학습을 사용하여 입력 sample에서 입력 구문을 자동으로 생성하려는 첫 번째 연구이며, 본 연구에서는 seq2seq 기반 신경망을 통해 자동으로 PDF를 생성할 수 있는 model을 제안하였고, 평가도 수행했다. Cheng와 [16] Joffe는 [17] PDF파일과 대상으로 하는 프로그램 실행 path 간의 상관관계를 찾기 위해 RNN과 seq2seq를 사용했으며, 상관관계를 사용하여 대상으로 하는 프로그램에서 새로운 path 탐색 가능성이 높은 seed corpus를 생성 했다. Fan는 [18] 네트워크 패킷을 입력으로 하는 응용 프로그램을 대상으로 하며, 네트워크 프로토콜을 인식하여 mutation을 수행한다. 해당 연구에서는 네트워크 프로토콜에 대한 black-box fuzzing test case를 자동으로 생성하는 방법을 제안했다. 이 방법은 seq2seq를 사용하여 고유하게 생성된 네트워크 프로토콜을 model의 입력값으로 사용하여 학습을 하고, 학습된 model을 통해 새로운 메시지를 생성한다. GANFuzz [19]는 GAN에서 생성된 model을 학습시켜, 제어 시스템에서 사용되는 네트워크 프로토콜을 추론할 수 있도록 하여, 잘 구성된 test case를 생성하도록 한다.

Li는 [20] WGAN 기반의 제어 시스템 프로토콜에 대한 fuzzing test case를 생성하는 방법을 제안했으며, 이 방법은 실제 데이터 프레임의 구조와 분포를 학습하기에 자세한 프로토콜의 구조를 몰라도 유사한 데이터 프레임을 생성할 수 있다.

앞서 설명한 바와 같이 context-aware 생성 연구는 context 인식을 위해 ML 기법을 사용하여 code coverage를 향상 시킬 수 있지만, 이러한 연구는 주로 context-aware에 초점이 맞춰져 있으므로 fuzzing의 목적인 취약점을 발견하기 위한 crash를 유발하는 seed corpus 생성과는 거리가 있다. 본 논문에서는 crash를 유발할 가능성이 높은 seed corpus를 생성하여 위의 연구들의 단점을 해결한다.

#### 3.2.2 crash를 위한 seed generation

앞에서 설명한 바와 같이 기존의 taint analysis 또는 symbolic execution을 이용하는 crash 트리거를 위한 seed corpus 생성 연구는 많은 전문 지식이 요구되거나 fuzzing 속도 및 확장성이 제한적이라는 단점이 존재한다. 따라서 crash를 위한 ML 기술을 사용하여 이 문제를 개선하려는 연구도 존재

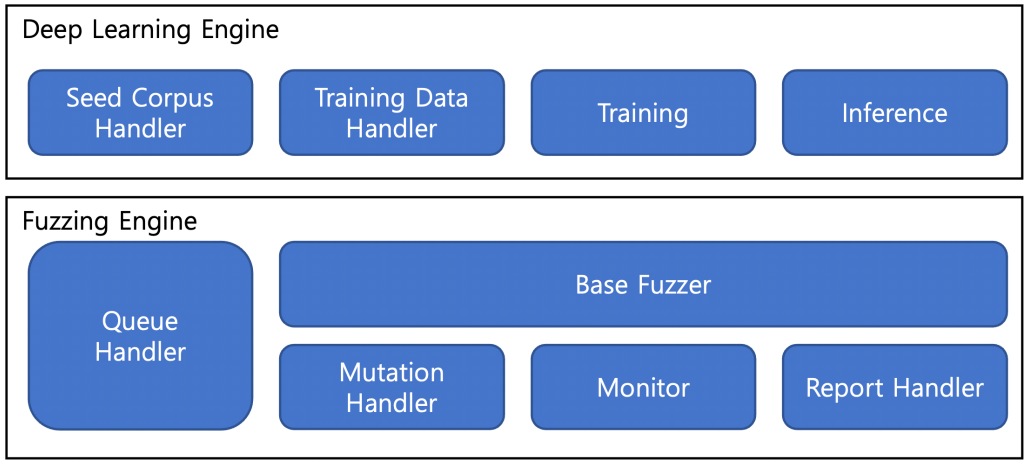


Fig. 4. DDRFuzz overview

한다.

Fast fuzzing은 [21] 무작위 mutation에 대한 테스트 효과를 향상시키기 위해 deep learning model을 사용하며, 이 방법은 AFL [22] 생성 샘플에서 features를 학습하여 GAN의 훈련을 통해 실행 path를 효율적으로 늘릴 수 있는 seed corpus를 생성한다. SmartSeed는 [3] 입력값을 읽어 균일한 유형의 vector로 변환한 후, WGAN (Wasserstein Generative Adversarial Network) 및 MLP (Multi-Layer Perceptron)를 활용하여 unique crash 또는 새로운 path를 찾을 수 있는 효율적인 seed corpus를 생성할 수 있다.

GAN은 입력값과 유사한 출력값을 생성하는 model이기 때문에 crash를 트리거할 가능성이 높은 seed corpus 생성이 가능하지만, 다양한 path에서 crash를 발생시키는데 한계가 있다는 단점이 존재한다. 하지만 본 연구에서는 GAN 대신 NLP (Natural Language Processing)에서 널리 사용되는 seq2seq model을 사용하여 fuzzer가 보다 다양한 path에서 crash를 유발할 수 있도록 한다.

## IV. DDRFuzz

본 섹션에서는 기존 fuzzer에 valuable seed corpus를 제공하는 DDRFuzz라는 새로운 seed corpus 생성 시스템을 제안한다. 먼저 DDRFuzz에 대해 전반적인 시스템 설명을 하고, seq2seq model 학습을 위한 데이터를 수집하는 방법과, 수집된 데이터로 valuable seed corpus를 생성하는 과정에 대해서 설명하고, 생성된 seed corpus로 fuzzing을 수행하는 단계에 대해서 설명한다.

### 4.1 DDRFuzz 개요

Fig. 4.에서 보여주는 바와 같이, DDRFuzz는 크게 데이터셋과 seq2seq model을 관리하기 위한 Deep Learning Engine과 fuzzer를 제어하는 Fuzzing Engine 두 부분으로 구성되어 있다. 이러한 DDRFuzz는 각 engine을 활용해 Data gathering, Seed generation, Fuzzing 세 단계로 동작한다. 이 섹션에서는 DDRFuzz가 동작하는 관점으로 설명하였으며, 위에서 언급한 3가지 단계에 대해서 설명한다. DDRFuzz의 각 단계 별 역할은 아래와 같다.

1) Data gathering: 먼저 Data gathering은 seq2seq model이 valuable seed corpus를 원활하게 추론할 수 있도록 데이터셋을 확보하기 위한 단계이다. AFL과 같은 기존의 fuzzer에 Initial seed seed corpus를 입력값으로 하여 fuzzing을 수행하고 unique crash나 새로운 path가 트리거 되는 등의 행위가 발생하면, fuzzing 수행 중 mutation 전/후 값을 하나의 데이터 쌍으로 사용하여, 학습에 사용될 데이터셋을 수집한다. 본 논문에서는 이 단계에서 수행하는 fuzzing을 pre-fuzz라고 정의하며, Data gathering은 DDRFuzz가 valuable seed corpus를 잘 생성할 수 있도록 학습 데이터를

수집하는 데만 사용되며 (pre-fuzz는 여러 응용 프로그램을 fuzzing하기 위한 seed corpus를 생성하는데 사용할 수 있음), 자세한 내용은 섹션 3.2에서 설명한다.

2) Seed generation: Seed generation은 valuable seed corpus를 생성하기 위한 단계로, 앞선 단계에서 수집한 데이터셋을 사용해서 seq2seq model을 학습하고, 학습된 model을 통해 fuzzing에 사용될 valuable seed corpus를 추론한다. 우리는 수집된 데이터를 deep learning model의 입력값으로 사용하기 위해 원시 데이터 형식을 embedding vector로 인코
3) 딩하는 embedding 방법을 제안한다. 자세한 내용은 섹션 3.3에서 설명한다.
4) Fuzzing: 학습된 seq2seq model에서 생성된 valuable seed corpus를 fuzzer(예: AFL)의 입력값으로 사용하여 대상 응용 프로그램을 fuzzing 한다. 또한 여기서 대상 응용 프로그램의 변화를 모니터링하며 unique crash나 새로운 path가 트리거 되는 등의 행위가 관찰되면, Data gathering 때와 마찬가지로 데이터 pair를 기록할 수 있으며, 이는 더 정교한 valuable seed corpus 생성을 위해 재학습 때 사용할 수 있다. 자세한 내용은 섹션 3.4에서 설명한다.

### 4.2 Data gathering

본 연구에서는 unique crash 또는 새로운 path를 트리거하는 입력값과 행위를 각각 'interesting seed'와 'interesting behavior'로 정의하였고, 높은 확률로 unique crash 혹은 path를 트리거 가능한 입력값을 'valuable seed'로 정의하였다.

우선, Valuable seed corpus를 생성하는 seq2seq model을 학습시키기 위해서는 빠르게 crash와 path를 탐지할 수 있는 seed corpus 데이터셋을 확보하는 것이 중요하다. 먼저, 본 논문에서는 공개되어 있는 확장자 별 seed corpus를 수집하였다. 하지만 공개되어 있는 seed corpus는 학습 데이터로 사용할 만큼 충분하지 않고, 어떤 입력값이 interesting seed인지 확인하기 어렵기 때문에 그대로 사용하기에는 한계점이 존재한다. 따라서 공개되어 있는 seed corpus를 입력값으로 하여 pre-fuzz를 수행하고, fuzzing 수행 중 interesting behavior 발견 시 mutation 되기

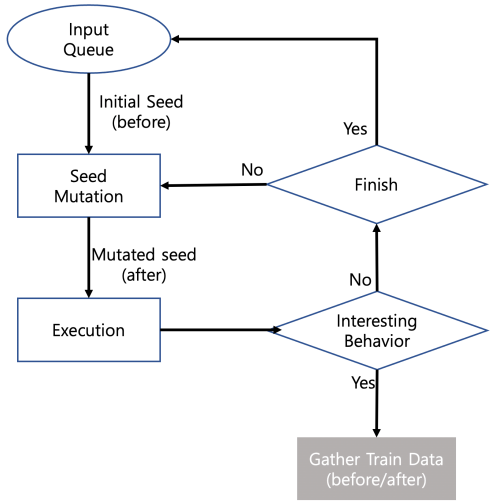


Fig. 5. Gathering process for training data

전/후의 입력값을 하나의 쌍으로 수집하여 데이터셋을 구축한다. 이러한 과정을 통해 interesting seed를 정확하게 식별할 수 있다.

Fuzzing 에서는 interesting seed와 interesting behavior가 중요한 의미를 갖는데, 그 이유는 첫째, 기존 연구 [23], [24], [25]에 따르면, coverage와 fuzzing path의 깊이를 늘리면 탐색된 crash 횟수가 증가할 가능성이 높다. 즉, fuzzing을 수행할 때 입력값은 넓은 범위의 code coverage를 가지거나 여러 조건문에 감싸져 있는 코드 영역에 도달하여 취약점을 트리거하는 등의 깊은 path를 탐색할 수 있는 값이 효율적인 입력값이라고 할 수 있다. fuzzer가 효율적인 입력값을 생성하기 위해서는 seed corpus가 굉장히 중요하며, 좋은 seed corpus는 fuzzer의 mutation을 최소화하여 원하는 path에 도달할 수 있으므로, 이러한 valuable seed corpus를 생성하기 위한 관점에서 새로운 path를 찾는 것은 fuzzer가 넓은 범위의 code coverage를 가질 수 있는 mutation을 수행했다라고 판단할 수 있으므로, fuzzer의 input queue에서 입력으로 사용된 값의 mutation 전/후 값에 대해서 태깅(tagging)을 수행하여 기록한다. 둘째, fuzzing의 궁극적인 목표는 더 많은 crash를 탐지하는 것이다. 따라서 입력값이 crash를 유발한다면, fuzzer가 프로그램 내에서 비정상적으로 메모리에 접근하여 예외를 발생시키는 수 있는 mutation을 수행했다라고 판단할 수 있으므로, 이러한 행위에 대해서도 mutation 전/후 값에 대해서 태

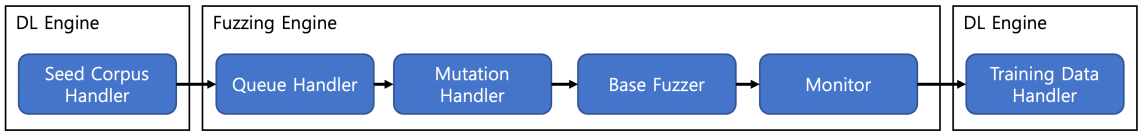


Fig. 6. Data gathering step

깅을 수행하여 기록한다.

Fig. 5.는 본 연구에서 데이터셋을 수집하기 위해 사용한 프로세스이다. 본 연구에서는 학습 데이터를 얻기 위해서 crash 또는 새로운 path를 트리거하는 interesting seed의 mutation 쌍을 저장하는 것에 주목했고, 인터넷을 통해 공개되어 있는 seed corpus를 수집하여 initial seed corpus를 구성한 후 pre-fuzz를 수행하였다. 단, 학습 데이터로 사용하기 위해서는 mutation 전/후 데이터 모두 필요하므로, 새롭게 crash 및 path가 탐색되는 행위인 interesting behavior가 발견될 경우, mutation 이전의 데이터와 이후의 데이터를 기록할 수 있도록 fuzzer(예:AFL)을 수정하였다.

본 연구에서 제안하는 학습데이터 수집 방식은 다음과 같은 이점이 있다. 첫째, 입력값 효율을 정확하게 평가할 수 있다. 학습에 사용되는 interesting seed corpus은 확실하게 fuzzing 대상 프로그램의 새로운 path 및 crash를 트리거할 수 있으므로, 이는 model이 생성하는 valuable seed corpus도 새로운 path 및 crash를 트리거 하는데에 큰 영향을 미친다. 둘째, 위와 같은 seed generation 전략은 매우 쉽고 빠르게 적용할 수 있다. 예를 들어, interesting behavior를 판단하는 기준으로 본 연구에서는 crash 또는 새로운 path를 트리거하는 것으로 하였는데, 이러한 판단 기준으로 변경하거나 확장하더라도 이러한 전략을 사용하는fuzzer를 간단하게 수정하여 적용 가능하다. 또한, parallel fuzzing을 진행할 경우 학습 데이터 수집을 훨씬 빠르게 수행할 수 있다. 예를 들어 수정된 fuzzer를 통해 parallel fuzzing을 진행할 경우, 특정 파일 형식에 대해서 일주일 내에 20,000개 이상의 mutation 쌍을 보다 빠르게 수집할 수 있다. 따라서 valuable seed corpus를 생성하기 위한 데이터셋을 빠르게 확보할 수 있고 적용도 어렵지 않다.

Fig. 6.은 위에서 설명한 Data gathering 단계를 통해 DDRFuzz가 데이터셋을 수집하기 위한 전체적인 절차를 나타냅니다.

### 4.3 Seed generation

본 연구에서는 valuable seed corpus 생성을 위해 seq2seq model을 채택했다. fuzzing 주요 목적인 crash를 빠르게 찾기 위해서는 mutation을 최소화하는 것이 중요하며, 이를 위해서 mutation은 효율적으로 수행되어야 한다. 본 논문에서는 interesting behavior을 발생시킬 수 있는 mutation에 초점을 두어, 입력값이 mutation 과정을 통해 생성된 출력값이 interesting behavior를 발생하게 되면 이를 mutation 전/후(input/output) 형태로 학습 데이터를 수집하였다. 이러한 input/ouput 형태의 학습 데이터는 NLP의 question/answering 문제와 동일하다고 판단하여 seq2seq model이 적합하다고 판단하였다.

Valuable seed corpus를 생성하는 seed generation 단계는 seq2seq model을 학습시키는 과정과 valuable seed corpus를 추론하는 과정 두 가지로 구분한다. 학습 과정에서는 앞서 수집한 데이터셋을 활용하여 mutation 전 데이터를 입력값, mutation 후 데이터를 answer로 사용하여 seq2seq model을 학습한다. 이 과정에서 목적은 높은 code coverage를 가지거나 crash를 잘 찾을 수 있도록 하는 valuable seed corpus를 생성하는 것이기 때문에 정확도가 높다고 무조건 좋다고 할 수는 없다. 실험 결과 섹션에서 model별 정확도에 따른 상세한 실험 결과와 의미를 설명하도록 한다. model 학습이 완료되면 valuable seed corpus를 추론하는 과정을 수행한다. 즉, 추론에 사용할 입력값을 학습된 seq2seq model의 입력으로 넣어서 fuzzing 단계에서 사용할 valuable seed corpus를 생성한다.

Fig. 7.는 Seed generation의 전체적인 절차를 나타낸 그림이며, 자세한 설명은 아래 3.3.1, 3.3.2에서 한다.

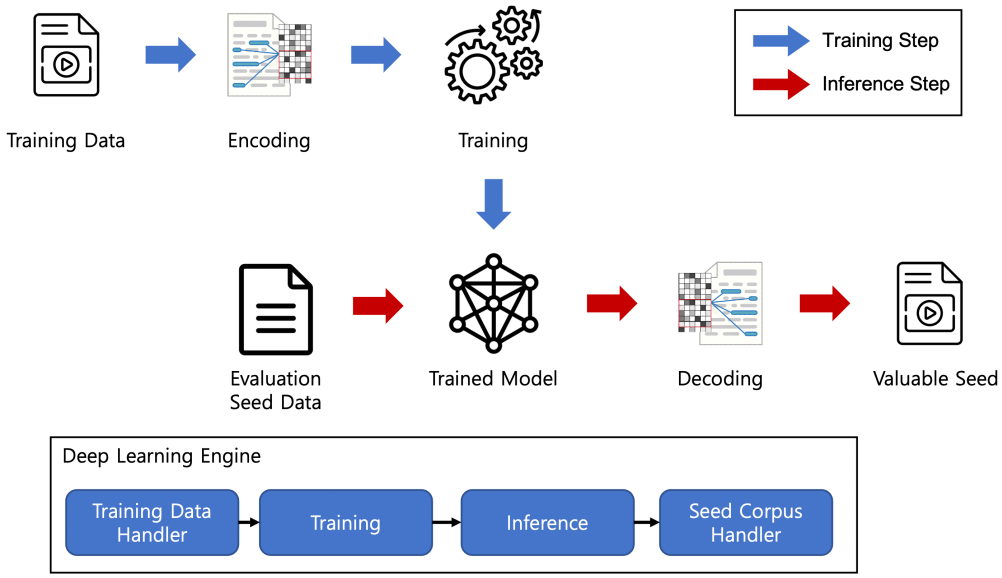


Fig. 7. Seed generation step

#### 4.3.1 Training Step

Bohme에 [23] 따르면 corrupted 파일이 unique crash 및 crash를 발견할 가능성이 더 크다고 한다. 하지만 이러한 seed corpus를 오픈 소스를 통해 많은 양을 확보하기에는 제한이 있다.

따라서 본 연구는 interesting behavior를 발생시키는 입력 값을 mutation 전/후 data 형태로 학습 데이터를 수집하였고, 이를 통해 mutation 과정의 knowledge를 학습할 수 있다. 이러한 seq2seq model을 학습하면 interesting behavior가 발생할 가능성이 높은 valuable seed corpus를 생성할 수 있을 것이라고 가정하고 연구를 진행했고, 위와 같은 가정을 증명하기 위해 실험을 통해 SOTA 연구보다 crash 및 coverage 측면에서 우수한 성능을 보이는 것을 확인하였다.

Training Step은 seq2seq model의 입력값으로 사용하기 위해 embedding vector로 생성하는 과정과 seq2seq model을 생성하는 과정으로 구분한다.

Encoding embedding vector는 앞서 확보한 seed corpus를 deep learning model의 입력값으로 사용하기 위해서는 다음과 같은 문제점을 해결해야 한다. DDRFuzz가 처리해야 하는 입력값의 format과 길이는 매우 다양하며, 이러한 다양한 파일의 format과 길이에 따라 입력값을 처리하는 routine을 조정하는 것은 어려우므로, 학습 데이터셋을 균일하게 처리할 수 있는 방법을 찾아야한다. 마지막으로 학습 데이터셋은 binary 형태로 표현되어 있다. Deep learning model은 입력 데이터를 vector로 표현되어야하므로 uniform vector matrices로 변환되어야 한다.

위에서 설명한 문제점을 해결하기 위해 본 연구에서는 word embedding을 이용한 방법과 Base64를 이용한 방법을 고려하였다. 먼저, word embedding 방식은 입력값의 context 정보를 고려한 encoding vector를 얻을 수 있다는 장점이 있으나, vocabulary in corpus의 크기가 충분하지 않은 경우 OoV (Out of Vocabulary) 문제가 발생할 수 있다. Base64 encoding 방식을 사용하면 입력값의 context 정보를 고려하지 않는다는 단점이 있으나, binary format의 학습 데이터를 간단하게 vector로 표현할 수 있고, encoding/decoding이 쉽다는 장점을 가지고 있다. 따라서 입력값의 context 정보가 중요하지 않은 format을 갖는 대상 프로그램의 경우에는 Base64 encoding을 사용하는 것을 권장하고, format의 context 정보가 중요한 format을 갖는 대상 프로그램의 경우에는 word embedding 방식을 사용하는 것을 권장한다. 본 연구에서는 입력값이 binary 형태의 멀티미디어 파일이므로 token 단위로 구분하거나 의미 정보를 포함하고 있지 않으므로 word

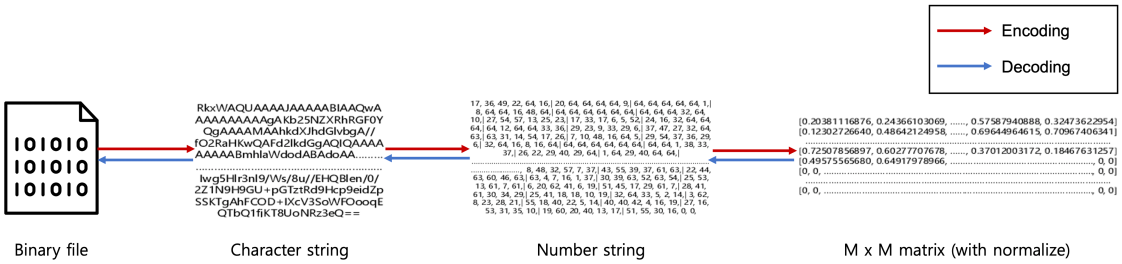


Fig. 8. Binary ecndoing process

embedding 방식 보다는 Base64 방식이 적합하다고 판단하였다.

Fig. 8.은 학습 데이터를 embedding vector 인코딩 하는 방식을 나타낸 그림이며, 다음과 같은 절차로 진행된다.

1) 모든 학습 데이터셋 파일은 binary 형식이기 때문에 이를 ASCII 문자열로 변환한다.
2) Base64를 사용하는 경우, ASCII 문자열을 Base64 format으로 변경하여 0 ~ 64개의 다른 문자로 표현할 수 있으므로 일련의 문자열을 얻을 수 있다. 그런 다음 model 학습의 정확성과 효율성을 위해 각 숫자를 정규화한다.
3) 고정된 길이를 맞춰주기 위해 최대 길이를 정하고 최대 길이보다 작은 파일은 제로 패딩 방식을 사용하여 패딩을 수행하고, 최대 길이보다 큰 파일은 잘라 낸다. 멀티미디어 파일의 경우 일반적으로 파일의 앞부분에 메타 데이터 등의 중요한 정보가 있으므로, 파일의 뒷부분에 패딩과 잘라내기를 수행하는 post padding/cut off 방식을 사용한다.

이러한 방법을 활용하여 파일 형식이 다르거나 파일 크기가 고정되지 않는 binary 파일들을 동일한 유형의 matrices로 변환할 수 있으며, 이러한 변환은 DDRFuzz의 확장성을 크게 향상 시킬 수 있습니다.

Seed generation은 기존의 seed corpus를 이용해서 새로운 valuable seed corpus를 생성하는 것이 목표이므로, 데이터 augmentation을 목적으로 oversampling 기법인 SMOTE, ADASyn 등의 방법을 사용하거나, GAN based generative model을 사용하는 것을 고려할 수 있다. 위의 방법은 입력 데이터와 유사한 데이터를 생성하는 것을 목표로 하므로 데이터 augmentation 목적으로는 우수한 성능을 나타낸다. 하지만 seed generation은 일반적인 데이터 augmentation과는 다른 특성을 고려해야 하므로, 기존의 데이터 augmentation 방법은 seed generation 목적으로는 적합하지 않다. 즉, seed generation은 입력값의 통계적인 특성을 고려하는 것보다는, exploitation과 exploration을 모두 고려하는 generation 전략이 필요하다. 여기서 exploitation은 입력값인 interesting seed corpus와 유사한 seed corpus를 만들어 비슷한 crash 및 path를 집중적으로 유도하는 특성을 의미하고, exploration은 다양한 종류의 seed corpus를 생성하여 검증 대상 프로그램의 다양한 영역을 검증하도록 유도하는 특성을 의미한다. SMOTE, ADASyn 그리고 GAN based generation model은 입력값과 유사한 데이터 augmentation을 수행하므로 exploitation 측면에서는 만족하지만, exploration 측면에서는 한계점을 가지므로, 본 연구에서는 생성된 seed corpus가 exploitation과 exploration 측면을 모두 만족할 수 있도록 seq2seq model을 채택하였다.

본 연구에서 우리의 초점은 seq2seq model 기반의 효율적인 fuzzing framework를 구축하고 그 효과를 입증하는 것이므로, 개선된 seq2seq model의 개발 연구는 향후 연구로 남겨둔다. 또한 DDRFuzz는 사용자가 application scenario에 맞추어 alternative deep learning model을 seed generation model로 선택할 수 있도록 설계하였다.

#### 4.3.2 Inference Step

이 섹션에서는 앞선 training step에서 학습된 model을 사용하여 더 효과적인 seed corpus를 생

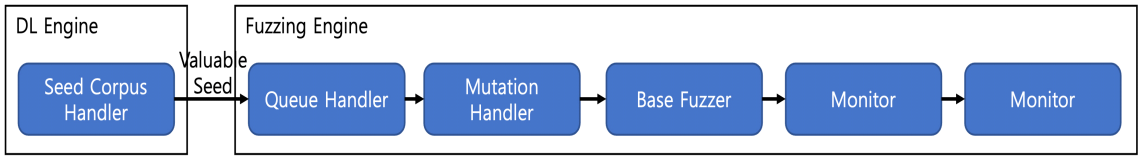


Fig. 9. fuzzing phase

성하는 방법을 소개한다. DDRFuzz의 학습 데이터는 Base64 format의 vector 형태이기 때문에 seq2seq model은 66개 항목으로 (64개의 Base64 format, 1개의 EOF, 그리고 1개의 패딩 구분자) 구성되는 vector 형태로 추론하도록 훈련된다. 따라서 fuzzing을 위한 binary 형태의 입력값을 얻으려면 생성된 vector 형태를 binary 파일로 변환해야 한다. 따라서 섹션 3.2에서 언급한 encoding embedding vector 생성 절차의 역순으로 decoding을 수행한다.

구체적으로, 첫 번째 단계는 패딩 값을 찾아 제거한다. 본 연구에서는 post padding을 사용하므로 출력값의 마지막부터 순차적으로 패딩 값을 제거하고, EOF 값을 만날 때까지 반복한다. 그런 다음 비정규화를 통해 66개의 정수를 구하는데 여기서 실수의 경우 소수점은 그대로 잘라서 정수로 변환한다. 둘째, 0 ~ 64 까지의 십진수 값을 Base64 문자로 변환하고 binary로 decoding하여 저장한다. 그 결과 우리는 fuzzing에 입력값으로 사용할 수 있는 파일을 얻을 수 있다. Fig. 4.에서 추론 단계를 통해 DDRFuzz가 seed corpus를 생성하기 위한 절차를 알 수 있다.

### 4.4 Fuzzing

앞선 seed generation을 통해 valuable seed corpus가 준비되면 fuzzing을 수행한다. Fuzzing 단계에서는 seed corpus를 fuzzer(예:AFL)의 입력값으로 사용하며, 여기서 valuable seed corpus는 fuzzer에 관계없이 AFL이나 libfuzzer 등 다양한 fuzzer에 걸쳐 사용할 수 있으므로 seed generation에서 생성된 valuable seed corpus는 fuzzing의 확장에 용이하다는 장점을 갖는다.

앞선 과정을 통해 자동으로 생성된 많은 양의 seed corpus들은 중복 fuzzing을 유발할 수 있으므로 seed trimming을 적용하여 유효한 seed corpus를 선택해야 한다. 즉, 동일한 coverage를 생성하는 용량이 큰 seed corpus를 사용하면 fuzzer의 성능이 저하된다. AFL은 이를 해결하기 위해 c-min 기능을 제공하며, c-min은 Fuzzing

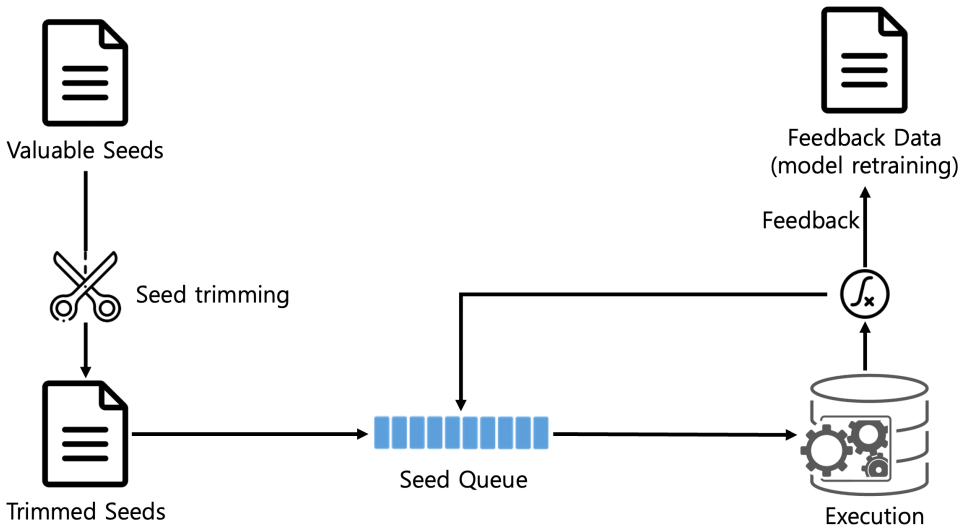


Fig. 10. DDRFuzz feedback mechanism

실행 전에 AFL의 dry run 수행 결과를 기반으로 중복되는 seed corpus를 제거한다. 이러한 기능을 통해 code coverage가 최대한 겹치지 않는 최소한의 seed corpus를 만들어 fuzzing 효율을 높일 수 있다. 따라본 본 연구에서는 AFL의 seed 간의 중복 최소화를 지원하는 AFL-cmin tool을 사용하여 seed trimming을 수행한다.

Fig, 9.에서 볼 수 있듯이 DDRFuzz를 실행하는 동안 interesting behavior를 유발하는 데이터 쌍을 기존 학습 데이터와 합쳐서 seq2seq를 재학습할 수 있다. 그렇게 하면 seed generation에서는 보다 더 정교한 valuable seed를 생성할 수 있지만 너무 과도하게 feedback mechanism을 사용하면 데이터 분포가 편향되어 overfitting 이 발생할 수 있는 문제가 있다.

Fig. 10.는 Fuzzing 단계에서 위에서 설명한 DDRFuzz가 valuable seed corpus를 활용하여 대상 프로그램을 fuzzing을 수행하는 전체적인 절차를 나타낸다.

## V. 실험 결과

### 5.1 평가 지표

DDRFuzz를 평가함에 있어 fuzzing 분야에서도 널리 사용되는 평가 metrics을 활용하였으며, 자세한 측정 항목은 아래와 같다.

- Code coverage: Code coverage는 fuzzing 효율을 측정하는 가장 일반적인 지표이며, fuzzing의 대상으로 하는 프로그램에서 실행된 code 문장의 수를 의미한다. 각 프로그램에 대해서 fuzzing 후 gcov 도구를 사용하여 coverage 결과를 계산하며, code coverage는 afl의 path counter 기반으로 측정하였다.
- Crash count: Crash count는 fuzzing 실행 중 대상 프로그램의 crash 수를 나타내며, code coverage와 함께 가장 많이 사용되는 fuzzing 성능 평가 지표이다.
- Vulnerability: 취약점을 발견하는 능력을 평가하기 위해 오픈 소스 프로젝트를 선정하여 취약점 탐지 결과를 분석했다.

### 5.2 실험 설정

DDRFuzz의 효율성을 평가하기 위해 4가지 평가 항목을 구성하였으며, 평가 항목은 아래와 같다.

- seed corpus 생성에 적합한 model 비교 및 평가: 본 연구는 효율적인 fuzzing을 위한 valuable seed corpus를 생성하기 위해 다양한 seq2seq model을 적용하였다. 이에 어떤 seq2seq model이 seed generation에 가장 효과적인 model인지 비교 분석을 먼저 수행한다. (섹션 참고 4.3)
- 자동으로 생성된 seed corpus의 효율성 평가: DDRFuzz의 효과성을 평가하기 위해 Initial seed corpus를 이용한 fuzzing과의 비교 분석과 기존 연구들과의 비교 분석을 진행한다. (섹션 참고 4.4)
- 각 model의 정확도에 따른 성능 비교 평가: Seed generation 측면에서는 높은 정확도가 반드시 높은 fuzzing 효율성 (성능)을 의미하지는 않는다. 따라서 생성된 seed corpus가 exploitation과 exploration 측면을 모두 만족하는 최적의 model 정확도 평가하기 위해 각 model의 정확도 fuzzing 성능을 비교 분석한다. (섹션 참고 4.5)
- 취약점 탐지 성능 평가: 마지막으로 DDRFuzz의 취약점 탐지 능력을 평가하기 위해 오픈 소스 프로젝트에 대한 Known/Unknown vulnerability 분석을 진행한다. (섹션 참고 4.6)

#### 5.2.1 Dataset

멀티미디어 프로젝트의 경우, 초반에 위치하는 입력에 대한 파일 format 검사를 통과하지 못한다면, 그 뒤에 기능을 제공해주는 main 코드 영역까지 도달할 수 없다. 따라서, seed corpus에 대한 중요성이 비교적 높은 수준으로 요구되기에 멀티미디어 프로젝트를 실험 대상으로 선정하였다. 그 중에서도 본 연구에 적합한 프로젝트를 선정하였으며, 그에 대한 선정 기준은 아래와 같다.

- 오픈소스 프로젝트에서 자체적으로 실행 파일 제공 여부
- ASAN 적용을 위한 32bit 빌드 가능 여부
- AFL 2.52b 버전을 활용하여 대상 프로젝트

Table 1. Project list and Dataset

| Multi media Type | Project | Version | Input Form at | Train Data set |
|---|---|---|---|---|
| Image | libavif<br>libtiff | v0.9.3<br>v4.0.9 | AVIF<br>TIFF | 3.9k<br>1.0k |
| Audio | mpg123<br>ffmpeg | v.1.26.9<br>v.4.4.1 | MP3<br>MP3 | 4.5k<br>4.5k |
| Video | libavc | android-1<br>2.0.0 r21 | H264 | 4.8k |

fuzzing 가능 여부 (coverage 증가 및 queue update)

- 적절한 입력값의 길이

프로젝트에서 자체적으로 실행 파일을 제공해주지 않는 경우, 해당 프로젝트에 대한 실행 파일을 직접 생성해 줘야 하기에 배제하였다. 또한 crash 분석을 위하여 필요한 ASAN은 32bit만을 지원하기에, 32bit 바이너리를 제공해 주지 않는 프로젝트도 배제하였다. 또한 실험에서 AFL 2.52b 버전을 사용하였으며, 해당 버전의 AFL fuzzing이 제대로 적용되지 않는 경우의 프로젝트도 배제하였다. 마지막으로 필요로 하는 입력값의 길이가 기본적으로 너무 긴 경우, 추후에 사용할 DDRFuzz의 필요 layer가 늘어난다는 문제점으로 배재하였다. 또한 model 학습은 한번만 수행하는 것이고 fuzzing은 지속적으로 수행하는 관점에서 학습에 오버헤드가 있다고 하더라도, 한번 학습하면 이후는 fuzzing만 수행하면 되므로, model 학습에 대한 오버헤드는 측정하지 않았다. 프로젝트는 입력의 유형에 따라 크게 3가지로 분류할 수 있으며, 각각 사진, 음성, 영상이다. 총 5개의 프로젝트를 선정하였으며, 프로젝트에서 입력으로 사용되는 format 유형은 4 가지 (avif, tiff, mp3, h264)이다. 따라서 각 입력 format 별 Initial seed corpus를 수집하고 그중 일부를 model 학습에 사용한다. 위와 같은 조건들을 충족하는 프로젝트와 구체적인 내용은 Table 1.과 같다.

#### 5.2.2 Implementation

실험을 위하여 사용한 PC 및 사용 프로그램은 Table 2.와 같다. 호스트로 다음과 같은 PC를 사용하였으며, fuzzing 대상 파일을 생성할 때 사용한 AFL, Initial seed corpus를 생성할 때 사용한 AFL4DDRFuzz, model 학습 및 추론에 사용한 tensorflow-gpu 및 Cuda 관련 버전 정보이다.

Table 3.은 model의 성능을 향상시키기 위하여 각 model 별 hyper parameter에 대한 최종 fitting 값을 정리한 표이며, parameter 별 의미는 다음과 같다. max length는 model의 입력값의 최대 길이를 의미 하며, 이는 곧 입력 vector의 layer 개수와 연결되므로 너무 길지 않으며, 최대한의 성능을 내도록 fitting 하였다. epoch은 전체 데이터셋에 대하여 학습을 완료한 횟수를 말하며, overfitting되지 않도록 early stop 시키도록 하였다. batch size는 전체 데이터셋에 대하여 한번에 학습을 시킬 수 없기 때문에, 이를 일정 크기로 나눈 것이며, 너무 작은 batch size는 model의 성능에 영향을 줄 수 있기 때문에 메모리 크기에 적합한 batch size를 선정하였다. embedding dim은 사용하는 vector의 크기를 줄이고, dense하게 표현하기 위한 embedding dim(dimension)을 fitting 하였다.

Table 2. Experiment Environment

| Categorty | Component | Details |
|---|---|---|
| PC | CPU<br>GPU<br>Memory | Intel(R) Core(TM) i7-10700k CPU @ 3.80GHz<br>NVIDIA GeForce RTX 2080 SUPER<br>64GB |
| Program | Python<br>tensorflow-gpu<br>AFL<br>AFLDDRFuzz<br>Cuda<br>CuDNN | Python 3.9.6<br>2.3.0<br>2.5.2b<br>https://github.com/onsoim/AFL4DDRFuzz<br>v10.1.243<br>8.2.0 |

Table 3. model parameters

| Model | Parameter | libavif | libavc | libtiff | mpg123 | ffmpeg |
|---|---|---|---|---|---|---|
| Simple-seq2seq | Max length | 1000 | 1000 | 1000 | 1000 | 1000 |
| | batch size | 8 | 8 | 8 | 16 | 16 |
| | epoch | 30 | 30 | 150 | 1000 | 1000 |
| | embedding dim | 64 | 64 | 128 | 128 | 128 |
| seq2seq-attention | max length | 1000 | 1000 | 1000 | 1000 | 1000 |
| | batch suze | 8 | 8 | 16 | 16 | 16 |
| | epoch | 30 | 30 | 160 | 1000 | 1000 |
| | embedding dim | 64 | 64 | 128 | 192 | 192 |
| transformer | max length | 1000 | 1000 | 1000 | 1000 | 1000 |
| | batch size | 8 | 8 | 16 | 16 | 16 |
| | epoch | 30 | 30 | 150 | 200 | 200 |
| | embedding dim | 64 | 64 | 128 | 128 | 128 |

### 5.3 Seed corpus 생성에 적합한 model 비교 및 평가

본 실험에서는 DDRFuzz에서 사용하는 3가지 model (Simple-seq2seq, seq2seq-attention, 그리고 transformer)에 대한 성능 비교를 통해 어떤 model이 seed generation에 가장 효과적인 model인지 평가하는 것을 목표로 한다. 총 4개의 확장자에 대해서 3개의 model을 통해 생성된 seed corpus를 활용하여 프로젝트 별로 6시간 동안 fuzzing을 수행하여, 새로운 path 개수와 crash 개수를 비교한다. Table 3.은 각 model의 성능에 큰 영향을 줄 수 있는 hyper-parameters를 정리한 표이며, 여러 조합을 사용하여 최적의 pre-training 결과를 가지는 값을 선정하였다. 앞으로의 실험들은 아래의 model들은 위와 Table 3.을 사용하여 학습한 model을 사용하였다.

Table 4.는 각 프로젝트 별 뛰어난 성능을 보인 model을 정리한 표이다. Table 4.에서 볼 수 있는 바와 같이 특정 과제별 성능이 가장 뛰어난 model에 대한 선택을 하는 것은 어렵지만, 입력값의 길이를 기준으로 정렬을 해보면 850 byte 이하의 과제에서는 seq2seq-attention model이, 850-1000 byte인 경우에는 simple-seq2seq model이 1000 byte를 초과하는 경우에는 transformer model의 성능이 제일 좋을 것을 확인할 수 있다. 이는 들어오는 입력값의 길이가 길어짐에 따라 vanishing gradient 문제로 인하여 정보가 더 잘 보존될 수 있는 transformer model의 성능이 제일 좋게 나오고, 그 이외에는 simple-seq2seq model 또는 seq2seq-attention model의 성능이 잘 나옴으로 판단된다.

### 5.4 자동으로 생성된 seed corpus의 효율성 평가

본 실험에서는 DDRFuzz를 통해 생성된 valuable seed corpus의 효율성을 확인하기 위해 original seed corpus와 관련 연구와의 비교 분석

Table 4. Comparison of Crash and Coverage between seq2seq models

| Project | Input length (Byte) | Simple-seq2seq | | seq2seq-attention | | transformer | |
|---|---|---|---|---|---|---|---|
| | | Crash | Path | Crash | Path | Crash | Path |
| libavif | 735.1 | 0 | 4006 | **0** | **4458** | 0 | 2400 |
| libtiff | 830.2 | 30 | 268 | **31** | **284** | 25 | 179 |
| ffmpeg | 942.3 | 5 | **2968** | 4 | 2773 | 2 | 2084 |
| mpg123 | 1979.9 | 0 | 1701 | 0 | 1740 | **0** | **1840** |
| libavc | 3753.6 | 0 | 6661 | 0 | 7083 | **0** | **7853** |

을 진행한다. DDRFuzz model은 앞선 섹션 4.3에서 가장 좋은 성능을 내는 model을 사용한다. 공정한 비교를 위해 original seed corpus는 Data gathering 시 initial seed corpus로 사용했던 seed 그대로 사용하고 이 중에는 OSS-Fuzz에서 사용하는 seed corpus도 포함되어 있다. 관련 연구와의 비교 분석은 섹션 2.1에서 언급한 바와 같이 기존 ML 기반의 seed generation 연구 중 현재 최고 수준인 (SOTA) SmartSeed를 대상으로 한다. 단, SmartSeed의 경우 소스 코드가 공개되어 있지 않아서 논문에 설명되어 있는 내용을 기반으로 직접 구현 (WGAN) 후 비교한다. 다른 model과의 공정한 비교를 위해 SmartSeed model의 학습 데이터는 DDRFuzz 학습 시 사용했던 데이터를 동일하게 사용했다. 실험은 섹션 4.3과 동일하게 6시간 동안 fuzzing을 수행하여, 새로운 path 개수와 crash 개수를 비교한다. Table 5.를 보면 기존의 original seed corpus 대비 SmartSeed를 통해 생성된 seed corpus나 DDRFuzz를 통해 생성된 결과가 더 향상되었으며 DDRFuzz는 SmartSeed의 seed corpus보다 더 성능이 좋은 것을 알 수 있다. 무엇보다 ffmpeg 프로젝트를 보면 SmartSeed를 통해 생성된 seed corpus로는 crash를 찾지 못했지만, DDRFuzz를 통해 생성된 seed corpus로는 crash를 발견할 수 있음을 알 수 있다.

Table 5. Project list and Dataset

| Project | Origin | | SmartSeed | | DDRFuzz | |
|---|---|---|---|---|---|---|
| | Crash | Path | Crash | Path | Crash | Path |
| ffmpeg | 0 | 359 | 0 | 2135 | 5 | 2968 |
| mpg123 | 0 | 863 | 0 | 1599 | 0 | 1840 |
| libtiff | 7 | 182 | 23 | 236 | 31 | 284 |
| libavif | 0 | 2834 | 0 | 2895 | 0 | 4458 |
| libavc | 0 | 6673 | 0 | 7603 | 0 | 753 |

### 5.5 각 model의 정확도에 따른 성능 평가 비교

섹션 4.2에서 언급한 바와 같이 seed generation을 위한 seq2seq model의 경우 높은 정확도가 fuzzing 높은 성능을 의미하지 않을 수 있다. 따라서 본 실험에서는 생성된 seed corpus가 exploitation과 exploration 측면을 모두 만족하는 최적의 정확도는 무엇인지 평가하기 위해 각 model의 정확도 별로 fuzzing 성능을 비교 분석한다. 이번 실험에서는 crash의 개수는 너무 적어 무의미하기에 배제하고 line coverage만을 기준으로 진행하였다.

Table 6.을 보면 최적의 정확도는 과제별로 또 model 별로 상이하다. 다만 과제별로 최적의 정확도 범위를 특정할 수 있는데, ffmpeg의 경우 모든 model에서 85-90% 인 경우 line coverage의 최대값을 달성할 수 있으며, mpg123의 경우에는 80-85%, libtiff는 90%, libavif는 75%, libavc는 75%에서 각각 line coverage의 최대값을 달성할 수 있었다.

또한 Fig. 11.은 model 정확도 별 coverage 맵을 시각화한 그림이다. 그림을 보면 initial seed corpus를 입력값으로 사용한 A-I, B-I은 눈에 띄는 클러스터는 3개가 있는 반면 DDRFuzz를 통해 생성된 seed corpus를 입력값으로 사용한 A-II, B-II 같은 경우는 눈에 띄는 클러스터가 6개가 존재하고 클러스터의 농도도 더 진해져 있는 것을 확인할 수 있으므로, model의 정확도가 높을수록 exploitation 속성이 증가하고 정확도가 낮을수록 exploration 속성이 증가한다는 것을 알 수 있다. 따라서, seed corpus가 이미 대상 프로젝트의 다양한 영역에 도달하는 경우에는 정확도가 높은 model을 사용하는 것을 권장하고, seed corpus가 대상 프로젝트의 제한적인 영역에 도달하는 경우에는 exploration 속성을 강화하기 위해 정확도가 낮은 model을 사용하는 것을 권장한다. 즉, model들 보

Table 6. Line coverage performance by model accuracy

| Project | Simple-seq2seq | | | | seq2seq-attention | | | | transformer | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 75% | 80% | 85% | 90% | 75% | 80% | 85% | 90% | 75% | 80% | 85% | 90% |
| ffmpeg | 2533 | 2405 | **2968** | 2864 | 2354 | 2559 | 2721 | **2773** | 1357 | 1857 | **2084** | 2027 |
| mpg123 | 1617 | **1720** | 1701 | 1701 | 1491 | 1685 | **1740** | 1639 | 1583 | 1724 | **1840** | 1726 |
| libtiff | 237 | 227 | 264 | **268** | 139 | 271 | 262 | **284** | 125 | 163 | 166 | **179** |
| libavif | **4417** | 2714 | 1182 | 1182 | 4458 | **610** | 1379 | 2705 | **2400** | 2276 | 1634 | 2164 |
| libavc | **6661** | 5412 | 5660 | 5660 | 7083 | **5649** | 5328 | 6124 | **7853** | 7126 | 4974 | 3990 |

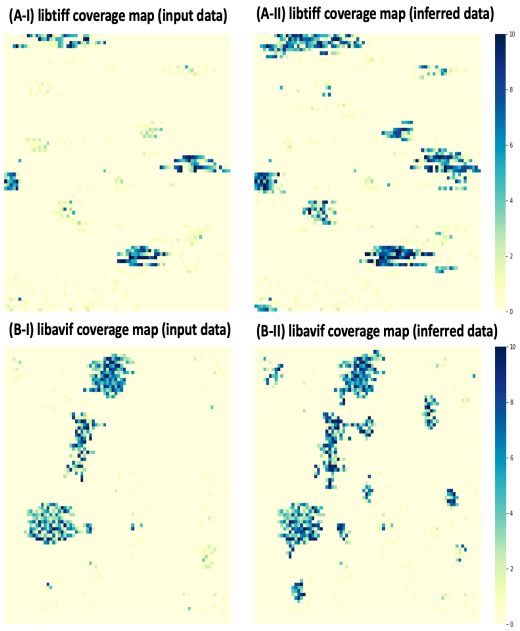


Fig. 11. Data distribution by accuracy

다는 과제별로 더 선호되는 정확도 범위가 있음을 알 수 있다.

### 5.6 취약점 탐지 성능

DDRFuzz의 취약점 탐지 능력을 검증하기 위해 실험에 사용한 5개의 오픈소스 프로젝트 중 가장 많은 crash가 발견되었고, 그간 많은 CVE가 제보된 libtiff를 선정하였다. 이 오픈소스 프로젝트를 선정한 이유는 그간 취약점이 많이 보고 되었음에도 불구하고 취약점이 지속적으로 발견되고 있는 만큼, 취약점이 많이 내재되어 있는 프로그램으로 가정할 수 있고, 기존의 연구들의 결과물과 DDRFuzz 의 성능 비교를 하기 용이할 것으로 판단했기 때문이다.
알려진 취약점을 탐지하는 능력을 평가하기 위해 알려진 CVE를 포함하는 이전 라이브러리 버전에서 실험을 진행하였다. 이미 알려진 취약점을 탐지할 수

Table 7. Known Vulnerability

| Project | Vul. Type | CVE ID |
|---|---|---|
| libtiff | Segmentation violation | (known) CVE-2014-9330 |
| libtiff | Heap-based buffer over-read | (known) CVE-2017-17942 |

Table 8. Unknown Vulnerability

| Project | Vul. Type | CVE ID |
|---|---|---|
| libtiff | Heap-based use-after-free | zero-day |

있는지 확인하기 위해 bmp 파일을 tiff 파일로 변환하는 모듈에서 실험을 진행하였고, Table 7.에 설명된 대로 알려진 취약점 2개를 탐지했다.
알려지지 않은 취약점도 알려진 취약점 탐지 실험과 동일하게 bmp 파일을 tiff 파일로 변환하는 모듈에서 실험을 진행하였다.

실험 결과 Table 8.에서와 같이 1개의 zero-day 취약점을 탐지하였으며, 해당 취약점은 해당 CVE mitre에 제보하였고, 개발자에게 취약점을 전달하여 패치될 예정이다. 또한 해당 취약점은 과거부터 지금까지 여러 사람들에 의해 취약점이 제보되어왔고, Google의 OSS-Fuzz에서도 지속적으로 fuzzing을 수행하고 있음에도 발견되지 않았고, SOTA 연구에서 진행되었던 SmartSeed에서는 발견되지 않았지만, DDRFuzz에서 의해 발견되었다는 점으로 보아 DDRFuzz의 취약점 탐지 능력은 충분히 의미가 있다고 할 수 있다.

### 5.7 실험 결과 요약

본 섹션에서는 DDRFuzz의 성능을 확인하기 위하여 다양한 조합의 실험을 진행하였다. line coverage 및 crash 즉 exploration과 exploitation 측면에서 최적의 조합을 실험을 통해 찾아보았다. 우선 model의 정확도가 무작정 높다고 좋은 것은 아니며, 각 프로젝트 (format) 별 선호하는 정확도 범위가 있음을 확인했다. 또한 model 역시 어느 것이 가장 좋다고 특정할 수는 없지만, 입력값의 길이에 따라 성능이 달라짐을 확인하였기에 사용하는 format과 그 입력이 되는 seed corpus의 평균 길이를 이용하여 model의 유형과 정확도 정도를 선정할 수 있다. 또한 다른 SOTA 연구들과 비교도 해본 결과 DDRFuzz에서 생성된 valuable seed corpus의 line coverage가 더 좋은 것을 알 수 있으며, OSS-Fuzz의 프로젝트로 선정되어 활발히 fuzzing이 진행됨에도 불구하고 OSS-Fuzz에서도 찾지 못한 취약점을 발견한 데에 큰 의의가 있다.

## VI. 향후 연구

본 연구는 다양한 fuzzing 기술에 광범위하게 적용 가능하고, 취약성을 감지하는데 효과적이지만, 향후개선할 여지는 여전히 남아 있다. 이 장에서는 현재 DDRFuzz에 대한 제한 사항을 작성하고 향후 연구를 위한 개선 사항에 대해 설명한다.

### 6.1 Valuable Seed의 확장

우리는 AFL을 수정하여 valuable seed corpus를 수집하고, 이를 deep learning model의 Dataset으로 활용한다. 새로운 path와 crash가 취약점을 탐지하는데 가장 중요한 요소이기는 하지만, 이 외에도 다양한 기준을 적용하여 valuable seed corpus의 수집을 확장할 수 있다. 예를 들어, crash 혹은 path를 트리거하지 않지만 이를 유발하게 하는 instruction으로 도달 할 수 있는 입력값은 취약점을 탐지할 가능성이 높다. valuable seed corpus의 수집 방법(도구) 또한 AFL 뿐 아니라 다양한 fuzzer를 이용할 수 있다. 특히 Vuzzer, T-Fuzz, Angora, Driller 등의 hybrid-fuzzer를 이용하면 Symbolic Execution이나 Taint Analysis 등의 기법을 적용하여 valuable seed corpus를 확장할 수 있다. 즉, valuable seed corpus의 수집 방법(도구)과 기준의 확장을 통해 좀 더 수준 높은 데이터셋을 확보할 수 있을 것으로 기대한다.

### 6.2 Seed selection에서 deep learning 기법 적용

본 연구는 효율적인 seed generation을 위해 deep learning approach를 적용하였다. 그러나 mutation 기반 fuzzer들의 seed corpus는 입력 범위가 지나치게 넓기때문에 어떤 seed를 입력값으로 사용하여 fuzzing을 해야 하는지에 대한 문제가 발생한다. 이러한 문제를 해결하기 위해 Revert et al. [2]에서는 coverage metric을 최대화하는 seed corpus의 최소 집합을 찾는 방식이 제안되었다. AFL fuzzer의 경우 pre-fuzz를 통해 fitness score를 측정하여 입력값의 우선순위를 부여하는 방식을 사용하고, fitness score가 동일한 경우 크기가 작은 입력값에 우선순위를 부여한다. DDRFuzz를 통해 생성된 seed corpus는 유사한 seed corpus가 많기 때문에 pre-fuzz를 통한 fitness score 방식의 Seed selection 기법은 효과적이지 않다. 따라서 Seed selection 방식에 deep learning 접근 방식을 적용하면 fuzzing의 효율성을 높일 수 있을 것으로 기대한다.

### 6.3 향상된 Feedback Mechanism 적용

본 연구는 더욱 정교한 valuable seed corpus 생성을 위해 Fuzzing engine에서 생성된 mutation 쌍 데이터셋을 다시 학습 데이터셋으로 활용하는 feedback mechanism을 적용하였다. 그러나 좀 더 개선된 feedback mechanism을 적용하기 위해서는 다음과 같은 사항을 고려해야 한다. 먼저, feedback을 통해 추가로 확보한 valuable seed corpus와 기존 valuable seed corpus와 중복 체크가 필요하다. 만약 중복된 valuable seed corpus에 대한 feedback을 누적하여 그대로 사용하면 dataset의 분포가 편향될 수 있다. 따라서 valuable seed corpus의 중복 체크가 필요하고, 중복 체크에는 유사도 기반과 seed trimming 기반 등 다양한 방법을 사용할 수 있다. 둘째로, 실시간으로 feedback mechanism을 적용하면 기존에 학습된 model에 점진적으로 learning을 수행하게 되므로 데이터가 누적될수록 과거의 데이터는 지워지고 신규 데이터 중심으로 결과값을 도출하는 Catastrophic Forgetting 문제가 발생할 수 있다. 따라서 이러한 오류를 해결하기 위해 continuous (incremental) learning 연구의 적용이 필요하다.

## VII. 결 론

Fuzzing은 취약점을 찾는데 효율적인 검증 방법 중 하나이며, fuzzing의 한계점을 개선하기 위해서 많은 연구가 진행되어 왔다. 특히 crash 트리거가 가능한 seed corpus가 fuzzing 효율성에 중요한 요소 하나라고 할 수 있다. 이에 dynamic taint analysis와 symbolic execution 기법 등을 적용하여 효율적인 seed corpus를 생성하는 연구들이 진행되었으나 기법의 복잡도가 높고 전문 지식이 필요함으로 인해 기존 연구들이 광범위하게 적용되지 못했다. 본 논문에서는 이러한 문제를 해결하기 위해 기존 fuzzer와 다양한 seed corpus 생성 기술을

분석했다. 또한 이들의 한계와 문제점을 보완하여 seq2seq model을 기반으로 seed corpus를 생성하는 시스템인 DDRFuzz를 제안하였고, 이는 우리가 아는 한 mutation-based fuzzing에서 seed augmentation을 위해 seq2seq model이 적용된 첫 번째 연구라 할 수 있다.

DDRFuzz는 Deep Learning Engine과 Fuzzing Engine으로 구성되어 있고, 먼저 학습 데이터 수집을 위해 AFL을 수정하여 새로운 crash와 path를 유발하는 seed corpus에 대해서 request/answer 데이터로 구분하여 수집하였고, Table 1.에 표시한 바와 같이 multimedia 파일 종류별로 학습 데이터를 수집하였다. Deep Learning Engine에서는 학습된 model을 통해 seed corpus를 augmentation하여 검증자가 프로젝트에 대한 고수준의 지식 없이 fuzzing을 수행할 수 있게 한다. Fuzzing Engine은 실제 fuzzing을 수행하는 engine이며, 본 연구진은 AFL을 사용했지만, 이를 제외한 여러 fuzzer (예:AFL)로 확장도 가능하다. 실험은 멀티미디어 파일을 입력값으로 하는 5개의 오픈소스 프로젝트에서 DDRFuzz를 평가했다. 본 연구를 통해 관련 연구와의 다양한 비교 실험에서 DDRFuzz가 coverage와 crash count 측면에서 가장 뛰어난 성능을 나타냄을 확인할 수 있었다. 또한 DDRFuzz 적용을 통해 3개의 취약점을 발견하였다. 이는 DDRFuzz가 취약점 탐지에 효과적임을 보여주는 결과이다. 마지막으로, 본 연구진은 DDRFuzz의 소스 코드를 GitHub에 공개하였고, 이를 통해 다른 연구자들이 본 연구의 결과를 재현하고, deep learning을 활용한 seed augmentation 연구를 확장할 수 있도록 하였다.

## References


[1] NVD, "CVE." https://cve.mitre.org/cve/, Accessed: Nov 2020.

[2] A. Rebert, S. K. Cha, T. Avgerinos, J. Foote, D. Warren, G. Grieco, and D. Brumley, "Optimizing seed selection for fuzzing", Proceedings of the 23rd USENIX Security Symposium, pp. 861-875, Aug. 2014.

[3] C. Lyu, S. Ji, Y. Li, J. Zhou, J. Chen, and J. Chen, "Smartseed: Smart seed generation for efficient fuzzing", arXiv, preprint arXiv:1807.02606, Jun. 2019.

[4] V. J. M. Man`es, H. Han, C. Han, S. K. Cha, M. Eg˪ele, E. J. Schwartz, and M. Woo, "The art, science, and engineering of fuzzing: A survey", IEEE Transaction on Software Engineering, pp. 2312-2331, Nov. 2021.

[5] K. Fang and G. Yan, "Emulation-instrumented fuzz testing of 4g/lte android mobile devices guided by reinforcement learning", European Symposium on Research in Computer Security, pp. 20-40, Sep. 2018.

[6] K. B¨ottinger, P. Godefroid, and R. Singh, "Deep reinforcement fuzzing", IEEE Security and Privacy Workshops (SPW), pp. 116-122, Oct. 2018.

[7] S. Karamcheti, G. Mann, and D. Rosenberg, "Adaptive grey-box fuzz-testing with thompson sampling", Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security, pp. 37-47, Oct. 2018.

[8] M. Rajpal, W. Blum, and R. Singh, "Not all bytes are equal: Neural byte sieve for fuzzing", arXiv, preprint arXiv:1711.04596, Nov. 2017.

[9] W. Drozd and M. D. Wagner, "Fuzzergym: A competitive framework for fuzzing and learning", arXiv, preprint arXiv:1807.07490, Jul. 2018.

[10] libFuzzer, "https://llvm.org/docs/LibFuzzer.html" accessed: Nov. 2020.

[11] X. Liu, R. Prajapati, X. Li, and D. Wu, "Reinforcement compiler fuzzing", arXiv, preprint arXiv:1801.04589, Jan. 2018.

[12] Z. Zhang, B. Cui, and C. Chen, "Reinforcement learning-based fuzzing

technology", International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 244 - 253, Jun. 2020.
[13] Y. Wang, Z. Wu, Q. Wei, and Q. Wang, "Neufuzz: Efficient fuzzing with deep neural network," IEEE Access, vol. 7, pp. 36340 - 36352, Mar. 2019.
[14] J. Wang, B. Chen, L. Wei, and Y. Liu, "Skyfire: Data-driven seed generation for fuzzing", 2017 IEEE Symposium on Security and Privacy (SP), pp. 579 - 594, Jun. 2017.
[15] P. Godefroid, H. Peleg, and R. Singh, "Learn&fuzz: Machine learning for input fuzzing", Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, pp. 50 - 59, Oct. 2017.
[16] L. Cheng, Y. Zhang, Y. Zhang, C. Wu, Z. Li, Y. Fu, and H. Li, "Optimizing seed inputs in fuzzing with machine learning", 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 244 - 245, Aug. 2019.
[17] L. Joffe, "Machine learning augmented fuzzing", 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 178 - 183, Oct. 2018.
[18] R. Fan and Y. Chang, "Machine learning for black-box fuzzing of network protocols", International Conference on Information and Communications Security, pp. 621 - 632, Apr. 2018.
[19] Z. Hu, J. Shi, Y. Huang, J. Xiong, and X. Bu, "Ganfuzz: a gan-based industrial network protocol fuzzing framework", Proceedings of the 15th ACM International Conference on Computing Frontiers, pp. 138 - 145, May. 2018.
[20] Z. Li, H. Zhao, J. Shi, Y. Huang, and J. Xiong, "An intelligent fuzzing data generation method based on deep adversarial learning," IEEE Access, vol. 7, pp. 49327 - 49340, Apr. 2019.
[21] N. Nichols, M. Raugas, R. Jasper, and N. Hilliard, "Faster fuzzing: Reinitialization with deep neural models", arXiv, preprint arXiv:1711.02807, Nov. 2017.
[22] AFLi, "American Fuzzy Loop." https://lcamtuf.coredump.cx/afl/, Accessed: Nov 2020.
[23] M. B¨ohme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox fuzzing as markov chain", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1032-1043, Oct. 2016.
[24] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "Vuzzer: Application-aware evolutionary fuzzing", NDSS, vol. 17, pp. 1 - 14, Feb. 2017.
[25] H. Peng, Y. Shoshitaishvili, and M. Payer, "T-fuzz: fuzzing by program transformation", 2018 IEEE Symposium on Security and Privacy (SP), pp. 697 - 710, May. 2018.

## 〈저자소개〉

김 동 영 (DongYoung Kim) 학생회원
2019년 2월: 학점은행제 정보보호학과 학사
2020년 9월~현재: 고려대학교 정보보호대학원 정보보호학과 석사과정
〈관심분야〉 Offensive Security, Vulnerability Analysis, Fuzzing

전 상 훈 (SangHoon Jeon) 학생회원
2006년 8월: 고려대학교 컴퓨터학과 학사
2011년 8월: 성균관대학교 디지털미디어통신공학 석사
2006년 7월~현재: 삼성전자 종합기술원 전문연구원
2019년 3월~현재: 고려대학교 정보보호대학원 정보보호학과 박사과정
〈관심분야〉 데이터 기반 취약점 및 악성코드 분석, 시스템 보안

류 민 수 (MinSoo Ryu) 학생회원
2020년 8월: 세종대학교 정보보호학과 학사
2020년 9월~현재: 고려대학교 정보보호대학원 정보보호학과 석사과정
〈관심분야〉 Effective Fuzzing, Vehicular Security, Reverse Engineering

김 휘 강 (Huy Kang Kim) 종신회원
1998년 2월: KAIST 산업경학학과 학사
2000년 2월: KAIST 산업공학과 석사
2009년 2월: KAIST 산업및시스템공학과 박사
2004년 5월~2010년 2월: 엔씨소프트 정보보안실장. Technical Director
2010년 3월~2014년 12월: 고려대학교 정보보호대학원 조교수
2020년 3월~현재: 고려대학교 정보보호대학원 교수
〈관심분야〉 온라인게임 보안, 자동차 보안, 침입탐지시스템, 네트워크 보안