바이트코드 분석을 통한 보안 취약점 검출 기술 동향

김대석○, 장주영, 배재호, 남재창
한동대학교 전산전자공학부
{naver, 21800637, human}@handong.ac.kr, jcnam@handong.edu

Survey on Security Vulnerability Detection Techniques based on Bytecode Analysis

Daeseok Kim○, Jooyoung Jang, Jaeho Bae, Jaechang Nam
School of Computer Science and Electronic Engineering, Handong Global University

요 약

본 연구는 보안 취약점 검출과 관련된 연구의 필요성을 조명한다. 조사 결과, 소스 코드 분석이나 바이너리 코드 분석을 통한 취약점 검출은 이미 상당수 진행되어 의미있는 성과들을 확인할 수 있었다. 그러나, 이러한 노력에도 불구하고 기존 연구들의 한계점들을 확인할 수 있었다. 본 문헌조사는 새로운 연구 방식의 방향성을 제시하기 위해 상대적으로 연구가 적게 이뤄진 보안 취약점 관련 자바 바이트코드 기반 연구 문헌조사에 집중했다. 문헌조사를 통해, 각 연구 방식에 따른 방향성과 한계점을 언급하고 선행 연구가 부족한 분야인 자바 바이트코드 연구를 통한 보안 취약점 검출이 필요하다는 결론에 도달했다.

1. 서론

본 논문은 보안 취약점 검출을 더욱 다양한 관점에서 연구할 필요성을 조명하기 위해 문헌조사를 진행하였다. 정보 보안에 대한 관심이 꾸준히 증대되고 있음에도 불구하고, 여전히 다양한 보안 취약 문제가 발생하고 있다. 예를 들어, 자바 분야에서 많은 보안 검출기법이 연구 및 논의되었음에도 2021년 자바 라이브러리의 일종인 Log4j[1]에 의해 재앙적인 보안 문제[2]가 발견되었다. 이는 현재의 기법에 안주하지 않고 더욱 다양한 관점에서 보안 문제를 검출해야 함을 의미한다. 이를 위해 바이트코드 분석, 소스 코드 분석, 바이너리 코드 분석 등 다양한 접근 방식을 기반으로한 선행 연구를 조사하였다.

문헌조사 결과, 소스 코드 분석이나 바이너리 코드 분석을 통한 취약점 검출은 이미 상당 수 진행되어 성과를 보이고 있었다[3, 4, 5]. 자바의 경우 이미 소스 코드를 기반으로 한 보안 취약점 검출모델이 많이 상용화되어있다[3, 6, 7]. 이는 GitHub를 비롯한 Version Control System(VCS)에서 다양한 오픈 소스 소프트웨어를 확보할 수 있었기 때문이다. 또한, C/C++의 경우 소스 코드, 바이너리 코드를 기반으로 한 다양한 연구가 이뤄지고 있다. 운영체제를 비롯한 컴퓨터의 주요 기능을 이루는 핵심 프로그램은 C/C++ 소스 코드를 컴파일하여 생성된 바이너리 코드로 구성되어 있다. 그러나 바이너리 코드는 보안에 취약한 모습을 보인다[8]. 이를 보완하기 위한 다양한 보안 취약점 연구가 진행되고 있다. 그러나, 이러한 노력에도 불구하고 여전히 보안의 위협은 존재한다.

본 문헌조사는 새로운 연구 방식의 방향성을 제시하기 위해 상대적으로 연구가 적게 이뤄진 바이트코드 기반 보안 취약점 탐색 연구에 집중했다. 지금까지 바이트코드 분야에서 행해진 Ethereum Virtual Machine(EVM), Java Server Page(JSP), 자바 바이트코드 등 여러 보안 취약점 검출을 다루는 방법에 대해 문헌조사를 실시했다.

EVM의 경우 소스 코드를 연구에 활용하기 어려운 스마트 컨트렉트의 배포방식으로 인해 바이트코드 연구가 활발히 진행되고 있다[9]. 하지만 이는 EVM 환경에 국한된 연구로써 Java Virtual Machine(JVM)과 같은 대중적인 플랫폼에 적용하기에는 무리가 있다. JVM 환경의 경우 SQL injection, JSP Webshell과 같은 특정 분야에만 치중된 보안 취약점 연구가 진행되었다는 한계를 지닌다. 또한 바이트코드 분석이 소스 코드 분석을 위한 보조 도구에 불과하거나 후속 연구가 부족하다는 한계점이 있었다.

이와 같은 문헌조사를 통해, 각 연구 방식에 따른 방향성과 한계점을 언급하고 선행 연구가 부족한 분야인 자바 바이트코드를 활용한 보안 취약점 검출 연구가 필요하다는 결론에 도달했다. 이는 바이트코드를 기반으로 하는 추가적인 보안 취약점 검출연구가 확장될 방향을 논의하고 제시해야 할 필요성을 보여준다.

2. 연구배경 및 방법

2021년 대두된 Log4j 보안 문제와 같이 오래전부터 존재해왔던 보안 취약점이 드러남에 따라 보안 취약점 검출 및 수정을 비롯한 그에 따른 후속 연구의 필요성이 강조되었다[1]. 이를 위해 다양한 접근 방식을 활용한

연구가 요구된다고 판단하였다. 그 중 바이트코드를 기반으로 하는 보안 취약점 검출법 연구에 초점을 뒀다.

본 조사는 주로 국외 논문을 다루었기 때문에 Google Scholar와 Google 검색 엔진을 통한 인터넷 조사와 대학 도서관에 비치된 자료들을 열람하는 방식으로 이루어졌다. 검색 기간은 특정 키워드에만 2018년부터 2022년으로 설정하였고 그 외 키워드에는 적용하지 않았다.

검색 키워드는 주로 Vulnerability Detection, Security, JVM, Byte Code와 Source Code, Java를 조합 사용하였다. 또한 검색된 논문 중 인용된 횟수가 많은 논문을 기준으로 해당 논문이 인용한 논문 목록을 활용하여 문헌조사가 이루어졌다.

3. 문헌조사 결과(총 참고 자료 53개)

해외 학술지 논문 및 기사 38편을 분석하였고, 5편의 문헌조사 결과를 참고했다. 또한 실제 사용되는 모델을 참고하기 위하여 10개의 정식 홈페이지를 사용하였다. 자바 바이트코드에 대한 보안 취약점을 검출하는 선행 연구의 수에 비해 이더리움 바이트코드와 C/C++의 바이너리 코드와 소스 코드를 통한 보안 취약점을 검출하는 연구가 많아 사용한 언어로 기준으로 대분류한 후, 접근 방식에 따라 다시 소분류 하여 요약하였다.

3.1 이더리움과 바이트코드

바이트코드를 이용한 보안 취약점 개선이 가장 활발하게 연구되고 있는 분야는 EVM 환경이다. Chen et al.은 이더리움 플랫폼에서 흔히 발견되는 29개의 주의해야 할 보안 취약점을 정리하며, 각 취약점을 검출하고 정정하기 위해 이더리움 바이트코드(EVM bytecode) 분석 기법을 주로 사용한다[9]. 공개된 이더리움 소스 코드의 비중으로도 바이트코드 분석에 연구가 집중되고 있다는 사실을 유추할 수 있다. 2022년 기준 이더리움 네트워크에 배포된 약 5000만 개의 이더리움 스마트 컨트렉트 중 단 0.3%(152,996)만이 소스 코드가 오픈되어있다[10, 11].

3.1.1 이더리움에서 바이트코드 연구가 활발한 이유

EVM 환경에서 바이트코드 연구가 활발히 이뤄지고 있는 이유는 스마트 컨트렉트 배포방식[12] 때문이다. 스마트 컨트렉트는 Solidity라는 고레벨(high-level) 언어로 작성되어 컴파일러를 통해 바이트코드로 변환되어 만들어진 객체이다. 그 후, 해당 객체는 이더리움 네트워크상의 다른 피어들에 전달되어 스마트 컨트랙트를 발생시킨 생산자도 임의로 수정할 수 없도록 무결성을 보장한다. 이에 따라 네트워크상의 사용자들은 바이트코드 객체만 확인할 수 있다. 따라서 모든 사용자는 역컴파일러(decompiler)[13]가 있지 않는 이상 해당 바이트코드의 원본 소스 코드 형태는 확인할 수 없다. 또한 배포된 스마트 컨트렉트들은 EVM내 에서 서로 객체에 내장된 바이트코드 기반 함수를 호출하는 경우가 많다. 위와 같은 이유로 EVM 환경에서 연구자들이 소스 코드를 이용한 연구를 진행하지 못하는 한계가 있다. 이러한 한계로 인해 이더리움 2.0에서는 소스 코드를 활용한 연구보다 바이트코드를 통한 연구가 활발하게 진행 중이다.

3.1.2 이더리움 바이트코드 분석을 통한 보안 취약점 검출및 수정 기법 연구 현황

이더리움 플랫폼은 주로 바이트코드 분석을 통해 보안 취약점을 검출한다. Kushwaha et al.은 이더리움 플랫폼에서 발견된 취약점의 근본원인에 따라 23개로 분류하였다[14]. 표 1의 S1~17은 재진입성, Denial of Service, 정수 오버플로우와 같은 Solidity 언어에서 비롯된 17가지 취약점을 일컫는다. 그밖에도 이더리움 가상머신(EVM)에 의한 VM 취약점 두 종류와 이더리움 블록체인 설계 결함으로 인한 트랜잭션 에러와 같은 Blockchain Design(BD) 취약점이 네 종류 존재한다. 표 1은 SmartCheck, Oyente, Mythril과 같은 대표적인 보안 취약점 검출도구들이 탐지할 수 있는 취약점을 보여준다.

표 1. 이더리움 플랫폼의 23개 대표적인 취약점과 이를 탐지할 수 있는 17개 검출 도구의 바이트코드 활용 여부[14]

Vulnerability Detection Tool	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	VM1	VM2	BD1	BD2	BD3	BD4	BYTECODE
SmartCheck	o	o	o	o	o	o	o		o										o	o				X
Oyente			o	o		o	o		o											o		o		O
Securify			o	o		o	o											o	o			o		O
Remix				o		o	o		o											o				X
Mythril						o				o			o							o		o		O
EasyFlow					o																			O
Vultrone								o																O
Gasper									o															O
Verx											o													X
MadMax												o												O
Mythx													o				o							X
SolidityCheck														o										O
ContractGuard															o									X
ContractFuzzer																o								O
EVMFuzzer																		o	o					X
Town Crier																							o	X
Hawk																					o			X

소개된 17개의 검출 도구 중 바이트코드를 분석하는 도구의 개수는 9개이며, 이들만으로 총 23종의 취약점 중 16종을 탐지할 수 있다. 또한 상용화된 스마트 컨트렉트 분석 도구 25개 중 15개가 바이트코드를 분석 대상으로 삼는다.

위와 같은 검출모델을 활용한 바이트코드 기반 보안 취약점 수정 기법을 ByteCode Rewriting[15]이라고 부른다. 이를 기반으로 파생된 모델 중 sGuard[16]와 개량형인 Elysium[17] 등이 실제로 성과를 보인다. sGuard는 4개의 취약점을 검출 후 수정하였다. Elysium의 경우 기본적으로 7개의 취약점을 검출 후 수정이 가능하며, 추가적인 patch template을 제공하여 적용할 수 있는 보안 취약점의 확장성을 보장한다. 이러한 특성을 통하여 Elysium은 EVM 바이트코드 관련 후속 연구를 진행할 때도 활용할 가능성이 충분히 있는 것으로 사료된다. 그러나 이는 앞서 언급한 것처럼 JVM과 같은 대중적인 플랫폼이 아니기에 그대로 타 플랫폼에 적용하기에는 무리가 있다.

3.2 자바 보안 취약점 관련 연구 현황

자바의 경우 보안 취약점을 검출하는 연구는 크게 소스 코드와 바이트코드 분석을 통하여 이루어졌다. 자바는 소스 코드를 통한 분석에 방점이 실려있고 바이트코드 분야에서는 선행 연구의 한계가 존재하거나 활발한 논의가 이루어지지 못했다.

3.2.1 자바 바이트코드 분석을 통한 보안 취약점 검출 기법

자바에서 바이트코드 분석을 통한 보안 취약점 검출기법에 관한 연구는 그리 활발하게 연구되지 못하고 있다. 유의미한 자바 바이트코드 분석을 통한 보안 취약점 검출 프레임워크[18]가 고안되었으나, 실용화되지 못하였고 후속 연구도 부족했다. Jackson et al.[19]은 자바 바이트코드를 활용해 보안 취약점 중 하나인 SQL injection을 검출하는 모델을 만들었다. 해당 연구에서는 자연어 기반 자바 바이트코드 보안 취약점 분석을 통해 SQL injection을 찾아냈다. 그러나 SQL injection만을 타겟으로 하는 보안 취약점 분석이라는 것에 한계가 있다. Pu et al.[20]은 JSP에서의 Webshell을 해결하기 위하여 진행되었다. 해당 연구에서는 Tomcat Server 환경에서 구동되는 JSP 파일을 바이트코드 단위로 컴파일하였다. 컴파일된 해당 코드에 대해 BERT(Bidirectional Encoder Representations from Transformers)[21]를 접목시켜 자연어 처리를 진행한 후 XGBoost 알고리즘을 활용해 Webshell을 검출한다. 이러한 방식을 접목해 유의미한 결과를 얻을 수 있었다. 그러나 이 방식 역시 자바의 전반적인 적용이 아닌 JSP에 한정된 보안 취약점 검출 방법이라는 한계점을 지닌다.

이 밖에도 자바 바이트코드 분석을 사용하는 모델로는 FindBugs[22]와 발전형인 SpotBugs[23] 가 있다. 이들은 사용자에게는 소스 코드를 제공받아 자신들이 가지고 있는 보안 취약점을 가진 바이트코드 패턴에 매칭시키는 방식으로 이루어진다[24]. 다만 바이트코드를 분석함에도 소스 코드에서 보안 취약점을 찾고자 하는 데에 초점이 맞춰져 있다. 이러한 점은 대다수의 논문에서 이들을 소스 코드 분석 모델로 언급하는 데에서 확인할 수 있다[3, 6, 7]. 많이 사용되는 모델임에도 두 개의 한계점이 있다. 첫째, 전체 보안 취약점 결과 중 35~91%가 위양성이다[25-33]. 둘째, 보안 취약점을 검출하기 위해서는 컴파일된 코드가 필요하다. 이는 개발 단계에서 코드를 스캔하려는 개발자를 번거롭게 한다는 단점이다[6].

3.2.2 기존 자바 바이트코드 관련 연구의 한계

연구 결과 자바의 바이트코드 분석을 통한 보안 취약점 검출기법에 관한 연구나 논의가 많이 이루어지고 있지 않았음을 알 수 있었다. 자바 바이트코드 연구들은 대부분 연구 결과가 발표된 후 오랜 시간이 지났지만, 후속연구가 부족하거나[18], 광범위한 보안 취약점 검출이 아니라 한 분야에만 특화된 모습을 보이기 때문이다[19, 20]. 또는 FindBugs나 SpotBugs처럼 바이트코드를 활용하여 검출하더라도 모델의 초점은 소스 코드 보완에 맞춰져 있는 모습을 보였다. 이러한 연구의 편향된 모습에 의해 자바 바이트코드 전반의 보안 취약점 검출이 이루어지고 있지 않은 것으로 판단된다.

이처럼 기존의 보안 취약점 검출기법은 한계점을 지니고 있다. 이는 보안 문제를 야기할 수 있다. 앞서 언급한 것처럼 Log4j가 오랫동안 산재해 있었음에도 2021년이 되어서야 발견되었고 보완하는 데에 많은 시간과 노력이 걸린 것[1]이 그 방증이다. 바이트코드 분석을 통한 보안 취약점 검출기법이 더욱 논의 되었다면 간과되어온 여러 보안 문제에 대해 다방면으로 분석할 수 가능성이 있을 수 있다.

4. 논의

4.1 소스 코드 분석 외의 기법이 보안 취약점 검출에 유의미한 성과를 보일 수 있는가?

바이트코드 혹은 바이너리 코드 분석 기법은 배포된 실행파일만으로 보안 취약점을 검사할 수 있다는 장점이 있다. 실제 명령어들과 그 동작을 파악하기 용이하여 보안 취약점 검출에 적합하기 때문이다. 특히 C/C++ 계열 프로젝트들은 소스 코드 분석 기법에 비해 컴파일 에러 혹은 최적화 문제에서 비교적 자유롭다는 장점[4]도 있다. 그 때문에 C/C++를 대상으로 한 많은 연구가 소스 코드 분석뿐만 아니라[34-37], 바이너리 코드도 함께 분석하며[4, 5, 38] 사용하고 있음을 알 수 있었다. 이처럼 C/C++에서는 소스 코드를 기반으로 한 모델과 바이너리 코드도 기반으로 한 모델의 각기 다른 검출범위를 포괄하여 보안 취약점 검출의 저변을 넓힌 것이다[3, 4, 5].

바이트코드를 통해 소스 코드 없이도 분석할 수 있다는 장점은 오픈 소스 라이브러리가 풍부한 자바에도 적합하다. 다양한 플랫폼에서 여러 언어와 교차해서 사용하거나[39], 외부 라이브러리를 참조하는 경우[40]에도 바이트코드 분석 기법은 도움이 될 수 있다.

그러나 문헌조사 결과 자바 보안 취약점 검출영역에서 바이트코드를 대상으로 진행한 연구는 제한적이다. 특정한 도메인에서만 작동하는 탐지 도구 혹은 아이디어 제안만이 존재한다[18-20].

4.2 자바에서의 바이트코드를 활용한 보안 취약점 탐색 기법의 수가 적은 이유에 대한 고찰

자바는 충분히 많은 JVM 기반의 소스 코드가 공유되어 있어[41], 바이트코드 기반 연구가 상대적으로 잘 이루어지지 않은 것으로 분석된다. 자바는 다른 언어나 플랫폼에 비해 다양한 목적과 구조의 소스 코드들에 대한 접근성이 매우 뛰어나다. 따라서 바이트코드를 기반으로 연구해야 할 관심과 이유가 상대적으로 적었음을 유추할 수 있었다. 이는 바이트코드를 통한 보안 취약점 검출과 관련된 기초 연구가 활성화되지 않은 것과 연관이 있다. 기존의 제한적인 도메인[19, 20]과 조건을 만족해야 하는 선행 연구들과 달리, 프로젝트에 일반적으로 적용할 수 있는 바이트코드 기반의 보안 취약점 탐지 연구가 필요하다.

4.3 학습 기반 연구가 보안 취약점 검출에 도움이 되는가?

보안 문제를 해결하기 위한 많은 최신 연구에서 학습 기반 도구들을 사용한다[42-46]. 특히 원시 데이터(primitive data)로 학습을 시키거나[43], 위협(Threat) 우선순위 문제[42] 등에 머신 러닝을 활용하는 사례가 자주 발견된다. 또한 EVM 환경에서도 머신러닝을 활용한 연구가 시도되었다. 범용성 높은 정적 분석 도구인 Mythril에 비해 머신러닝을 활용한 ABCNN은 오탐률이나 검출시간면에 있어 더 개선된 성능을 보인다[47]. 그러나 머신 러닝은 지속적인 재학습과 매개변수 조정이 필요하며, 개별 위협에 맞추어 학습된 서로 다른 모델(Classifier)을 사용해야 하는 한계점이 있다[48].

딥러닝은 머신러닝보다 더 많은 데이터를 자가 학습할 수 있고, 새로운 유형을 탐지하기 유리하다[45, 49, 50]. 그 예로, SmartConDetect[51]는 스마트 컨트렉트 상의 취약한 코드 패턴을 Solidity 코드로 사전 훈련된 BERT 모델을 사용하여 탐지한다. Choi et al.의 연구는 바이트코드에도 naturalness가 존재함을 확인하여[52], 소스 코드와 AST에만 사용되던 자연어 처리 딥러닝 모델을 바이트코드에도 활용할 수 있는 가능성을 열었다.

5. 결론

본 연구는 자바 보안 취약점 검출을 위해 바이트코드 분석의 필요성을 제고한다. 자바는 많은 오픈 소스를 제공하며[53], 전 세계에서 4번째로 많이 사용되는 프로그래밍 언어이다[41]. 재앙적인 보안 취약점으로 불릴 만큼 위협적이었던 Log4j 사태[2]가 자바에 기반을 두고 있는 만큼 자바 소프트웨어에서의 보안이 중요해졌다. 기존의 자바 보안 취약점 검출연구는 소스 코드에 집중되어 있다. C/C++ 영역에서 소스 코드와 바이너리 코드가 동시에 보안 취약점 검출을 위해 논의, 연구된 것과 같이 자바에도 유사한 접근 방식이 적용되어야 한다. 소스 코드 분석만으로 보장하지 못한 취약점을 바이트코드로 접근할 때 보완할 수 있기 때문이다. 비록 환경이 다르지만 EVM 바이트코드를 활용하여 보안 취약점을 검출, 보완하는 기법들이 활발히 연구되고 있다. 그러나 자바 영역에서는 보안 취약점 검출에 바이트코드를 사용한 선행연구가 드물다. 따라서 유용하고 범용성 있는 바이트코드 기반 보안 취약점 탐지 기법을 연구해야할 필요성이 있다. 더불어 보안 취약점 검출문제를 머신러닝과 딥러닝을 활용해 해결하려는 시도가 최근 들어 늘어나고 있다[47, 49-51]. 해당 모델들은 기존의 모델에 비해 성능이 향상된 모습을 보인다. 따라서 향후 자바 바이트코드 분석을 통한 보안 취약점 검출연구에 최신 딥러닝 기법을 적용하여 검출 성능을 개선할 여지가 크다.

※이 논문은 과학기술정보통신부의 소프트웨어중심대학 지원사업(2017-0-00130)과 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행하였음. (No. 2021R1F1A1063049).

6. 참고 문헌

[1] Hiesgen, Raphael, et al. "The Race to the Vulnerable: Measuring the Log4j Shell Incident." *arXiv preprint arXiv:2205.02544* (2022).
[2] “Apache log4j Flaw: A Fukushima Moment for the Cybersecurity Industry.” Tenable®, 17 Dec. 2021, https://www.tenable.com/blog/apache-log4j-flaw-a-fukushima-moment-for-the-cybersecurity-industry.
[3] Kaur, Arvinder, and Ruchikaa Nayyar. "A comparative study of static code analysis tools for vulnerability detection in C/C++ and JAVA source code." *Procedia Computer Science* 171 (2020): 2023-2029.
[4] Song, Dawn, et al. "BitBlaze: A new approach to computer security via binary analysis." *International conference on information systems security*. Springer, Berlin, Heidelberg, 2008.
[5] Brumley, David, et al. *BitScope: Automatically dissecting malicious binaries*. Technical Report CS-07-133, School of Computer Science, Carnegie Mellon University, 2007.
[6] Mahmood, Rahma, and Qusay H. Mahmoud. "Evaluation of static analysis tools for finding vulnerabilities in Java and C/C++ source code." *arXiv preprint arXiv:1805.09040* (2018).
[7] Lenarduzzi, Valentina, et al. "A critical comparison on six static analysis tools: detection, agreement, and precision." *Journal of Systems and Software* 198 (2023): 111575.
[8] Shoshitaishvili, Yan, et al. "Sok:(state of) the art of war: Offensive techniques in binary analysis." *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016.
[9] Chen, Huashan, et al. "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses." *ACM Computing Surveys (CSUR)* 53.3 (2020): 1-43.
[10] Google. 2022. BigQuery - Ethereum Dataset.

Retrieved Sep 20, 2022 from https://console.cloud.google.com/bigquery?project=bigquery-public-data&d=crypto_ethereum&p=bigquery-public-data&page=dataset
[11] Ortner, M., Eskandari, S. (n.d.). Smart Contract Sanctuary. Retrieved from https://github.com/tintinweb/smart-contract-sanctuary
[12] "Introduction to Smart Contracts." Edited by Paul Wackerow, Ethereum.org, 2 Sept. 2022, https://ethereum.org/en/developers/docs/smart-contracts/.
[13] Grech, Neville, et al. "Elipmoc: advanced decompilation of Ethereum smart contracts." *Proceedings of the ACM on Programming Languages* 6.OOPSLA1 (2022): 1-27.
[14] Kushwaha, Satpal Singh, et al. "Systematic review of security vulnerabilities in ethereum blockchain smart contract." IEEE Access (2022).
[15] Ayoade, Gbadebo, et al. "Smart contract defense through bytecode rewriting." *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2019.
[16] Nguyen, Tai D., Long H. Pham, and Jun Sun. "SGUARD: towards fixing vulnerable smart contracts automatically." *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.
[17] Ferreira Torres, Christof, Hugo Jonker, and Radu State. "Elysium: Context-Aware Bytecode-Level Patching to Automatically Heal Vulnerable Smart Contracts." *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*. 2022.
[18] Hong, Tang, et al. "The vulnerability analysis framework for java bytecode." *2009 15th International Conference on Parallel and Distributed Systems*. IEEE, 2009.
[19] Jackson, Kevin A., and Brian T. Bennett. "Locating SQL injection vulnerabilities in Java byte code using natural language techniques." *SoutheastCon 2018*. IEEE, 2018.
[20] Pu, Ao, et al. "BERT-Embedding-Based JSP Webshell Detection on Bytecode Level Using XGBoost." *Security and Communication Networks* 2022 (2022).
[21] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).
[22] Hovemeyer, David, and William Pugh. "Finding bugs is easy." *Acm sigplan notices* 39.12 (2004): 92-106.
[23] "SpotBugs Manual." SpotBugs Manual - Spotbugs 4.7.3 Documentation, https://spotbugs.readthedocs.io/en/stable/.
[24] Grindstaff, C. "FindBugs, Part 1: Improve the quality of your code." (2004).
[25] Aggarwal, Ashish, and Pankaj Jalote. "Integrating static and dynamic analysis for detecting vulnerabilities." *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*. Vol. 1. IEEE, 2006.
[26] Boogerd, Cathal, and Leon Moonen. "Prioritizing software inspection results using static profiling." *2006 Sixth IEEE International Workshop on Source Code Analysis and Manipulation*. IEEE, 2006.
[27] Heckman, Sarah, and Laurie Williams. "On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques." *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. 2008.
[28] Heckman, Sarah, and Laurie Williams. "A model building process for identifying actionable static analysis alerts." *2009 International conference on software testing verification and validation*. IEEE, 2009.
[29] Kim, Sunghun, and Michael D. Ernst. "Prioritizing warning categories by analyzing software history." *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007.
[30] Kim, Sunghun, and Michael D. Ernst. "Which warnings should I fix first?." *Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 2007.
[31] Kremenek, Ted, et al. "Correlation exploitation in error ranking." *ACM SIGSOFT software engineering notes* 29.6 (2004): 83-93.
[32] Kremenek, Ted, and Dawson Engler. "Z-ranking: Using statistical analysis to counter the impact of static analysis approximations." *International Static Analysis Symposium*. Springer, Berlin, Heidelberg, 2003.
[33] Choi, Yoon-ho, and Jaechang Nam. "WINE: Warning miner for improving bug finders." *Information and Software Technology* 155 (2023): 107109.
[34] D. Wheeler, "Flawfinder," 2017. [Online]. Available: https: //www.dwheeler.com/flawfinder/
[35] RATS [online] available: "https://security.web.cern.ch/security/recommendations/en/codetools/rats.shtml,", 2019
[36] CPPCHECK [online] available: ": http://cppcheck.sourceforge.net/, 2019
[37] Scovetta, Michael. "Yasca." Scovettacom, 2008, https://www.scovetta.com/yasca/.
[38] Balakrishnan, Gogul, et al. "Codesurfer/x86—a platform for analyzing x86 executables." *International Conference on Compiler Construction*. Springer, Berlin, Heidelberg, 2005.
[39] Figueiredo, Alexandra, Tatjana Lide, and Miguel Correia. "Multi-language web vulnerability detection." *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2020.
[40] Hong, Tang, et al. "The Vulnerability Analysis of Java Bytecode Based on Points-to Dataflow." *International Journal of Computer and Information Engineering* 3.6 (2009): 1540-1543.
[41] "TIOBE Index for December 2022." *TIOBE*, 3 June 2022, https://www.tiobe.com/tiobe-index/.
[42] Le, Triet HM, Huaming Chen, and M. Ali Babar. "A survey on data-driven software vulnerability assessment and prioritization." *ACM Computing Surveys* 55.5 (2022): 1-39.
[43] Sarker, Iqbal H., et al. "Internet of things (iot)

security intelligence: a comprehensive overview, machine learning solutions and research directions." *Mobile Networks and Applications* (2022): 1-17.
[44] Hwang, Seung-Yeon, Dong-Jin Shin, and Jeong-Joon Kim. "Systematic Review on Identification and Prediction of Deep Learning-Based Cyber Security Technology and Convergence Fields." *Symmetry* 14.4 (2022): 683.
[45] Suresh, P., et al. "Contemporary survey on effectiveness of machine and deep learning techniques for cyber security." *Machine Learning for Biometrics*. Academic Press, 2022. 177-200.
[46] Momeni, Pouyan, Yu Wang, and Reza Samavi. "Machine learning model for smart contracts security analysis." *2019 17th International Conference on Privacy, Security and Trust (PST)*. IEEE, 2019.
[47] Sun, Yuhang, and Lize Gu. "Attention-based machine learning model for smart contract vulnerability detection." Journal of Physics: Conference Series. Vol. 1820. No. 1. IOP Publishing, 2021.
[48] Apruzzese, Giovanni, et al. "On the effectiveness of machine and deep learning for cyber security." *2018 10th international conference on cyber Conflict (CyCon)*. IEEE, 2018.
[49] Jasim, Ammar D. "A survey of intrusion detection using deep learning in internet of things." *Iraqi Journal For Computer Science and Mathematics* 3.1 (2022): 83-93.
[50] Rodriguez, Eva, et al. "A survey of deep learning techniques for cybersecurity in mobile networks." *IEEE Communications Surveys & Tutorials* 23.3 (2021): 1920-1955.
[51] Jeon, Sowon, et al. "SmartConDetect: Highly Accurate Smart Contract Code Vulnerability Detection Mechanism using BERT." *Proceedings of the 2021 KDD Workshop on Programming Language Processing, Virtual Conference*. 2021.
[52] Choi, Yoonho, and Jaechang Nam. "On the Naturalness of Bytecode Instructions." (2022).
[53] The Top Programming Languages, Github, https://octoverse.github.com/2022/top-programming-languages.