서 동 수 교수 지도

석사학위 청구논문

기계학습에 기반한 보안약점의 탐지 및 분류에 관한 연구

2018

성신여자대학교 대학원

컴퓨터학과

이 원 경

기계학습에 기반한 보안약점의 탐지 및 분류에 관한 연구

서 동 수 교수 지도

이 논문을 석사학위논문으로 제출함

2018년 5월

성신여자대학교 대학원

컴퓨터학과

이 원 경

인 준 서

이원경의 석사학위 논문으로 인준함

2018년 5월

심사위원장 ..... (서명 또는 인)

심 사 위 원 ..... (서명 또는 인)

심 사 위 원 ..... (서명 또는 인)

성신여자대학교 대학원

논 문 개 요

소프트웨어에 내재되어있는 결함을 통해 발생하는 보안 사고가 늘어나고 있는 추세이다. 이에 소프트웨어 보안 사고는 발생 후에 대처를 하는 것보다 사전에 소프트웨어의 결함을 제거함으로써 예방하는 것이 그에 따른 피해를 줄이는 데 효과적이라는 인식이 전반적으로 공감대를 얻고 있으며, 이러한 인식의 일환으로 우리나라는 2012년 소프트웨어 개발 보안 제도를 만들어 시행 중이다. 소프트웨어 개발보안제도의 핵심은 프로그램의 구현단계에서의 보안 활동인 시큐어코딩이다. 시큐어코딩과 연계된 개발보안 활동은 보안 취약점이 내재되어 있는 소스코드의 특징과 패턴을 찾아내는 것이며, 소스코드의 정확한 분석을 위해 자동화된 정적 분석기의 활용을 권고하고 있다.

그러나 현실적으로는 코딩 방법과 개발자의 스타일에 따라 다양한 형태로 소스코드가 작성 될 수 있기 때문에 보안에 취약하다고 판단되는 패턴을 탐지하기 위한 룰셋을 만들어주고 정적 분석 도구에 적용 시키는 일은 여전히 전문가의 개입을 필요로 한다. 또한 소스코드 분석은 고려해야 될 변수가 많으며, 어떤 취약점을 탐지할 건지에 따라 소스코드를 분석하는 관점을 달리해야하기 때문에 정적분석 도구의 복잡성을 높이고 정확한 진단을 어렵게 만든다는 문제가 있다.

본 연구에서는 이러한 문제를 완화시키기 위한 방법으로 기계학습 알고리즘의 도입을 제안하는 바이다. 기계학습은 최근 음성신호, 자연어 처리, 사물개체 인식 등과 같이 사람의 인지 능력으로 판단할 수 있던 문제 영역을 해결하는 성과를 보이고 있다. 이와 같은 최근 연구 사례를 보았을 때, 전문가의 의해 정의되었던 안전하지 않은 소스코드의 룰셋을 기계학습을 이용해

보안 취약점을 식별하고 탐지할 수 있는지 실험 결과를 통해 알아보고자 한다. 또한 실험 결과를 기반으로 기계학습을 이용한 정적분석의 가능성을 평가하고 발전방향을 제시하고 연구를 마친다.

목 차

표 목 차

그 림 목 차

I. 서 론

1. 연구의 배경 및 목적

수십 년에 걸쳐 이루어진 인터넷의 보급과 소프트웨어의 성장은 산업 분야에 그치지 않고 일상생활 플랫폼에까지 소프트웨어가 활용되며 웹과 모바일 애플리케이션의 보편화를 가속시켰다. 그러나 소프트웨어의 활용이 빠르게 확산되는 만큼 보안사고 및 해커의 공격이 늘어나게 되며 피해 규모 또한 늘어나고 있는 추세다.

이러한 보안 사고는 소프트웨어에서 제거되지 않은 치명적인 결함에 의해 발생하며, 악의적인 공격자가 결함을 통해 매개변수 변조, URL 변조와 같은 데이터 삽입형 공격을 수행하는 것뿐만이 아니라, SQL 인젝션, 크로스 사이트 스크립트, 명령어 삽입과 같은 보안 사고는 확인되지 않은 입력 값에 의해 예상치 못한 실행 오류가 발생되기도 한다[1]. 즉, 소프트웨어의 결함을 제거하는 것은 단순히 오류나 버그를 없애고자 하는 목적뿐만이 아니라 공격의 대상이 되는 보안 취약점을 제거하는 측면에서도 중요한 부분으로 볼 수 있는 것이다.

2002년 미국 국립표준기술연구소(NIST, National Institute of Standards and Technology)는 소프트웨어의 결함이 제품출시 이후 발견되고 수정할 경우 개발단계에서 수정하는 것과 비교하여 최대 30배의 비용 차이가 난다는 결과를 발표했다[3]. 이는 소프트웨어의 결함은 발생 직후 대처하는 것보다 사전에 제거하는 것이 비용, 품질 측면에서 효과적이라는 것을 알 수 있게 하는 부분이다.

이와 같이 취약점 제거를 위한 사전 대처에 대한 인식이 증대되며 체계적

인 방법과 관리를 통해 소프트웨어의 결함을 제거하고 하는 필요성이 거론되었다. 이에 따라 Microsoft와 OWASP는 소프트웨어 개발 생명 주기의 모든 단계에서 고려해야 할 보안 사항 또는 모범 사례를 제시하는 S-SDLC (Secure Software Development Life Cycle) 보안 방법론[4][13]을 마련하였고 안전한 소프트웨어를 만들기 위한 가이드를 제안하고 있다.

[표 1] 소프트웨어 개발단계별 결함 수정 비용[3]

Where Errors are Introduced	Where Errorsare Found				
	Requirements Gathering and Analysis / Architectural Design	Coding / Unit Test	Integration and Component / RAISE System Test	Early Customer Feedback / Beta Test Programs	Post-product Release
Requirements Gathering and Analysis / Architectural Design	1.0	5.0	10.0	15.0	30.0
Coding / Unit Test	-	1,0	10.0	20.0	30.0
Integration and Component / RAISE System Test	-	-	1.0	10.0	20.0

시큐어코딩은 소스코드를 작성하는 과정에서 지켜야 할 안전한 코딩 규칙과 보안에 위협을 줄 수 있는 소스코드 취약 목록을 제시하여 소프트웨어의 결함과 취약점을 줄이는 것을 목적으로 한다. 현재 우리나라는 행정자치부와 한국인터넷진흥원이 공공 분야의 웹 보안 감사 기준을 시큐어코딩으로 채택해 개발 단계에서의 보안 활동에 대한 인식을 확산 시키고 있을 뿐만 아니라[5], 공공기관에 한정적으로 활용되었던 시큐어코딩을 금융업으로까지 적용 분야를 넓힐 계획을 밝혀 앞으로 시큐어코딩의 적용 분야는 더욱 확대

될 것으로 전망된다.

시큐어코딩을 위해 수행되는 정적분석은 어떤 취약점 유형이 소스코드에 내포되어 있는지 파악하는 것이 선행 작업으로 이루어져야 한다. 이를 위해 국내외로 OWASP, CWE, 한국인터넷진흥원과 같은 기관들은 보안 취약점 유형을 특징에 따라 분류하고 위험도를 평가하여 정보를 제공하고 있으며, 정적분석 도구들도 이를 기준으로 취약점 유형을 분류하는 기능을 수행한다. 이런 점에 있어 기계학습의 지도학습(supervised learning)을 통한 분류, 비지도 학습(unsupervised learning)을 통한 군집화(clustering)와 같은 기법을 활용하여 취약점 유형별로 소스코드의 패턴을 탐지하고 분류를 수행할 수 있을 것으로 보인다. 또한, 취약점 유형에 따라 효과적으로 탐지가 가능한 정적분석 기법이 달라 기존의 정적분석 도구들이 점차 복잡성이 증가하고 있던 문제점을 기계학습 모델이 대용량의 훈련 데이터를 이용하여 자동으로 학습함으로써 해결 될 수 있다는 점도 기대해 볼 수 있다.

이에 본 논문은 기존 정적분석 도구의 정확한 취약점 유형 식별과 탐지에 기여하기 위한 방법으로 기계학습 알고리즘의 도입을 제안하는 바이다. 기계학습의 여러 모델 중에서도 일반 신경망보다 깊은 신경망 층을 쌓아 비선형적인 예측이 가능한 심층신경망(DNN, Deep Neural Network)과 이미지 내의 패턴을 학습하여 개체를 식별하는데 활용되고 있는 합성곱 신경망(CNN, Convolutional Neural Network)을 적용해 보안에 취약한 소스코드 패턴의 탐지가 가능한지 실험을 진행하고 Accuracy 수치를 중심으로 한 실험 결과를 제시하고자 한다.

2. 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 통해 정적분석과 시큐어 코딩의 정의와 활용 현황에 대해 알아보고 실험에 사용될 기계학습

모델에 대해 설명한다. 3장에서는 기계학습 실험에 필요한 훈련 데이터에 대한 설정과 기계학습 모델에 따라 고려해야 될 실험 변수에 대해 알아보고 신경망 모델의 특성에 맞춰 실험 설정을 해준다. 4장에서는 3장에서의 실험 결과를 제시하여 기계학습 모델별 탐지율의 정확도를 비교하고 5장에서는 결론 및 향후 계획을 통해 발전 방향에 대해 논의를 하고 마무리 한다.

Ⅱ. 관련 연구

1. 시큐어코딩

1) 소프트웨어 개발 보안 방법론

소프트웨어가 개발되어지는 과정을 요구사항 분석, 설계, 개발, 테스트, 운영의 단계로 나누어 관리하는 소프트웨어 개발 생명 주기와 같이 소프트웨어의 보안 활동도 체계적인 개발 보안 방법론을 통해 보안사고의 피해를 최소화하고 신속하게 대응하려는 인식이 증대되며 S-SDLC (Secure Software Development Life Cycle)가 제안되었다. S-SDLC는 소프트웨어 개발 생명주기의 모든 단계에서 고려해야 할 보안 사항 또는 모범 사례를 제시한 방법론으로 안전한 소프트웨어를 만들기 위한 가이드의 역할을 한다[13].

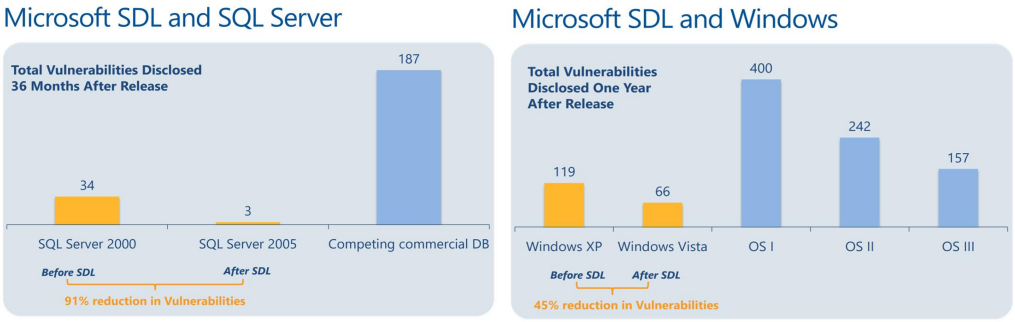
Microsoft사는 2004년 소프트웨어 취약성의 수와 심각도를 줄이기 위해 개발 보안 프로세스인 MS-SDL을 설계하였다[4]. 교육, 요구사항 분석, 설계, 구현, 테스트, 운영, 대응으로 총 7단계를 나눈 방법론을 채택하여 보안 활동을 구성하였다.

1. TRAINING	2. REQUIREMENTS	3. DESIGN	4. IMPLEMENTATION	5. VERIFICATION	6. RELEASE	7. RESPONSE
1. Core Security Training	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	Execute Incident Response Plan
	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

(그림 1) MS-SDL Process [4]

이후 MS사는 실제 자사 제품 개발에 MS-SDL 도입 이전과 이후의 효과를 비교한 결과 Windows 운영체제의 경우 119개였던 취약점이 66개로 감

소하여 45%의 취약점이 줄어들었다. 타사의 운영체제 제품 중 가장 적은 취약점의 개수가 157개인 것과 비교 해봐도 월등히 줄어든 걸 알 수 있다. SQL Server 제품에서는 34개의 취약점이 3개로 감소하여 91%의 절감 효과를 보였으며 경쟁사의 데이터베이스 제품에서 발견된 취약점의 개수가 187개인 것과 비교했을 시 약 61배의 차이가 나는 것을 확인 할 수 있다. 이로써 MS-SDL과 같은 소프트웨어 개발 보안 방법론이 소프트웨어의 보안 수준을 높이고 취약성 사고의 심각성과 비용을 낮춘다는 것을 입증하며[8], 체계적인 단계를 거쳐 취약점을 제거하는 보안 활동이 수행되면 결과적으로 소프트웨어의 보안성, 품질이 향상된 소프트웨어의 개발이 가능함을 보였다.



(그림 2) Microsoft products : Vulnerabilities reduction after SDL implementation

2) 시큐어코딩

시큐어코딩은 S-SDLC와 같은 보안 방법론의 단계 중 구현단계에 해당하는 보안활동으로 소스코드를 작성하는 과정에서 지켜야 할 안전한 코딩 규칙과 보안에 위협을 줄 수 있는 소스코드 취약 목록을 제시하여 소프트웨어가 배포되기 전 정적분석을 통해 소프트웨어의 결함과 취약점을 줄이는 것을 목적으로 한다. 현재 우리나라는 행정안전부와 한국인터넷진흥원이 2012년 소프트웨어 보안 의무 제도를 마련하며 공공 분야의 웹 보안 감사 기준

을 시큐어코딩으로 채택해 개발 단계에서의 보안 활동에 대한 인식을 확산시키고 있다[5]. 뿐만 아니라 공공기관에 한정적으로 활용되었던 시큐어코딩이 핀테크의 활성화에 힘입어 금융업으로 적용 분야를 넓힐 계획을 밝혀 중요성이 증대되고 있다.

3) 보안 취약점 유형

보안 취약점은 어떤 기준으로 점검할 것인지에 따라 취약점이 분류되는 카테고리와 세부 취약점 유형, 진단항목의 개수가 결정된다. 국내에서는 행정자치부와 한국인터넷진흥원이 소프트웨어 개발보안 가이드를 배포하여 보안 취약점 진단항목과 분류 기준에 대한 설명을 제공하고 있으며, 국정원 또한 공공기관 및 일반 회사가 운영하는 공개 게시판에서 공격된 취약점 8개 분야를 선정하고 발표한 바 있다. 국외에서는 CWE (Common Weakness Enumeration), CVE (Common Vulnerabilities and Exposures), OWASP (Open Web Application Security Project) 중심으로 취약점을 관리하는 데이터베이스를 제공하고 있다[10][11][12].

이번 장에서는 국내 사례로 한국인터넷진흥원과 행정자치부, 국외 사례로 OWASP의 보안 취약점 유형에 대해 알아본다.

① 한국인터넷진흥원, 행정자치부

우리나라는 2012년 시큐어코딩 의무화 법안을 발표하면서 한국인터넷진흥원과 행정자치부에서 행정기관 및 공공기관 정보시스템 구축과 운영 시 지켜야하는 개발 보안 관련 지침과 소프트웨어 개발 보안 가이드를 배포하고 있다. 공공기관의 전자정부 시스템의 경우 해당 개발 보안 가이드를 준수하여 소프트웨어를 개발하도록 지시되고 있다.

2017년 개정된 소프트웨어 개발 보안 가이드에서는 분석 단계에 적용해야

할 보안 요구사항의 4개 항목과 구현단계에서 발생할 수 있는 보안 취약점 유형 47개의 연관관계를 정리하여 정보를 제공하고 있다[5]. 보안 취약점 제거는 구현단계에서만 한정적으로 수행해야하는 활동이 아니라 분석·설계 단계에서부터 보안취약점 기준들이 고려돼야함을 강조하기 위함이다. [표 2]와 [표 3]은 한국인터넷진흥원에서 제시하고 있는 47개의 보안 약점을 분석, 설계, 구현 단계별로 연관관계를 표로 정리한 것이다.

[표 2] 분석·설계 단계 보안요구항목 분류와 구현 단계 47개 보안약점 기준 [5]

입력값 검증 보안기능 에러처리 세션통제

구분	구현 단계 보안약점					
입력 데이터 검증 및 표현	SQL 삽입	경로 조작 및 자원 삽입	크로스사이트 스크립트	운영체제 명령어 삽입	위험한 형식 파일 업로드	신뢰되지 않은 URL 주소로 자동 접속 연결
	XQuery 삽입	XPath 삽입	LDAP 삽입	크로스사이트 요청 위조	HTTP 응답분할	정수형 오버플로우
	보안기능 결정에 사용되는 부절절한 입력값	메모리 버퍼 오버플로우	포맷 스트링 삽입			
보안 기능	적절한 인증 없는 중요기능 허용	부적절한 인가	중요한 자원에 대한 잘못된 권한 설정	취약한 암호화 알고리즘 사용	중요정보 평문 저장	중요정보 평문 전송
	하드코드 된 비밀번호	충분하지 않은 키 길이 사용	적절하지 않은 난수값 사용	하드코드 된 암호화 키	취약한 비밀번호 허용	사용자 하드디스크에 저장되는 쿠키를 통한 정보노출
	주석문 안에 포함된 시스템 주요정보	솔트 없이 일방향 해시함수 사용	무결성 검사 없는 코드 다운로드	반복된 인증시도 제한 기능 부재		
시간 및 상태	검사시점과 사용시점	종료되지 않은 반복문 재귀함수				

에러 처리	오류메시지를 통한 정보노출	오류 상황 대응 부재	부적절한 예외 처리			
코드 오류	Null Pointer 역참조	부적절한 자원 해제	해제된 자원 사용	초기화되지 않은 변수 사용		
캡슐화	잘못된 세션에 의한 데이터 정보노출	제거되지 않고 남은 디버그 코드	시스템 데이터 정보노출	Public 메서드로부터 반환된 Private 배열	Private 배열에 Public 데이터 할당	
API 오용	DNS Lookup에 의존한 보안 결정	취약한 API 사용				

[표 3] 구현 단계 보안약점과 분석·설계 단계 보안요구항목의 연관관계 [5]

구분	s분석·설계 단계 보안요구항목	구현 단계 보안약점
입력 데이터 검증 및 표현	DBMS 조회 및 결과 검증	SQL 삽입
	XML 조회 및 결과 검증	XQuery 삽입 XPath 삽입
	디렉터리 서비스 조회 및 결과 검증	LDAP 삽입
	시스템 자원 접근 및 명령어 수행 입력값 검증	경로조작 및 자원삽입 운영체제 명령어 삽입
	웹 서비스 요청 및 결과 검증	크로스사이트 스크립트
	웹 기반 중요기능 수행 요청 유효성 검증	크로스사이트 요청 위조
	HTTP 프로토콜 유효성 검증	신뢰되지 않은 URL 주소로 자동접속 연결 HTTP 응답분할
	허용된 범위내 메모리 접근	포맷스트링 삽입 메모리 버퍼 오버플로우
	보안기능 동작에 사용되는 입력값 검증	보안기능 결정에 사용되는 부적절한 입력값 정수형 오버플로우 Null Pointer 역참조
	업로드·다운로드 파일 검증	위험한 형식 파일 업로드 무결성 검사 없는 코드 다운로드
보안 기능	인증대상 및 방식	적절한 인증 없는 중요기능 허용 DNS lookup에 의존한 보안결정
	인증수행 제한	반복된 인증시도 제한기능 부재
	비밀번호 관리	하드코드된 비밀번호 취약한 비밀번호 허용
	중요자원 접근통제	부적절한 인가 중요한 자원에 대한 잘못된 권한 설정

	암호키 관리	하드코드된 암호화 키 주석문 안에 포함된 시스템 주요 정보
	암호연산	취약한 암호화 알고리즘 사용 충분하지 않은 키 길이 사용 적절하지 않은 난수 값 사용 솔트없이 일방향 해시함수 사용
	중요정보 저장	중요정보 평문저장 사용자 하드디스크에 저장되는 쿠키를 통한 정보노출
	중요정보 전송	중요정보 평문전송
에러 처리	예외처리	오류메시지를 통한 정보노출 시스템 데이터 정보노출
세션 통제	세션통제	잘못된 세션에 의한 데이터 정보노출

② OWASP (Open Web Application Security Project)

OWASP는 신뢰할 수 있는 애플리케이션을 설계, 개발, 인수, 운영 및 유지 관리를 할 수 있도록 지원하는 비영리 개방형 커뮤니티이다[13]. 전 세계의 기업, 교육기관, 개인이 모두 참여 할 수 있으며, 3년을 주기로 OWASP Top 10을 발간하여 웹에서 발생할 수 있는 대표적인 취약점 10개를 발표하고 있다.

OWASP Top 10-2017에서 발표한 대표적인 취약점 10개와 그에 대한 설명은 [표 4]와 같으며, (그림 3)은 선정된 취약점 10개의 위험도를 평가하여 점수로 나타내고 있다[13].

[표 4] OWASP Top 10 Application Security Risks - 2017 [14]

분류	설명
A1:2017-Injection	명령 또는 쿼리의 일부로 신뢰할 수 없는 데이터가 삽입되어 의도하지 않은 명령을 실행하거나 적절한 권한 없이 데이터에 액세스 할 수 있게 되는 공격이다. SQL 삽입, NoSQL 삽입, OS 삽입, LDAP 삽입과 같은 결함이 이에 해당한다.

A2:2017-Broken Authentication	인증 및 세션 관리의 관련된 애플리케이션 기능이 잘못 구현되어 공격자가 암호, 키 또는 세션 토큰을 손상시키거나 다른 구현 결함을 이용하여 다른 사용자의 ID를 임시 또는 영구적으로 도용하는 취약점이다.
A3:2017-Sensitive Data Exposure	많은 웹 애플리케이션과 API는 금융, 의료 정보와 같은 개인정보를 제대로 보호하지 않기 때문에, 공격자들이 신용 카드 사기, 신분 도용 또는 기타 범죄를 수행하기 위해 보안에 취약한 데이터를 훔치거나 수정하는 취약점이다.
A4:2017-XML External Entities (XXE)	오래되거나 잘못 구성된 XML 프로세서는 XML 문서 내에서 외부 엔티티 참조를 평가한다. 외부 엔티티를 사용하여 파일 URI 핸들러, 내부 파일 공유, 내부 포트 검색, 원격 코드 실행 및 서비스 거부 공격을 사용하여 내부 파일이 공개 될 수 있는 취약점이다.
A5:2017-Broken Access Control	인증된 사용자가 수행할 수 있는 작업에 대한 제한이 제대로 적용되지 않는 경우가 많다. 공격자는 이러한 결함을 이용하여 다른 사용자의 계정 액세스, 중요한 파일 보기, 다른 사용자의 데이터 수정, 액세스 권한 변경 등과 같이 무단으로 데이터에 액세스 할 수 있는 취약점이다.
A6:2017-Security Misconfiguration	가장 흔하게 발생하는 문제로, 일반적으로 안전하지 않은 기본 구성, 불완전하거나 임시적인 구성, 개방형 클라우드 저장소, 잘못 구성된 HTTP 헤더 및 중요한 정보가 포함된 상세 오류 메시지로 인해 발생한다. 이를 보완하기 위해서 모든 운영체제, 프레임워크, 라이브러리 및 애플리케이션을 안전하게 구성해야 할 뿐만 아니라 최신의 업데이트 상태로 소프트웨어를 유지해야 한다.
A7:2017-Cross-Site Scripting (XSS)	신뢰할 수 없는 외부 값을 적절한 검증 없이 웹 브라우저로 전송하는 경우 발생되는 취약점으로, 사용자 세션을 가로채거나 홈페이지 변조, 악의적인 사이트 이동 등의 취약점이 발생된다.
A8:2017-Insecure Deserialization	보완되지 않은 직렬화는 원격 코드 실행으로 이어지는 경우도 있다. 비활성화 결함으로 인해 원격 코드가 실행되지 않더라도 재생 공격, 주입 공격 및 권한 상승 공격을 포함한 공격을 수행하는 데 사용 할 수 있다.

A9:2017-Using Components with Known Vulnerabilities	슈퍼유저권한으로 운영되는 취약한 라이브러리, 프레임워크 및 기타 소프트웨어 모듈로 인해 데이터유실 및 서버 권한획득과 같은 취약성이 발생된다.
A10:2017-Insufficient Logging & Monitoring	로깅 및 모니터링이 부족하고 장애 대응과 통합되지 않거나 효과적이지 않을 경우 공격자는 추가 공격 시스템을 유지하고 더 많은 시스템을 사용하며 데이터를 조작, 추출 또는 폐기 할 수 있다.

RISK	Threat Agents	Attack Vectors	Security Weakness		Impacts		Score
		Exploitability	Prevalence	Detectability	Technical	Business	
A1:2017-Injection	App Specific	EASY: 3	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	8.0
A2:2017-Authentication	App Specific	EASY: 3	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A4:2017-XML External Entities (XXE)	App Specific	AVERAGE: 2	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE: 2	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT: 1	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	MODERATE: 2	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE: 2	WIDESPREAD: 3	DIFFICULT: 1	MODERATE: 2	App Specific	4.0

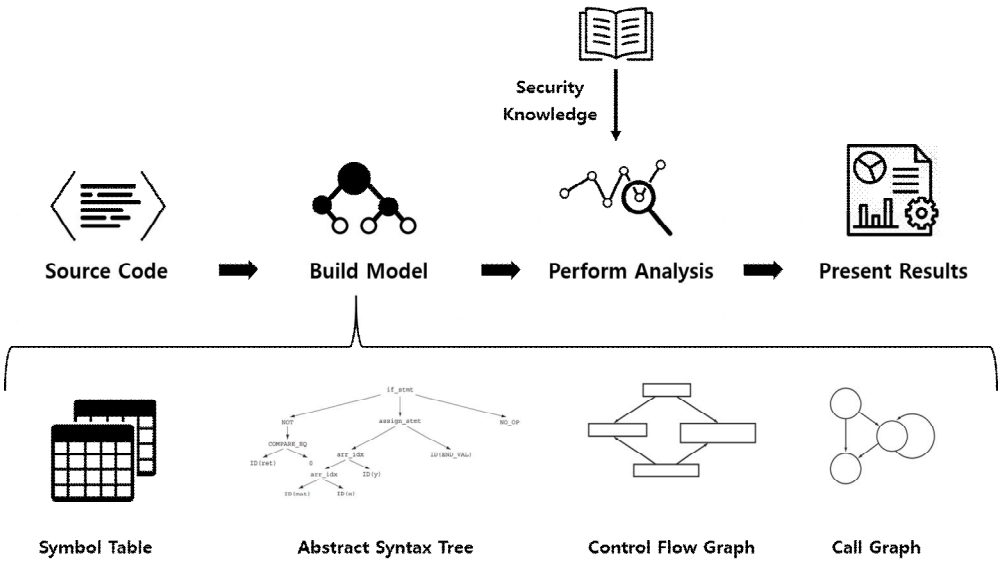
(그림 3) OWASP Top 10 Risk Factor Summary [14]

4) 정적분석

정적분석은 프로그램을 실행시키지 않고 소스코드의 그 자체를 분석하는 테스팅 기법으로 소스코드 모듈 안의 논리적인 구조를 점검하는 용도로 쓰인다. 소스코드의 문법, 코딩규칙, 보안 취약점, 실행 오류 등을 점검하여 발생할 수 있는 취약점과 결함을 찾아내는 것이 정적분석의 역할이며 상세설계 및 코딩을 진행하면서 결함을 찾아내어 사전에 조치를 취할 수 있다는

점에서 개발 효율성과 비용 절감을 유도한다는 장점을 가진다[6].

정적분석을 하기 위해선 우선적으로 텍스트 형태의 코드문을 규칙과 구조에 기반한 특정 모델로 변환시킨 뒤 보안약점의 패턴과 비교하여 진단을 하는 과정을 거쳐야 한다. 이러한 과정을 정적분석 기법이라고 부르며 소스코드를 변환시키는 모델의 종류에 따라 정적분석 기법의 종류가 나뉘게 된다. [표 5]는 시큐어코딩에서 사용하는 대표적인 정적분석 기법으로 소스코드에서 탐지하려는 대상과 목적에 따라 사용되는 기법과 그에 따른 특징을 정리하였다.



(그림 4) Static Analysis Process

[표 5] 정적분석 기법 종류[2][7]

정적 분석 기법	설명
패턴 매칭 (Pattern Matching)	- 보안 약점이 있는 함수를 패턴으로 등록하고 소스코드를 탐색해 동일 패턴이 발견될 때 검출하는 방법 - 가장 단순한 방법이나 부정확하며 많은 오탐(False Positive)을 발생시킴

어휘 분석 (Lexical Anaysis)	- 소스코드로부터 분리한 토큰 열이 데이터베이스에 등록된 보안 약점 함수의 패턴과 일치하는지 확인하는 방법 - 패턴 매칭에 비해 탐지 기능이 개선되었으나 여전히 오탐의 비율이 높음
AST 분석 (Abstract Syntax Tree Analysis)	- 파싱 결과물로 생성된 추상구문트리를 탐색하며 수행되지 않는 명령, 미사용 변수, 적절하지 않은 파라미터의 함수 등의 보안 약점을 탐지하는 방법 - 세밀한 데이터 분석보다 코딩 스타일에 관련된 보안 약점을 탐지하는데 적합
타입 체커 (Type Checker)	- 타입의 불일치 오류에 관련된 취약성을 탐지하는 방법 - 타입 불일치에 대한 정보를 찾을 수 있는 취약성이 작다는 한계가 있음
자료흐름 분석 (Data Flow Analysis)	- 정적 분석 방법 중 보안 약점 탐지에 기본이 되는 기법으로 버퍼 오버플로우, 정수 오버플로우, 널 포인터 역참조 오류 등 시스템에 위협이 되는 위험한 오류 탐지에 효과적인 방법 - 정교한 분석이 가능하고 대부분의 사용도구에서 채택하고 있는 기법

위의 표를 통해서 알 수 있듯이 여러 보안 취약점을 정적분석 기법으로 탐지하기 위해선 하나의 기법만 사용 할 수 없고 복합적으로 기법을 적용해야 한다. 데이터 중심, 코드 패턴 중심, 함수 호출 흐름 중심 등 보안 유형에 따라 소스코드의 어떤 특성을 중심으로 분석할 것인지 관점을 달리해야 하기 때문이다.

국내외로 정적분석 도구를 활용하여 보안 약점을 탐지하고 위협으로부터 안전한 시스템을 만들고자 하는 필요성이 증가함에 따라 기업에 의해 운영되는 상용화 도구와 대학, 연구기관과 같은 비영리기관에서 제공하는 오픈소스 기반의 도구들이 활용되고 있다.

앞서 기재된 [표 5]에서 보았듯이 정적분석 기법은 많은 기술을 필요로 하기 때문에 그만큼 분석 도구 또한 복잡한 구조를 가지게 된다. 이러한 이유로 국내외에서 상용화되고 있는 정적분석 도구들은 전문가에 의해 개발, 유지보수가 되는 만큼 프로그램의 가격이 고가인 경우가 많다. 그렇다보니 중소기업, 연구 목적의 비영리 기관의 경우 상용화된 정적분석 도구를 도입하는 데에 진입벽이 높아 PMD, FindBugs, Yasca, CppCheck와 같은 오픈소스로 제공되는 정적분석 도구를 활용하고 있다.

NIST SAMATE에서는 국내외에 출시되어 있는 정적분석 도구 목록을 제공하고 있다[8]. 또한 도구별 지원하는 프로그래밍 언어, 탐지 가능한 유형, 유무료 라이센스 구분, 특징 등의 설명을 수록하여 분석의 목적에 맞는 도구를 선택할 수 있다. [표 6]은 NIST SAMATE에서 공개한 정적분석 도구와 국내에서 상용화되고 있는 정적분석 도구를 정리한 표이다.

[표 6] 정적분석 도구 종류 [6][8]

도구명	제조사	라이센스	지원언어
Astree	AbsInt.	상용화	C
SPARK tool set	AdaCode	상용화	SPARK (Ada subset)
Kiuwan	Kiuwan	상용화	Abap, ActionScript, ASP, COBOL, C/C++, C#, JAVA, Python, PHP, JSP 등
QA-C, QA-C++, QA-J	PRQA	상용화	Ruby on Rails
SPARROW	파수닷컴	상용화	C/C++, Java, JSP, JavaScript, C#, ASP(.NET), Objective-C, PHP, VB.NET, VBScript, HTML, SQL, XML
Fortify SCA	Micro Focus	상용화	C#, Visual Basic, JavaScript, VB Script
Code Inspector	슈어 소프트	상용화	C/C++, Java 등

CAST AIP	CAST	상용화	Abap, .NET, ASP.NET, VB.NET, C3, LINQ to Objects, LINQ to Data Sets, C/C++, Visual C, Java JDK, HTML, XHTML, MS SQL, JSP, Javascript 등
Sofos Coding	한컴 시큐어	상용화	C/C++, Java 등
CodePrisom / Security Prisom	지티원	상용화	C, JSP, Java 등
Code Ray	트리니티 소프트	상용화	C, Java 등
FindBugs	GNU	오픈소스	Java, Groovy, Scala
PMD	BSD	오픈소스	Java
Roslyn Security Guard	GNU	오픈소스	C#
Yasca	GNU	오픈소스	Java, C/C++, JavaScript, ASP, ColdFusion, PHP, COBOL, .NET 등
CppCheck	GNU	오픈소스	C, C++
WAP	GNU	오픈소스	PHP
pylint	GNU	오픈소스	Python

정적분석 도구의 중요성이 증가하고 국내외로 배포되고 있는 도구들을 통해 소프트웨어의 품질을 높이고자 하는 연구가 지속적으로 진행되고 있음을 알 수 있지만 그럼에도 불구하고 정적분석 도구는 여러 요인으로 인해 문제점이 남아 있으며 다음과 같이 기술적 요인, 환경적 요인, 도구가 가진 성능에 따라 문제점을 정리할 수 있다.

① 소스코드 분석의 어려움

정적분석의 주요 진단 대상이 되는 소스코드는 일반적인 텍스트 정보와는 달리 복잡한 구조를 가지고 있어 정적분석 결과의 정확도를 높이는데 어려

움으로 남아있다. 프로그래밍 언어 종류, 문법, 개발자 스타일 등 여러 방법으로 소스코드가 작성될 수 있어 비정형적인 특성을 가지면서도 컴파일 단계를 거치기 위해 파싱이 가능하도록 문법적 구조를 지켜야 되는 정형적인 특징도 가져 일반적인 자연어 처리로는 분석의 한계가 있다.

그렇기 때문에 소스코드를 특정한 모델로 변환하여 분석을 하는 패턴 매칭, 어휘 분석, AST 분석과 같은 정적분석 기법이 연구되고 활용되고 있지만[15], 소스 코드 품질에 영향을 주는 요인으로 함수 복잡도 수준, 함수 간 호출 및 연관관계 수준, 전역변수 수, 함수 및 모듈 수, 사용되지 않는 변수 및 함수 수준, 주석 수준, 그리고 코딩 표준 준수 수준 등이 있어 고려해야 될 변수가 많아 소스코드 분석의 복잡성을 낮추는 데에 한계가 있다[6].

② 소프트웨어 개발 환경적 요인

현 시대의 소프트웨어는 빅데이터, 클라우드, 모바일, 보안, 미디어, 임베디드와 같은 다양한 형태의 기술이 융복합적으로 통합되어 개발되어진다[6]. 그리고 소프트웨어에 적용되는 기술의 개수만큼 고려해야 되는 서로 다른 개발 언어와 환경도 비례하게 증가하므로 소스코드의 품질이 지나친 편차를 보이며 구현단계에서 발생하는 결함을 최소화 하는 것이 어려워지고 있다.

또한 급격하게 변화하는 기술만큼 보안 취약점을 이용한 공격도 변화하고 고도화 되고 있다. 이로 인한 새로운 취약한 소스코드 패턴의 등장은 정적분석 도구에 룰셋을 추가하고 검증을 하는 작업이 요구 되는데, 이와 같은 유지보수 작업은 분석을 위한 전문가의 개입 빈도수를 높이고 계속해서 추가되는 룰셋으로 인해 정적분석 도구의 내부 프로그램 복잡도가 높아진다는 문제점이 있다.

[표 6]을 통해서 볼 수 있듯이 오픈소스 기반의 정적분석 도구가 진단 할 수 있는 프로그래밍 언어의 종류가 한정적이고, 룰셋이 다양하지 못한다는

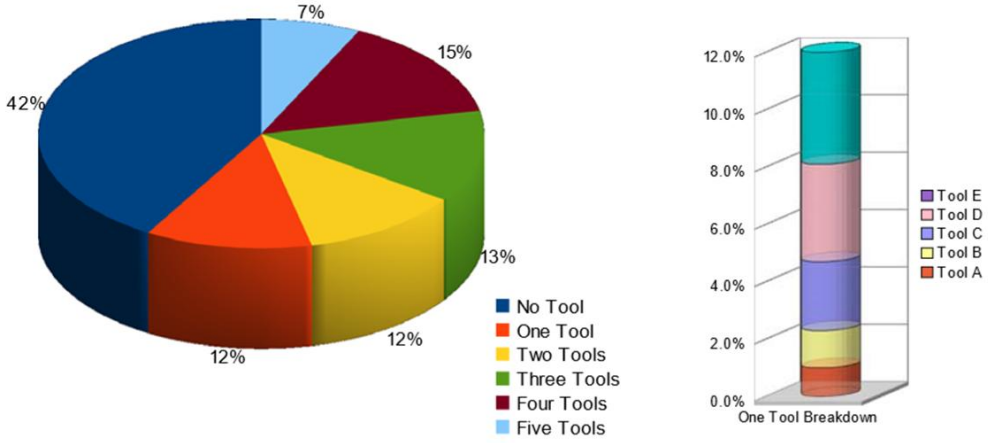
평가를 받고 있는 것 또한 이와 같은 환경적 요인으로 인한 것이라고 볼 수도 있다.

③ 높은 오탐율

실제로는 소스코드의 결함이 없지만 잘못 탐지하여 정적분석 도구가 문제가 있다고 탐지하는 것을 오탐(False Positive)이라고 하며, 정적분석 도구는 오탐율이 높다는 문제점이 있어 탐지 정확도를 높이고 오탐율을 줄이는 연구가 지속되고 있다.

2012년 SCALe (Source Code Analysis Laboratory)에서 발표한 보고에 따르면, 210개의 테스트 케이스 중에서 40%가 넘는 케이스가 C/C++ 소스코드를 분석하는 도구 5개에서 모두에서 진단을 받지 못했고 오직 7.2%만이 5가지 도구 모두에서 성공적으로 진단된다고 하였다. 마찬가지로 177개의 테스트 케이스 중 39.7%가 Java 소스코드를 분석하는 6개의 도구에서 진단되지 않은 것으로 보고되었다[9].

이러한 이유로 정적분석 도구의 높은 오탐율로 인한 문제를 고려하지 않을 수 없는 상황이다. 우선적으로 정적분석 도구의 오탐율이 높을 경우 소스코드 진단 비용이 증가하는 문제를 피할 수 없다. 점검자가 오탐된 부분을 Eye-Check 하거나 다른 검증도구를 사용하여 재진단해야 되는 추가 작업이 요구되어지기 때문이다. 또한 문제가 없는 소스코드를 오탐하여 소프트웨어를 변경하게 되고 그로 인해 발생하는 유지보수 비용도 문제점으로 볼 수 있다.



(그림 5) C and C++ “Breadth” Case Coverage [Landwehr 2008] [9]

2. 기계학습 (Machine Learning)

1) 기계학습의 정의

기계학습은 명시적으로 프로그래밍 하지 않아도 컴퓨터가 대량의 샘플 데이터에서 일정한 규칙을 찾아내고 찾아낸 규칙을 기반으로 다른 데이터를 분류하거나 미래를 예측해내는 기술을 말한다. 사람의 경험과 유사한 역할을 하는 기계학습의 샘플 데이터는 빅데이터 시대에 힘입어 대량으로 수집, 저장, 관리, 분석되어지면서 기계학습의 성능을 높이는 결과를 가져왔다. 이처럼 기계학습이 대량의 데이터를 통해 스스로 학습을 한다는 점은 전통적인 프로그래밍 기법과 비교했을 때 다음과 같은 이유에서 강점을 가진다.

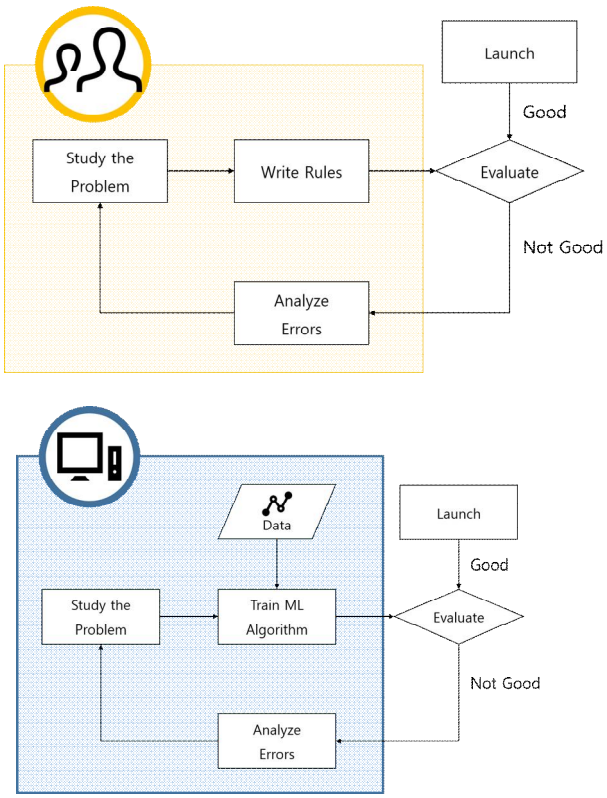
기존의 프로그래밍 기법으로 소프트웨어를 만들 경우 특정 규칙을 탐지하는 기능을 만들기 위해선 규칙이 나타나는 특정 패턴을 수작업으로 찾아야 했으며 이러한 패턴을 내포하고 있는 데이터 필드를 선택하는 작업도 전문가의 개입으로 진행돼야 했다. 이와 같이 사람의 개입이 필요한 작업은 새

로운 규칙이 발생하거나 프로그램의 기능이 충분히 제 역할을 하지 못해 반복하는 것이 일반적이며 반복을 통해 프로그램이 유지보수 된다. 탐지해야 될 규칙의 양이 사소하거나 복잡하지 않을 경우에는 전통적인 프로그래밍 방법으로 이러한 기능을 구현하는 데에 큰 문제가 없지만, 이와 반대로 계속해서 변화하는 환경의 데이터를 탐지해야 되거나, 데이터의 규칙이 복잡할 경우 기존의 방법은 프로그램의 규모를 점차 크고 복잡하게 만들며 이로 인해 유지보수를 어렵게 만든다는 문제가 있다.

이와는 대조적으로 기계학습 기법에 기반을 둔 프로그램은 탐지해야 될 분류 유형과 그렇지 않은 유형의 예제들을 통해서 비정상적으로 자주 발생하는 패턴을 탐지하여 자동으로 학습을 한다. 기존의 프로그래밍 기법에선 전문가의 개입을 통해 수작업으로 이루어지던 분석 작업이 기계학습이 자동으로 분석하기 때문에 프로그램은 훨씬 더 짧아지고 유지보수가 더 쉬워진다는 강점을 가진다. 또한 기계학습은 사람이 예상하지 못한 상관관계나 새로운 추세를 복잡한 데이터 속에서 드러내 문제 영역을 더 잘 이해 할 수 있게끔 사람에게 도움을 주는 역할을 하기도 한다. (그림 6)은 전통적인 프로그래밍 기법과 기계학습에서 프로그램을 구현할 때 거치는 과정을 보여주는 그림으로, 전통적인 방법에선 사람의 직접적 개입을 필요로 하는 단계들이 기계학습에선 컴퓨터가 대신한다는 차이점을 보여준다.

위의 내용을 통해 다시 한 번 정리하자면 기계학습을 기반으로 한 프로그램은 다음과 같은 영역에서 사용이 유용하다는 것을 알 수 있다[16].

① 기존의 솔루션에 많은 수작업 또는 긴 규칙 목록이 필요한 문제
② 전통적인 접근법으론 좋은 해결책이 없는 복잡한 문제
③ 변동이 심한 환경
④ 복잡한 문제를 가지고 있으나 대량의 데이터를 보유하여 통찰력이 확보 될 수 있는 영역



(그림 6) The traditional Approach and Machine Learning Approach

2) 기계학습의 유형

기계학습 시스템에는 매우 다양한 유형이 존재하며 어떤 기준으로 학습 데이터를 분류하고 결과 값이 어떤 형태로 나오는지에 따라 기계학습의 유형을 분류 할 수 있다. 기계학습 유형은 단 하나의 유형만을 선택하여 사용할 수 있는 것이 아니라 목적에 따라 여러 유형을 조합하거나 일부 기능을 수용하여 모델을 구축하는데 사용 할 수 있기 때문엔 분류되는 기준과 특징을 파악하여 필요한 유형을 선택해야 한다. [표 7]은 기계학습의 유형을 정리한 것으로 기계학습을 분류하는 기준과 약식 설명을 포함하고 있다.

[표 7] 기계학습의 유형

분류 기준	기계학습 유형	설명
정답 라벨의 제공 유무	지도학습 (Supervised Learning)	학습데이터와 정답 라벨이 쌍으로 함께 입력데이터로 쓰이는 학습 유형
	비지도 학습 (Unsupervised Learning)	정답 라벨이 없이 학습 데이터만 입력되는 학습 유형
	반지도학습 (Semi-Supervised Learning)	부분적으로 정답 라벨이 있는 학습 데이터를 사용하고 일반적으로 정답 라벨이 없는 학습 데이터를 함께 사용하는 학습 유형
	강화학습 (Reinforcement Learning)	학습 시스템을 관찰하는 Agent를 두어 가장 많은 보상 받기 위해서 가장 좋은 전략이 무엇인지 스스로 알아내는 학습 유형
업로드 되는 학습 데이터의 개수 차이	온라인학습 (Online Learning)	학습 데이터가 되는 샘플이 생길 때마다 모델을 업데이트 하는 학습 유형
	배치학습 (Batch Learning)	모델을 만들 기 전 모든 학습 데이터를 탐색하고 데이터를 한 번에 메모리에 적재하여 학습 시키는 유형
일반화 방법의 차이	인스턴스 기반 학습 (Instance-Based Learning)	데이터가 입력되면 좌표 상 인접해 있는 유사한 데이터 k개를 선택하여 일반화하고 데이터를 분류하는 학습 유형
	모델 기반 학습 (Model-Based Learning) [combining]	만들어져 있는 예측 모델을 통해 입력되는 데이터를 일반화하여 분류하는 학습 유형

기계학습의 유형 중에서도 지도학습과 비지도 학습은 구축한 모델의 결과값의 형태가 뚜렷한 차이를 보이는 유형이다. 대표적으로 지도학습은 분석을 통해 나온 결과 값을 분류(Classfication)하거나 회귀 예측(Regression)하는 데에 쓰이며, 비지도 학습은 유사한 특성을 가진 데이터들을 묶는 군집

화(Clustering)에 쓰인다. 이와 같이 기능에 따라 기계학습 유형은 또 다시 세분화되고 그에 따라 사용할 수 있는 알고리즘의 종류와 개수도 다양하게 있다. [표 8]은 분류, 회귀 예측, 군집화를 비교 정리한 표로 적용 사례와 예측 결과를 통해 3개의 기능적 차이를 확인 할 수 있다.

[표 8] 분류, 회귀 예측, 군집화

	분류 (Classification)	**회귀 예측 (Regression)**	**군집화 (Clustering)**
기계학습 유형	지도학습	지도학습	비지도 학습
예측 결과	어느 범주에 속하는지 예측	특정 어느 하나의 값을 예측	유사한 특성을 가진 데이터 무리
적용 사례	- 스팸메일 분류 - 질병 발생 여부 판별 - 이미지 속 숫자 분류	- 기상예보 기온 예측 - 주식가격 예측 - 매출액 예측 - 제품의 판매량 예측	- 유사한 구매 이력을 가진 고객들을 군집 - 천문학 데이터를 이용해 유사한 특성을 가진 별 탐색 - 전자상거래 사용자가 좋아할 만한 물건을 추천
알고리즘	Logistic Regression SVM Random Forest Decision Tree	Linear Regression SGD Regression Bayesian Regression	k-Means Algorithm Hierarchical Clustering PCA SVD

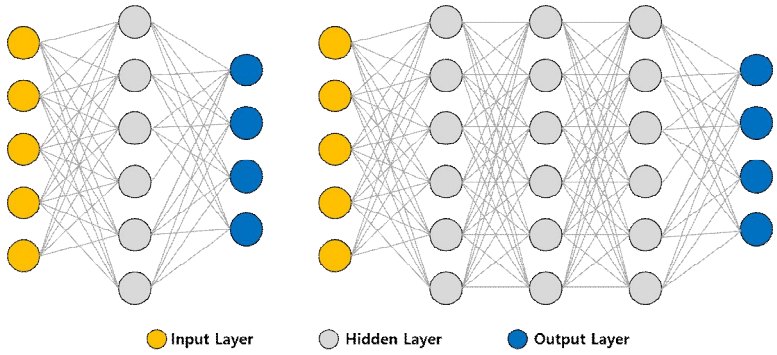
3) 심층신경망 (DNN, Deep Neural Network)

심층신경망은 입력층과 출력층을 제외한 퍼셉트론 노드가 서로 연결되어 있는 은닉층(Hidden Layer)이 2개 이상의 계층으로 쌓여 있는 인공 신경망을 말한다.

은닉층이 1개의 계층으로 구성되어 있는 단일신경망은 입력 데이터와 추론하고자 하는 결과의 관계가 선형적인 경우에 추론 결과를 분류하고 식별

(Classification)의 기능이 뛰어나지만 식별해야 될 결과의 유형의 많아질 경우 선형적 분석이 어려워져 단일신경망으론 정확한 추론 결과를 얻기 힘들다. 또한 입력 데이터가 가지고 있는 특징(Feature)의 수가 많을 경우에도 선형적 분석이 어려워져 여러 개의 특징들 중에서 입력 데이터 중에서 유의미한 패턴을 파악하는데 필요한 몇 개의 특징을 전문가가 수식이나 도구, 방법론을 통해 선별해줘야 한다.

그에 비해 심층신경망은 각각의 은닉층이 이전 계층의 출력 값을 다음 계층의 입력 값으로 전달하는 구조를 가지고 있기 때문에 데이터의 고유한 특징을 추상화하고 학습을 과정에서 자동적으로 특징 추출이 이루어진다. 그로 인해 데이터 간의 비선형적인 관계를 예측해 단일신경망으론 추론할 수 없던 식별 문제를 해결하게 되었다. (그림 7)은 단일신경망과 심층신경망의 구조와 특징의 차이를 보여준다.



(그림 7) 단일 신경망과 심층신경망의 구조 차이

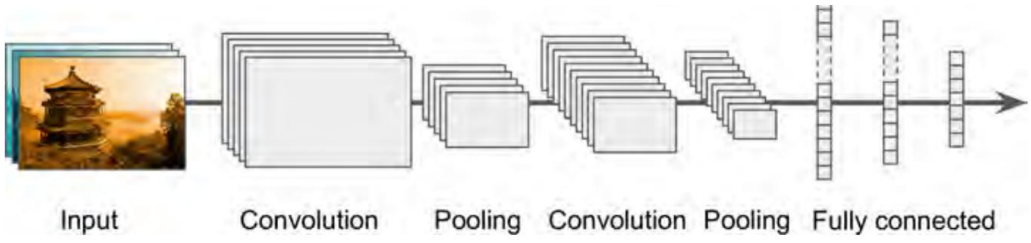
심층신경망은 단순한 기계학습 방법으론 풀기 어려웠던 복잡한 패턴의 학습이 신경망을 통해 가능함을 증명하며 현재 각 데이터의 특성에 맞는 알고리즘 기법과 심층신경망을 결합해 사람의 경험으로 판단하고 추론하여 풀 수 있던 문제영역에서 활용되고 있다.

4) 합성곱 신경망(CNN, Convolutional Neural Network)

합성곱 신경망은 인간의 시신경 구조를 모방하여 만들어진 신경망 모델로 인간이 시각 정보를 처리하는 과정을 흉내 낸 기계학습 기법으로 이미지 인식과 같은 시각적 분석이 필요한 분야에 적합한 신경망 모델이다.

일반적인 심층신경망의 경우 크기가 작은 이미지에서는 문제없이 동작하지만, 이미지의 크기가 커질 경우 분석에 필요한 파라미터의 양이 방대해지면서 작동이 멈춘다는 문제점이 있으며, 이미지 속 개체의 위치, 크기, 각도 등에 변화가 일어나도 성능이 떨어지게 된다. 그 결과로 심층신경망을 이용한 이미지 개체 인식의 Accuracy는 74% 정도에 그치고 마는 한계가 있다.

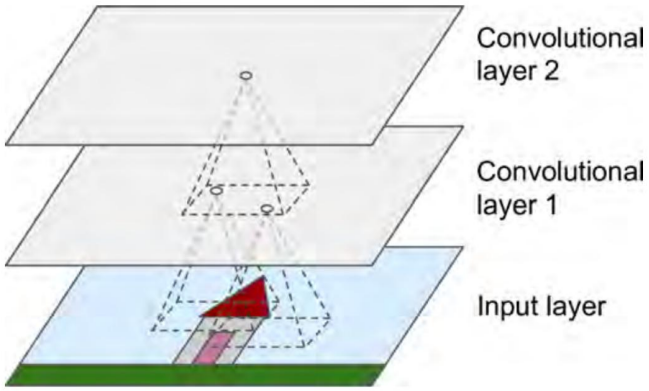
이러한 문제를 해결하기 위해 등장한 것이 바로 합성곱 신경망 모델이다. 시각 피질의 뉴런은 국소(Local)의 제한된 영역에 위치한 시각 자극에만 반응하며, 이러한 국소의 자극들이 하위 레벨에서 모여 상위 레벨에서 조합되고 복잡한 패턴을 감지하는 메커니즘을 가진다[16]. 합성곱 신경망은 이런 시신경 뉴런의 동작을 모방한 것으로 하위 계층에서 이미지의 국소 영역들을 데이터로 수집하고 상위 계층에서 국소의 이미지 데이터들을 조합하여 복잡한 패턴 정보를 감지하는 합성곱(convolutional) 계층과 풀링(pooling) 계층을 삽입하여 일반 심층신경망이 가졌던 이미지 인식의 문제를 해결하였다.



(그림 8) Typical CNN Architecture [16]

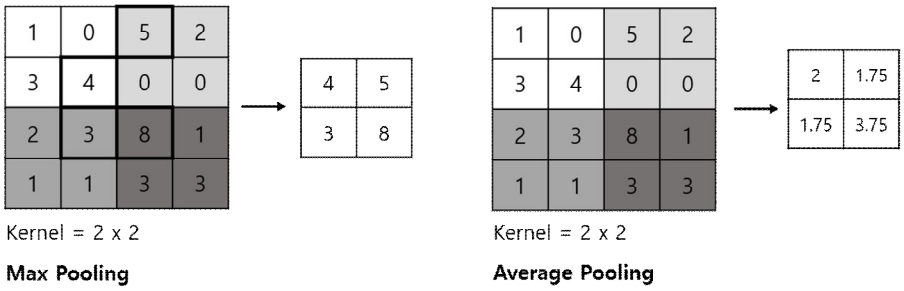
합성곱 신경망의 구조에는 심층신경망에서 보았던 Fully Connected 계층과 시그모이드(sigmoid) 활성화 기능과 같은 이미 알고 있는 구성요소도 있지만, 합성곱 계층과 풀링 계층이라는 새로운 구성 요소가 포함되며, 시각적 정보를 분석하는데 최적화된 결과를 가져오게 되었다.

○ 합성곱 계층 : 이미지 데이터에서 국소 영역들의 특징을 추출하는 계층이다. 커널이라고 불리는 원본 픽셀 값보다 작은 크기의 창(Window)이 이미지 데이터 위를 슬라이딩하며, 이미지 픽셀 값과 커널의 가중치 값을 합성곱하여 특징을 추출하게 된다.



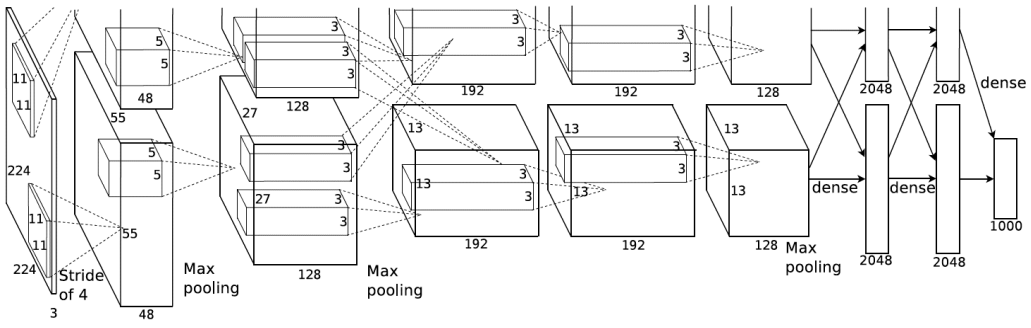
(그림 9) CNN layers with rectangular local receptive fields [16]

○ 풀링 계층 : 계산 부하, 메모리 사용량 및 매개 변수 수를 줄이기 위해 입력된 이미지 데이터를 특정 연산으로 축소시켜 과다 입력으로 인한 위험을 줄이는 계층이다. 이미지 커널의 크기와 이동 보폭이 정해지면, 커널은 일정한 간격(Stride)으로 이미지 위를 움직이며 커널 공간 안에 있는 수치 데이터의 최대/평균값을 선택하여 추출하게 된다. 최댓값을 선택하는 것은 최대 풀링(Max Pooling)이라고 하며, 평균값의 선택은 평균 풀링(Average Pooling)이라고 한다.



(그림 10) Max Pooling, Average Pooling

합성곱 신경망을 이용한 이미지 속 개체 인식의 성능은 2012년에 열린 ILSVC대회 (Imagenet Large Scale Visual Recognition Challenge, Imagenet)에서 발표된 AlexNet 신경망 모델에 의해 다시 한 번 입증되었다. AlexNet은 합성곱 계층과 풀링 계층을 2개 이상의 계층으로 쌓아올린 Deep-CNN으로 26%대 머물러있던 오류율을 15%대로 줄여 ILSVC 대회에서 우승하게 된다[17]. 그 이후 AlexNet 연구 사례를 확장한 GoogLeNet, ResNet과 같은 신경망 모델들이 등장했으며 5% 미만의 오류율을 기록하며, 현재 영상 인식 (Video Receognition), 객체 추적(Object Tracking), 자율 주행 자동차(Self-Driving Car) 플랫폼과 같은 이미지/영상 분야에 합성곱 신경망이 활용되고 있다.



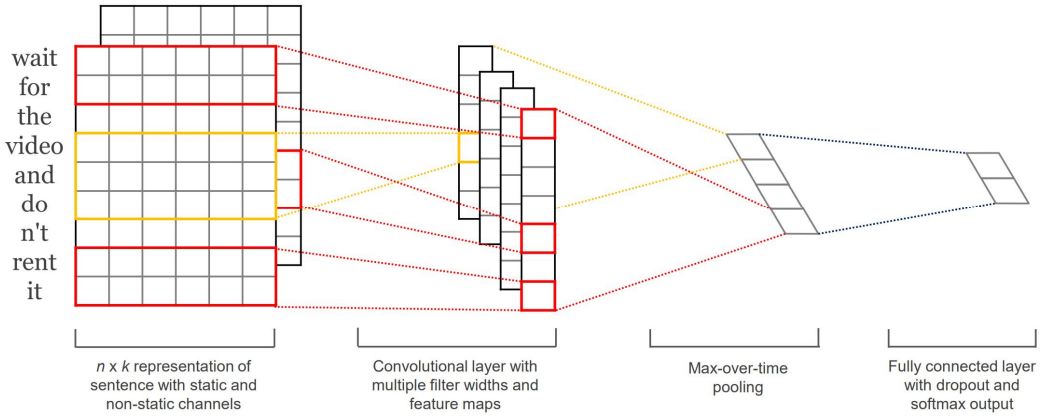
(그림 11) AlexNet Architecture [17]

5) 기계학습을 이용한 취약점 탐지 연구 사례

합성곱 신경망은 시각적 인식 분야에만 국한되어 사용되는 것은 아니다. 음성 인식이나 자연어 처리(NLP)와 같은 분야에도 적용되어 텍스트 데이터 속에 숨겨진 정보를 찾는 사례 연구들이 발표되고 성과를 거두고 있다.

2014년 Yoon Kim에 의해 발표된 논문 “Convolutional Neural Networks for Sentence Classification”[18]에서는 합성곱 신경망을 이용한 Sentence Classification Model을 발표했다. 기존의 합성곱 신경망에선 이미지 픽셀 한 개가 하나의 벡터가 되었다면, CNN Sentence Classification 모델에선 문장을 이루고 있는 단어가 하나의 벡터로 표현된다. 그리고 단어를 벡터로 표현하기 위해 학습 데이터에 대한 사전 훈련으로 word2vec의 적용 유무와 초기화 방식의 차이를 두어 비교 실험하였다.

이와 같은 합성곱 신경망을 이용한 자연어처리 연구 사례가 생겨남에 따라 소스코드 덩어리를 하나의 문자열인 텍스트 정보로 받아들여 기계학습 기법을 적용시키고 원하는 결과를 얻고자 하는 연구 사례들이 생겨났다. 소프트웨어 공학적인 시각에서 소스코드는 분석을 통해 소프트웨어의 품질 향상, 취약점 탐지, 소프트웨어 결함 예측과 같은 목적을 달성 할 수 있기 때문에 2000년대에 들어 Bayesian Belief Networks (BBN)와 같은 전통적인 기계학습 기법의 적용이 제안되어 왔으며[21], 최근 들어 딥러닝의 성능이 향상됨에 따라 소스코드를 텍스트 형태로 벡터화하고 딥러닝 기법으로 분석하는 연구들이 늘어나고 있다. 본 논문과 같이 기계학습 기법을 적용하여 소스코드의 취약점을 탐지하고자 하는 연구 사례도 마찬가지로 생겨나고 있는 추세이다. 따라서 이번 장에서는 최근 발표된 논문 ‘Automated software vulnerability detection with machine learning’[22]을 통해 기계학습을 통한 취약점 탐지 사례를 알아보고 결과를 중심으로 내용을 정리하고자 한다.



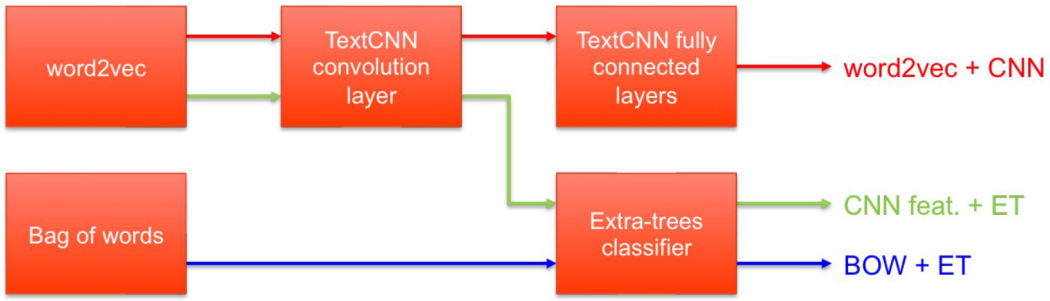
(그림 12) Model architecture with two channels for an example sentence [18]

Harer & Kim의 논문은 1) Build-based feature extraction, Source-based feature extraction 2가지 Method를 중심으로 소스코드의 특징를 추출하고 분석하는 방향을 설정하며, 2) 랜덤 포레스트와 같은 전통적인 기계학습 기법과 딥러닝 기법의 성능을 비교하여 결과를 제시하고 있다. 2가지 측면의 실험 설정을 통해 취약점 탐지에 적합한 소스코드 특징 추출 방법과 최적의 성능을 내는 기계학습 기법을 추측할 수 있는 방향을 제시하고, 최종적으로 소스코드에 취약점이 포함되어 있는지 없는지의 여부를 식별하는 것을 목표로 하고 있다.

소스코드의 특징을 추출하는 2가지 방법 중 Build-based feature extraction은 Clang과 LLVM을 이용하여 소스코드를 CFG, opcode vector, use-def matrix와 같은 IR 모델로 변환하고 벡터화 시키는 방법으로, operation의 실행 여부와 흐름, 변수의 사용 여부와 같은 프로그래밍 언어의 구조 정보를 포함하고 있는 데이터 변환 방식이다. 이와는 대조적으로 Source-based feature extraction은 소스코드를 기존의 자연어 처리 기법인 Bag of words, word2vec을 적용하여 분석하는 방식으로 문자열의 형태로

소스코드를 변환하는 방식이기 때문에 Build-based feature extraction에 비해 프로그래밍 언어의 구조나 의미와 같은 내재되어 있는 정보를 파악하긴 어렵다는 차이점을 가진다.

기계학습 기법은 총 3개의 모델을 구성하여 실험 결과를 비교하고 있다. ①Extra-trees classifier로 식별하는 모델, ②합성곱 신경망으로 식별하는 모델, ③ 합성곱 신경망 + Extra-trees classifier로 식별하는 모델로 나뉘어 실험이 진행되면 ①번 모델이 전통적인 기계학습 기법을 적용한 모델이라면 ②번은 딥러닝의 대표 모델, ③번은 이 2개의 모델을 합친 복합 모델을 대표한다.



(그림 13) Illustration of the three types of source-based models tested [22]

위와 같은 데이터 변환 과정과 기계학습 모델의 적용을 통해 해당 논문은 합성곱 신경망과 Extra-Trees Classifier를 결합한 복합 모델이 ROC 0.82를 기록하며 최적의 성능을 보임을 보였으며, Source-Based Feature Extraction 기반으로 합성곱 신경망과 같은 딥러닝 모델을 적용하는 경우가 전통적인 기계학습의 기법보다 나은 성능을 보이고 취약점이 있고 없음의 탐지 또한 가능함을 보였다.

Ⅲ. 실험 설정

1. 실험 개요

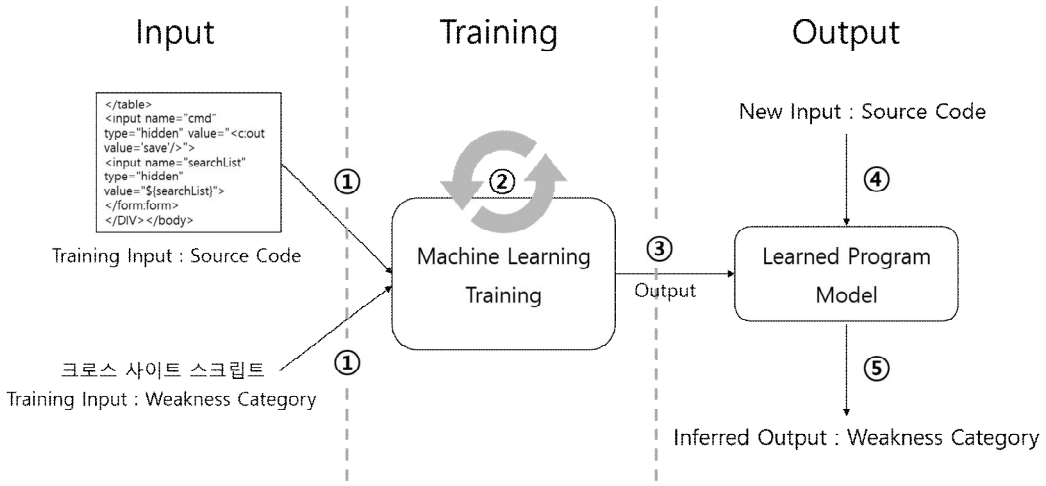
앞 장의 관련 연구를 통해 심층신경망과 합성곱 신경망이 복잡한 데이터 간의 관계를 파악하고 자연어 처리 기법과 결합되어 텍스트 정보를 판별하는 사례들을 보았다. 연구 사례에서 텍스트 데이터를 토큰으로 만들어 벡터로 표현했던 것과 같이 보안 취약점을 가진 복잡한 형태의 소스코드도 하나의 문장으로 표현하고 위와 같은 동일한 과정을 거쳐 수치 데이터로의 변환하여 실험이 가능할 것으로 보인다.

이번 3장에서는 프로그램을 구성하고 있는 전체 소스코드를 정적 분석기에 넣어 얻게 된 결과 리포트를 입력 데이터로 활용하여 실험을 진행한다. 정적분석기는 보안 취약점이 있다고 판단되는 소스코드 라인의 앞뒤 3줄과 그에 해당하는 보안 취약점 유형열 결과로 내놓기 때문에 보고된 취약한 소스코드를 하나의 문장, 문자열 스트림으로 입력할 수 있고, 보안 취약점 유형 결과를 정답 라벨로 활용할 수 있어 지도 학습 모델로 훈련시킬 수 있다. 그러므로 훈련을 위한 입력 데이터는 정적분석기를 통해 얻어진 결과인 소스코드와 정답 영역인 보안 취약점 유형의 쌍이 될 것이다.

본 실험은 입력 데이터로부터 어떤 결과를 유추해내는 모델을 만드는 것이 목적이므로 기계학습의 한 방법인 지도학습(Supervised Learning)이 이번 실험에 적합하다. 그러므로 훈련을 위한 입력데이터는 문제 영역인 소스코드와 정답 영역인 보안 취약점 유형의 쌍이 될 것이다. (그림 14)는 이번 실험의 입력-훈련-결과의 과정을 도식화 한 것으로 본 과정을 통해 만들어진 모델에 새로운 입력 데이터를 넣어 보안 취약점 유형을 추론하는 모습을

보여준다.

실험에 쓰일 프로그래밍 언어는 Google Brain Team에서 제공하는 오픈소스 라이브러리인 Tensorflow이며, Python 2.7 버전을 사용하여 모델을 구현하고 실험을 진행한다.



(그림 14) 실험 모델의 입력-훈련-결과 과정

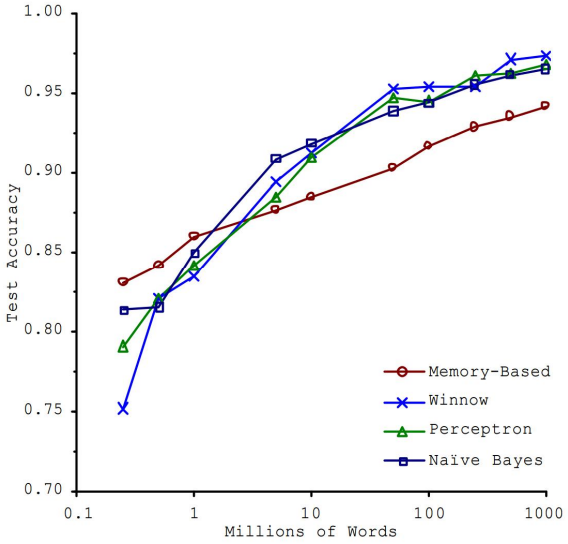
2. 실험 데이터 설정

1) 실험 데이터 개수 설정

Microsoft사의 연구원 Michele Banko, Eric Brill은 2001년 발표한 논문을 통해 자연어의 의미적인 모호함을 탐지하는 것과 같은 복잡한 문제는 훈련에 필요한 데이터의 양이 늘어날수록 학습의 성과가 높아지는 결과를 보이며, 알고리즘을 개발하는데 드는 비용과 말뭉치(Corpus) 데이터를 개발하는데 드는 비용은 Trade-Off[1] 관계임을 암시하는 말을 담았다[19]. 즉, 풀기 어려운 복잡한 패턴을 가진 문제영역일수록 훈련을 할 수 있는 데이터의 양

1) 서로 대립되는 요소 사이의 균형

이 충분해야하며, 그로 인해 알고리즘의 성능도 높일 수 있음을 뜻한다.



(그림 15) Learning Curves for Confusion Set Disambiguation [19]

기계학습을 이용한 실험도 마찬가지이다. 학습 데이터의 양이 충분하지 않으면 모델의 성능도 충분한 결과를 내지 못 할 수 있다. 손글씨로 쓰여진 숫자 0부터 9를 구별하는 MNIST 실험의 경우 훈련, 검증, 테스트를 위한 데이터 셋을 총 70,000개 제공하고 있다[20]. 훈련 데이터는 55,000개로 1개의 숫자 당 5,500개의 이미지 파일이 훈련을 위해 제공되고 있는 것이다. 검증 데이터는 5,000개이며, 테스트 데이터는 10,000개이다. 이를 토대로 본 실험도 그에 준하는 데이터양을 가진 보안 취약점 유형의 소스코드를 사용할 것이다.

이번 실험을 위해 한국인터넷진흥원으로부터 제공된 데이터의 개수는 총 68,863개이며 보안 취약점 유형은 37개이다. 본 실험은 내재되어 있는 보안 취약점의 유형을 탐지하고 분류하는 것이 목표인 만큼 주어진 68.863개의 데이터 중에서 정탐으로 판별된 소스코드 데이터만을 사용할 것이다. 이 중

에서 훈련에 쓰일 수 있을 만큼의 충분한 개수를 가진 취약점 유형은 크로스 사이트 스크립트(15,671개), 널 포인터 역참조(10,771개), 부적절한 예외처리(7,422개), 시스템 데이터 정보 노출(4,383개)로 총 4개이다. [표 9]는 보안 취약점 유형별 데이터 개수를 정리한 것으로 상단에 표기된 4개의 항목이 본 논문의 실험에 훈련 데이터로 쓰일 것이다.

[표 9] 보안 취약점 유형별 데이터 개수

취약점 유형	정탑 개수	오탑 개수	합계
01.03. 크로스사이트 스크립트	15,671	12,492	28,163
05.01. 널(Null) 포인터 역참조	10,771	2,179	12,950
04.03. 부적절한 예외처리	7,422	13	7,435
06.03. 시스템 데이터 정보 노출	4,383	941	5,324
01.01. SQL 삽입	4,356	460	4,816
04.02. 오류 상황 대응 부재	1,622	5	1,627
05.02. 부적절한 자원 해제	834	575	1,409
06.01. 잘못된 세션에 의한 데이터 정보 노출	820	388	1,208
01.02. 경로 조작 및 자원 삽입	740	1,014	1,754
02.04. 취약한 암호화 알고리즘 사용	455	28	483
02.09. 적절하지 않은 난수 값 사용	235	265	500
01.12. 정수형 오버플로우	217	309	526
06.02. 제거되지 않고 남은 디버그 코드	212	13	225
01.11. HTTP 응답분할	165	197	362
06.04. Public 메소드부터 반환된 Private 배열	133	10	143
01.06. 신뢰되지 않는 URL 주소로 자동 접속 연결	123	237	360
06.05. Private 배열에 Public 데이터 할당	123	3	126
02.16. 반복된 인증시도 제한 기능 부재	107	124	231
02.07. 하드코드된 비밀번호	106	24	130
03.01. 경쟁조건: 검사시점과 사용시점(TOCTOU)	93	165	258

01.04. 운영체제 명령어 삽입	50	156	206
03.02. 종료되지 않는 반복문 또는 재귀 함수	45	31	76
02.12. 사용자 하드디스크에 저장되는 쿠키를 통한 정보 노출	41	17	58
02.14. 솔트 없이 일방향 해쉬 함수 사용	28	6	34
01.13. 보안기능 결정에 사용되는 부적절한 입력값	26	8	34
01.05. 위험한 형식 파일 업로드	16	4	20
02.03. 중요한 자원에 대한 잘못된 권한 설정	7	0	7
02.06. 중요정보 평문전송	6	79	85
02.11. 취약한 비밀번호 허용	5	9	14
02.08. 충분하지 않은 키 길이 사용	4	0	4
02.13. 주석문 안에 포함된 시스템 주요정보	4	19	23
02.10. 하드코드된 암호화 키	2	9	11
07.01. DNS lookup에 의존한 보안 결정	1	205	206
01.15 포맷스트링 삽입	0	10	10
02.15. 무결성 검사없는 코드 다운로드	0	40	40
05.03. 해제된 자원 사용	0	4	4
XML External Entity Injection	0	1	1

2) 과적합과 검증데이터 설정

기계학습 모델이 훈련 데이터에 과도하게 적응해서 새로운 데이터를 제대로 예측하지 못하는 문제를 과적합(Overfitting)이라고 하며, 은닉층이 2개 이상으로 이루어진 심층신경망에서는 과적합이 일어나기 쉬워 주의해야 한다. 일반적으로 과적합을 방지하기 위해서 다음과 같은 솔루션을 적용한다.

① 정규화(Regularization) : 모델을 단순하게 만들어 훈련 데이터의 특징 개수 줄이기

② 더 많은 학습 데이터 수집

③ 노이즈 데이터 제거 : 데이터 오류 수정 및 특이치 제거

위와 같은 솔루션을 도입하였더라도 과적합이 100% 일어나지 않을 거라 보장 할 수 없다. 따라서 모델이 과적합 되지 않았는지 확인하기 위해 훈련에 사용되지 않은 새로운 데이터를 입력하여 성능을 모니터링을 해줄 필요가 있다. 이러한 과적합을 평가하는 데에 쓰이는 데이터를 검증데이터라고 하며, 훈련 과정 도중에 만들어진 모델에 검증데이터를 넣었을 때 성능이 월등히 떨어지면 과적합이 일어난 것으로 판단한다.

본 논문은 심층신경망을 이용한 보안 취약점 식별 실험을 포함하고 있으므로 모델의 성능 평가를 위해 훈련데이터(Training), 검증데이터(Validation), 테스트데이터(Testing)를 분리하여 사용할 것이다. 각 보안 취약점 유형별 데이터의 10%는 검증 데이터로 사용되며, 완성된 모델의 성능을 평가하는 용도로 테스트 데이터가 각 보안 취약점 유형별로 100개가 사용될 것이다. 최종적으로 보안 취약점 유형별로 사용될 데이터의 개수는 아래의 [표 10]과 같다.

[표 10] 보안 취약점 유형별 실험 데이터셋

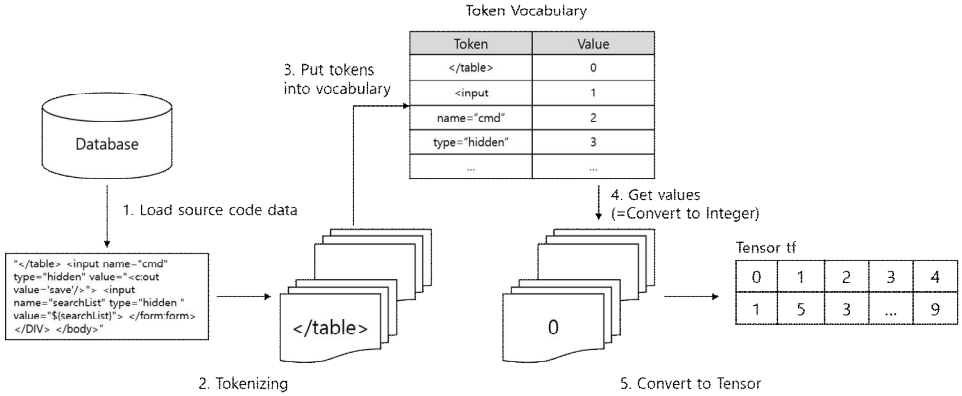
취약점 유형	Training	Validation	Testing	Total
크로스 사이트 스크립트	14,014	1,557	100	15,671
널 포인터 역참조	9,604	1,067	100	10,771
부적절한 예외처리	6,590	732	100	7,422
시스템 데이터 정보 노출	3,855	428	100	4,383

3) 데이터 변환 과정

문자열 스트림으로 입력될 소스코드는 이번 예측 모델 구현에 쓰이는 Tensorflow API의 기본 단위인 텐서(Tensor)로 변환되어야 한다. 텐서는 다차원 데이터 배열의 형태를 가지며 데이터 플로우 그래프를 사용하여 수치 연산을 진행하는 Tensorflow에서 간선으로 나타낸다. 수치 연산을 나타

내는 노드 간의 사이를 연결하는 것이 텐서의 역할로 텐서는 정수, 실수와 같은 수치 데이터로 표현되어야 한다.

따라서 본 실험에서 텐서에 해당하는 소스코드 문자열 스트림을 숫자로 변환해주어야 하며, 이를 위해 (그림 16)과 같은 단계로 나누어 소스코드 스트림을 텐서로 변환하게 된다.



(그림 16) 소스코드 텐서 변환 과정

① 소스코드가 저장되어 있는 DB로부터 데이터를 가져온다.

② 공백을 기준으로 소스코드를 분리하여 여러 개의 토큰으로 만들어준다.

③ 토큰을 Vocabulary에 저장한다. 토큰에 고유 ID를 부여하는 작업이다.

④ 토큰을 Key로 사용하여 Vocabulary로부터 ID 값을 가져온다.

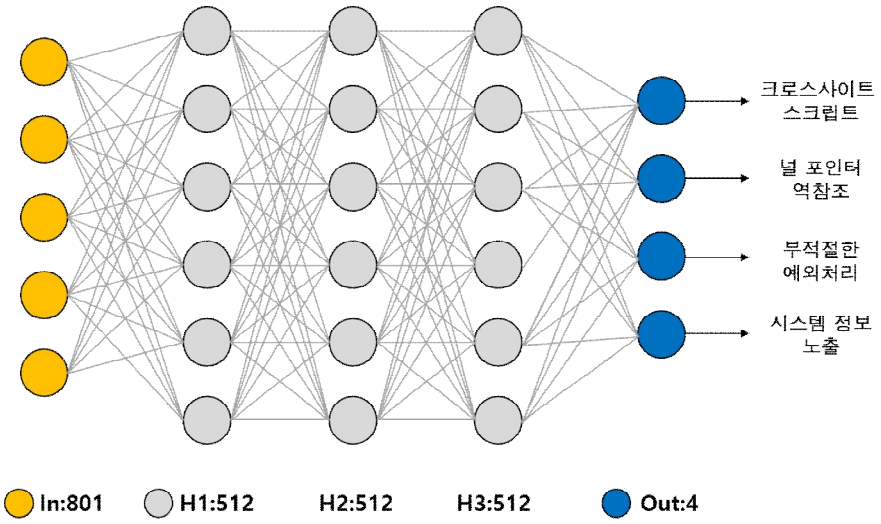
⑤ ID를 텐서로 변환한다.

3. 심층신경망을 이용한 보안 취약점 식별 실험 설정

1) 심층신경망 실험 구조

심층신경망을 이용한 보안 취약점 시별 실험에서의 신경망은 총 3개의 은

닉층을 가지며 각 은닉층은 512개의 노드를 가진다. 입력층은 가장 많은 토큰으로 이어진 소스코드의 개수를 기준으로 노드 개수가 결정된다. 따라서 이번 실험에서 가장 긴 토큰 개수인 801개가 입력층의 노드 개수가 된다. 출력층 노드는 식별해야 되는 보안 취약점 유형을 나타내므로 4개로 구성되며, 각 노드는 크로스 사이트 스크립트, 널 포인트 역참조, 부적절한 예외처리, 시스템 정보 노출을 대표한다. (그림 17)은 이번 심층신경망 실험의 구조를 보여주기 위해 도식화한 모형으로 입력-은닉-출력의 계층과 노드들을 보여준다.



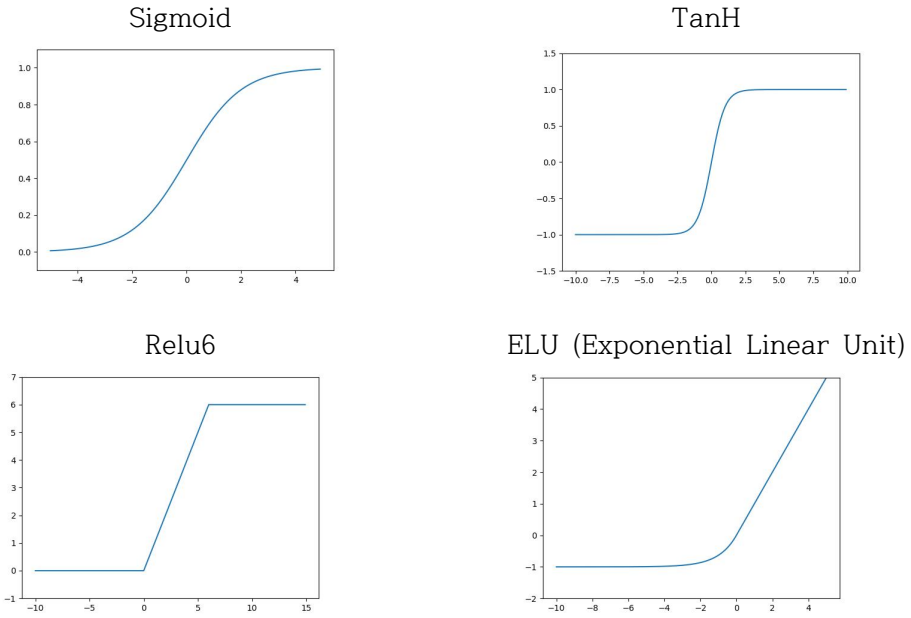
(그림 17) 심층신경망 실험 구조

2) 활성화 함성(Activation Function)

(그림 17)에서 볼 수 있듯이 노드는 자신이 속해있는 계층의 전후 계층들의 노드들과 모두 연결되어 있다. 이때 각 노드들은 자신과 연결되어있는 모든 노드들과 통신을 하지는 않는다. 이전 계층의 노드 값이 특정 분계점에 도달했을 때에만 다음 계층의 노드를 활성화 시킬 것을 지시한다. 즉, 입력 값이 어떤 분계점에 도달해야만 출력이 발생하는 것이다. 이처럼 특정

분계점을 넘어서는 경우에 출력 신호를 생성하는 함수를 활성화 함수(Activation Function)라고 한다.

심층신경망 구조에서는 어떤 활성화 함수를 사용하느냐에 따라 오류율과 정답률이 달라진다는 변수가 있다. 따라서 이번 실험에서는 (그림 18)에서와 같은 대표적인 비선형 활성화 함수인 Sigmoid, Tanh와 선형 활성화 함수인 Relu6, ELU를 적용하여 결과를 비교할 예정이다.



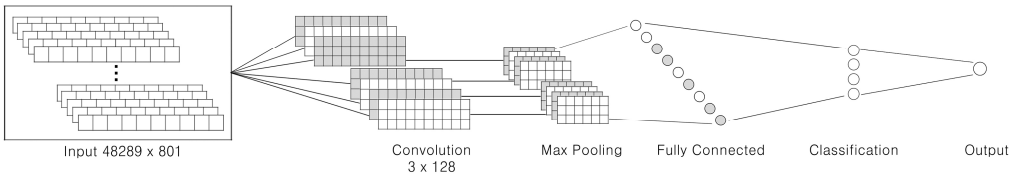
(그림 18) 활성화 함수

4. 합성곱 신경망을 이용한 보안 취약점 식별 실험 설정

1) 합성곱 신경망 실험 구조

힙성곱 신경망을 이용한 보안 취약점 식별 실험은 앞선 심층신경망과 다른 점은 합성곱 계층과 풀링 계층을 거친다는 것이다. 이 2단계의 과정은

입력으로 받은 데이터가 어떤 개체에 유사한지 수치 데이터를 점수화하고 계산하기 적절한 크기의 데이터로 압축, 축소를 시켜준다. 압축된 입력 데이터는 수치 연산이 가능한 신경망의 노드와 간선으로 변환이 되는 Fully Connected 과정을 거치게 된다. (그림 19)는 이미지 개체 인식을 위한 합성곱 신경망의 구조와 본 논문에서 진행할 실험할 텍스트를 훈련데이터로 입력받는 합성곱 신경망 실험의 구조를 보여준다.



(그림 19) 소스코드 취약점 유형 분류를 위한 힙성곱 신경망 구조

2) 정규화(Regularization)

앞서 심층신경망 실험의 검증 데이터 설정에서 과적합과 이를 해결하기 위한 방법으로 정규화를 언급했었다. 정규화는 데이터를 정답 곡선 안에 포함시키기 위해 과하게 굴곡졌던 신경망 모델을 곱게 펴주는 방법으로 데이터의 특징 개수를 줄여 모델을 단순하게 만들어주는 기능이다.

정규화를 하는 방법에는 L1 정규화와 L2 정규화가 있으며, 두 개의 방식은 가중치에 제약을 거는 수식에서 차이가 있다. L1 정규화는 절대 값의 합을 적용하여 상수와 같은 영향이 미미한 작은 가중치들은 0으로 수렴이 되게 만들고 중요한 몇 개의 가중치들만 영향을 미치게 만드는 함수라면, L2 정규화는 모든 파라미터에 제곱만큼의 크기로 가중치에 제약을 걸어 몇 개의 입력 데이터에 강하게 가중치를 적용하기 보단 모든 입력 데이터에 영향력을 약하게 적용되게 만드는 함수이다.

(수식 1)은 L1 정규화를 적용하는 수식을 나타내며, (수식 2)는 L2 정규화

의 수식을 나타낸다.

$$(\text{수식 } 1) \quad S = \sum_{i=0}^{n} \left| y_i - h(x_i) \right|$$

$$(\text{수식 } 2) \quad S = \sum_{i=0}^{n} (y_i - h(x_i))^2$$

일반적인 합성곱 신경망 모델들은 심층신경망에 비해 입력받는 이미지 데이터의 픽셀 수에서부터 차이가 나서 고려해야 될 데이터의 특징이 많아져 정규화를 적용하여 과적합을 방지하는 경우가 많다. 본 논문의 합성곱 신경망 실험은 합성곱 신경망은 동일한 변환 과정을 거쳐 입력 데이터의 개수에서 차이가 나지는 않는다. 하지만 소스코드를 텍스트 형태로 입력 받았다는 점에서 신경망 모델이 복잡해질 가능성이 높다. 따라서 L1 정규화와 L2 정규화를 적용하여 과적합을 방지함과 동시에 정규화 함수가 실험 결과에 영향을 미치는지 확인하고자 한다.

Ⅳ. 실험 결과

1. 심층신경망을 이용한 보안 취약점 식별 결과

심층신경망을 이용한 보안 취약점 식별 결과는 아래의 [표 11]과 같다. Test Accuracy는 훈련과 검증을 거쳐 만들어진 기계학습 모델에 훈련에 사용되지 않은 새로운 소스코드를 보안 취약점 유형별로 100개를 입력했을 경우 답을 맞힐 확률을 나타낸다. 이번 심층신경망 실험은 활성화 함수 Tanh를 사용했을 경우 훈련데이터에 대한 Accuracy 0.79, 검증 데이터에 대한 Accuracy 0.74, 테스트 데이터에 대한 Accuracy가 0.83으로 나타나며 4개의 활성화 함수에 대한 모델 중 가장 높은 Test Accuracy 성능을 보인다.

[표 11] 심층신경망 실험 결과

Activation Function	Accuracy / Loss	Training Average	Validation Average	Test Accuracy
Sigmoid	Accuracy	0.66	0.65	0.62
	Loss	0.83	0.83	
Tanh	Accuracy	**0.79**	0.74	**0.83**
	Loss	0.53	0.66	
Relu6	Accuracy	0.77	0.73	0.79
	Loss	0.85	0.68	
ELU	Accuracy	0.77	**0.75**	0.80
	Loss	7.15	5.76	

부록에 삽입된 (그림 21)은 훈련데이터의 Accuracy와 과적합을 판단하기 위해 설정된 검증데이터의 Accuracy를 함께 나타낸 그래프이다. 데이터의

양을 나타내는 x축이 오른쪽으로 갈수록 y축이 일정한 높이에서 수렴하고 있으므로 실험은 적절한 횟수로 훈련이 진행되고 종료되었음을 볼 수 있다. 또한 훈련데이터 Accuracy에 비해 검증데이터 Accuracy가 낮게 나타나고 있지만 [표 11]을 통해 수치를 비교하면 최대 차이는 5% 안팎이므로 과적합이 일어났다고 보기 어렵다.

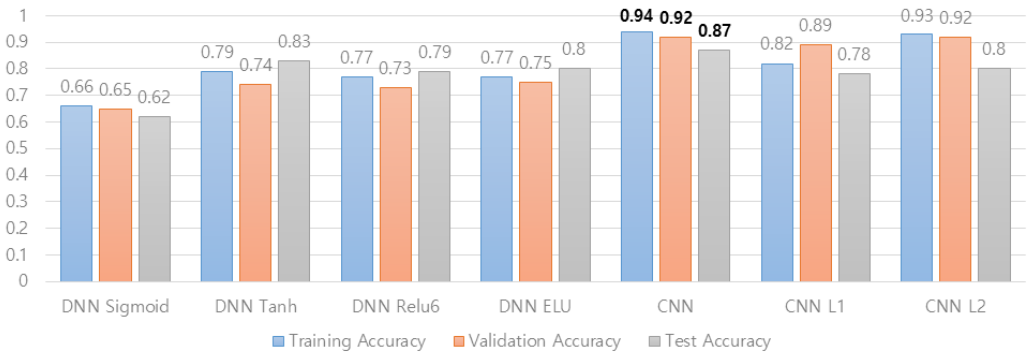
2. 합성곱 신경망을 이용한 보안 취약점 식별 결과

합성곱 신경망을 이용한 보안 취약점 식별 결과는 아래 [표 12]와 같다. 이번 실험은 정규화 함수를 적용하지 않았을 경우, 훈련데이터에 대한 Accuracy 0.94, 검증데이터에 대한 Accuracy 0.92, 테스트 데이터에 대한 Accuracy는 0.87로 합성곱 신경망 모델 중 가장 높게 나타났다. 또한 [표 12]를 통해 훈련데이터와 검증데이터의 정확도를 비교 했을 때, Accuracy의 차이가 3% 미만으로 과적합이 일어나지 않았음을 확인 할 수 있다.

부록에 삽입된 (그림 22)는 합성곱 신경망 모델별로 훈련데이터의 Accuracy와 과적합을 판단하기 위해 설정된 그래프이다. 심층신경망과 마찬가지로 합성곱 신경망의 결과 그래프도 y축의 값이 일정한 높이에서 수렴하고 있으며, L1 정규화 모델의 경우 훈련데이터와 검증데이터의 Accuracy가 7% 정도의 차이를 보이나 [표 12]를 통해 알 수 있듯이 검증 데이터의 Accuracy가 훈련 데이터 보다 높은 것으로 과적합이 일어났다고 볼 수 없다.

[표 12] 합성곱 신경망 실험 결과

		Training Average	Validation Average	Test Accuracy
Default	Accuracy	**0.94**	**0.92**	**0.87**
	Loss	0.2303	0.2372	
L1 Function	Accuracy	0.82	0.89	0.78
	Loss	0.0548	0.0543	
L2 Function	Accuracy	0.93	0.92	0.80
	Loss	1.41E-05	1.229E-05	



(그림 20) 심층신경망, 합성곱 신경망 모델별 Accuracy

Ⅴ. 결론 및 향후 연구

본 연구에서는 소프트웨어 보안취약점을 탐지하고 분류하는 과정에서 반복되어 나타나는 취약점 패턴을 대상으로 기계학습을 시킴으로써 소소코드에 한 특성이 올바르게 학습될 수 있는지를 실험하였고, 이 과정에서의 학습정확성을 분석하였다. 본 연구를 통해 얻어진 주요 내용을 정리하면 다음과 같다.

- 소스코드의 취약점 탐지를 위해 심층신경망과 합성곱 신경망을 이용한다.
- 소스코드는 공백을 기준으로 분리하고 고유한 정수형 ID를 부여하여 변환한다.
- 심층신경망은 활성화 함수 4가지를 변수로 두어 비교한다.
- 합성곱 신경망은 정규화 함수 적용 여부를 변수로 두어 비교한다.
- 최종 결과, 합성곱 신경망을 이용했을 경우 최고 Accuracy 87.54%를 기록했다.

Harer & Kim의 연구에서 소개한 사례와 비교할 경우, 사례 연구는 ROC 수치를 통해 취약점이 포함되어 있는지 없는지 여부를 확인하고 이를 통해 오탐율 축소가 가능한지 확인하는 것을 목적으로 한다면, 본 논문의 실험은 한국인터넷진흥원이 분류한 47개의 취약점 중 4가지 취약점 유형을 분류하는 결과를 냄으로써 정적분석 도구를 통한 취약점 탐지의 사전 단계에 기여할 수 있는지 가능성을 확인하는 데에 목표를 가진다는 점에서 차이를 두고 있다. 하지만 사례 연구와 본 논문의 결과를 확인했을 때, 두 연구 실험 모두 딥러닝을 이용한 취약점 탐지가 가능함을 보였으며, 딥러닝 기법 중에서

도 이미지 인식 처리에 적합하다고 알려져 있던 합성곱 신경망이 가장 좋은 결과를 보인다는 공통된 결과를 보여 합성곱 신경망에 기반한 학습이 정적 분석에 대한 탐지율 향상에 기여할 수 있다는 긍정적인 기법이라 판단한다.

취약점 유형이 무엇인지에 따라 특정 IR 모델로 소스코드를 변형시켜서 분석해야하는 기존의 정적분석 도구와 달리 본 실험에서는 소스코드를 토큰화하여 정수형 ID를 부여해주는 변환 과정만 거쳤다는 점에서 데이터 변형이 과정이 간소화 될 수 있음을 확인했다. 이는 향후 기계학습이 정적분석을 지원하는 도구로 사용될 경우 프로그램의 복잡도를 낮출 가능성도 기대해볼 수 있는 부분이다.

시큐어코딩은 소스코드의 복잡한 구조로 인한 정적분석이 어렵고, 빠르게 변화하는 소프트에어 개발 환경만큼 보안 취약점을 이용한 공격이 고도화되고 있는 환경적인 문제로 인해 소스코드 진단도구의 정확한 진단이 어렵다는 한계와 함께 오탐율이 증가하는 문제점을 안고 있다. 실제로 정탐된 소스코드와 오탐된 소스코드의 형태는 매우 유사하여 단순한 구조의 신경망 모델로는 충분한 결과를 내기 어려울 것으로 보인다. 따라서 데이터의 앞뒤 문맥을 중심으로 데이터의 상관관계를 파악하여 문제를 해석하는 데에 뛰어난 RNN(Recurrent Neural Network)과 Agent를 두어 가장 많은 보상을 받기 위해서 가장 좋은 전략이 무엇인지 스스로 알아내는 강화 학습과 같은 새로운 유형의 신경망 모델을 적용하여 정교한 정오탐에 대한 연구를 진행해볼 필요가 있다.

마지막으로 합성곱 신경망을 이용한 본 실험은 1개의 합성곱 계층을 갖는 구조로 실험이 진행되었다. 합성곱 신경망도 심층신경망과 마찬가지로 합성곱 계층이 깊을수록 부정확한 학습 문제를 해결 할 수 있다. 따라서 2개 이상의 합성곱 계층을 가지는 Deep-CNN 구조의 모델로 실험을 발전시켜 성능이 향상되는지 확인해볼 필요가 있다.

위의 제시된 방향을 통해 90% 이상의 Test Accuracy를 도출하는 것을 목표로 하는 실험과 정적분석을 위한 기계학습 모델이 상용화되기 위해 필요한 보완점과 활용 방안에 대해 지속적으로 연구해 나갈 예정이다.

참 고 문 헌

[1] Livshits, V. B., & Lam, M. S. (2005). Finding Security Vulnerabilities in Java Applications with Static Analysis. In USENIX Security Symposium (Vol. 14, pp. 18-18).

[2] Huang, Y. W., Yu, F., Hang, C., Tsai, C. H., Lee, D. T., & Kuo, S. Y. (2004). Securing web application code by static analysis and runtime protection. In Proceedings of the 13th International Conference on World Wide Web (pp. 40-52).

[3] Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, RTI Project, 7007(011).

[4] Howard, M., & Lipner, S. (2006). The security development lifecycle (Vol. 8). Redmond: Microsoft Press.

[5] 행정자치부 (2017). 전자정부 SW 개발·운영자를 위한 소프트웨어 개발 보안 가이드.

[6] 박정현, 박영식, 정효택. (2017). SW 개발 R&D 프로젝트에서 소스 코드 품질을 위한 정적분석.

[7] 김보미. (2015). 시큐어코딩을 위한 PMD 룰 셋 확장(석사학위). 성신여자대학교 일반대학원.

[8] Software Assurance Metrics And Tool Evaluation. (n.d.). Source Code Security Analyzers. National Institute of Standards and Technology [Online]. Retrieved from https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html.

[9] Seacord, R. C., Dormann, W., McCurley, J., Miller, P., Stoddard, R.,

Svoboda, D., & Welch, J. (2012). Source code analysis laboratory (scale) (No. CMU/SEI-2012-TN-013). CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

[10] Christey, S., & Martin, R. A. (2007). Vulnerability type distributions in CVE.

[11] 손영주. (2017). 실시간 보안 규칙 적용을 위한 AOP 기반 시큐어 코딩 (석사학위). 숭실대학교 정보과학대학원

[12] 배종민. (2017). 공개 소프트웨어 점검도구를 활용한 소프트웨어 보안약점 진단규칙 개선. 건국대학교 정보통신대학원

[13] De Win, B., Scandariato, R., Buyens, K., Grégoire, J., & Joosen, W. (2009). On the secure software development process: CLASP, SDL and Touchpoints compared. Information and Software Technology, 51(7), 1152-1171.

[14] Williams, J., & Wichers, D. (2017). The Ten Most Critical Web Application Security Risks. OWASP Foundation.

[15] Chess, B., & West, J. (2007). Secure programming with static analysis (pp. 72-83). Pearson Education.

[16] Aurélien, G. (2017). Hands-On Machine Learning with Scikit-Learn & TensorFlow. O'Reilly Media, Inc, Sebastopol (CA).

[17] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (pp. 1097-1105).

[18] Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

[19] Banko, M., & Brill, E. (2001). Scaling to very very large corpora for

natural language disambiguation. In Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (pp. 26-33). Association for Computational Linguistics.

[20] LeCun, Y., Cortes, C., & Burges, C. J. (2010). MNIST handwritten digit database. AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, 2.

[21] Zhang, D. (2000). Applying machine learning algorithms in software development. In Proceedings of the 2000 Monterey Workshop on Modeling Software System Structures in a Fastly Moving Scenario (pp. 275-291).

[22] Harer, J. A., Kim, L. Y., Russell, R. L., Ozdemir, O., Kosta, L. R., Rangamani, A., ... & McConley, M. W. (2018). Automated software vulnerability detection with machine learning. arXiv preprint arXiv:1803.04497.

ABSTRACT

A Study on Detection and Classification of Security Vulnerabilities Based on Machine Learning

Lee Won-Kyung
Department of Computer Science
Graduate School of
Sungshin University

Security incidents caused by defetcs inherent in software are on the rise. Thus, it is generally understood that preventing software security incidents by removing software defects in advance rather than dealing with them after the incident is effective in reducing the damage. Since 2012, as part of such efforts Korean Government has developed the guidelines for secure software development.

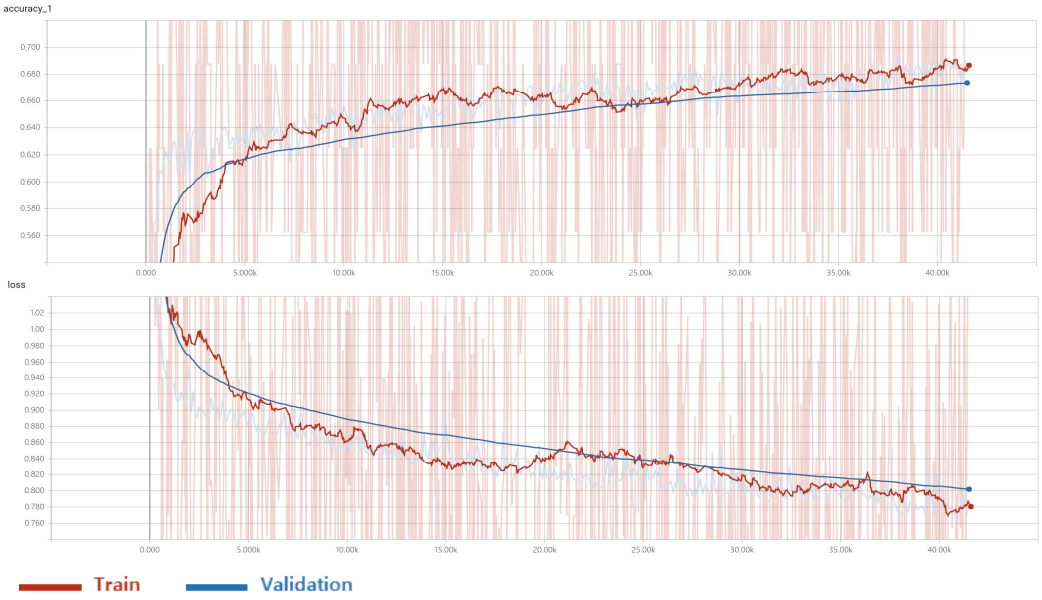
The core of the guide is secure coding, a security activity at the implementation stage of the program. Applyin security activities associated with secure coding are identifying the characteristics and patterns of source code with inherent security vulnerabilities, and recommend the use of automated static analyzers for accurate analysis of source code.

In reality, however, source can be written in a variety of forms depending on the coding method and the developer's style, creating and

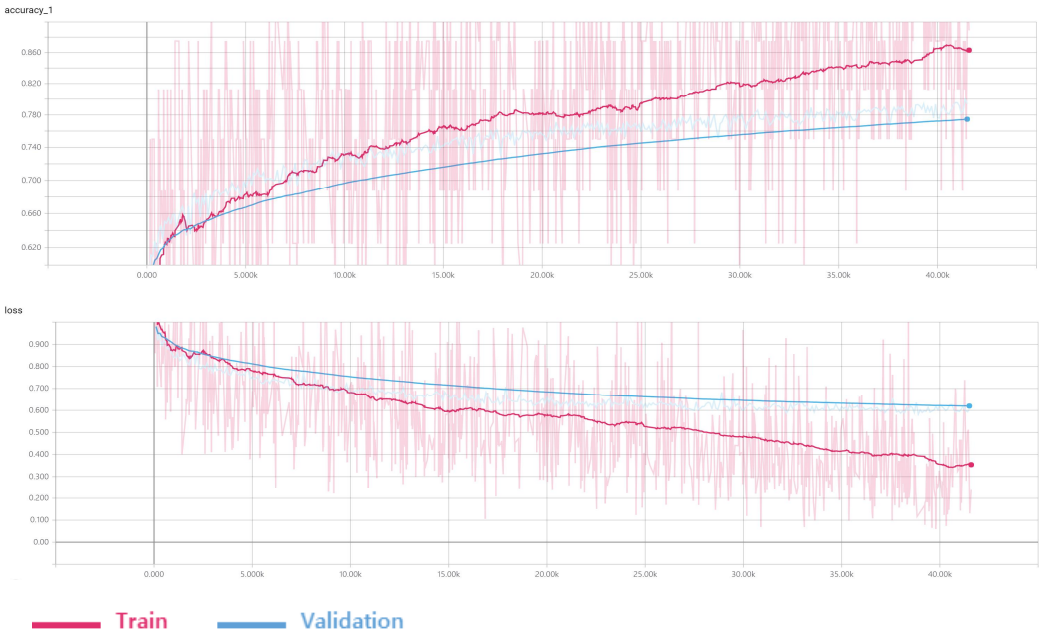
applying ruleset to detect patterns that are considered vulnerable to security still requires expert involvement. In addition, There are many variables to consider for source code analysis and the view of analyzing source code must vary depending on which vulnerabilities to defects. Thus there are problems of increasing complexity of static analysis tools and making accurate diagnosis difficult.

This paper proposes steps of applying of machine learning algorithms as a way to mitigate these problems. Machine learning techniques have been successfully adopted in the area of natural language processing, and recognition of objects. With recent research findings, we identify and detect security vulnerabilities for the vulnerable source code using machine learning. We also evaluate the possibility of enhancing static analysis performance using machine learning based on experimental results as a future study.

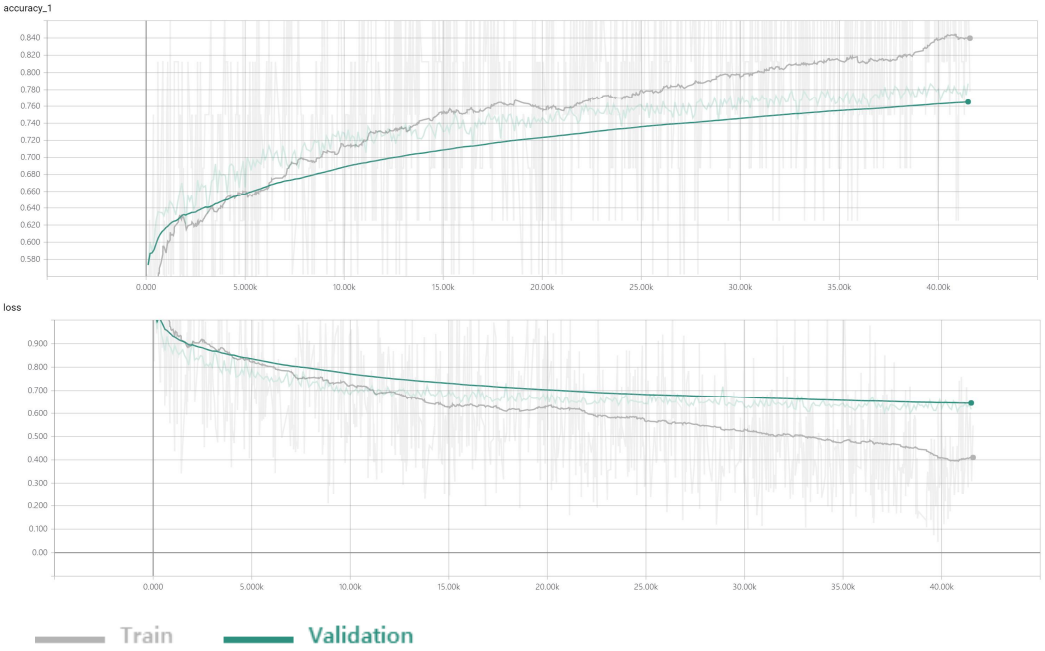
[부록]



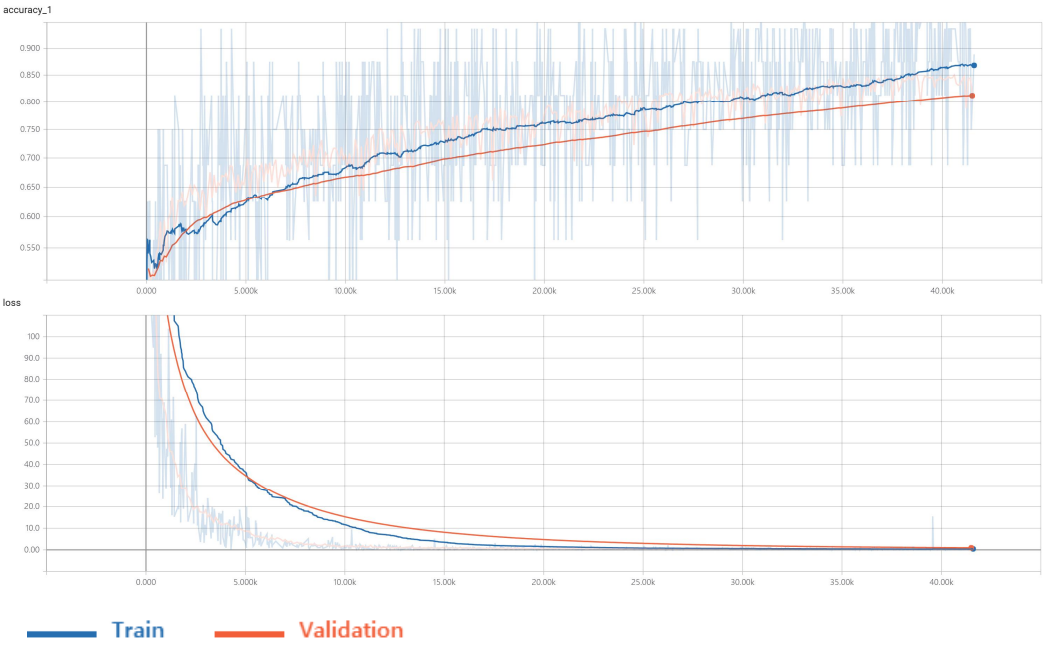
(그림 21-a) Sigmoid 심층신경망 모델 결과 그래프



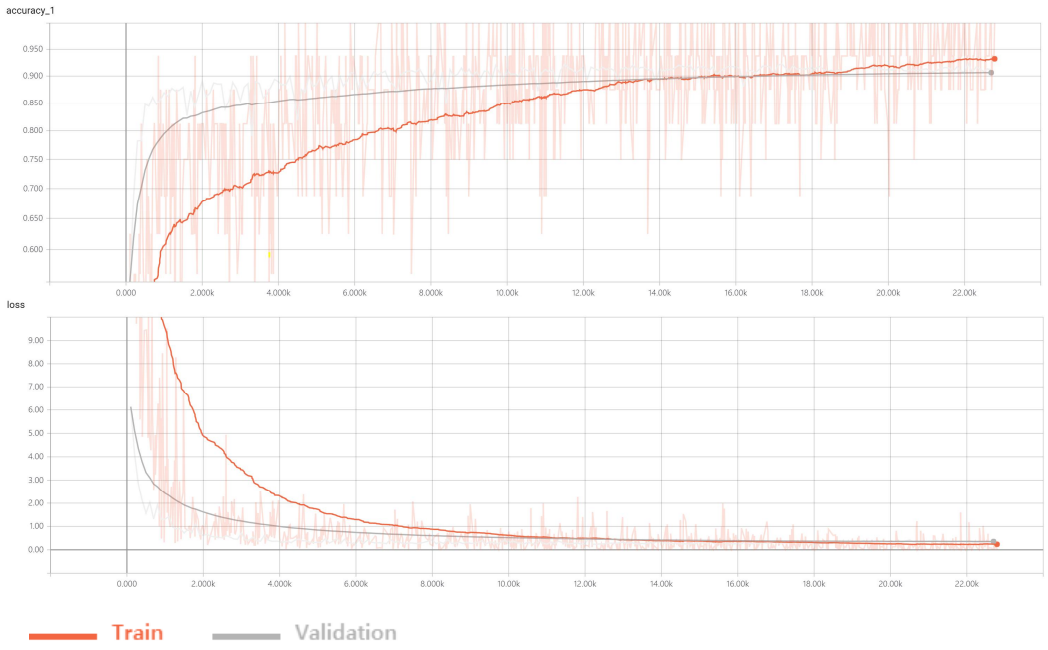
(그림 21-b) Tanh 심층신경망 모델 결과 그래프



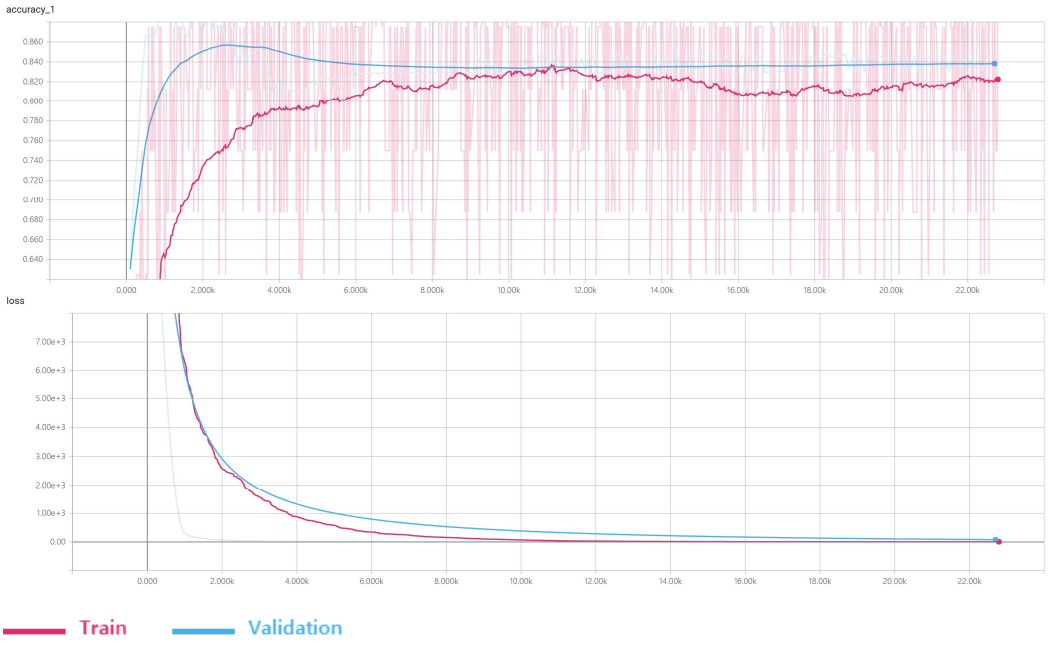
(그림 21-c) Relu6 심층신경망 모델 결과 그래프



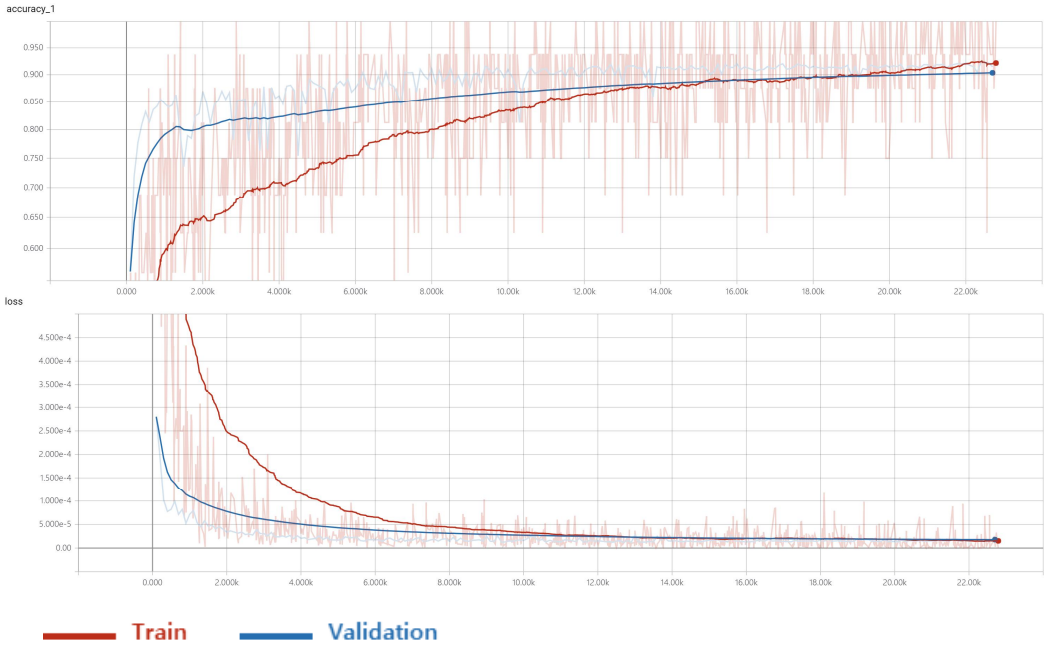
(그림 21-d) ELU 심층신경망 모델 결과 그래프



(그림 22-a) 기본 합성곱 신경망 모델 결과 그래프



(그림 22-b) L1 함수를 적용한 힙성곱 신경망 모델 결과 그래프



(그림 22-c) L2 함수를 적용한 힙성곱 신경망 모델 결과 그래프