Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.3, Jun. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.3.557

신경망을 이용한 소프트웨어 취약 여부 예측 시스템*

최 민 준,[1†] 구 동 영,[2] 윤 주 범[1‡]
[1]세종대학교, [2]한성대학교

Software Vulnerability Prediction System Using Neural Network*

Minjun Choi,[1†] Dongyoung Koo,[2] Joobeom Yun[1‡]
[1]Sejong University, [2]Hansung University

요 약

소프트웨어의 증가에 따라 소프트웨어의 취약점도 함께 증가하고 있다. 다양한 소프트웨어는 다수의 취약점이 존재할 수 있으며 취약점을 통해 많은 피해를 받을 수 있기 때문에 빠르게 탐지하여 제거해야 한다. 현재 소프트웨어의 취약점을 발견하기 위해 다양한 연구가 진행되고 있지만, 수행 속도가 느리거나 예측 정확도가 높지 않다. 따라서 본 논문에서는 신경망 알고리즘을 이용하여 소프트웨어의 취약 여부를 효율적으로 예측하는 방법을 제안하며 나아가 기계학습 알고리즘을 이용한 기존의 시스템과 예측 정확도를 비교한다. 실험 결과 본 논문에서 제안하는 예측 시스템이 가장 높은 예측률을 보였다.

ABSTRACT

As the number and type of software increases, those security vulnerabilities are also increasing. Various types of software may have multiple vulnerabilities and those vulnerabilities as they can cause irrecoverable significant damage must be detected and deleted quickly. Various studies have been carried out to detect the vulnerability of the current software, but it is slow, and prediction accuracy is low. Therefore, in this paper, we describe a method to efficiently predict software vulnerability by using neural network algorithm and compare prediction accuracy with conventional system using machine learning algorithm. As a result of the experiment, the prediction system proposed in this paper showed the highest prediction rate.

Keywords: Artificial Intelligence, Neural Network, Machine Learning, Fuzzing, Software Vulnerability

I. 서 론

2017년 소프트웨어 산업 실태조사[1]에 따르면 업무처리를 위한 소프트웨어뿐만 아니라, 4차 산업혁명 시대를 맞아 인공지능, 빅데이터 IoT, 클라우드 등을 위한 다양한 분야를 위한 많은 소프트웨어가 새롭게 개발되고 있다. 하지만 소프트웨어의 증가에 따라 필연적으로 소프트웨어 취약점도 증가하고 있다. 보안솔루션업체인 Avecto 社의 조사에 의하면 최근 5년간 Microsoft 社 소프트웨어의 취약점이 111% 증가하였다고 밝혔다[2].

이러한 소프트웨어 취약점을 통해 해커들이 시스템을 공격할 경우 개인이나 기업은 막대한 피해를 받을 수 있기 때문에 소프트웨어 취약점은 사전에 탐지하여 제거해야 한다. 현재 소프트웨어 취약점을 검사하는 도구는 대표적으로 퍼징(fuzzing)[3] 기법과 기호실행(symbolic execution)[4]을 이용한 도구들이 있다. 하지만 기존의 도구들은 복잡하고 시간이

Received(02. 22. 2019), Modified(2019. 04. 15), Accepted(04. 16. 2019)
* 본 연구는 한국연구재단 연구과제 (NRF-2018R1D1A1B07047323) 지원으로 수행하였습니다.
† 주저자, choiminjun7077@gmail.com
‡ 교신저자, jbyun@sejong.ac.kr(Corresponding author)

오래 걸리는 치명적인 단점이 있다. 위 단점을 극복하기 위해 기계학습과 신경망 알고리즘을 이용하여 취약 여부를 예측하는 연구들[5][6][7][8][9][10][11]이 있지만, 예측정확도가 높지 않거나 소스코드가 필요한 한계점 등이 존재한다. 따라서 본 논문에서는 기존 연구의 한계점을 개선한, 소프트웨어 취약 여부 예측 시스템을 제안한다.

2장에서는 기존 연구인 기계학습 및 신경망을 이용하여 취약 여부를 예측하는 방법에 대해 설명하며 3장에서는 본 논문에서 제안하는 예측 시스템, NVFinder(Neural Network-Vulnerability Finder)의 취약 여부 예측 방법을 설명한다. 4장에서는 NVFinder의 데이터 분류 알고리즘에 대해 알아보고 5장에서는 NVFinder와 기존 시스템의 성능평가를 진행한다. 끝으로 결론을 제시한다.

II. 관련 연구

의료, 기후 등 각 분야에 활발히 적용되고 있는 신경망 기술[12][13]은 인간의 두뇌를 모방한 방식으로, 구글 딥마인드의 알파고[14]와 이세돌 바둑 경기 이후 우리 사회에 자리매김한 인공지능(Artificial Intelligence, AI)을 말한다[15]. 보안 분야에서도 신경망을 이용한 연구들이 지속되고 있다[16][17]. 하지만 기계학습이나 신경망을 이용하여 소프트웨어의 취약 여부를 예측하는 최근 3년 이내 연구는 많지 않다.

현재 소프트웨어 취약점을 검사하는 도구는 대표적으로 기호실행과 퍼징 기법을 이용한 소프트웨어 검사 도구가 있다. 하지만 퍼징 기법을 이용한 소프트웨어 검사 도구는 소프트웨어를 검사할 때마다 검사하고자 하는 소프트웨어의 입력 값을 무작위로 변경하고 검사하기 때문에 분석 시간이 오래 걸리며 경우에 따라 소프트웨어 구조를 파악하여야 한다는 단점이 있다.

또한 기호실행을 이용한 소프트웨어 검사 도구는 소프트웨어를 검사할 때마다 검사하고자 하는 소프트웨어 분기문의 미지수 값을 기호로 나타내어 모든 분기를 검사하기 때문에 대규모 소프트웨어의 경우 메모리 부족으로 인한 오류가 발생할 수 있고 시간이 오래 걸린다는 단점이 있다. 따라서 소프트웨어의 구조를 분석하지 않고 빠른 시간 내 취약하다고 예측되는 소프트웨어를 파악하기 위해 신경망과 기계학습을 이용한 기존의 취약 여부 예측 방법에 대해 살펴본다.

2.1 신경망을 이용한 취약 여부 예측 방법

신경망을 이용한 취약 여부 예측 방법[9]은 Juliet test suite[18]를 사용해 취약한 어셈블리 코드와 취약하지 않은 어셈블리 코드를 학습하고 새로운 어셈블리 코드를 입력 값으로 주었을 때, 신경망을 통해 취약 여부를 예측한다. 실험을 통해 높은 예측률을 보여주었지만, CPU 구조가 다를 경우 어셈블리 코드도 달라질 수 있기 때문에 다시 학습 데이터를 추출해야 한다. 따라서 동일한 구조를 가지는 CPU로만 예측 실험을 해야 정확하다는 단점이 있다.

2.2 기계학습을 이용한 취약 여부 예측 방법

기계학습을 이용한 취약 여부 예측 방법은 크게 소스코드 또는 바이너리를 이용한 예측 방법이 있다.

먼저 소스코드를 이용한 취약 여부 예측[5][6][7][8]은 취약점이 존재하는 취약한 소스코드 데이터베이스와 일반적인 소스코드 데이터베이스를 학습하여 예측 모델을 생성한다. 예측은 새로운 소스코드를 입력 값으로 주었을 때, 예측 모델을 통해 소스코드의 취약 여부를 예측한다. 하지만 소스코드를 대상으로 한 예측 방법은 소스코드 없이 바이너리만 있으면 취약 여부 예측이 불가능하다.

바이너리를 이용한 예측 방법[10][11]은 먼저 퍼저(fuzzer)를 이용해 바이너리 취약 여부를 확인한다. 그리고, 취약한 바이너리 함수 호출 리스트와 취약하지 않은 함수 호출리스트를 분류한 후 학습하고 새로운 함수 호출 값을 입력으로 주었을 때, 예측 모델을 통해 취약 여부를 예측한다. 해당 방법은 소스코드가 없어도 취약 여부를 예측할 수 있는 장점이 있지만, 다른 방법에 비해 예측률이 낮다. 따라서 본 논문에서는 바이너리를 이용한 예측방법에 기계학습이 아닌 신경망을 적용하여 예측률을 높이기 위한 실험을 진행하였다.

III. NVFinder의 학습 및 취약 여부 예측 방법

NVFinder는 바이너리 취약 여부를 예측하기 위해 바이너리 취약 여부 데이터와 바이너리가 동작할 때의 함수를 학습한다.

바이너리가 동작할 때의 함수는 신경망 알고리즘이 새로운 바이너리의 취약 유형 분석을 위한 특징(feature)으로 활용된다. 예측은 예측하고자 하는

바이너리의 특징을 주면 학습한 특징과 비교 후 취약 여부 데이터 값에 따라 새로운 바이너리의 취약 여부를 분류한다.

3.1 학습

학습 데이터를 가지고 있다면 학습 과정을 생략해도 된다. 하지만 본 논문에서는 다양한 도구를 연구하기 위해 직접 ptrace[19]로 바이너리가 동작할 때의 특징을 추출하였으며, zzuf[20] 퍼저로 바이너리 취약 여부 데이터를 추출하여 학습에 이용될 데이터를 생성하였다.

3.1.1 바이너리 특징 데이터 추출

ptrace로 바이너리가 실행될 때의 특징 데이터를 추출하기 위해 먼저, 우분투 기본 바이너리 등 827개 바이너리의 실행 명령어 리스트를 Fig.1.과 같이 75,072개 생성하였다. 그리고 생성된 명령어 리스트를 이용해서 바이너리를 동작시킨 후 ptrace로 바이너리가 실행될 때의 특징 총 75,072개를 추출하였다.

1	/usr/bin/acyclic -n file:/home/Neural-Predictor/seeds/tar-classic.tar
2	/usr/bin/acyclic -n file:/home/Neural-Predictor/seeds/exfat.img
3	/usr/bin/acyclic -n file:/home/Neural-Predictor/seeds/spkac.cnf
4	/usr/bin/acyclic -n file:/home/Neural-Predictor/seeds/abiword.abw
5	/usr/bin/acyclic -n file:/home/Neural-Predictor/seeds/msdos.exe
•	
•	
75068	/usr/bin/shred file:/home/Neural-Predictor/seeds/archives/lha.lzh
75069	/usr/bin/shred file:/home/Neural-Predictor/seeds/archives/rar30.rar
75070	/usr/bin/shred file:/home/Neural-Predictor/seeds/archives/7zip.7z
75071	/usr/bin/shred file:/home/Neural-Predictor/seeds/archives/rar2.rar
75072	/usr/bin/shred file:/home/Neural-Predictor/seeds/archives/gzip.gz

Fig. 1. Binary Command List

3.1.2 바이너리 취약 여부 데이터 추출

취약 여부 데이터를 추출하기 위해 zzuf 퍼저로 바이너리를 검사하였다. 퍼저에 따라 입력값이 다르지만, zzuf 퍼저는 3.1.1 바이너리 특징 데이터 추출 과정에서 생성한 바이너리 실행 명령어 리스트를 입력 값으로 주면 해당 바이너리를 검사할 수 있다. 따라서 바이너리 취약 여부 데이터도 총 75,072개 추출되었다.

각 방법을 통해 추출한 바이너리 특징 데이터와 취약 여부 데이터를 Fig.2.와 같이 1:1 매핑 하였으며, 공백 등 노이즈 값을 제거한 총 7,758개의 정상적인 데이터를 학습하여 예측 모델을 생성 하였다.

Normal : 0
Binary Feature Vulnerability : 1

	A	B	C
1	testcase/_usr_bin_bc:14800:/usr/bin/bc	isatty:0=Num32B0 isatty:0=Num32B8 s	0
2	testcase/_usr_bin_cmp:25035:/usr/bin/cmp	strrchr:0=SPtr32 strrchr:1=Num32B8 se	0
3	testcase/_usr_bin_head:2368:/usr/bin/head	strrchr:0=SPtr32 strrchr:1=Num32B8 se	0
4	testcase/_usr_bin_alsaucm:851:/usr/bin/alsaucm	calloc:0=Num32B8 calloc:1=Num32B8	0
•			
•			
7755	testcase/_usr_bin_gcc-4.8:41:/usr/bin/gcc-4.8	strlen:0=SPtr32 sbrk:0=Num32B0 malc	1
7756	testcase/_usr_bin_gipddecode:8178:/usr/bin/gipddecode	getopt:0=Num32B8 getopt:1=SPtr32 g	0
7757	testcase/_usr_bin_ginstall-info:5987:/usr/bin/ginstall-info	setlocale:0=Num32B8 setlocale:1=GxP	0
7758	testcase/_usr_bin_gencat:3598:/usr/bin/gencat	setlocale:0=Num32B8 setlocale:1=GxP	1

Fig. 2. Learning Data List

3.2 예측

예측은 새로운 특징을 입력 값으로 주었을 때 학습 과정에서 생성한 예측 모델로 바이너리의 취약 여부를 예측한다. 우선 예측 모델의 자연어 처리를 위해 Bag-of-word 알고리즘[21]을 이용하여 데이터를 벡터 상에 위치시킨다. 그리고 새로운 특징을 벡터 상에 위치시켰을 때, 새로운 특징이 취약한 영역, 또는 취약하지 않은 영역에 가까이 위치하는지에 따라 취약 여부를 예측한다. 학습과 예측 과정을 정리하면 Fig.3.과 같다.

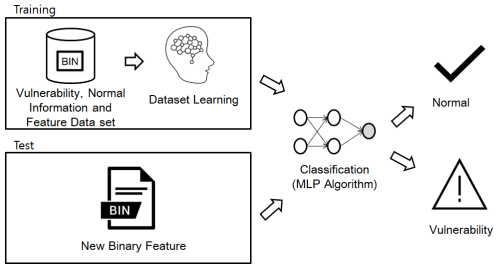


Fig. 3. Prediction Process

IV. 다층 퍼셉트론

본 논문에서 제안하는 NVFinder는 데이터 분류를 위해 신경망 알고리즘인 다층 퍼셉트론을 이용하였다.

신경망 알고리즘인 다층 퍼셉트론은 단층 퍼셉트론으로 해결할 수 없는 XOR문제를 해결하기 위한 퍼셉트론이다[22]. 예를 들어 단층 퍼셉트론의 경우 Fig.4.와 같이 입력층과 출력층밖에 없기 때문에 Fig.5., Fig.6.과 같이 AND, OR문제는 해결할 수 있지만, XOR 문제는 해결할 수 없다. 하지만 다층 퍼셉트론의 경우 Fig.7.과 같이 입력과 출력층 사이에 은닉층이 있어 Fig.8.과 같이 XOR문제를

해결할 수 있다. 따라서 복잡한 데이터 분류 문제를 해결할 때 단층 퍼셉트론보다 다층 퍼셉트론이 적합하다[22].

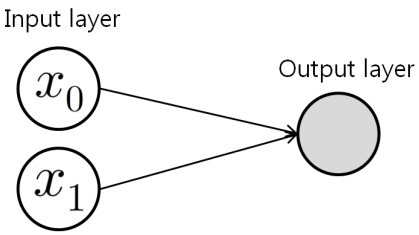
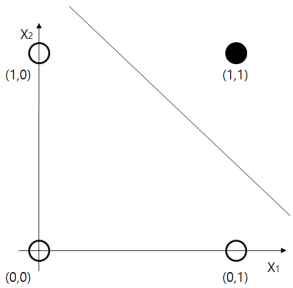
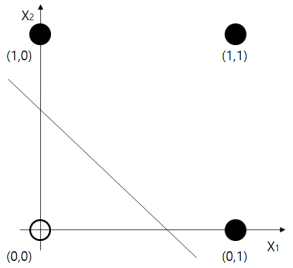


Fig. 4. Single-layer Perceptron Diagram[22]



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 5. AND operation of Single-layer Perceptron[22]



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 6. OR operation of Single-layer Perceptron[22]

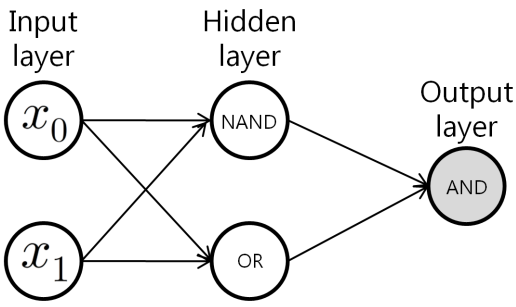
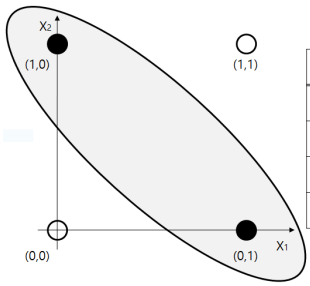


Fig. 7. Multi-layer Perceptron Diagram[22]



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 8. XOR operation of Multi-layer Perceptron[22]

V. NVFinder 성능평가

기계학습 및 신경망 알고리즘은 매개변수에 따라 예측 정확도가 달라질 수 있다. 따라서 성능평가는 NVFinder에 이용된 다층 퍼셉트론의 최적 매개변수 값을 찾기 위해 매개변수 변화에 따른 예측 정확도를 비교하며 나아가 기계학습 알고리즘으로 바이너리 취약 여부를 예측하는 기존 도구 VDiscover [10], VulPredictor[11]와 본 논문에서 제안하는 NVFinder와의 예측 정확도를 비교한다.

5.1 성능 측정 기준

실험은 우분투 14.04 32bit에서 진행하였으며 신경망 분류, 데이터 계층화 등을 효율적으로 구현[23]할 수 있는 오픈소스 도구 scikit-learn[24]을 이용하였다. 또한 과대적합(overfitting)을 줄이기 위해 교차검증[25],[26]을 시행하였다. 교차검증은 학습 데이터 중 무작위 25%를 예측 실험을 위한 데이터로 이용하였고, 75%를 학습모델 생성을 위한 데이터로 이용하였다. 4.2 NVFinder 예측 정확도 비교의 실험 결과는 위 과정을 10번 반복하여 평균값을 측정한 결과이고 1번 반복 시간은 약 20분 소요되었다. 그리고 알고리즘 성능을 평가하기 위해 오차 행렬(confusion matrix)[27]로 Precision, Recall, F-measure, Accuracy를 계산하였다. 평가항목인 Precision은 1이라고 예측한 것 중 실제 1이라고 예측한 것을 비율(TP/(TP+FP))로 나타낸 것이며 Recall은 실제 값이 1인 것 중 1이라고 예측한 것을 비율(TP/ (TP+FN))로 나타낸 것이다. 나머지 F-measure는 Precision과 Recall의 조화평균(2 * ((Recall * Precision)/(Recall+Precision)))을 나타낸 것이

Table 1. Comparison of prediction accuracy according to alpha parameter change

No.	Parameter	Precision	Recall	F-measure	Accuracy
1	alpha = 0.1	90%	84.5%	83.5%	83.9%
2	alpha = 0.01	79.8%	82.7%	80%	81.2%
3	alpha = 0.001	91.5%	86.8%	85.8%	86.8%
4	alpha = 0.0001	91.6%	87.2%	86.9%	87.9%
5	alpha = 0.00001	84.5%	84.4%	82.5%	86.1%
6	**alpha = 0.000001**	**92.5%**	**86.9%**	**85.5%**	**88.1%**
7	alpha = 0.0000001	89.2%	83.4%	82.8%	86.5%

고 Accuracy는 전체에서 올바르게 예측한 것이 몇 개인지를 비율((TP+TN)/(TP+TN+FP+FN))로 나타낸 것이다.

5.2 매개변수 변화에 따른 예측 정확도 비교

본 논문에서 이용된 다층 퍼셉트론은 데이터를 학습하기 위해 solver매개변수를 이용하는데, 그 중 본 실험에서는 안정적인 학습 및 데이터 최적화가 가능한 lbfgs[28]를 이용하였다. 또한, 데이터 결정 경계를 조절하여 과대적합을 줄일 수 있는 alpha 매개변수를 변경하여 예측 실험을 진행하였고[28], 그 외 나머지 매개변수는 scikit-learn에서 제공하는 기본 값을 이용하였다.

실험 결과 다층 퍼셉트론 알고리즘은 Table 1., Fig.9.와 같이 alpha의 값을 0.000001로 하였을 때 가장 높은 예측률을 보였다. alpha 값을 세밀하게 조절하면 유연한 결정 경계를 가질 수 있기 때문에 평균적으로 alpha 값이 낮을 때 높은 예측 결과를 보인 것이라고 판단된다.

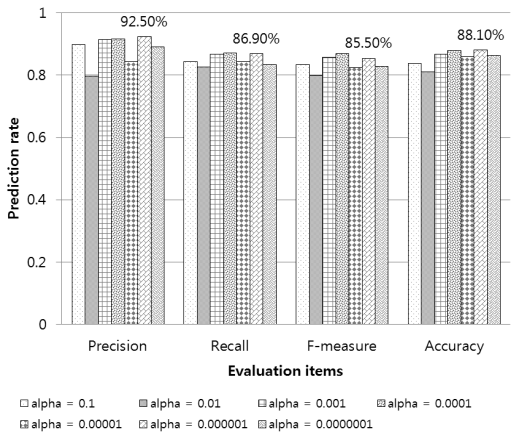


Fig. 9. Comparison Graph of prediction accuracy according to alpha parameter change

5.3 기존 도구와 예측 정확도 비교 실험 결과

기계학습 알고리즘을 이용한 기존 도구인 VDiscover, VulPredictor 보다 신경망을 이용한 NVFinder로 바이너리 취약 여부를 예측할 때 전체적인 예측 정확도가 증가하였으며 비교 결과는 Table 2., Fig.10.과 같다. 트레이스로 추출한 학습 데이터가 벡터 상에 일정하게 분포되어있지 않아서 XOR 연산에 적합한 다층 퍼셉트론의 예측률이 가장 높게 나온 것이라고 판단된다.

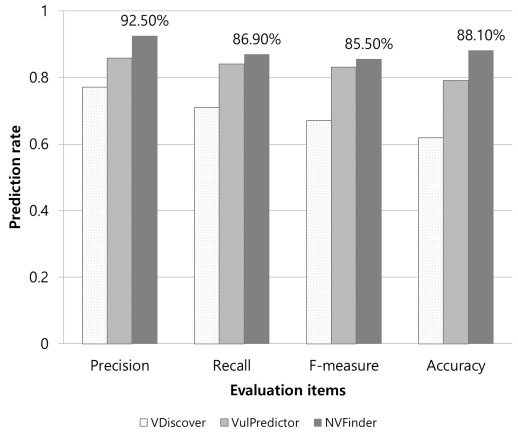


Fig. 10. VulPredictor, VDiscover, NVFinder Prediction Accuracy Comparison Graph

Table 2. VulPredictor, VDiscover, NVFinder Prediction Accuracy Comparison

Tool	Algorithm	Precision	Recall	F-measure	Accuracy
VDiscover	Random Forest	77%	70.9%	67%	61.8%
VulPredictor	k-Nearest Neighbors	85.7%	84%	83%	79.1%
NVFinder	**Multi-layer Perceptron**	**92.5%**	**86.9%**	**85.5%**	**88.1%**

VI. 결 론

소프트웨어 증가에 따라 필연적으로 소프트웨어 취약점도 함께 증가하고 있다. 소프트웨어 취약점을 통해 해커가 시스템을 공격한다면 개인이나 기업은 막대한 손실을 받을 수 있다. 따라서 본 논문은 막대한 손실을 줄 수 있는 소프트웨어 취약점을 사전에 탐지하여 제거하기 위해 신경망을 적용하여 소프트웨어 취약 여부를 예측하였고, 더 나아가 기존 기계학습을 이용한 취약 여부 예측 시스템인 VDiscover, VulPredictor와 예측 비교 실험을 진행하였다. 실험 결과 기계학습을 이용한 기존의 도구보다 신경망을 이용한 NVFinder로 바이너리 취약 여부를 예측할 때 전체적인 예측 수치가 향상되었다.

본 논문에서 제안하는 소프트웨어 취약 여부 예측 시스템을 통해 소프트웨어 취약점에 대한 대책을 보다 빠르고 정확하게 마련할 수 있을 것으로 예상한다.

References

[1] SPRI, "Software Policy & Research Institute" https://www.spri.kr/download/21696, Sep. 2018.

[2] AVECTO, "Microsoft Vulnerabilities Report 2017" https://www.avecto.com/resources/reports/microsoft-vulnerabilities-report-2017, Sep. 2018.

[3] Miller, Barton P., Louis Fredriksen, and Bryan So. "An empirical study of the reliability of UNIX utilities." Communications of the ACM, vol. 33, no. 12, pp. 32-44, Dec. 1990.

[4] King, James C. "Symbolic execution and program testing." Communications of the ACM, vol. 19, no. 7, pp. 385-394, Jul. 1976.

[5] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller. "Predicting vulnerable software components." ACM, pp. 529 - 540, Oct. 2007.

[6] Nguyen, Viet Hung, and Le Minh Sang Tran. "Predicting vulnerable software components with dependency graphs." Proceedings of the 6th International Workshop on Security Measurements and Metrics. ACM, pp. 3:1-3:8. Sep. 2010.

[7] Yonghee Shin, and Laurie Williams. "An empirical model to predict security vulnerabilities using code complexity metrics." Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. ACM, pp. 315-317. Oct. 2008.

[8] Yonghee Shin, and Laurie Williams. "Is complexity really the enemy of software security?." Proceedings of the 4th ACM workshop on Quality of protection. ACM, pp. 47-50, Oct. 2008.

[9] Young Jun Lee, Sang-Hoon Choi, Chulwoo Kim and Ki-Woong Park. "Learning Binary Code with Deep Learning to Detect Software Weakness." KSII The 9th International Conference on Internet (ICONI) 2017 Symposium, 2017.

[10] Grieco, G., Grinblat, G. L., Uzal, L.,

Rawat, S., Feist, J., and Mounier, L. "Toward large-scale vulnerability discovery using Machine Learning." Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. ACM, pp. 85-96, Mar. 2016.

〔11〕 Minjun Choi, Juhwan Kim and Joobeom Yun, "Software Vulnerability Prediction System Using Machine Learning Algorithm." Journal of the Korea Institute of Information Security & Cryptology, 28(3), pp. 635-642, Jun, 2018.

〔12〕 Kang Yoon Lee and Junhewk Kim. "Artificial Intelligence Technology Trends and IBM Watson References in the Medical Field" Korean Medical Education Review, 18(2), pp. 51-57, 2016.

〔13〕 Seonhwa Choi. "Neural Network Model for Prediction of Damage Cost from Storm and Flood" Journal of KIISE, 38(3), pp. 115-123, Mar. 2011

〔14〕 Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search", nature, vol. 529, no. 7587, pp. 484-489. 2016

〔15〕 Deawon Kim, et al. "How AlphaGo does Change People's Perception of Introduction of Artificial Intelligence into Intellectual Work?" Cybercommunication Academic Society, 33(4), pp. 107-158, Oct. 2016.

〔16〕 Iltaek Kwon and Eul Gyu Im. "Study on Application of Recurrent Neural Network to Extract Malware API Call Pattern", Communications of KIISE, pp. 1081-1083, Jun. 2017.

〔17〕 Jae-Hyun Seo, "A Comparative Study on the Classification of the Imbalanced Intrusion Detection Dataset Based on Deep Learning" Journal of KIIS, 28(2), pp. 152-159, Apr. 2018.

〔18〕 NIST, "Juliet Test Sutie" https://samate.nist.gov/SRD/testsuite.php, Mar. 2019.

〔19〕 python-ptrace, "python-ptrace" http://python-ptrace.readthedocs.org, Jul. 2018.

〔20〕 zzuf, "zzuf－multi-purpose fuzzer" http://caca.zoy.org/wiki/zzuf, Jul. 2018.

〔21〕 Witten, Ian H., et al. "Data Mining: Practical machine learning tools and techniques.", Morgan Kaufmann, 2016.

〔22〕 Euijoong Kim "Introduction to Artificial Intelligence, Machine Learning, Deep Learning with Algorithms." 2016.

〔23〕 Szymanski, P., & Kajdanowicz, T. "Scikit-multilearn: a scikit-based Python environment for performing multi-label classification." The Journal of Machine Learning Research, 20(1), pp. 209-230, Feb. 2019.

〔24〕 Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of Machine Learning Research, pp. 2825-2830, Oct. 2011.

〔25〕 Domingos, Pedro. "A few useful things to know about machine learning." Communications of the ACM, vol. 55, no. 10, pp. 78-87, Oct. 2012.

〔26〕 Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. "A practical guide to support vector classification.", pp. 1-16, 2003.

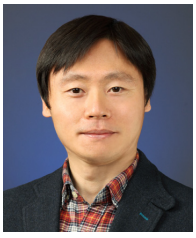
〔27〕 Batista, Gustavo EAPA, Ronaldo C. Prati, and Maria Carolina Monard. "A study of the behavior of several methods for balancing machine learning training data." ACM Sigkdd Explorations Newsletter, vol. 6, no. 1, pp. 20-29, Jun. 2004.

[28] Müller, Andreas C., and Sarah Guido. "Introduction to machine learning with Python: a guide for data scientists." 2016.

〈저자소개〉

최 민 준 (Minjun Choi) 학생회원
2014년 2월: 인천대학교 메카트로닉스공학과, 컴퓨터공학부 복수전공 학사
2018년 8월: 세종대학교 일반대학원 정보보호학과 석사
〈관심분야〉 네트워크 보안, 임베디드 보안, 시스템 보안

구 동 영 (Dongyoung Koo) 종신회원
2009년 2월: 연세대학교 컴퓨터.산업공학과 졸업
2012년 2월: 한국과학기술원 전산학과 석사
2016년 2월: 한국과학기술원 전산학부 박사
2016년 3월~2017년 3월: 고려대학교 정보대학 컴퓨터학과 연구교수
2017년 4월~현재: 한성대학교 기계전자공학부 조교수
〈관심분야〉 정보보호, 응용 암호, 네트워크 보안, 클라우드/엣지/포그 컴퓨팅 보안



윤 주 범 (Joobeom Yun) 종신회원
1999년 2월: 고려대학교 컴퓨터학과 학사
2001년 2월: 서울대학교 컴퓨터공학과 석사
2012년 2월: KAIST 전산학과 박사
2001년 3월~2015년 2월: ETRI부설연구소 선임연구원
2015년 3월~현재: 세종대학교 정보보호학과 부교수
〈관심분야〉 네트워크 보안, 시스템 보안, 클라우드 컴퓨팅 보안