1-day 취약점 분석 워크플로우 개선을 위한 패치 기반 바이너리 분석 도구

신명진, 조세빈, 박태준*

전남대학교(학부생), 전남대학교(학부생) *전남대학교(교수)

audwls2160@gmail.com, sebin_jo@outlook.com, *taejune.park@jnu.ac.kr

A Patch-Based Binary Analysis Tool for Improving 1-day Vulnerability Analysis Workflows

Shin Myeong Jin, Jo Se Bin, Park Tae June*

Chonnam National Univ., Chonnam National Univ., *Chonnam National Univ.

요 약

1-day 취약점 분석은 보안 패치가 공개된 이후 패치 전•후 실행 파일의 차이를 분석하여 변경 내용의 보안적 의미와 악용 가능성을 해석하는 과정이다. 최근 패치 공개 이후 실제 공격 코드가 빠르게 등장하면서, 1-day 분석은 공격자와 방어자 모두에게 중요한 분석 단계로 자리 잡고 있다. 그러나 실제 분석 과정에서는 변경 함수 탐색과 여러 도구를 반복적으로 사용하는 복잡한 워크플로우로 인해 분석가의 수작업 부담이 주요 병목으로 작용한다. 본 논문은 이러한 문제를 1-day 분석의 자동 판정 문제가 아닌 분석 워크플로우 비용 문제로 재정의하고, 이를 지원하기 위한 패치 기반 바이너리 분석 보조 도구를 제안한다. 제안하는 도구는 패치 전•후 실행 파일을 입력으로 받아 diffing 기법을 통해 변경된 함수 후보를 식별하고, 해당 함수의 패치 전•후 디컴파일 결과를 하나의 분석 흐름으로 통합하여 제공한다. 이를 통해 분석가는 반복적인 도구 전환 없이 변경 지점을 확인하고, 패치 의미 해석에 집중할 수 있다. 본 연구는 자동 취약점 탐지가 아닌 분석가 중심(analyst-in-the-loop) 접근을 통해 1-day 분석 과정의 반복 작업을 줄이고, 실질적인 분석 효율을 지원하는 도구 설계를 제시한다.

Ⅰ. 서 론

보안 패치가 공개된 이후 패치 전•후 코드 차이를 분석하는 1-day 취약점 분석은 공격자와 방어자 모두에게 중요한 단계이다. 최근 연구들은 공개 패치 이후 원데이(one-day) 익스플로잇이 짧은 시간 내에 등장하고 확산되는 사례가 반복적으로 관측되고 있음을 보여준다[1]. 이러한 환경에서 1-day 분석의 목적은 새로운 취약점을 자동으로 탐지하는 것이 아니라, 패치로 인해 무엇이 변경되었고 그 변경이 어떤 보안적 의미를 가지는지를 신속히 이해하는 것이다.

한편 보안 패치는 릴리스 이후에도 수정•보완이 반복되는 패치 진화(patch evolution) 특성을 가지며[2], 이는 1-day 분석이 일회성이 아닌 반복적으로 수행되어야 하는 작업임을 의미한다. 그러나 실제 1-day 분석 과정은 여전히 많은 수작업에 의존하며, 기존의 연구들은 이미 패치가 공개된 상황에서 분석가가 수행하는 반복적인 분석 작업을 줄이는 문제에는 상대적으로 주목하지 않았다.

본 연구는 이러한 문제 인식에 기반하여 1-day 분석을 자동 판정 문제가 아닌 분석 워크플로우 비용 문제로 재정의한다. 이를 위해 패치 전•후 바이너리를 입력으로 받아 변경 함수 탐색과 디컴파일 결과 확인을 하나의 분석 흐름으로 통합하는 분석 보조 도구를 설계•구현한다.

Ⅱ. 관련 연구

기존의 1-day 취약점 분석 연구들은 보안 패치로 인해 변경된 코드 영역을 패치 전•후 바이너리 diffing이나 패치된 코드의 특징 분석을 통해 식별하고, 이를 기반으로 1-day 또는 유사한 0-day 취약점을 자동으로 탐지하는 데 초점을 맞추고 있다[3-4].

이러한 선행 연구들은 취약점 탐지 정확도 향상에는 효과적이지만, 실제 분석가가 수행하는 1-day 분석 과정 자체를 단순화하는 데에는 상대적으로 초점을 두지 않는다. 특히 diffing 이후의 변경 함수의 디컴파일 결과를 반복적으로 확인하고 패치 의미를 해석하는 과정은 여전히 수동 개입에 크게 의존한다.

한편 패치 진화에 대한 연구는 보안 패치 분석이 단발성 작업이 아니라, 반복적이고 누적적인 분석 과정을 요구함을 보여준다[2]. 이는 분석 결과의 재사용성과 워크플로우 통합이 중요함을 시사한다.

Ⅲ. 1-day 분석 워크플로우와 병목 분석

일반적인 1-day 분석 워크플로우는 (1) 패치 전•후 바이너리 확보, (2) 분석을 위한 BinExport 파일 생성 (3) 함수 단위 diffing을 통한 변경 함수 후보 식별, (4) 변경 함수의 패치 전•후 디컴파일 결과 비교, (5) 패치 의미 해석 및 악용 가능성 판단으로 구성된다. 이 중 (3)와 (4) 단계는 서로 다른 도구에서 수행되는 경우가 많아, 분석가는 diffing 결과를 확인한 뒤 다시 디컴파일 도구로 이동해 동일 함수를 탐색해야 한다.

이러한 반복 작업은 변경 함수 수가 많아질수록 반복되며, 패치 진화로 인해 동일한 코드 베이스에 대해 유사한 분석을 여러 차례 수행해야 하는 상황에서 더욱 두드러진다[2]. 즉, 1-day 분석의 주요 병목은 변경 지점을 찾는 알고리즘 자체가 아니라, 변경 지점 탐색과 디컴파일 결과 확인이 분리되어 발생하는 반복 작업에 있다.

Ⅳ. 도구 설계 및 구현

제안하는 도구는 패치 전•후 바이너리가 모두 확보된 1-day 분석 시나리오를 전제로 하며, 분석가 중심(analyst-in-the-loop) 접근을 따른다. 본 도구는 취약점 존재 여부를 자동으로 판정하는 것이 아닌, 변경 지점을 신속히 파악하고 패치 의미 해석에 집중할 수 있도록 분석 흐름을 단순화하고 반복 작업을 줄이는 것을 목표로 한다.

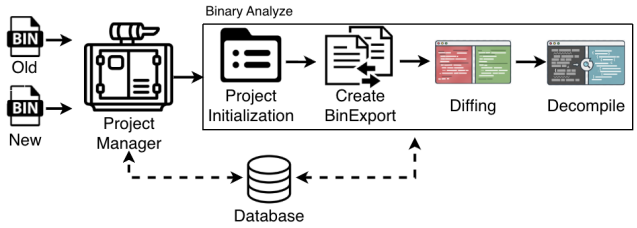

그림 1. 제안된 도구의 워크플로우

그림 1은 제안하는 도구의 전체 분석 파이프라인을 나타낸다. 분석 과정은 크게 Project Manager와 Binary Analyze단계로 구성된다. 먼저 패치 전•후 실행 파일이 입력되면, Project Manager는 실행 파일 쌍에 대한 해시 기반 프로젝트를 생성하고, 동일한 파일에 대한 분석 이력이 존재할 경우 이를 재사용한다. 이를 통해 반복 분석에 따른 비용을 줄일 수 있다.

Binary Analyze 단계에서는 headless 환경의 Ghidra를 이용해 각 파일에 대한 정적 분석을 수행하고, 함수 경계와 제어 흐름 정보를 수집한 뒤 디컴파일 환경을 초기화한다[5]. 이후 분석 결과는 BinExport 형식으로 변환되어 BinDiff의 입력으로 사용된다[6-7]. BinDiff는 두 실행 파일 간 함수 단위 유사도를 계산하여 변경된 함수 후보를 식별한다. 이 단계는 분석 범위를 효과적으로 축소하는 역할을 하며, 분석가는 변경된 함수를 중심으로 패치 내용을 검토할 수 있다.

식별된 변경 함수에 대해서는 패치 전•후 디컴파일 결과가 자동으로 수집되며, 함수 주소, 함수 간 유사도, 디컴파일 코드 및 어셈블리 코드가 구조화되어 관리된다. 분석가는 도구 전환이나 수동 함수 탐색 없이 단일 분석 흐름 내에서 변경 지점과 패치 내용을 확인할 수 있다.

그림 2. 통합 분석 결과를 보여주는 웹 기반 인터페이스

분석 결과는 데이터베이스에 저장되며, CLI 및 웹 기반 인터페이스를 통해 직관적으로 확인할 수 있다. 그림 2는 웹 기반 인터페이스에서의 분석 결과 화면을 보여주며, 변경된 함수 목록과 패치 전•후 디컴파일 결과가 함께 제공되는 모습을 나타낸다. 이를 통해 분석가는 변경 함수 선택, 패치 전• 후 코드 비교, 분석 결과 확인을 하나의 화면에서 수행할 수 있다. 이러한 설계는 그림 1에 나타난 통합된 전체 분석 파이프라인의 결과를 시각적으로 제공함으로써, 반복 작업을 줄이고 분석가가 패치 의미 해석에 집중할 수 있도록 지원한다.

V. 평가

본 연구의 평가는 제안하는 도구가 실제 1-day 분석 과정에서 분석가의 반복 작업을 줄이고 분석 흐름을 단순화하는 데 기여하는지를 확인하는 데 목적이 있다. 공개된 CVE 패치 사례를 대상으로 패치 전•후 실행 파일 쌍을 분석하였으며, 자동화 성능 비교가 아닌, 분석가의 작업 흐름 변화에 초점을 두어 평가를 수행하였다.

표 1은 기존 1-day 분석 방식과 제안 도구 적용 시의 워크플로우 차이를 정성적으로 비교한 결과를 보여준다. 기존 방식에서는 diffing 결과 확인 이후 도구 전환, 변경 함수의 수동 탐색, 결과의 반복 비교가 필요하였다. 반면, 제안하는 도구는 diffing 결과와 디컴파일 정보를 단일 분석 흐름으로 통합하여 제공함으로써, 이러한 준비 및 확인 단계에서 발생하던 반복 작업을 제거한다. 이는 1-day 분석을 자동 취약점 탐지 문제가 아닌 분석 워크플로우 비용 문제로 재정의한 본 연구의 관점을 실질적으로 뒷받침한다.

구분	기존 분석 방식	제안 도구
diffing 결과 확인	BinDiff에서 별도 확인	통합 인터페이스 사용
디컴파일 도구 전환	필요 (수동 전환)	불필요
변경 함수 탐색	수동 주소/이름 탐색	자동 연결
패치 디컴파일 비교	개별적으로 반복 수행	단일 화면에서 제공
분석가 개입 형태	반복적 수작업 중심	의미 해석 중심
분석 결과 재사용	제한적(재확인 필요)	DB 기반 재사용 가능

표 1 기존 1-day 분석 방식과 제안 도구의 워크플로우 비교

또한 변경 함수 목록과 패치 전•후 디컴파일 결과를 직접 연결해 제공함으로써, 디컴파일 정보 접근성을 개선한다. 표 1에서 나타난 바와 같이, 도구 전환과 수동 함수 탐색이 제거되어 코드 탐색보다는 패치 의미 해석에 집중할 수 있다. 이러한 접근성 개선 효과는 변경 함수 수가 많아질수록 더욱 두드러진다. 마지막으로, 분석 결과를 구조적으로 저장하고 재사용할 수 있도록 설계되어, 반복적인 1-day 분석 시나리오에서도 실용적으로 활용 가능함을 보여준다.

VI. 결론

본 논문에서는 1-day 취약점 분석을 자동 탐지 문제가 아닌 분석 워크플로우 비용 문제로 재정의하고, 이를 지원하기 위한 패치 기반 바이너리 분석 보조 도구를 설계•구현하였다. 제안하는 도구는 기존 diffing 기법을 활용하되, 변경 함수 탐색과 디컴파일 결과 확인 과정을 하나의 분석 흐름으로 통합함으로써 분석가의 반복 작업을 줄이고, 패치 의미 해석과 활용 가능성 판단에 보다 집중할 수 있도록 지원한다.

평가 결과, 분석 워크플로우를 단순화함으로써 1-day 분석 과정의 단계적 효율을 개선할 수 있음을 확인하였다. 특히 분석 결과를 구조적으로 저장하고 재사용할 수 있도록 설계한 점은, 패치 진화 환경에서 반복적으로 수행되는 1-day 분석 시나리오에 적합함을 보여준다.

본 연구는 새로운 취약점 탐지 알고리즘을 제안하지 않지만, 분석가 중심(analyst-in-the-loop) 관점에서 분석 도구의 역할과 가치를 재조명한다는 점에서 의의를 가진다. 향후 연구에서는 제안하는 분석 파이프라인 위에 취약점 패턴 분석이나 의미 기반 코드 비교 기법을 결합함으로써, 분석가의 판단을 추가적으로 보조하는 방향으로 확장이 가능할 것으로 기대된다.

ACKNOWLEDGMENT

본 과제(결과물)는 교육부와 한국연구재단의 재원으로 지원을 받아 수행된 첨단분야 혁신융합대학사업 및 과학기술정통부의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2022R1C1C1006967).

참 고 문 헌

[1] L. Maar et al., "Defects-in-Depth: Analyzing the Integration of Effective Defenses against One-Day Exploits in the Android Kernel," USENIX Security Symposium, 2024.

[2] Z. Xie et al., "Unveiling the Characteristics and Impact of Security Patch Evolution," ASE (IEEE/ACM International Conference on Automated Software Engineering), 2024.

[3] L. Gao et al., "SigmaDiff: Semantics-Aware Deep Graph Matching for Pseudocode Diffing," NDSS Symposium, 2024.

[4] C. Dong et al., "Fine-Grained 1-Day Vulnerability Detection in Binaries via Patch Code Localization," arXiv:2501.17413, 2025.

[5] National Security Agency, "Ghidra Software Reverse Engineering Framework," 2019-.

[6] Google, "BinDiff: Binary Code Comparison Tool," GitHub Repository, 2011-2024.

[7] Google, "BinExport: Export Disassemblies for BinDiff," GitHub Repository, 2011-.