정 보 보 호 학 회 지
제28권 제2호, 2018. 4

보안 취약점 자동 탐색 및 대응기술 동향

장 대 일*, 김 태 은*, 김 환 국*

요 약

머신러닝 및 인공지능 기술의 발전은 다양한 분야 활용되고 있고, 이는 보안 분야에서도 마찬가지로 로그 분석이나, 악성 코드 탐지, 취약점 탐색 및 대응 등 다양한 분야에서 자동화를 위한 연구가 진행되고 있다. 특히 취약점 탐색 및 대응 분야의 경우 2016년 데프콘에서 진행된 CGC를 필두로 바이너리나 소스코드 내의 취약점을 정확하게 탐색하고 패치하기 위해 다양한 연구가 시도되고 있다. 이에 본 논문에서는 취약점을 탐색 및 대응하기 위해 각 연구 별 탐색 기술과 대응 기술을 분류 및 분석한다.

Ⅰ. 서 론

다양한 플랫폼에서 다양한 언어로 개발되는 소프트웨어의 취약점이 발표되고 있지만[1], 플랫폼 및 언어별 문법이나 규칙 및 적용 가능한 취약점 등이 상이하기 때문에 각 소프트웨어에 맞는 취약점 탐색 방법과 대응책을 찾는 것은 쉬운 일이 아니다. 이러한 문제를 해결하기 위해 소스코드, 바이너리 혹은 시스템 상에서 문제를 해결하기 위한 다양한 연구가 진행되고 있다.

취약점을 탐색하기 위해 머신러닝이나 딥러닝 알고리즘을 적용하여 취약점을 예측하거나 기존과 유사한 취약점의 패턴을 탐지하기 위한 연구들이 진행되고 있고, 탐색된 취약점에 자동으로 대응하기 위해 소프트웨어의 행위나 상태를 패치하기 위한 많은 연구가 진행되고 있다. 이에 본 논문에서는 소프트웨어를 분석하기 위한 기법들을 언급하고 자동으로 취약점을 찾고 대응하기 위한 지능형 취약점 탬색 및 대응 연구에 대한 동향을 분석하고자 한다.

본 논문의 2장에서는 소프트웨어를 분석하고, 취약점을 탐색하기 위한 기법을 소개하고, 3창에서는 지능형 취약점 탐색 기법 연구에 대한 분석 및 분류를 수행한다. 4장에서는 지능형 취약점 대응 기법 연구에 대한 분석을 수행하고 마지막으로 5장에서 결론을 맺는다.

Ⅱ. 관련연구

2.1. 소스코드 및 바이너리 분석 기법

소프트웨어의 소스코드 및 바이너리를 분석하기 위한 기술은 특징에 따라 정적 분석, 동적 분석 및 하이브리드 분석으로 분류할 수 있다. 각 분석 방법의 특징은 아래와 같다.

2.1.1. 정적 분석 기법

정적분석은 소프트웨어의 소스코드나 바이너리를 실행하지 않은 상태에서 대상에 맞는 분석을 수행하였을 때 나타나는 오류나 논리적인 문제 혹은 취약점을 찾아내는 방법이다. 이를 위해 대상 소프트웨어의 실행 가능한 경로, 변수가 가질 수 있는 값의 범위 등을 분석하고, 대상 내 탐색이 가능한 실행 경로나 값의 범위가 프로그램의 오류를 발생시키는 조건을 만족하는 지 검사하여 오류가 발생하는 소스코드나 바이너리의 인스트럭션을 찾아내는 기술이 정적 분석 기술이다. 정적 분석은 소프트웨어를 실행하지 않은 상태에서 소스코드 및 바이너리를 대상으로 분석하기 때문에 개발 도중에 적용 가능하고, 비교적 짧은 시간에 많은 범위를 분석할 수 있다. 하지만 동적 분석에 비해 많은 수의 허위 취약점

이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2017-0-00184, 자기학습형 사이버 면역 기술 개발)
* 한국인터넷진흥원 보안기술R&D2팀 (dale@kisa.or.kr)

에 대한 알람이 존재할 수 있다. 이를 위해 기호실행 등을 이용한다.

2.1.2. 동적 분석 기법

동적 분석은 통제 가능한 가상 환경에서 소스코드나 바이너리를 실행시켜 오류를 찾아내는 방법이다. 바이너리에서 사용하는 API, 네트워크 활동 정보, 접근하는 파일, 레지스트리, 메모리 정보 등 다양한 정보를 확인할 수 있다. 소스코드 동적 분석은 대표적으로 디버깅 과정이 포함되며 유닛 테스트 또한 동적 분석의 범위에 들 수 있다. 이를 위해 동등 분할, 경계값 분석, 결정 테이블 테스팅, 상태 전이 테스팅, 유스케이스 테스팅, 조합 테스팅 등 다양한 테스팅 기법을 이용할 수 있다. 하지만 입력되는 테스트케이스가 프로그램의 속성을 분석하는데 사용되며, 입력 및 런타임 상태가 무한히 가능하기 때문에 동적 분석을 통해서는 프로그램의 수행 가능한 전체 동작을 분석하기는 매우 힘들다. 따라서 동적 분석은 일부 취약점을 분석하지 못할 가능성이 존재한다.

2.1.3. 하이브리드 분석 기법

하이브리드 분석 방법은 정적/동적 분석이 갖는 한계를 보완하고자 2가지 방법을 혼용하는 분석 방법이다. 예를 들어 동적 분석을 사용하여 식별된 문제를 코드에서 문제가 되는 부분을 찾아 더 명확하게 파악하거나, 정적 분석 결과 생성된 취약점 정보의 허위 취약점을 식별하기 위해 동적 분석을 사용하는 등의 접근방법을 사용할 수 있다.

2.2. 취약점 탐색 기법

정적 분석 및 동적 분석 등을 이용하여 취약점을 탐색하기 위한 기법은 다음과 같다.

2.2.1. 소프트웨어 침투테스트

소프트웨어 침투테스트는 소프트웨어를 더 안전하게 만들기 위해 합법적인 해킹과정을 통해 취약점을 찾아내는 것이다[2]. 일반적인 취약점 진단의 경우 잠재적인 보안에 관련된 이슈를 찾기 위한 과정이고, 침투테스트는 탐색한 취약점이 어떻게 약용될 수 있는지 확인하는 과정이 추가된다. 이를 위해 탐색, 스캐닝, 공격, 접근권한 유지의 과정을 거치게 된다. 탐색 과정은 타겟 소스나 바이너리에 대한 가능한 모든 정보를 수집하는 과정이고, 스캐닝을 통해 대상이 어떤 역할을 수행하는지, 어떤 취약점을 가지고 있는지 분석한다. 스캐닝 단계를 통해 얻어진 취약점 정보를 기반으로 익스플로잇을 생성 및 공격을 수행하고, 소프트웨어를 수정하는 등의 과정을 통해 공격 성공 후 접근 권한을 유지하기 위한 과정을 거치게 된다.

이러한 침투테스트는 소프트웨어 보안을 측정하는데 가장 일반적으로 적용되는 메커니즘이고, 단위 및 시스템 수준에서 침투테스트를 적용하고 결과에 대한 위험 분석을 통해 많은 보안 위협을 관리할 수 있다[3].

2.2.2. 퍼징 테스트

퍼징 테스트는 소프트웨어 테스트 기법으로 데스트 대상에 다양하고 무작위의 데이터를 입력하여 소프트웨어 내부의 충돌, 검증 실패 및 메모리 누수 등의 예외를 발생시키는 것을 목적으로 한다. 이를 위해 화이트박스, 블랙박스, 그래이 박스 등 다양한 방법을 활용한다[4,5]. 화이트박스 테스트는 소스코드 리뷰를 통해 소프트웨어의 취약점을 발견하는 방법이고, 블랙박스 테스팅은 소프트웨어의 내부 구조나 작동원리를 모르는 상태에서 소프트웨어 동작을 검사하는 방법이다. 그레이 박스 테스팅은 블랙박스 테스팅과 화이트박스 테스팅의 중간단계로 소프트웨어의 내부 구조 중 일부만 알고 시험을 수행하는 기법이다. 테스트를 위한 테스트케이스를 만들 때 내부 구조 정보를 활용하며, 시험은 블랙박스 테스팅 형태로 수행된다.

2.2.3. 데이터 흐름 분석

취약점을 탐색하기 위해 생성되는 테스트케이스에 대해서 소프트웨어가 받는 영향을 측정 기법으로, 취약한 메모리 영역의 추적에 주로 사용된다. 입력된 테스트케이스를 오염되었다고 판단한 뒤, 해당 입력이 악의적인 목적으로 버그 혹은 에러를 발생시키는지 판단한다. 동적 혹은 정적 데이터 흐름 분석을 사용한다.

Ⅲ. 보안 취약점 탐색 기법

소프트웨어 취약점 분석 및 탐색 분야에서 지난 수년간 다양한 연구가 시도되었고 지능형 취약점 탐색에 대한 많은 연구가 발표되었다.

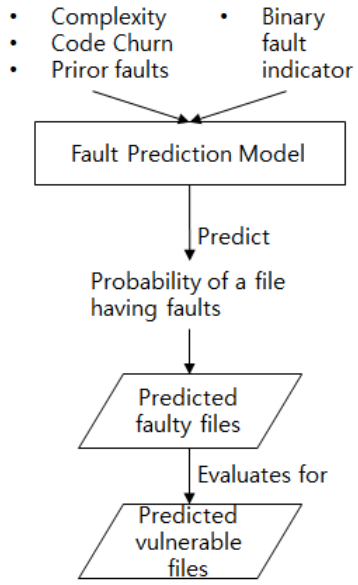
3.1. 소스코드 대상 보안 취약점 탐색

3.1.1. 취약점 예측 모델

취약점 예측 모델은 알려진 소프트웨어 메트릭을 기반으로 한 예측 모델을 구축하고 소프투에어 아티팩트의 취약 상태를 평가하기 위한 연구이다.

Zimmermann et al.은 2010년에 윈도우즈 비스타를 대상으로 바이너리의 결함 예측을 기반으로 취약점의 존재를 예측하기 위한 연구를 진행하였다[6]. Spearman의 순위 상관관계를 사용하여 메트릭과 바이너리 당 취약점 수 사이에 통계적으로 유의미한 상관관계가 있음을 보이고 있지만, 효과 자체는 미비하다. Meneely et al.은 개발자 활동 메트릭과 소프트웨어 취약점 간의 관계를 조사하였다[7]. 개발자 활동 메트릭에는 소스 파일을 변경한 고유 개발자 수, 파일에 대한 커밋 수 등을 포함하고 있다. 3가지 오픈소스를 대상으로 베이지안 네트워크를 이용하여 상관 관계를 분석하였다. Doyle et al.은 워드프레스나 미디어위키같은 오픈소스 웹 응용프로그램에 대해 소프트웨어 메트릭과 취역점 간의 관계를 분석하였다[8]. 정적 분석을 이용하여 취약성 밀도, 소스코드 크기, 순환 복잡성 등을 비롯한 다양한 메트릭을 분석하고, 예측 가능성을 위해 Spearman의 순위 상관관계를 SAVD와 메트릭을 통해 계산한다. 이를 통해 함수 별 평균 사이클로메틱 복잡도가 취약점 탐색에 효율적임을 보였다.

Shin and Williams는 복잡도 및 코드 이탈 메트릭을 기반으로 하는 장애 예측 모델이 취약점 탐색에 사용 가능함을 보였다[9]. 이를 위해 18가지의 복장도 메트릭과 5가지 코드 이탈 메트릭 및 오류 내역 메트릭을 사용하여 파이어폭스에 대해 취약점 탐색을 수행하였다. 그 결과 논문에서 사용한 전통적인 메트릭을 기반으로 하는 오류 예측 모델이 취약점 예측에도 사용될 수 있다고 주장한다. 다른 논문에서는 소프트웨어 취약점을 지표로 실행 복잡도 메트릭을 조사했다[10]. 소프트



〔그림 1〕 복잡도와 코드이탈 메트릭 기반의 취약점 예측 모델

웨어 취약점을 탐색하기 위한 실행 복잡도 및 정적 복잡도 메트릭스의 비교 검증을 수행하였다. 이를 위해 통계에 대한 차별적 분석과 예측 분석을 모두 수행하였는데 파이어폭스와 와이어샤크를 대상으로 실험하였을 때 실행 복잡도 메트릭스가 더 나을 지표임을 주장하고 있다.

Mpshtari et al.에서는 취약점 예측 모델의 이전 연구들에 대한 한계를 해결하기 위해 소프트웨어의 취약한 위치를 예측하기 위해 복잡도 메트릭스를 이용한 새로운 프레임워크를 제안하였다[11]. 5개의 오픈소스에서 수집한 데이터를 기반으로 프로젝트간 취약점 예측에 활용하였다. Walden et al.은 소프트웨어 특성과 텍스트 마이닝을 기반으로 취약한 소프트웨어 구성 요소를 예측하기 위한 연구를 수행하였다. 이를 위해 223개의 취약점이 포함된 3개의 오픈소스를 대상으로 취약점 데이터를 생성한 후 12개의 코드 복잡도 메트릭을 선정하여고, 텍스트 마이닝을 위해 소스 파일을 먼저 토큰화한 이후 토큰을 제거하거나 변경하여 최종 토큰의 빈도를 계산하였다. 취약점에 대한 예측을 수행하기 위해 Random-Forest를 1차 분류 알고리즘으로 선택하였다. 그 결과 텍스트 마이닝에 기초한 예측 기술의 취약점 탐색 정확도가 더 높다고 주장하였다. Perl et al.은 취약점 커밋을 식별하기 위해 코드 리포지토리에 포함된

메타 데이터를 코드 메트릭과 함께 사용하는 방안을 연구하였다[13]. 총 66개의 GitHub 프로젝트에서 170,860개의 커밋을 포함하는 데이터를 대상으로 관련 CVE를 640개의 취약점 컨트리뷰트 커밋에 매핑하였다. 해당 프로젝트에서 프로젝트, 작성자, 커밋 및 파일 등 다양한 GitHub 메타 데이터를 포함한 데이터를 추출하고 SVC를 사용하여 줄립 커밋에서 취약점 기여 커밋을 식별하는 VCCFinder를 개발하였다. younis et al.은 악용될 가능성이 더 높은 취약점을 포함하는 코드의 속성을 확인하기 위한 연구를 수행하였다[14]. 리눅스 커널 및 아파치 웹 서버의 익스플로잇이 가능한 취약점 83개를 포함한 취약점 183개를 수집하여 취약점의 특성에 따라 4가지 카테고리로 나눈 후 소스코드 라인수, 패스 복잡도, 패스 수, 중첩도, 정보 흐름, 호출된 함수, 호출한 함수 및 호출 수 등 8개의 소프트웨어 메트릭을 선정하였다. wrapper subset 선택 접근법을 사용하여 취약점 예측 정확도 84%를 기록하였다.

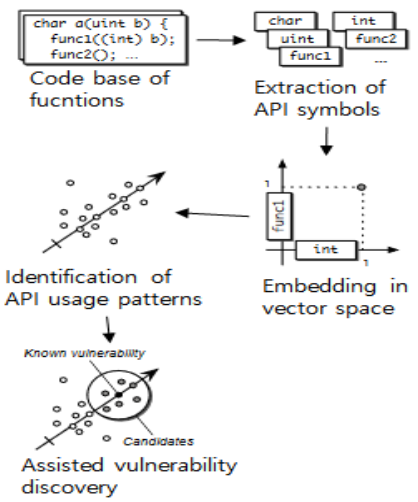
3.1.2. 취약한 코드 패턴 탐지

취약한 코드 패턴 탐지 연구는 취약점 코드 샘플에서 코드 세그먼트의 패턴을 추출하고 패턴 매칭을 사용하여 소프트웨어의 취약점을 탐지하고 찾아내는 연구를 의미한다.

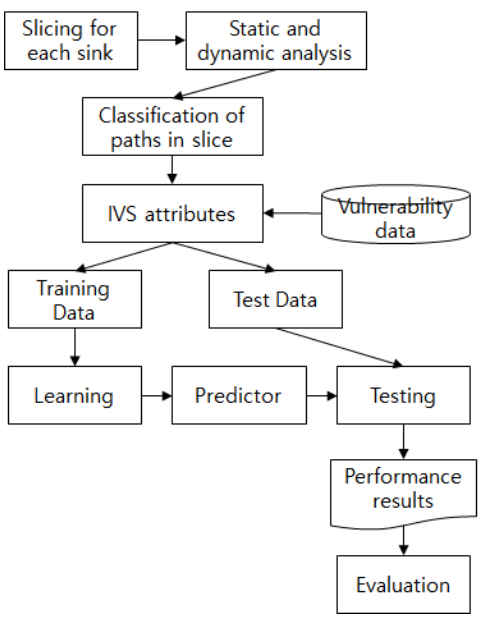
Yamaguchi et al은 알려진 보안 취약성에서 관찰된 프로그래밍 패턴을 기반으로 알려지지 않은 취약점을 식별하기 위한 방법을 연구하였다[15,16].

취약점 추정이라는 개념을 도입하여 코드베이스[17]에서 최근에 발견된 취약점의 유사한 인스턴스를 검색하여 취약점을 탐색하는 방법이다. 이를 위해 추상화된 신텍스 트리를 생성하고 이를 벡터 공간에 매핑시킨다. 코드 기반의 AST에서 자주 발생하는 코드의 구조 패턴을 식별하고, 코사인 거리를 이용하여 출력 행렬의 행을 비교함으로써 취약점 추상을 수행한다. 이러한 기법을 이용하여 최근 취약점을 시드로 사용하여 제로데이 취약점을 찾기도 하였다.

Shar et al.은 정적 속성의 제한된 예측 정확도와 라벨링 된 테스트 셋 필요성에 대한 문제를 해결하기 위해 소프트웨어의 실행 중 수집 가능한 동적 속성을 기반으로 취약점을 탐색하기 위한 방법을 연구하였다[18]. 제안된 동적 속성은 특정 입력 유효성 검사 및 명령문 수 등을 사용한다. 총 15개의 정적속성과 7개의 동적 속성을 추출하여 k-means와 유크리드 거리를 이용하여 취약점 탐색을 수행하였다. 또한 비지도학습 기반의 분류 모델을 통해 평균 76%의 정확도를 달성하였다. 이후 프로그램 슬라이스에서 다른 실행 경로를 추출하기 위해 정적 역방향 프로그램 슬라이싱 및 제어 종속성 정보를 사용하여 취약점 탐색을 위한 추가적인 연구를 수행하였다[19].



〔그림 2〕 코드 베이스를 이용한 취약점 추정 기법



〔그림 3〕 하이브리드 분석을 통한 취약점 예측 모델

Grieco et al은 제한된 시간과 예산을 통해 OS 규모의 큰 프로그램을 테스트하기 위한 방법을 연구하였다[20]. 바이너리 코드에 대한 기계학습을 기반으로 확장 가능한 취약점 분석 방법을 제안하였다. 경량의 정적 및 동적 분석을 사용하여 바이너리에서 쉽게 발견되는 메모리 손상 취약점을 포함하여 다양한 취약점에 대한 탐색이 가능하다.

3.1.3. 기타 탐지 방법

Wijayasekara et al.은 공개 버그 데이터베이스를 마이닝하고 분석하여 숨겨진 영향력을 갖는 취약점(Hidden impact vulnerabilities, HIV)을 찾기 위한 연구를 수행하였다[21]. HIV는 보안 영향을 받고 취약점으로 분류되기 전에 버그 데이터베이스를 통해 대중에게 공개되는 소프트웨어 버그로 리눅스 커널과 MySQL에서 보고된 취약점의 상당 수가 HIB(Hidden Impact Bug)로 보고되었음을 발견하였다. 이후 추가적인 연구를 통해 공개된 버그 데이터베이스에서 텍스트 마이닝과 버그 보고서 분석을 통해 HIB를 식별하기 위한 방법론을 제안하였다[22]. Alvares et al.은 악용 가능한 메모리 손상 취약점을 찾기 위해 AI를 이용한 프로그램 분석 및 인텔리전스에 기반한 하이브리드 방식을 제안하였다[23]. 프로그램 소스 코드를 개별적으로 분할하고 변수에 대한 정보를 추출한다. 또한 가능한 모든 할당 연산은 추상화 및 내부 절차에 따른 데어터 흐름 분석을 기반으로 각 함수에 도달 가능한 패스와 지역변수의 상태에 대한 다차원 검색 공간을 생성한다. 인텔리전스를 통해 지역 변수에 대한 상태 유발을 일으키는 입력 데이터를 식별한다. 이를 통해 적절한 시간에 취약점을 유발하는 입력값을 찾을 수 있다. Medeiros et al.은 정적 오연 데이터 흐름 분석을 통해 취약점 발견 알고리즘에서의 높은 오탐 문제를 해결하고자 노력했다[24]. 이를 위해 데이터 마이닝 및 기계 학습에서 사용하는 오탐을 예측하여 웹 어플리케이션 취약성을 자동 발견 및 수정하는 새로운 하이브리드 방식을 제안하였다. 오탐을 유발하는 14가지 속성 집합을 선택한 후 32개의 오탐을 포함하는 76개의 취약점 보고서를 가지고 학습을 수행하였을 때 로지스틱 회귀 알고리즘이 가장 좋은 성능을 보였다.

3.2. 바이너리 대상 보안 취약점 탐색

3.2.1. 기호실행(Symbolic Execution)

기호실행(symbolic execution)은 소프트웨어의 입력값에 대한 실행 경로를 분석하기 위한 기법이다. 입력값에 대한 실행 경로를 가지고 있다면, 역으로 실행 경로에 접근하기 위한 입력값의 생성도 가능하기 때문에 취약점 탐색을 위한 위치까지 범위를 탐색하는 용도로 많이 사용된다.

Li et al.은 기호실행을 활용하여 C++ 프로그램 대상의 자동화된 테스트케이스를 생성하는 방안을 제안했다[34]. C언어 기반의 기호실행 모듈을 C++로 확장하면서 C++의 예외처리, C++ RTTI, 메모리 모델등을 특징으로 추가하여 테스트케이스 생성에 효율성을 높이고자 노력하였다. 또한 c++용 라이브러리 최적화와 개체 수준에서 실행 및 추론, 특정 유형의 효율적인 Solver를 구현하였다. Luo et al.은 안드로이드를 대상으로 시스템 서비스 콜을 기호실행 분석함으로써 취약점을 탐색하기 위한 프레임워크인 CENTAUR를 제안하였다[35]. Android프레임워크의 심볼 실행을 가능하게 하는 미들웨어 특성 및 특이한 복잡성과 같은 프레임워크의 몇가지 고유한 특성으로 인해 많은 새로운 문제가 발생하한다. 이를 해결하기 위해 콘크리트 실행과 기호실행을 단계적으로 적용하여 패스를 최적화 하고, 경량화된 오염분석을 통해 특정 접근 패턴을 추적하여 악성 앱에서 파생된 변수를 기호 실행으로 식별한다.

3.2.2. 스마트 퍼징(Smart Fuzzing)

퍼징은 버그를 발견하는 효과적인 소프트웨어 테스트 기법이다. 하지만 실제 어플리케이션의 크기와 경로의 복잡성을 고려할 때 단순한 퍼징은 버그 및 취약점을 찾는데 효과적이지 않다. 이러한 이유로 퍼지을 수행하는데 있어 기계학습이나 템플릿 등을 활용하는 스마트 퍼징에 대한 많은 연구가 진행되고 있다.

Rawat et al.은 어플리케이션 또는 입력 형식에 대한 사전 지식이 필요하지 않은 퍼징 기법을 연구하였다[36]. 어플리케이션을 인식하는 진화적 퍼징 정책을 토대로 탐지 범위를 확대하기 위해 정적 및 동적 분석에 기초한 데이터 흐름을 분석하고, 이를 응용하여 소프트

웨어의 기본 속성을 추론하였다. 그 결과 어플리케이션에 제한이 없는 접근 방식에 비해 효율적으로 테스트케이스를 생성할 수 있다.

Ⅳ. 보안 취약점 자동 대응 기술 연구

지능형 취약점 대응 기법은 탐색된 취약점에 대해 효율적이고 자동화된 취약점 패치 및 대응을 위한 기법으로 크게 3가지 영역으로 구분할 수 있다.

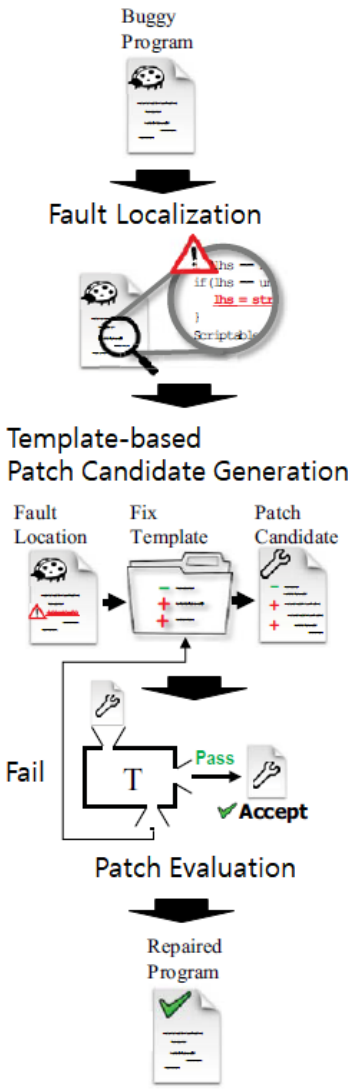
4.1. 소프트웨어 행위 패치 기반 취약점 대응 기술

행위 패치는 대상 소프트웨어의 동작은 변경하는 것으로 소스코드 뿐만아니라 바이너리를 대상으로도 수행가능한 방법이다. 이때 오프라인 또는 온라인으로도 수행할 수 있다. 소프트웨어의 행위를 패치하기 위한 방법으로는 코드를 추가/제거/대체하는 방법과 선행 조건을 추가하거나 제거하는 방법 등이 존재한다.

Weimer et al.은 테스트 케이스 기반의 자동화된 패치 시스템인 Genprog를 제안하였다[25, 26]. GenProg는 확장된 형태의 유전 프로그래밍을 사용하여 소프트웨어의 패치를 진행한다. 이때 필요한 기능은 유지하면서 발견된 결함에는 취약하지 않도록 패치를 생성한 후 테스트케이스를 이용하여 검증을 수행한다. 그 결과 120만 라인의 소스코드에서 6만 라인의 수정된 코드 및 8가지 타입의 에러를 수정하엿다. 또한 비정상 침입 탐지를 통한 closed-loop 패치에 대한 PoC를 수행하였으며, 패치의 품질을 위해 퍼징, 변형된 버그 발생 케이스 등 다양한 테스트를 수행할 수 있다. Arcuri는 유전 프로그래밍 기반의 소프트웨어 패치와 검색 기반 소프트웨어 테스팅을 이용한 테스트 케이스 생성 방안을 연구하였다[27]. 이를 위해 소프트웨어 패치와 테스트 케이스 생성 간 서로 연관성을 분석하여 공동 진화 프레임워크를 구축하여 소프트웨어 패치의 신뢰도를 향상시켰다.

Debroy와 Wong은 변이 엔진을 사용하여 프로그램 에러의 자동화된 수정 방안을 연구하였다[28]. 프로그램을 수정하기 위해 산술, 관계형, 논리식, 증가 및 감소, 또는 할당 연산자를 같은 클래스의 다른 연산자로 대체하는 형태로 패치를 수행한다. 이때 결함 상태의 위치는 스펙트럼 기반의 결함 위치 추정 기술을 사용한다.

Kim et al.은 자바 코드의 버그를 자동으로 수정하는 방법인 PAR 시스템을 제안하였다[29]. PAR는 복구 템플릿을 기반으로 패치를 생성하는데, 10개의 복구 템플릿 각각은 일반적인 종류의 버그를 수정하는 방법을 정의한다. 이를 위해 6만개 이상의 사람이 작성한 패치를 수동으로 검사한 결과 몇 가지 공통적인 수정 패턴을 발견하여, 해당 패턴을 활용한 패치를 자동을 생성하였다. 119개의 실제 버그 중 27개에 대한 패치를 성공적으로 생성하였다.

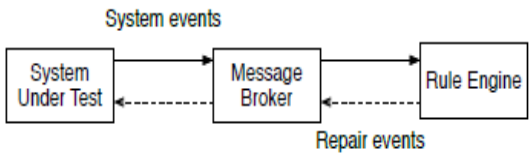


〔그림 4〕 템플릿 기반의 패치 생성 연구

4.2. 소프트웨어 상태 패치 기반 취약점 대응 기술

상태 패치는 프로그램의 상태를 변경하는 것으로 입력, 힙, 스택, 환경을 변경하거나 링크드 리스트에서 주가를 자동을 끊는 형태의 패치를 의미한다.

Candea et al.은 마이크로 리부트 개념을 도입하면서 세부적으로 재부팅이 가능한 계측정 구성요소를 통해 오류 발생 시 가장 작은 구성요소에서 가장 큰 구성요소로 재부팅 시도를 통해 상태를 패치하기 위한 연구를 수행하였다[30]. Smirnov et al.은 악의적인 페이로드를 통해 하이재킹을 시도하는 공격을 탐지하고 복구하는 코드 계측 시스템인 Dira를 제안하였다[31]. 공격 탐지에 필요한 구성 요소를 생성하고, 공격으로 인해 손상된 프로그램의 상태를 공격 이전 상태로 복원하기 위한 공격 복구 구성 요소를 생성하는 형태로 자동화된 패치를 진행한다. Ammann et al.은 소프트웨어 입력값의 제어를 통해 오류를 수정하기 위한 연구를 수행하였다[32]. 프로그램의 일부 입력이 실패하였을 때 상태를 변경하기 위한 방법 중 하나는 입력값을 수정하는 것으로, 수정된 입력값은 오류를 발생시키지 않는다는 것을 가정한다. Lewis et al.은 런타임 오류 모니터를 정의하여 이벤트 기반 소프트웨어에 대한 복구 방법을 제안하였다[33]. 입력, 출력 및 상태의 추상화로 구성된 이벤트는 메시지 브로커로 전송되고, 메시지 브로커는 이벤트를 테스트하기 위한 프로그램과 룰 엔진으로 이벤트를 전송한다. 룰 엔진에서는 이벤트에 대한 검증을 수행하고 에러로 판단된 행위가 발생할 경우 에러를 해결하기 위한 패치 이벤트를 전송한다.



〔그림 5〕 이벤트 기반 소프트웨어 복구 방안

Ⅴ. 결 론

본 논문에서는 지능형 취약점 탐색 및 대응을 위한 연구 동향을 소개하였다. 지능형 취약점 탐색 연구는 크게 2가지로 분류할 수 있다. 첫째는 취약점 예측 모델을 통해 취약점을 탐색하는 연구이고, 둘째는 기존 취약점을 활용하여 대상 소스코드나 바이너리 내에 동일한 패턴의 취약점이 존재하는지 탐색하는 연구이다. 지능형 취약점 대응에 관한 연구도 크게 2가지로 나눌 수 있는데, 첫 번째는 소프트웨어의 행위를 패치하는 연구이다. 코드를 대체, 추가, 변경하거나 특정 입력값 혹은 데이터 흐름 중 조건을 통해 흐름을 변경하는 방법에 대한 연구가 주로 수행되었다. 두 번째는, 소프트웨어의 오류 상태를 정상 상태로 패치하는 연구로 분류할 수 있다.

그러나 취약점 탐색 결과에 대한 높은 오탐율이나 미탐율 해결하기 위한 연구 및 취약점이 패치된 소프트웨어의 성능에 대한 검증 방안에 대한 추가적인 연구가 필요하다.

참 고 문 헌

[1] CVE Details, https://www.cvedetails.com/

[2] B. Arkin, S. Stender, G. McGraw, “Software penetration testing,” *IEEE Security and Privacy*, 3(1), pp. 84-87, 2005.

[3] Matt Bishop, “About penetration testing”, IEEE Security & Privacy, pp.84-87, 2007.

[4] Patrice Godefroid, “Random testing for security: Blackbox vs. whitebox fuzzing”, In Proceedings of the 2nd International Workshop on Random Testing (RT’07), 2007.

[5] M. E. Khan, F. Khan, “A comparative study of white box, black box and grey box testing techniques”, International Journal of Advanced Computer Science and Applications (IJACSA), 2012.

[6] Thomas Zimmermann, Nachiappan Nagappan, Laurie Williams, “Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista”, In Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST’10), pp. 421-428, 2010.

[7] Andrew Meneely, Laurie Williams, “Strengthening the empirical analysis of the relationship between linus’ law and software security”, In Proceedings of the ACM/IEEE

International Symposium on Empirical Software Engineering and Measurement (ESEM'10), 2010.

[8] MaureenDoyle, JamesWalden, "An empirical study of the evolution of PHPweb application security", In Proceedings of the 3rd International Workshop on Security Measurements and Metrics (MetriSec'11), pp.11-20, 2011.

[9] Yonghee Shin, Laurie Williams, "Can traditional fault prediction models be used for vulnerability prediction?", Empir. Softw. Eng, pp.25-59, 2013.

[10] Yonghee Shin, Laurie Williams, "An initial study on the use of execution complexity metrics as indicators of software vulnerabilities", In Proceedings of the 7th International Workshop on Software Engineering for Secure Systems(SESS'11), pp.1-7, 2011.

[11] Sara Moshtari, Ashkan Sami, Mahdi Azimi, "Using complexity metrics to improve software security", Computer Fraud & Security, pp.8-17, May 2011.

[12] James Walden, Jeffrey Stuckman, Riccardo Scandariato, "Predicting vulnerable components: Software metrics vs text mining",In Proceedings of the 25th International Symposium on Software Reliability Engineering (ISSRE'14), pp.23-33, 2014.

[13] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, Yasemin Acar, "VccFinder: Finding potential vulnerabilities in open-source projects to assist code audits", In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15), pp.426-437, 2015.

[14] Awad Younis, Yashwant Malaiya, Charles Anderson, Indrajit Ray, "To fear or not to fear that is the question: Code characteristics of a vulnerable function with an existing exploit", In Proceedings of the 6th ACM Conference on Data and Application Security and Privacy (CODASPY'16), pp.97-104, March 2016.

[15] Fabian Yamaguchi, Felix Lindner, Konrad Rieck, "Vulnerability extrapolation : Assisted discovery of vulnerabilities using machine learning", In Proceedings of the 5th USENIX Workshop on Offensive Technologies, 2011.

[16] Fabian Yamaguchi, Felix Lindner, Konrad Rieck. "Generalized vulnerability extrapolation using abstract syntax trees", In Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC'12), pp.359-368, 2012.

[17] Sean Heelan, "Vulnerability detection systems: Think cyborg, not robot", IEEE Security and Privacy, pp.74-77, 2011.

[18] Lwin Khin Shar, Hee Beng Kuan Tan, Lionel C. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis", In Proceedings of the 35th International Conference on Software Engineering (ICSE'13), pp.642-651, 2013.

[19] Lwin Khin Shar, Lionel C Briand, Hee Beng Kuan Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning", IEEE Transactions on Dependable and Secure Computing, pp.688-707, 2015.

[20] Gustavo Grieco, Guillermo Luis Grinblat, Lucas Uzal, Sanjay Rawat, Josselin Feist, Laurent Mounier, "Toward Large-scale Vulnerability Discovery Using Machine Learning", In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (CODASPY'16), pp.85-96, March 2016.

[21] Dumidu Wijayasekara, Milos Manic, Jason L. Wright, Miles McQueen, "Mining bug databases for unidentified software vulnerabilities", In Proceedings of the 5th International Conference on Human System Interactions (HSI'12), pp.89-96, 2012.

[22] Dumidu Wijayasekara, Milos Manic, Jason L. Wright, Miles McQueen, "Vulnerability identification and classification via text mining

bug databases", In Proceedings of the 40th Annual Conference of the IEEE Industrial Electronics Society (IECON'14), 2014.

[23] Dumidu Wijayasekara, Milos Manic, Jason L. Wright, Miles McQueen, "Applications of computational intelligence for static software checking against memory corruption vulnerabilities", In Proceedings of the IEEE Symposium on Computational Intelligence in Cyber Security (CICS'13), pp. 59-66, 2013.

[24] Iberia Medeiros, Nuno F. Neves, Miguel Correia, "Automatic detection and correction of web application vulnerabilities using data mining to predict false positives", In Proceedings of the 23rd International Conference on World Wide Web (WWW'14), pp. 63-74, 2014.

[25] W.Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically Finding Patches Using Genetic Programming", In Proceedings of the International Conference on Software Engineering, 2009.

[26] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, "GenProg: A Generic Method for Automatic Software Repair", IEEE transactions on software engineering, pp 54-72, 2012.

[27] A. Arcuri, "Automatic Software Generation and Improvement Through Search Based Techniques", PhD thesis. The University of Birmingham, 2009.

[28] V. Debroy, W.Wong, "Using Mutation to Automatically Suggest Fixes for Faulty Programs", In Proceedings of the International Conference on Software Testing, Verification and Validation, pp. 65-74, 2010.

[29] D. Kim, J. Nam, J. Song, S. Kim, "Automatic Patch Generation Learned From Human-Written Patches", In: Proceedings of ICSE, 2013.

[30] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, A. Fox, "Microreboot: a Technique for Cheap Recovery", In: Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation, pp. 31-44, 2004.

[31] A. Smirnov, T. Chiueh, "DIRA: Automatic Detection, Identification, and Repair of Control-hijacking Attacks", The 12th Annual Network and Distributed System Security Symposium, 2005.

[32] P. E. Ammann, J. C. Knight, "Data Diversity: An Approach to Software Fault Tolerance", Ieee transactions on computers, pp. 418-425, 2005.

[33] C. Lewis, J. Whitehead, "Runtime Repair of Software Faults Using Event-driven Monitoring", In Proceedings of the 32nd acm/ieee international conference on software engineering(icse '10), pp. 275-280, 2010.

[34] Guodong Li, Indradeep Ghosh, and Sreeranga P. Rajan, "KLOVER: A Symbolic Execution and Automatic Test Generation Tool for C++ Programs", IEEE Software, pp. 33-37, 2017.

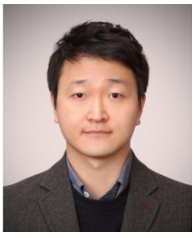
[35] L. Luo, Q. Zeng, C. Cao, K. Chen, J. Liu, L. Liu, N. Gao, M. Yang, X. Xing, and P. Liu, "System service call-oriented symbolic execution of android framework with applications to vulnerability discovery and exploit generation," In Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services. ACM, pp. 225-238, 2017.

[36] S. Rawat, V. Jain, A.Kumar, L. Cojocar, C. Giuffrida, H. Bos, "Vuzzer: Application-aware evloutionary fuzzing," In Proceedings of the Network and Districuted System Security Symposium(NDSS), 2017.

〈저 자 소 개〉

장 대 일 (Daeil Jang)
정회원

2016년~현재 : 한국인터넷진흥원 선임연구원
2012년~2015년 : ASU 박사 후 연구원
2012년 8월 : 전남대학교 정보보호협동과정 이학박사
2010년 8월 : 전남대학교 정보보호협동과정 석사
관심분야 : 악성코드/취약점 분석, IoT 보안



김 태 은 (Taeeun Kim)
정회원

2013년 ~ 현재 : 한국인터넷진흥원 선임연구원
2007년 2월 : 숭실대학교 컴퓨터학과(석사)
2005년 2월 : 백석대학교 정보통신학부(학사)
관심분야 : 네트워크 보안, 모바일 보안, 정보보안

김 환 국 (Hwankuk Kim)
종신회원

2017년 2월 : 고려대학교 정보보호대학원 공학박사
2007년~현재 : 한국인터넷진흥원 보안기술R&D2팀 팀장
2017년~현재 : TTA PG503(사이버보안) 의장
2001년~2006년 : 한국전자통신연구원 정보보호본부 연구원
관심분야 : ISMS, IoT 취약점 분석, 무선 네트워크 보안