한국정보보호학회
Korea Institute of Information Security & Cryptology

2025년 한국정보보호학회 동계학술대회

CISC-W'25

Conference on Information Security and Cryptography Winter 2025

2025년 11월 27일(목)~28일(금)

곤지암리조트

주최·주관 한국정보보호학회 Korea Institute of Information Security & Cryptology

후원 국가정보원 NATIONAL INTELLIGENCE SERVICE, 과학기술정보통신부, 행정안전부, KISA 한국인터넷진흥원, ETRI 한국전자통신연구원 Electronics and Telecommunications Research Institute, NSR 국가보안기술연구소 National Security Research Institute, KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information www.kisti.re.kr

2025년 한국정보보호학회 동계학술대회 CISC-W'25

Conference on Information Security and Cryptography Winter 2025

목 차

파생 프로젝트 취약점 탐지를 위한 다중 시드 풀 퍼저

전세옥[1] 윤상권[2] 최광훈[3] 김석휘[4] 김건오[4]

[1,2,3]전남대학교 (대학원생, 대학생, 교수)
[4]케이사인

A Multi-Seed Pool-Based Fuzzer for Clone-Induced Vulnerabilities

Seok Jeon[1] Sanggwon Yoon[2] Kwanghoon Choi[3]
Seokhwi Kim[4] Keonoh Kim[4]

[1,2,3]Chonnam National University
(Graduate student, Undergraduate student, Professor)
[4]KSign

요 약

오픈소스 복사·붙여넣기 개발로 인해 원본 프로젝트의 취약점이 구조가 다른 파생 프로젝트로 전파되는 사례가 증가하고 있다. 기존 기법은 코드 변형이나 구조 차이에 취약해 탐지에 한계가 있다. 본 연구는 취약점의 함수 호출 경로 기반으로 시드를 분류하고, 취약점 인접 경로 실행 시드 풀을 우선 퍼징하는 다중 시드 풀 퍼저를 제안한다. 실험 결과, AFLFast 대비 약 30% 높은 탐지 효율을 보였으며, 취약점 탐지 횟수는 향상되었다.

I. 서론

오픈소스 생태계의 확산으로 복사·붙여넣기 개발이 증가하면서, 원본 프로젝트의 취약점이 파생 프로젝트로 전파되는 사례가 늘고 있다. 복제된 코드는 파생 프로젝트 상황에 따라 일부 수정되어 사용되기 때문에 원본 프로젝트에서 적용한 취약점 패치를 그대로 적용하기 어렵다. 이로 인해 파생 프로젝트에서 취약점을 다시 분석해야 하는 문제가 발생하고 있다.

기존 취약점 탐지 기법들은 파생 프로젝트에 전파된 취약점을 탐지하는 데 한계를 보인다. 유사도 기반 기법(VUDDY)[1]은 함수의 구문이 동일할 때만 유효하다 보니, 단순한 코드 삽입에도 탐지가 실패할 수 있다. 딥러닝 계열(SySeVR) [2]은 원본 취약점에서 슬라이스를 추출해 AI 모델을 학습하는데 슬라이스 추출이 어렵다 보니, 탐지 신뢰도가 낮다. 지향성 퍼징[3][4]은 원본 프로젝트의 CFG/DFG 등 구조 정보에 의존해 구조가 다른 파생 코드에는 적용이 어렵다.

이에 본 연구에서는 재분석 없이도 취약점을 탐지할 수 있는 다중 시드 풀 퍼저를 제안한다. CVE 취약점이 처음 발견되어 PoC를 분석할 때 취약점 유발 함수로의 콜 스택 트레이스를 분석한다. 여기서 분석한 취약점 위치에 도달하기까지의 함수 호출 경로를 보관해두었다가 오픈소스를 복사해 사용하는 프로젝트에서 동일 취약점을 탐지하는 데 활용하고자 한다. 초기 CVE 분석에서 획득한 취약점의 함수 호출 경로에 따라 시드들을 여러 풀로 분류하고, 취약점 인접 시드 풀을 더 자주 선택하게 함으로써 탐지 효율을 높인다. 실험에서는 취약점을 재분석하는 기존 AFLFast 퍼저 대비 약 30% 개선된 탐지 효율을 보였으며, 시드풀 선택 확률 p 값에 따라 커버리지가 감소하지만 취약점 탐지 횟수가 증가함을 보였다. 구현된 퍼저의 코드는 https://github.com/seokjeon/MultiPoolFuzz에 공개되어 있다.

II. 관련 연구

유사도 기반 기법인 VUDDY[1]는 구문이 동일한 코드를 탐지하는 방법(Type 1 또는 Type 2)으로 파생 프로젝트에 도입된 코드가 수정되면 분석이 어렵다. 다중 시드 풀 퍼저는 실행 호출 경로를 기준으로 시드를 분류·탐색하여 코드 변형에 강건하다.

딥러닝 기반 기법인 SySeVR[2]은 AI 모델 학습을 위해 취약 슬라이스 추출이 필요하지만, 다중 시드 풀 퍼저는 테스트 기반으로 슬라이스 없이 취약점 함수 호출 경로만으로 취약점 탐지가 가능하다.

지향성 퍼징[3][4]은 크래쉬 리포트에서 획득한 취약점 위치 정보를 이용하여 취약점에 도달하는 경로를 찾는 방법이다. 다중 시드 풀 퍼저는 지향성 퍼징과 유사한 점이 있지만, 지향성 퍼저는 제어 흐름 그래프(CFG)나 자료 흐름 그래프(DFG) 등 정적 프로그램 분석 정보에 의존하여 파생 코드에 적용하기 어렵다. 반면, 다중 시드 풀 퍼저는 파생 프로젝트에서 별도의 정적 프로그램 분석 없이 취약점의 함수 호출 경로만으로 파생 프로젝트에서 취약점을 탐지할 수 있다.

III. 다중 시드 풀 퍼저

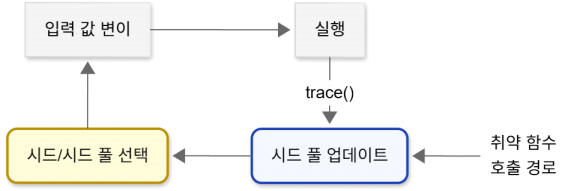

그림 1 다중 시드 풀 퍼저 개요

다중 시드 풀 퍼저는 그림 1과 같이 오픈소스에서 탐지된 취약점의 함수 호출 경로 정보를 활용해, 도달 깊이에 따라 시드 풀을 구성하고, 취약점 유발 함수(sink)에 가까운 풀에 우선순위를 부여하는 그레이박스 퍼저이다.

3.1 다중 시드 풀 퍼저

퍼저의 다중 시드 풀은 취약점 함수 호출 경로의 길이 k에 따라 총 k+1개의 시드 풀로 구성된다. 각 풀은 도달한 깊이 d를 key로 하는 딕셔너리 형태로 관리되며, 예를 들어 취약 함수 호출 경로가 A→B→C→D (k=4)인 경우, 풀은 {4: [], 3: [], 2: [], 1: [], 0: []}로 구성된다.

3.2 깊이에 따른 시드 분배

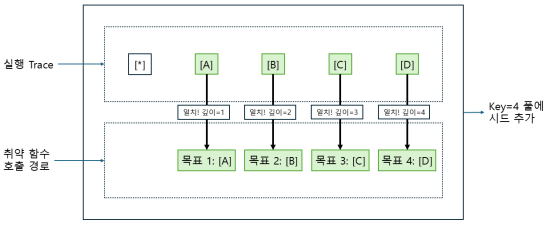

그림 2 취약 함수 호출 경로 도달 깊이에 따른 다중 시드 풀 업데이트

그림 2와 같이 시드는 실행 결과로 수집된 트레이스를 기반으로, 취약 함수 호출 경로에서 어느 함수까지 도달했는지를 계산해 해당 깊이의 풀에 분류된다. 함수 호출 순서만 일치하면 도달한 것으로 간주하며, 중간에 다른 함수가 있어도 유연하게 매칭된다.

예를 들어 D까지 도달한 시드는 key=4 풀에, A까지만 도달한 시드는 key=1 풀에 저장된다. 취약 함수 경로에 도달하지 않으면, 초기 탐색용인 key=0 풀에 저장된다.

3.3 스케쥴링 정책

퍼저는 스케쥴링 정책을 통해 퍼징할 시드를 선택해야 한다. 다중 시드 풀 퍼저의 경우 시드뿐만 아니라 시드 풀도 여러 개 존재하기 때문에 유망한 시드 풀에서 유망한 시드 선택이 필요하다.

3.3.1 풀 선택 방법

풀은 확률 함수 rand()에 의해 결정된다. $p \in (0,1)$일 때, rand()를 순차적으로 호출해 처음 1이 나온 풀을 선택한다. 모두 0이면 마지막 풀을 선택하며, 선택된 풀이 비어 있으면 다음 풀로 이동해 동일한 방식으로 결정한다. 이 방식은 순서상 앞에 배치된 취약점 인접 풀에 더 많은 퍼징 횟수를 부여한다.

3.3.2 시드 선택 방법

시드를 선택할 때는 AFLFast의 시드 스케쥴링 방식처럼 경로 빈도의 역수(path_frequency)를 가중치로 사용해, 반복 경로는 완화하고 희귀 경로에 탐색 기회를 부여한다.

IV. 평가

평가는 다중 시드 풀 퍼저가 파생 프로젝트에서 전파된 취약점을 얼마나 효율적으로 탐지하는지, 코드 커버리지가 줄어들지는 않는지, 시드풀 선택 확률 p가 커짐에 따라 100회의 퍼징 테스트에서 동일 취약점의 발견 횟수가 증가하는지를 중심으로 수행되었다.

4.1 실험 환경

제안한 다중 시드 풀 퍼징 기법의 검증을 위해 Fuzzing Book[5]의 파이썬 예제 프로그램을 확장하여 실험에 활용하였다. 해당 예제는 "bad!" 문자열이 입력되면 b1() → a2() → d3() → ex() → leaf_ crash() 경로로 실행되며, 마지막 함수에서 예외가 발생한다. 이를 확장한 실험 대상 프로그램은 총 30개의 함수로 구성되며, 입력 값에 따라 "just_good", "just_soso", "real_good", "real_bad!" 경로로 분기된다. "real_bad!" 경로는 원본 예제와 동일한 취약점 경로를 포함하고 있어 leaf_crash()에서 예외가 발생한다.

비교 대상은 Fuzzing Book[5]에 구현된 AFLFast 기반 퍼저(표기: aNone)이며, 다중 시드 풀 퍼저(표기: m{p})의 경우 p 값 변화에 따른 성능 차이를 분석하기 위해 p=0.1부터 0.9까지 실험을 반복하였다. 결과의 일관성을 확보하기 위해 퍼저 별로 100개의 고정 랜덤 시드를 사용해 각 시드당 30만 회의 퍼징을 수행했다.

4.2 효율성 평가

그림 3에 따르면, p 값이 클수록 취약점 탐지까지의 평균 퍼징 횟수가 감소하였다. 특히 p=0.9일 때는 aNone 대비 탐지 효율이 약 30% 향상되었다.

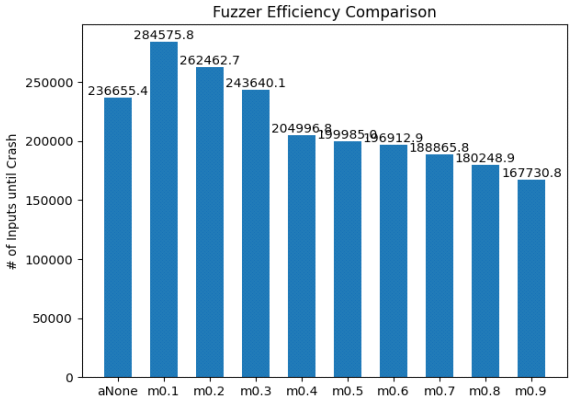

그림 3 p 값 변화에 따른 다중 시드 풀 퍼저와 AFLFast의 취약점 탐지까지의 퍼징 횟수 비교

4.3 코드 커버리지 평가

그림 4는 탐색 효율성과 커버리지 간의 trade-off를 보여준다. 초기(0~50k) 구간에서는 모든 퍼저가 유사한 속도로 커버리지를 확장했지만, 후반(100k 이후)에는 p 값이 클수록 최종 커버리지가 낮아졌다. 이는 높은 p 값이 취약점 함수 호출 경로 탐색을 집중시키기 때문이다.

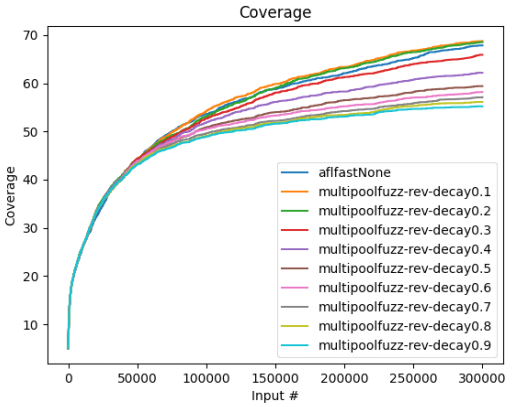

그림 4 p 값 변화에 따른 다중 시드 풀 퍼저와 AFLFast의 코드 커버리지 비교

4.4 크래시 발견 가능성 평가

그림 5에 따르면, p 값이 증가할수록 동일 실행 횟수 내에서 크래시 발견 횟수가 늘어났다. 특히 p ≥ 0.4에서는 aNone보다 성공률이 크게 높았다. 이는 다중 시드 풀 퍼저의 스케줄링이 동일한 입력 수에서도 취약점 발견 횟수를 효과적으로 증가시킨다는 점을 보여준다.

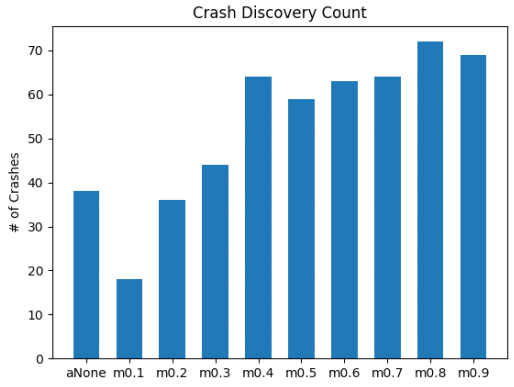

그림 5 p 값 변화에 따른 다중 시드 풀 퍼저와 AFLFast의 동일 취약점 발견 횟수 비교

V. 결론

본 연구는 오픈소스 생태계에서 복사·붙여넣기 개발로 인해 파생 프로젝트에 전파되는 취약점을 효과적으로 탐지하기 위해 다중 시드 풀 퍼저를 제안했다. 실험 결과, 제안 기법은 기존 AFLFast 대비 약 30% 높은 탐지 성능을 보였으며, 취약점 경로 탐색에 집중함으로써 커버리지는 다소 줄었지만 탐지 가능성은 향상되었다. 이는 다중 시드 풀 퍼저가 파생 프로젝트의 취약점을 재분석하는 AFLFast보다 더 효율적임을 시사한다. 향후 연구에서는 본 기법을 AFL++ 퍼저에 통합하고, 다양한 C/C++ 프로그램을 대상으로 실제 CVE 취약점 탐지 실험을 수행할 계획이다.

Acknowledgments

이 논문은 2024 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구결과임 (RS-2024-00398993, 전기자동차 충전기 보안위협 대응 기술 개발)

본 연구는 한국인터넷진흥원(KISA)-정보보안특성화대학 지원사업의 지원을 받아 수행된 연구임.

[참고문헌]

[1] S. Kim, S. Woo, H. Lee and H. Oh, "VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery," 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 2017, pp. 595-614, doi: 10.1109/SP.2017.62.

[2] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu and Z. Chen, "SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities," in IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 4, pp. 2244-2258, 1 July-Aug. 2022, doi: 10.1109/TDSC.2021.3051525.

[3] Marcel Bohme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. 2017. Directed Greybox Fuzzing. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). Association for Computing Machinery, New York, NY, USA, 2329-2344.

[4] Kim, Tae Eun, et al. "{DAFL}: Directed Grey-box Fuzzing guided by Data Dependency." 32nd USENIX Security Symposium (USENIX Security 23). 2023.

[5] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler, The Fuzzing Book, CISPA Helmholtz Center for Information Security, 2024. [Online]. Available: https://www.fuzzingbook.org/.