Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.28, NO.6, Dec. 2018 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.6.1463

Variational Autoencoder를 활용한 필드 기반 그레이 박스 퍼징 방법

이 수 림,† 문 종 섭‡
고려대학교 정보보호대학원

A Method for Field Based Grey Box Fuzzing with Variational Autoencoder

Su-rim Lee,† Jong-sub Moon‡
Graduate School of Information Security, Korea University

요 약

퍼징이란 유효하지 않은 값이나 임의의 값을 소프트웨어 프로그램에 입력하여, 보안상의 결함을 찾아내는 소프트웨어 테스팅 기법 중 하나로 이러한 퍼징의 효율성을 높이기 위한 여러 방법들이 제시되어 왔다. 본 논문에서는 필드를 기반으로 퍼징을 수행하면서 커버리지, 소프트웨어 크래쉬와 연관성이 높은 필드가 존재한다는 것에 착안하여, 해당 필드 부분을 집중적으로 퍼징하는 새로운 방식을 제안한다. 이 때, Variational Autoencoder(VAE)라는 딥 러닝 모델을 사용하여 커버리지가 높게 측정된 입력 값들의 특징을 학습하고, 이를 통해 단순 변이보다 학습된 모델을 통해 재생성한 파일들의 커버리지가 균일하게 높다는 것을 보인다. 또한 크래쉬가 발생한 파일들의 특징을 학습하고 재생성 시 드롭아웃을 적용하여 변이를 줌으로써 새로운 크래쉬를 발견할 수 있음을 보인다. 실험 결과 커버리지가 퍼징 도구인 AFL의 큐의 파일들보다 약 10% 정도 높은 것을 확인할 수 있었고 Hwpviewer 바이너리에서 초기 퍼징 단계 시 발생한 두 가지의 크래쉬를 사용하여 새로운 크래쉬 두 가지를 더 발견할 수 있었다.

ABSTRACT

Fuzzing is one of the software testing techniques that find security flaws by inputting invalid values or arbitrary values into the program and various methods have been suggested to increase the efficiency of such fuzzing. In this paper, focusing on the existence of field with high relevance to coverage and software crash, we propose a new method for intensively fuzzing corresponding field part while performing field based fuzzing. In this case, we use a deep learning model called Variational Autoencoder(VAE) to learn the statistical characteristic of input values measured in high coverage and it showed that the coverage of the regenerated files are uniformly higher than that of simple variation. It also showed that new crash could be found by learning the statistical characteristic of the files in which the crash occurred and applying the dropout during the regeneration. Experimental results showed that the coverage is about 10% higher than the files in the queue of the AFL fuzzing tool and in the Hwpviewer binary, we found two new crashes using two crashes that found at the initial fuzzing phase.

Keywords: software testing, fuzzing, vulnerability, deep learning, VAE(Variational Autoencoder)

Received(09. 21. 2018), Modified(1st: 11. 02. 2018, 2nd: 11. 08. 2018)Accepted(11. 08. 2018)

† 주저자, ynca333@korea.ac.kr
‡ 교신저자, jsmoon@korea.ac.kr(Corresponding author)

I. 서 론

소프트웨어의 보안 취약점을 악용한 사이버 공격이 증가하면서 초기에 보안 취약점을 발견하여 대처하기 위한 보안 소프트웨어 테스팅의 중요성이 높아지고 있다. 퍼징(fuzzing)이란 유효하지 않은 값이나 임의의 값을 프로그램에 입력하여 보안상의 결함을 찾아내는 소프트웨어 테스팅 기법 중 하나로 대상 어플리케이션에 대한 구체적인 이해 없이도 취약점을 유발시킬 수 있다는 장점이 있다[1].

퍼징은 블랙박스 퍼징, 화이트박스 퍼징, 그레이박스 퍼징 3가지로 분류할 수 있다. 블랙박스 퍼징은 퍼징 대상 프로그램에 대한 분석 없이 단순히 랜덤하게 퍼징을 수행하기 때문에 가볍지만 커버리지(coverage)를 높이기는 힘들다. 이에 반해, 화이트박스 퍼징은 프로그램 소스코드에 대한 분석을 통해 퍼징을 수행하기 때문에 무겁지만 커버리지를 효율적으로 높일 수 있다는 장점이 있다[2]. 여기서 커버리지란, 소프트웨어의 테스팅을 논할 때 얼마나 테스트가 충분한가를 나타내는 지표중 하나로 퍼징 대상 프로그램의 입력 값이 프로그램을 어느 정도로 깊게 실행시켰는가를 의미하며 퍼징의 효율성을 측정하는 하나의 기준 지표로 사용된다. 그레이박스 퍼징은 블랙박스와 화이트 박스 퍼징의 중간 형태로 화이트 박스 퍼징에 비해 상대적으로 가벼운 프로그램 분석을 사용하기 때문에 블랙박스 퍼징보다는 새로운 경로를 찾아낼 가능성이 높으면서도 화이트박스 퍼징보다는 가벼운 분석을 사용하여 시간 효율적이다. 따라서 커버리지를 높이기 위한 커버리지 기반의 그레이박스 퍼징에 관한 연구가 많이 진행되어 왔다.

본 논문에서는 딥 러닝 모델인 Variational Autoencoder(VAE) 모델을 적용하여 기존의 그레이박스 퍼징을 커버리지와 새로운 크래쉬(crash) 발견의 측면에서 효율성을 높일 수 있는 방안을 제안한다. 커버리지 측면에서, 기존의 그레이박스 퍼징은 단순히 새로운 경로를 발견한 입력 값을 큐(queue)에 넣어 다음 변이(mutation)의 대상이 되도록 하는 방식을 사용하기 때문에 큐에 있던 입력 값을 대상으로 새로운 변이를 수행할 시 커버리지 값이 떨어질 수 있는 가능성이 존재하며 새로 발견한 경로가 있는 입력 값은 확인하나 어느 부분을 변이시켜 새로운 경로를 발견했는지에 대한 정보는 기록하지 않는다. 따라서 본 논문은 커버리지가 높게 측정된 입력 값들을 학습 데이터로 삼아 새롭게 재생성(reconstruction)하는 방식을 이용하여 커버리지 값을 균일하게 높게 유지하면서 변이시키는 방식을 제안한다. 이 때, 재생성하는 필드(field)는 필드 기반으로 퍼징을 수행하면서 어떤 필드를 변이시켰을 때 커버리지가 높아졌는지에 대한 정보를 확인하여 정하게 된다.

또한 새로운 크래쉬 발견의 측면에서 기존의 그레이박스 퍼징은 크래쉬가 발생한 입력 값은 확인할 수 있으나 어느 부분이 원인이 되어 크래쉬가 발생했는지에 대한 정보는 기록하지 않는다. 따라서 새롭게 제안하는 방식에서는 필드 기반으로 퍼징을 수행하면서 크래쉬가 난 원인이 된 필드를 특정 지을 수 있기 때문에 동일한 내용의 크래쉬가 발생한 파일들의 해당 필드들을 학습하고 재구성하는 방식으로 해당 부분을 집중적으로 퍼징함으로써 새로운 크래쉬를 발견할 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 커버리지를 높이기 위한 다양한 퍼징, 좋은 입력 값 생성을 위한 딥 러닝 모델을 적용한 퍼징 방법에 대해 소개한다. 3장에서는 VAE를 사용하여 입력 값의 특정 필드를 재생성하는 방식으로 기존의 그레이박스 퍼징 방법보다 커버리지와 새로운 크래쉬 발견의 측면에서 효율성을 높일 수 있음을 보인다. 4장에서는 3장에서 제안된 방법을 통한 실험 결과를 보이고 마지막 5장에서는 결론과 앞으로의 연구 방향을 제시한다.

II. 관련 연구

2.1 기호 실행(Symbolic Execution)을 적용한 화이트박스 퍼징

기호 실행이란 퍼징 대상 소프트웨어의 실행 가능한 모든 경로를 분석하는 기술로 입력 값을 실제 값이 아닌 특정 기호로 두고 분기문이나 반복문 등에 따라 최종 지점까지 가는 경로 식을 기록하는 기술이다. 새로운 경로를 탐색하기 위한 정확한 식이 존재하기 때문에 커버리지를 효율적으로 높일 수 있지만 프로그램 분석에 많은 시간이 소요된다는 문제와 경로 폭발(path explosion)문제가 존재한다[3].

기호 실행의 이러한 문제점을 해결하기 위한 다양한 방법들이 제안되었지만 이 중 특히 'Driller'라는 퍼저는 기존의 랜덤 퍼징 방식과 기호 실행을 결합한 하이브리드 퍼징 방식을 사용한다[3]. Driller의 퍼징 엔진은 더 이상 새로운 경로를 찾을 수 없을 때까

지 랜덤 퍼징을 수행하다가 퍼징 대상 프로그램의 복잡한 검사에 도달했을 때, 선택적으로 기호 실행을 수행하는 방식으로 기호 실행의 장점을 도입하면서 경로 폭발 문제를 해결했다.

2.2 바이너리 계측을 활용한 그레이박스 퍼징

그레이 박스 퍼징이란 가벼운 프로그램 분석을 사용하는 퍼징 방식을 말하는데 대표적인 그레이박스 퍼징 도구로 'American Fuzzy Lop(AFL)'[12]이 있다. 위 Fig 1은 AFL 구조로 AFL은 유전 알고리즘을 사용해 유효한 입력을 생성하고 다양한 계측을 통한 피드백 루프로 해당 입력이 새로운 경로를 발견했는지 평가한다. 새로운 경로를 발견한 값은 버리지 않고 큐에 유지하여 다음 변이의 대상으로 삼는다. 다음 데이터 생성 방식은 유전자 알고리즘에서 사용하는 교배 방식이나, 변이를 사용하여 생성하는데 비트 플립(bit flip), 삽입(insertion), 삭제(deletion)와 같은 방식으로 수행된다[13].

입력 값이 새로운 경로를 발견했는지를 알기 위해선 바이너리 계측이 필요한데 그 중 동적 바이너리 계측(Dynamic Binary Instrumentation)은 런타임 중 실행 코드를 삽입하여 바이너리의 동작을 분석하는 방법이며, 대표적인 도구로 'PIN'[14], 'DynamoRIO'[15] 등이 있다. AFL은 소스코드가 주어진 퍼징 대상 프로그램에 대해선 'afl-gcc'를 사용하여 새롭게 컴파일하여 계측한 후 퍼징하는 방식을 사용하지만 대상 프로그램에 대한 소스코드가 주어지지 않았을 경우도, 'QEMU'[16] 라는 에뮬레이터를 사용하여 퍼징을 지원한다.

최근에는 이러한 AFL의 효율성을 향상시키거나 AFL의 단점을 보완하기 위한 연구가 많이 진행되고 있다. 대표적으로 'VUzzer'[4]는 퍼징 대상 프로그램이 매직 바이트를 요구하는 경우 AFL이 변이를 수행할 위치와 값을 생각하지 않고 단순히 랜덤 변이를 수행하기 때문에 시간 비효율적이라는 점에 착안하여 이에 대한 단점을 정적, 동적 어플리케이션 분석을 통해 보완한 '어플리케이션 인지' 퍼저이다.

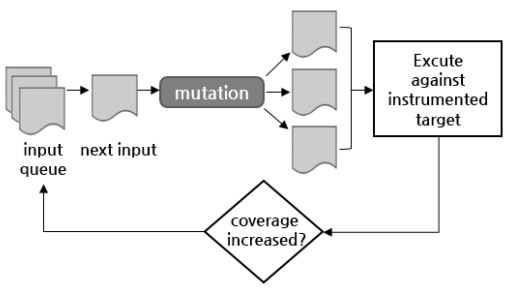

Fig. 1. Structure of Amercan Fuzzy Lop

2.3 딥 러닝 모델을 적용한 퍼징

문법 구조가 복잡한 입력 형태를 갖는 프로그램을 퍼징하는 경우 퍼징을 위해 입력 값을 생성해 내는 것이 시간이 오래 걸린다. 이러한 문제를 해결하기 위해 딥 러닝 모델 중 생성 모델을 사용하면 프로그램에 통과가 되기 위한 'well-formed'한 형태이면서도 프로그램 에러를 유발시키기 위해 적당히 'ill-formed'한 입력 값을 자동으로 생성할 수 있다. 대표적으로 비지도 기반의 Seq2Seq를 사용한 퍼저[5]의 경우 문법이 복잡한 pdf를 자동으로 생성해내기 위해 학습 대상을 pdf object로 하여 신경망 기반의 학습을 수행하였다. 신경망 기반의 통계 학습 기법을 사용하면 학습된 모델의 확률 분포에 기초하면 새로운 입력을 생성해낼 수 있다.

2.4 Variational Autoencoder(VAE)

VAE는 비지도 기반의 생성 모델로 베이지안 추론을 기반으로 하는 변분 추론과 오토인코더(autoencoder)를 바탕으로 복잡한 데이터의 분포를 추정하는 방법론이다. 오토인코더의 잠재 변수(latent variable)인 z는 데이터를 생성하는 분포가 알려져 있는 원인적인 변수인 반면, VAE에서의

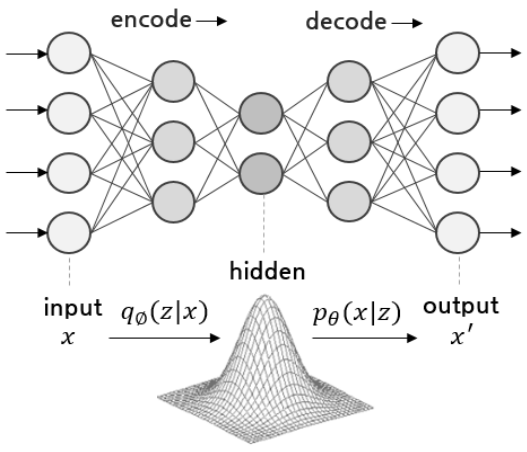

Fig. 2. Structure of Variational Autoencoder

z는 연속적인 분포를 갖는 확률 변수(random variable)라고 할 수 있다. z의 분포는 학습 과정에서 학습 데이터로부터 학습되며 이 과정을 인코딩 과정이라 한다.

VAE의 구조는 Fig 2에 나타나 있으며 Fig 2에서 VAE는 크게 인코더와 디코더로 나눌 수 있다. 인코더는 주어진 x로부터 z를 얻는 신경망으로, 입력 x가 주어졌을 때 z의 확률 분포 $p(z|x)$는 사후 확률이기 때문에 이 값을 직접 계산하기 어렵다. 이때 변분 추론(Variational Inference)이라는 방법을 사용하게 된다. 변분 추론이란 계산하기 힘든 p 대신에 정규 추정 분포 q를 사용하는 것으로 $q(z|x)$를 도입하여 $p(z|x)$로 근사시키는 방법이다.

VAE의 디코더는 z로부터 x를 만들어내는 신경망으로 출력인 x의 확률분포인 $p(x)$를 알기 위해 $p(x|z)$를 학습해야 한다. 이 때 디코더는 정규분포를 전제로 하며 인코더가 만들어낸 z의 평균과 분산을 모수로 한다. 따라서 VAE의 인코더는 학습 데이터를 latent variable인 z로 압축하고 디코더는 이 z로부터 처음 입력 데이터를 재구성하게 된다. 실제로 잘 학습된 VAE의 경우 디코더 부분만 따로 분리해 생성 모델로 사용할 수 있는데 임의의 z값을 디코더에 넣으면 다양한 데이터를 생성할 수 있다.

III. 제안하는 방법론

3.1 VAE를 활용한 퍼징 개요

개요에서 설명한 VAE 퍼징의 전체적인 모식도는 아래 Fig 3과 같고 Fig 3의 각 부분이 의미하는 바는 아래와 같다.

- fuzzing: 필드를 선정하기 위한 초기 퍼징
- field based mutation: m_i 파일을 필드 기반으로 변이시켜 m_{i+1} 파일을 생성
- analysis: 새롭게 생성된 m_{i+1} 파일에 대한 분석을 수행하는 곳으로 커버리지 값이 측정되고 크래쉬 발생 여부가 분석됨
- mutated files: 일정 시간 퍼징을 수행하면서 생긴 변이된 파일들
- field information: 커버리지에 영향력이 있는 필드와 크래쉬가 발생한 원인이 된 필드에 대한 정보(선정된 퍼징할 필드)
- extract fields: VAE 모델에 입력 값을 주기 위해, 선정된 필드를 파일에서 추출(VAE 퍼징

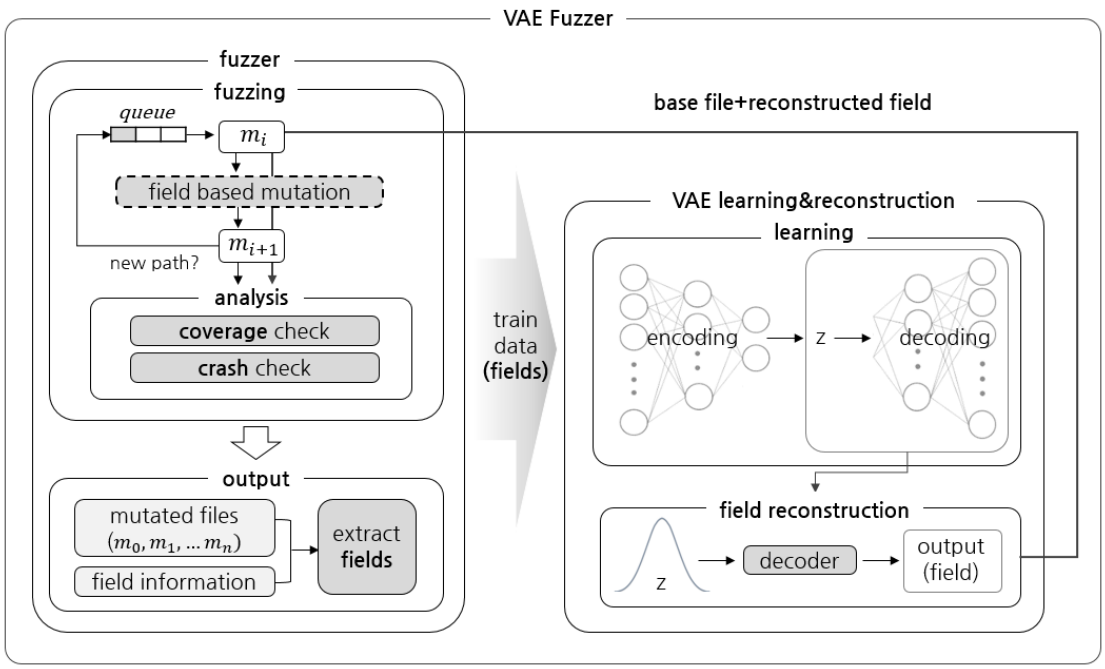

Fig. 3. Overall Structure of VAE Fuzzer

방식에 따라 필드를 추출할 파일의 집합이 다름)

- train data: VAE 모델을 학습시키기 위한, 'extract fields'에서 추출된 필드의 바이트 값
- VAE learning&reconstruction: 위 'train data'를 학습 데이터(필드의 바이트 값)로 사용하여 VAE 모델을 학습시키고 필드를 재생성하는 부분으로 학습 데이터와 목적에 따라 VAE1, VAE2로 나뉨
- field reconstruction: 학습된 VAE 모델의 디코더 부분을 생성 모델로 사용해 필드를 재생성

기존의 그레이박스 퍼징은 새로운 경로를 발견한 입력 값을 확인하여 다음 변이의 대상으로 삼기 위해 버리지 않고 유지하지만 어느 부분이 변형되어 새로운 경로를 발견할 수 있었는지, 어느 부분이 변형되어 크래쉬가 발생하게 되었는지에 대한 정보는 활용하지 않는다. 따라서 본 논문에서 새로 제안하는 퍼징 방식은 해당 정보를 사용해 집중적으로 퍼징할 필드를 특정지어 퍼징 영역을 좁히는 방식을 시도한다. 퍼징할 영역을 좁힌 후에는 VAE라는 딥 러닝 모델을 활용하게 된다.

이 때 VAE를 활용한 퍼징 방식(VAE 퍼징)은 목적에 따라 두 가지 방식으로 진행된다. 첫 번째 방식은 커버리지를 균일하게 높이기 위한 방식(VAE1)이고 두 번째 방식은 발생한 크래쉬를 사용하여 새로운 크래쉬를 발견하기 위한 방식(VAE2)이다.

Table 1. Two Types of VAE Fuzzing

	VAE1	VAE2
Analysis Information	Coverage values of mutated files	Crash location (to see if it is unique)
Collecting Information	Field informations that has an impact on coverage	Field informations that caused the crash
Input Data (Train Data)	Fields that has an effect on coverage of the top 10% files with the highest coverage	Fields that has an effect on crash of files with the same crash
Purpose	To increase coverage	To find unique crash

딥 러닝을 사용하여 복잡한 입력 문법을 학습한 후 퍼징 입력 값을 임의로 생성해 에러를 유발시키는 최근의 연구들([5],[6])은 학습 데이터 구성에 정해진 기준이 없는 반면에 본 논문은 '크래쉬'와 '커버리지'를 기준으로 목적에 맞는 학습 데이터 따로 구성한다는 차별점이 존재한다. 이러한 방식을 사용하면 단순히 에러를 유발해내는 랜덤한 입력 값을 생성해내는 것 이상으로 각 기준에 맞는 중요한 특징 값을 학습하면서 변이가 동시에 일어날 수 있으며 퍼징 범위가 줄어들기 때문에 시간 효율적이라는 장점을 갖는다.

결국, 학습된 VAE 모델의 디코더를 사용하여 재생성한, 학습 데이터로 입력한 필드의 특징을 갖게 된 필드는 베이스 파일의 해당 필드 부분을 덮어 쓰게 되고 이러한 방식으로 특정 필드만 재구성된(변이된) 파일들은 다시 퍼저에 입력되어 커버리지, 크래쉬에 대한 정보가 분석된다. 이때는 이미 변이된 파일들의 집합이 퍼저에 입력되어 분석만 되는 것이기 때문에 Fig 3의 'field based mutation'은 수행하지 않는다. 위 Table 1은 두 가지 방식의 VAE 퍼징을 위해 필요한 분석, 수집 정보와 각 VAE 퍼징의 학습 데이터, 목적을 정리한 표이다.

3.2 필드 기반 퍼징 방식

보통 파일은 내부적으로 복합 파일 구조를 가지는데, 예를 들어, 한글 파일(hwp)은 내부적으로 스토리지(storage)와 스트림(stream)을 구별하기 위한 이름을 갖고 하나의 스트림에는 일반적인 바이너리나 레코드 구조로 데이터가 저장된다[18]. 본 논문에서는 이러한 파일 데이터가 나뉘는 영역을 '필드'라 칭한다. 결국 이 필드가 VAE 생성 모델의 입력 값이자 출력 결과가 되는 것이다.

필드 기반의 퍼징을 수행하려면 먼저 학습, 재생성의 대상이 되는 필드를 선정하는 과정이 필요하기 때문에 처음 입력으로 들어오는 시드(seed) 파일에 대한 구조 분석, 필드 파싱이 선행되어야 한다. 시드란 퍼징에서 처음으로 들어오는 입력 값을 의미한다. 분석된 필드 정보를 사용하여 필드 단위로 변이를 수행하는데 변이시킬 필드와 필드 오프셋을 랜덤으로 선정하여 비트 플립(bit flip)을 하거나 다른 필드의

데이터를 삽입(insertion) 하거나 하는 방식을 사용한다.

이렇게 필드를 정의하고 필드를 기반으로 퍼징을 수행하는 이유는, 소스코드가 주어지지 않은 바이너리 형태의 프로그램에 대해 어느 부분이 원인이 되어 커버리지가 향상되었거나 크래쉬가 발생하게 되었는지에 대한 위치 정보를 필드 단위로 확인하기 위함이다. 필드를 확인하여 위치를 특정 지을 수 있다면 그 부분만 집중적으로 퍼징할 수 있게 된다.

3.3 필드의 선정 및 VAE 학습 데이터 생성

필드 기반의 퍼징을 수행하면서 각 퍼징 방식에 맞는 정보를 분석, 수집한다. 이 때 퍼징은 VAE를 학습할 학습 데이터인 필드가 충분히 많이 확보되도록 mutated file(m0,m1,...)의 수가 충분히 많아질 때까지 진행한다.

하지만 단순히 변이된 파일을 많이 생성하는 것이 아니라 커버리지에 영향력이 있는 필드와 크래쉬가 발생한 지점의 필드를 선정하기 위해, 변이를 수행하면서 각 퍼징 방식(VAE1, VAE2)에 맞게 새롭게 생긴 입력 값에 대한 분석 과정 또한 필요하다. VAE1 방식에서 필요한 분석은 '3.3.1. 동적 바이너리 계측과 필드 선정'에서 설명하고 VAE2 방식에서 필요한 분석은 '3.3.2. 크래쉬 분석과 필드 선정'에서 설명한다.

3.3.1 동적 바이너리 계측과 필드 선정

VAE1 방식을 위해선, '커버리지 측정'과 '커버리지에 영향력이 있는 필드 선정' 이 필요하다.

커버리지 측정을 위해, 동적 바이너리 계측을 수행해주는 도구인 PIN을 사용하여 퍼징 대상 바이너리에 대한 해당 입력 값의 커버리지 값을 측정한다. 커버리지를 측정하는 대표적인 유형 중 하나인 베이직 블록 커버리지(basic block coverage)는 코드 내의 모든 베이직 블록이 한번 이상 수행되었는지를 판단하는 방법으로 분기문의 참 거짓 실행 경로에 해당하는 각 지점에 대응되는 오브젝트 코드가 런 타임에 실제로 실행되었는지의 여부를 확인하는 방식을 사용한다[7].

m_i 파일의 특정 필드가 변이되어 m_{i+1} 파일이 생성되었고 m_{i+1} 파일의 커버리지 값이 m_i 파일의 값보다 크다면 해당 필드가 커버리지에 영향력을 행사하는 필드라고 여기고 해당 필드에 대한 횟수를 추가한다. 여기서 횟수가 가장 큰 필드를 커버리지에 가장 영향력이 있는 필드로 선정하고 따라서 해당 필드는 VAE1의 방식에서 재생성할 필드가 된다.

3.3.2 크래쉬 분석과 필드 선정

VAE2 방식을 위해선, 크래쉬가 발생한 파일에 대해, '크래쉬가 발생한 위치'와 어떤 필드의 데이터가 변이되어 크래쉬가 발생했는지 그 '크래쉬 발생 원인이 된 필드'가 필요하다. 크래쉬가 발생한 위치를 확인하는 이유는 두 가지가 있는데 첫 번째는, 해당 크래쉬가 새로운 크래쉬인지 확인해야하기 때문이

Table 2. Pseudo Code for Field Based Fuzzing using Dynamic Binary Instrumentation

```
1  do field parsing of seed file
2
3  while not stopping criterion do
4    // Field based mutation
5    field, offset=random_select()
6    m_{i+1}=mutation(m_i, field, offset)
7
8    // Coverage analysis using DBI tool
9    if coverage(m_{i+1}) 〉 coverage(m_i) then
10     count cov_field[field]
11   // Crash analysis using gdb
12   if m_{i+1} occurs segmentation fault then
13     crash_info=crash_analysis(core)
14     append m_{i+1} to crash[crash_info]
15
16 // Select files for each vae fuzzing types
17 tocov_files=select top 10% coverage files
18 same_crash_files=crash[crash_info]
19
20 // Select field for each vae fuzzing types
21 cov_field=max field of cov_field
22 crash_field=field info of crash_info
23
24 // Extracting fields from selected files
25 vae1_data=extract(cov_files,cov_field)
26 vae2_data=extract(crash_files,crash_field)
```

다. 두 번째는, 같은 위치에서 발생한 크래쉬는 같은 종류의 크래쉬라고 볼 수 있으므로 같은 종류의 크래쉬를 같은 집합으로 구성해 같은 집합으로 구성된 크래쉬 파일들의 크래쉬가 발생한 원인이 된 필드들을 VAE2 퍼징 방식에서의 학습 데이터로 입력하기 위함이다.

크래쉬가 발생한 위치는 크래쉬가 발생하면서 생긴 코어 덤프(core dump) 파일을 gdb로 분석하는 과정에서 도출되는 함수 명을 기준으로 확인한다.

3.3.3 VAE 학습 데이터 생성

위 Table 2는 각 VAE 퍼징 방식 VAE1, VAE2에 입력으로 들어갈 입력 데이터를 구성하는 방식에 대한 의사코드이다. 변이된 파일들 중 커버리지가 높은 상위 파일들(cov_files)에서 커버리지에 가장 영향력이 있다고 선정된 필드(cov_field)를 추출하여 VAE1에 입력으로 들어갈 입력 데이터(vae1_data)를 구성한다.

크래쉬가 발생한 파일들 중 크래쉬가 발생한 위치와 크래쉬가 발생하게 된 원인이 된 필드로 구성된 크래쉬 정보(crash_info)가 같은 파일들은 같은 집합(crash_files)으로 구성된다. 같은 집합으로 구성된 파일들(same_crash_files)에서 크래쉬의 원인이 된 필드(crash_field)를 추출하여 VAE2에 입력으로 들어갈 입력 데이터(vae2_data)를 구성한다.

3.4 VAE 학습

각 퍼징 방식 VAE1, VAE2에 맞는 학습 데이터가 충분히 확보되면 해당 학습 데이터는 VAE 모델 학습을 위해 모델에 입력된다. VAE1과 VAE2는 커버리지가 높은 파일들로 학습 데이터가 구성되는지 동일 크래쉬가 발생된 파일들로 구성되는지에 따라 달라지는 것이고 학습 및 필드 재생성 방식은 동일하다. 이 때, 학습 데이터는 '3.3.3 VAE 학습 데이터 생성'에서 살펴본 대로 필드 단위로 구성되어 있으며 실제 학습은 해당 필드의 바이트 단위로 이루어지게 된다.

각 VAE1, VAE2 방식에 맞게 학습 데이터가 VAE 모델에 입력되면 인코더와 디코더가 학습된다. 각 퍼징 방식에 맞는 필드의 바이트 값들이 벡터 형태로 인코더 신경망에 입력되고 따라서 입력 노드 수는 필드의 크기가 된다. 인코더는 입력 벡터를 낮은 차원의 잠재 변수인 로 압축하고 디코더는 인코더가 출력해낸 의 평균과 분산을 모수로 하여 입력 벡터의 확률 분포를 학습해 입력과 같은 차원의 출력 값을 생성해낸다.

Table 3. Node Numbers of VAE Layers

	Encoder Layer		Latent Layer	Decoder Layer	
	1st	2nd		1st	2nd
1	1024	512	64	512	1024
2	256		32	256	

위 Table 3은 VAE 모델의 레이어(layer) 구성과 각 레이어에 맞는 노드(node) 수이다. 본 논문에서 사용한 시드 파일의 경우 중요히 여긴 필드 2개가 각각 3318 바이트, 517 바이트였기 때문에 각 크기에 맞게 아래와 같이 1, 2번으로 구성을 나누었고 입력 벡터 크기에 맞게 1, 2번 구성을 선택하여 사용하였다. 신경망에 사용되는 각 노드의 출력 신호를 결정하기 위해 적용하는 함수인 활성화 함수는 'ReLU(Rectified Linear Unit)'를 사용했다. ReLU는 $f(x)=\max(0,x)$ 로 표현되는 함수로 $x>0$이면 0을 출력하고 그 외의 경우에는 x를 그대로 출력한다. ReLU는 학습 속도가 매우 빠르고 미분 계산을 할 필요가 없다는 장점이 있어 최근 딥러닝에 많이 사용되고 있는 활성화 함수이다[11].

3.5 VAE 디코더를 이용한 필드 재생성

VAE의 학습이 완료되면, 학습된 VAE 모델의 디코더 부분을 분리해 생성 모델로 사용할 수 있다. 이 때 재생성을 위한 z는 정규 분포에서 샘플링 하고 학습된 디코더에 입력해 재 생성된 필드를 얻는다. 이 재 생성된 필드는 베이스 파일에 붙여 하나의 새로운 파일을 만들게 되고 이렇게 충분한 퍼징 데이터(퍼징 대상 바이너리에 입력으로 들어갈 데이터)가 생성되면 이 파일들은 다시 퍼저에 입력된다. 수식은 아래 (1)에 나타나 있다. 샘플링 된 z를 디코더에 입력할 때는, 재생성 과정에서 약간의 변이를 가하기 위해 z에 드롭아웃을 적용한다.

$$p(x|z)=N(x|f_u(z),f_\sigma(z)^2\times I) \quad (1)$$

IV. 실험 및 평가

4.1 실험 환경

아래 Table 4는 퍼징이 이루어진 가상 머신, 필드 재생성을 위한 VAE 학습에 사용된 시스템의 환경과 실험에 사용한 퍼징 대상 프로그램의 버전이다.

본 논문에서는 실험을 위해 OLE(Object Linking & Embedding)[17] 구조를 갖는 파일을 입력 값으로 받는 파일 기반 프로그램을 퍼징 대상 프로그램으로 삼았다. OLE 구조를 갖는 파일은 복합 구조 형식을 갖고 있기 때문에 필드의 수가 많고 그 특성이 다양하여 본 논문이 제안하는 방식에 잘 맞는다.

Table 4. Experiment Environment

Fuzzing System	
OS	Ubuntu 16.04
VAE Training System	
OS	Ubuntu 16.04
CPU	AMD Ryzen 7 1800X
RAM	64GB
GPU	NVIDIA GeForce GTX 1080 Ti
Target Software for test	
hwpviewer	2014(recent version for linux)
catdoc	0.94.4
abiword	3.0.0

4.2 커버리지 기반의 VAE 퍼징

4.2.1 필드 선정을 위한 초기 퍼징

Table 5, 6은 Table 2의 의사코드 대로 초기 퍼징을 iteration 1500번까지 수행하면서 m_i의 커버리지 값이 이전 파일인 m_{i-1}의 값보다 큰 경우 변이된 필드가 무엇이었는지 카운트 한 결과를 0~1 사이로 스케일한 결과이다.

실험 결과, 한글 파일(hwp)에서 가장 커버리지에 영향력이 있다고 선정된 필드는 문서의 요약정보가 저장되는 '\005HwpSummaryInformation' 이고 그 다음은 문서에 첨부된 OLE 바이너리 데이터가 저장되는 'BinData/BIN0002.OLE'로 나타났다[18]. 워드 파일(doc)에서 커버리지에 영향력이 있다고 설정된 필드는 'SummaryInformation'으로 OLE 속성에 대한 정보가 저장되는 필드이다[19]. 워드 파일의 경우 'catdoc', 'abiword' 두 가지 바이너리로 실험을 진행했고 똑같이 제일 높은 횟수를 기록한 하나의 필드를 선정했다.

Table 5. Hwp File Field Count

Field	Size	Value
\005HwpSummaryInformation	**517**	**0.105**
BinData/BIN0001.png	295	0.091
BinData/BIN0002.OLE	**3318**	**0.190**
BodyText/Section0	3131	0.068
DocInfo	1084	0.064
DocOptions/_LinkDoc	524	0.103
PrvImage	3599	0.099
PrvText	1146	0.098
Scripts/DefaultJScript	136	0.096
Scripts/JScriptVersion	13	0.086

Table 6. Doc File Field Count

Field	Size	value	
		catdoc	abiword
CompObj	110	0.083	0.082
DocumentSummaryInformation	468	0.079	0.090
SummaryInformation	**420**	**0.147**	**0.170**
1Table	11181	0.078	0.090
Data	379188	0.084	0.088
ObjectPool/1595941772/CompObj	101	0.077	0.076
ObjectPool/1595941772/Ole	20	0.076	0.088
ObjectPool/1595941772/EPRINT	7644	0.084	0.084
ObjectPool/1595941772/ObjInfo	**6**	**0.111**	0.074
ObjectPool/1595941772/Workbook	2943	0.073	0.075
WordDocument	13649	0.108	0.082

4.2.2 VAE 학습/재생성 및 커버리지 측정 결과

아래 Table 7은 기존 그레이 박스 퍼징 방식보다 본 논문에서 제안한 방식이 커버리지가 높음을 보이기 위해 위해서, VAE1 방식으로 재생성된 파일들(베이스 파일에 재생성된 필드를 덮어씌운 파일)과 AFL 퍼저의 큐에 들어있는 파일들의 커버리지 값을

Table 7. Coverage Based VAE Fuzzing Result

	hwp viewer	abi word	cat doc
Existing greybox fuzzing	0.450	-	-
AFL	-	0.468	0.447
VAE fuzzing	**0.549**	**0.549**	**0.552**

비교한 결과이다. AFL 퍼저로 퍼징을 수행하면, 출력 디렉토리로 지정한 디렉토리에 큐 디렉토리가 생성되고 새로운 경로를 발견한 파일들은 해당 큐 디렉토리에 저장된다.

실험을 위한 퍼징 대상으로 'hwpviewer', 'abiword', 'catdoc'을 선정했는데 비교를 위한 AFL 퍼저의 경우 소스코드가 없는 바이너리 기반의 퍼징을 수행해주는 'qemu mode'가 GUI 어플리케이션을 지원해주지 않기 때문에 catdoc을 제외하고 hwpviewer, abiword는 제대로 된 비교를 할 수가 없다. 따라서 비교를 위해 hwpviewer는 AFL 대신 기존의 그레이 박스 퍼징 방식을 모사하여 제작한 퍼저와 비교를 했고 abiword는 '--display=:1' 옵션을 통해 디스플레이 모드를 끄는 방법을 사용했다. 기존 그레이 박스 퍼징 방식을 모사한 퍼저 제작은 변이된 파일에 대한 커버리지를 측정한 후 이 값이 큐에 있는 파일들의 값보다 크다면 새로운 경로를 찾았다 여기고 큐에 넣어 다음 변이의 대상이 되도록 하는 방식을 사용했다.

Table 7의 'VAE Fuzzing'의 경우 중간에 VAE 모델 학습 과정이 포함되기 때문에 반복 횟수를 기준으로 값을 비교하는 것은 타당하지 않아 1시간을 기준으로 변이의 대상이 되는 파일들을 대상으로 커버리지 값의 평균을 내어 비교하였다. 여기서 변이 대상이란 'AFL', 'Greybox Fuzzing'의 경우 퍼징 수행 1시간 경과 기준으로 큐에 들어있는 파일들을 의미하며 본 논문에서 제안한 'VAE Fuzzing'의 경우 학습된 모델을 사용하여 재생성된 파일들을 말한다. 또한 커버리지 측정값은 0~1 사이의 값으로 스케일하였다.

실험 결과 catdoc의 경우 AFL이 0.447, VAE 퍼징이 0.552 의 결과를 보였고, abiword의 경우 AFL이 0.468, VAE 퍼징이 0.531의 결과를 보였으며 hwpviewer의 경우 기존 그레이 박스 퍼징 방식으로 수행한 것이 0.450, VAE 퍼징이 0.549의 결과를 보였다. 즉, AFL의 큐의 파일들보다 VAE 퍼징을 사용했을 경우 약 0.1 정도 커버리지 값이 높은 것을 확인할 수 있었다.

4.3 기존 크래쉬를 이용한 새로운 크래쉬 탐지

4.3.1 기존 크래쉬의 필드 선정

'hwpviewer' 바이너리를 기준으로 초기에 충분한 학습 데이터를 확보할 때까지 퍼징을 수행하는 단계에서 발생된 크래쉬는 두 가지였는데 하나는 'CHncProperty::Get(unsigned int*)' 함수에서 발생한 것으로 해당 크래쉬가 발생하게 된 원인이 된 필드는 'BinData/BIN0002'였다. 즉, 변이시킬 필드로 'BinData/BIN0002' 필드가 랜덤 선택이 되었고 이 필드의 특정 부분이 변이된 후 해당 크래쉬가 발생한 것이다. 다른 하나는 'CHncProperty::ReadFromStream(IStream*)' 함수에서 발생한 크래쉬로 원인이 된 필드는 'HwpSummaryInformation'으로 확인되었다.

4.3.2 VAE 학습/재생성 및 크래쉬 탐지 결과

크래쉬가 발생한 함수는 크래쉬가 발생하면서 생긴 코어 덤프(core dump) 파일을 gdb로 분석하는 과정에서 도출되는 함수 명을 기준으로 확인한다. gdb로 코어 덤프 파일을 분석하기 위해선 'gdb 〔프로그램 명〕 〔코어 덤프 파일〕' 명령을 사용했다. 아래 Table 8은 본 논문의 실험 결과 중 발생한 크래쉬로 생성된 코어 덤프 파일을 gdb로 분석한 결과의 한 예시이다. 'hwpviewer m29.hwp'로 인해 코어 파일이 생성되었으며 세그멘테이션 결함(segmentation fault)을 의미하는 시그널 'SIGSEGV'이 발생했음을 알 수 있다. 세그멘테이션 결함이란 프로그램이 허용되지 않은 메모리 영역에 접근을 시도하거

Table 8. Crash Analysis Using GDB

```
...
1 Core was generated by `hwpviewer m29.hwp'.
2 Program terminated with signal SIGSEGV, Segmentation fault.
3 #0 0x0000000000db17b1 in HwpViewAdaptor::close() ()
...
```

Table 9. Segmentation Fault in hwpviewer

Reconstruction Field	Function that occurs crash	
BinData/Bin0002.OLE	Original Crash	CHncProperty::Get(unsigned int*)
	New Crash	CHwpList::_UpdateCtrlCh(CHncRootCtx*, CHncPara*, HNCPOS, unsigned int, int)
HwpSummaryInformation	Original Crash	CHncProperty::ReadFromStream(IStream*)
	New Crash	HwpViewAdaptor::close

나 허용되지 않은 방법으로 메모리 영역에 접근을 시도할 경우 발생하는 특수한 오류를 말한다[21]. 본 논문에서는 세그멘테이션 결함을 '크래쉬'라 칭했다.

위 Table 9는 기존 크래쉬를 이용하여 새로운 크래쉬를 탐지해낸 결과이다. 동일한 함수에서 크래쉬가 발생한 파일들의 원인이 된 필드를 학습 데이터로 삼아 학습시킨 VAE 모델을 사용하여 해당 필드를 재생성한 후 베이스 파일에 붙이는 식으로 충분한 파일들을 재생성하였다. 이 때 학습에 사용한 파일은 약 50개이고 재생성 파일은 약 100개이다.

실험 결과, 이 재생성된 파일들 중 특정 파일에서 'CHwpList::_UpdateCtrlCh('CHncRootCtx*, CHncPara*, HNCPOS, unsigned int, int)' 함수에서 발생한 크래쉬와 'HwpViewAdaptor::close()' 함수에서 발생한 크래쉬를 새로 발견할 수 있었다.

V. 결 론

본 논문에서는 커버리지, 크래쉬와 연관성이 높은 필드가 존재한다는 것에 착안하여 해당 필드 부분을 집중적으로 퍼징할 영역으로 선정하고, 학습된 VAE의 디코더를 사용하여 재생성하는 방식으로 변이시키는 새로운 방식을 제안했다. VAE 모델에 커버리지 값이 높게 측정된 입력 값들의 특징을 학습시킬 수 있어, 모델을 통해 재 생성된 파일들이 단순 변이를 통해 생성된 파일보다 커버리지 값이 균일하게 높은 것을 볼 수 있었다. 또한 크래쉬가 발생한 파일들의 원인이 된 필드의 특징을 학습하고 재생성 시 드롭아웃을 적용하여 변이를 줌으로써 새로운 크래쉬를 발견할 수 있음을 보였다.

하지만 필드를 선택하는 기준이 세밀하지 못하고 AFL과의 직접적인 비교가 힘들다는 한계점이 있었다. 이러한 한계점은 향후 연구에서 발전시킬 수 있을 것이다.

References

[1] H. Liang, X. Pei, X. Jia, W. Shen and J. Zhang, "Fuzzing: State of the Art," IEEE Transactions on Reliability, pp. 1199-1218, Sep. 2018

[2] M. Böhme, V. Pham and A. Roychoudhury, "Coverage-based Greybox Fuzzing as Markov Chain," IEEE Transactions on Software Engineering, Dec. 2017

[3] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Driller: Augmenting fuzzing through selective symbolic execution," NDSS, vol. 16, pp. 1 - 16, Feb. 2016

[4] S. Rawat, V. Jainz, A. Kumarz, L. Cojocar, C. Giuffrida and H. Bos, "VUzzer: Application-aware Evolutionary Fuzzing," NDSS Symposium 2017, Feb. 2017

[5] 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, Nov. 2017

[6] C. Cummins, P. Petoumenos, A. Murray and H. Leather, "Compiler Fuzzing through Deep Learning," ACM SIGSOFT International Symposium on Software Testing and Analysis, pp.95-105, July 2018

[7] M. Mustafa and K. Jeffrey, "Efficient

Instrumentation for Code Coverage Testing," ISSTA '02 Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis, pp.86-96, July 2002

[8] Y. Noller, R. Kersten and C. S. Păsăreanu, "Badger: Complexity Analysis with Fuzzing and Symbolic," ISSTA 2018 - Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 322-332, July 2018

[9] M. Jurczyk, "Effective file format fuzzing," Black hat Europe, Nov. 2016

[10] N. Nichols, M. Raugas, R. Jasper and N. Hilliard, "Faster Fuzzing: Reinitialization with Deep Neural Models," arXiv preprint arXiv:1711.02807v1, Nov. 2017

[11] A.Severyn and A.Moschitti, "Twitter Sentiment Analysis with Deep Convolutional Neural Networks," SIGIR '15 Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 959-962, Aug. 2015

[12] Lcamtuf, "american fuzzy lop" http://lcamtuf.coredump.cx/afl/

[13] Lcamtuf, "Technical whitepaper for afl-fuzz" http://lcamtuf.coredump.cx/afl/technical_details.txt

[14] Intel, "Pin- A Dynamic Binary Instrumentation Tool" https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[15] DynamoRIO, "DynamoRIO" http://www.dynamorio.org/

[16] QEMU, "QEMU version 3.0.0 User Documentation" https://qemu.weilnetz.de/doc/qemu-doc.html

[17] Wikipedia, "OLE" https://en/wikipedia.org/wiki/OLE

[18] Hancom, "HWP File Format" https://www.hancom.com/etc/hwpDownload.do

[19] Microsoft, "Summary Information Property Set" https://msdn.microsoft.com/en-us/library/dd943476(v=office.12).aspx

[20] Lcamtuf, "American Fuzzy Lop READ DME", http://lcamtuf.coredump.cx cx/afl/README.txt

[21] Wikipedia, "Segmentation Fault" https://en.wikipedia.org/wiki/Segmentation_fault

〈저자소개〉

이 수 림 (Su-rim Lee) 학생회원
2017년 2월: 숭실대학교 경영학과 학사
2017년 3월~현재: 고려대학교 정보보호대학원 금융보안학과 석사과정
〈관심분야〉 정보보호, 시스템 보안, 기계학습

문 종 섭 (Jong-sub Moon) 종신회원
1981년 2월: 서울대학교 계산통계학과 학사
1983년 2월: 서울대학교 계산통계학과 석사
1991년 2월: Illinois Institute of Technology 전산학과 박사
1993년 3월~현재: 고려대학교 전자 및 정보공학부 교수
2001년 2월~현재: 고려대학교 정보보호대학원 겸임교수
〈관심분야〉 정보보호, 운영체제, 침입탐지