*Journal of The Korea Institute of Information Security & Cryptology* ISSN 1598-3986(Print)
VOL.28, NO.3, Jun. 2018 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.3.635

# 기계학습 알고리즘을 이용한 소프트웨어 취약 여부 예측 시스템*

최 민 준,† 김 주 환, 윤 주 범‡
세종대학교

# Software Vulnerability Prediction System Using Machine Learning Algorithm*

Minjun Choi,† Juhwan Kim, Joobeom Yun‡
Sejong University

요 약

4차 산업혁명 시대에 우리는 소프트웨어 홍수 속에 살고 있다. 그러나, 소프트웨어의 증가는 필연적으로 소프트웨어 취약점 증가로 이어지고 있어 소프트웨어 취약점을 탐지 및 제거하는 작업이 중요하게 되었다. 현재까지 소프트웨어 취약 여부를 예측하는 연구가 진행되었지만, 탐지 시간이 오래 걸리거나, 예측 정확도가 높지 않았다. 따라서 본 논문에서는 기계학습 알고리즘을 이용하여 소프트웨어의 취약 여부를 효율적으로 예측하는 방법을 설명하며, 다양한 기계학습 알고리즘을 이용한 실험 결과를 비교한다. 실험 결과 k-Nearest Neighbors 예측 모델이 가장 높은 예측률을 보였다.

ABSTRACT

In the Era of the Fourth Industrial Revolution, we live in huge amounts of software. However, as software increases, software vulnerabilities are also increasing. Therefore, it is important to detect and remove software vulnerabilities. Currently, many researches have been studied to predict and detect software security problems, but it takes a long time to detect and does not have high prediction accuracy. Therefore, in this paper, we describe a method for efficiently predicting software vulnerabilities using machine learning algorithms. In addition, various machine learning algorithms are compared through experiments. Experimental results show that the k-nearest neighbors prediction model has the highest prediction rate.

**Keywords:** Machine Learning, Fuzzing, Prediction, Vulnerability, Confusion Matrix

## I. 서 론

현대 사회에서 컴퓨터는 개인과 기업, 더 나아가 국가에서 여러 업무를 효율적으로 처리하기 위해 사용하고 있으며 업무 처리를 위해 사용되는 소프트웨어에는 문서, 음성, 영상, 작성, 편집을 위해 다양한 소프트웨어가 사용된다. 다양한 소프트웨어들은 다수의 취약점이 존재할 수 있고[1], 해당 취약점은 개인이나 기업 자산에 막대한 손해를 끼칠 수 있기 때문에 탐지하여 제거해야 한다.

이러한 소프트웨어의 취약점을 탐지하기 위해 다양한 접근법이 사용된다[2]. 그 중 첫 번째로 퍼징 기법은 취약점 발견 확률을 높이기 위해 대상 분석, 입력 분석 등 여러 단계를 거쳐야 하며 데이터 형태,

Received(02. 13. 2018), Modified(1st: 05. 02. 2018, 2nd: 06. 14. 2018), Accepted(06. 14. 2018)
* 본 연구는 한국연구재단 연구과제(NRF-2018R1D1A1B07047323) 지원으로 수행하였습니다.
† 주저자, choiminjun7077@gmail.com
‡ 교신저자, jbyun@sejong.ac.kr(Corresponding author)

포맷 구조 이해 여부 등에 따라 덤, 스마트 등 여건에 맞는 퍼징 기법을 선택[3]해야 하는 단점이 있다. 두 번째로 프로그램 흐름을 분석하여 취약점을 발견하는 기호 실행[4] 기법의 경우 분석할 바이너리마다 스크립트를 작성해야 하며 스크립트 작성을 위해 라이브러리 사용법까지 숙지하여야 하는 단점이 있다[5]. 또한, 이 방법들은 여러 소프트웨어를 검사할 때 엄청난 시간이 걸린다는 치명적인 한계점도 존재한다.

빠른 시간 내 효율적인 방법으로 취약점을 탐지하고 분석하려면 많은 소프트웨어 중 취약점이 있을 것으로 예상되는 소프트웨어를 찾을 필요가 있다. 이에 본 논문에서는 취약점이 있다고 예상되는 소프트웨어를 찾기 위해 기계학습 알고리즘 중 지도학습 알고리즘을 이용하여 소프트웨어의 취약 여부를 예측하는 방법을 설명하며 더 나아가 지도학습 알고리즘의 종류인 Random Forest, Logistic Regression, Support Vector Machine, k-Nearest Neighbors를 이용하여 예측 정확도를 비교한다.

2장에서는 기존의 취약점 탐지 방법에 대한 연구와 한계점에 대해 살펴보고 3장에서는 기계학습 알고리즘으로 소프트웨어의 취약 여부를 예측하는 개념을 설명하며 4장에서는 실험을 통해 다양한 기계학습 알고리즘의 실험 결과를 비교한다. 끝으로 5장에서 결론을 제시한다.

## II. 관련 연구

소프트웨어의 취약점을 검사하는 자동화된 소프트웨어 검사 기법 중 퍼징 기법은 소프트웨어에 무작위 데이터를 입력하여 오작동을 유발하며 유발된 오작동으로 취약 여부를 탐지한다[6]. 현재까지 퍼징 기법의 취약점 탐지율을 높이기 위해 다양한 연구가 진행되고 있다[7][8]. 하지만 퍼징 기법은 다수의 취약점을 탐지하기 위해 입력 값을 증가시킬수록 많은 실행 시간이 소요될 수 있고, 상황에 따라 소프트웨어의 포맷 구조 등을 이해해야 한다. 또한, 테스트 소프트웨어를 변경할 때마다 위 과정을 반복해야 하므로 번거롭다는 단점이 있다.

이와 비슷한 도구로 프로그램 흐름을 분석하여 소프트웨어의 취약점을 탐지하는 기호 실행 기법은 실행 시 메모리 누수가 발생할 수 있고 퍼징 기법과 마찬가지로 테스트 소프트웨어를 변경할 때마다 위 과정을 반복해야 하므로 번거로우며 많은 실행 시간이 소요될 수 있다. 따라서 본 장에서는 퍼징 기법, 기호 실행 기법보다 효율적으로 취약점을 분석하기 위해 국내, 국외 연구 중 기계학습을 이용하여 취약 여부를 예측하는 방법에 대해 살펴본다.

### 2.1 의존성 그래프를 이용한 취약 여부 예측

의존성 그래프를 이용한 취약 여부 예측[9]은 취약한 소스코드 데이터베이스와 일반적인 소스코드를 비교하여 결과를 추출하고, 이 추출된 결과와 일반적인 소스코드를 의존성 그래프로 나타낸 특징을 합쳐 학습 모델을 생성한다. 예측은 새로운 소스코드를 입력 값으로 주었을 때, 새로운 소스코드를 의존성 그래프로 특징을 나타내며 나타낸 특징을 기반으로 학습 모델을 통해 취약 여부를 예측한다.

하지만 소스코드를 대상으로 한 예측 방법들[1][10][11]은 의존성 그래프를 이용한 취약 여부 예측처럼 소스코드를 공개하지 않고 실행 파일만 제공할 경우 취약점 예측이 불가능하다.

### 2.2 어셈블리코드를 이용한 취약 여부 예측

어셈블리 코드를 이용한 취약 여부 예측[12]은 취약한 바이너리의 어셈블리 코드와 취약하지 않은 바이너리의 어셈블리 코드를 벡터 상에 위치시켜 취약 여부를 예측한다. 어셈블리 코드는 프로그램 구조를 상세하게 나타낼 수 있기 때문에 어셈블리 코드를 학습하게 되면 높은 정확도를 기대할 수 있다. 하지만 ARM, X86 과 같이 CPU 구조가 다를 경우 어셈블리 코드도 다르기 때문에 동일한 구조를 가지는 CPU로 추출된 어셈블리 코드로만 예측 실험을 해야 하는 단점이 있다.

### 2.3 트레이스(Trace)를 이용한 취약 여부 예측

트레이스를 이용한 취약 여부 예측[13]은 트레이스와 소프트웨어 보안 검사 도구를 통해 바이너리의 취약 여부 결과 및 특징을 추출하고 이 추출된 결과로 학습 모델을 생성한다. 예측은 테스트할 바이너리의 특징을 트레이스를 통해 추출하고 추출된 특징을 기반으로 취약 여부를 예측한다. 예측 실험 결과 기계학습 알고리즘 중 지도학습 알고리즘의 한 종류인 Random Forest 알고리즘 예측률이 가장 높았음을 보여주었다. 트레이스를 이용한 취약 여부 예측은

소스코드 없이 실행 파일만 제공해도 취약 여부를 예측할 수 있다는 장점이 있지만, 예측 정확도가 높지 않다는 단점이 있다. 따라서 본 논문에서는 트레이스를 이용한 예측 방법을 기반으로 VulPredictor라는 취약점 예측 도구를 제작하여 예측률을 높이기 위한 실험을 진행하였다.

## III. VulPredictor의 취약 여부 예측 방법

본 장에서는 트레이스 및 기계학습 기법 등을 이용해 취약 여부를 예측하는 도구인 VulPredictor가 어떻게 학습하고 예측하는지 살펴본다.

### 3.1 학습 방법

VulPredictor는 예측을 하기 위해서 학습할 데이터가 있어야 한다. 학습 데이터를 모으려면 전처리 과정을 진행하여야 하며 전처리 과정을 통해 생성된 바이너리가 실행될 때의 특징 데이터와 바이너리의 취약 여부 데이터를 학습한다.

먼저 바이너리가 실행될 때의 특징 데이터를 모으기 위해 우분투 14.04 32bit 설치 시 기본적으로 설치되는 바이너리 및 추가로 설치한 바이너리 827개를 이용하였다. 그리고 이 바이너리를 실행할 수 있는 75,072개의 바이너리 명령어 리스트를 작성하였다. 작성된 명령어 리스트를 바이너리에 입력하여 제대로 실행되는지 실행이 된다면 어떠한 특징을 가지는지 ptrace[14]로 확인해 보았다. ptrace 확인 결과 75,072개의 명령어 중 14,886개의 명령어만 바이너리에 입력하였을 때 바이너리가 제대로 동작하였고, 제대로 동작하는 14,886개의 명령어를 바이너리에 입력 후 바이너리가 실행될 때의 System Call을 ptrace로 추출하여 특징으로 사용하였다. 바이너리 취약 여부 데이터는 바이너리에 제대로 동작하는 14,886개의 명령어를 주었을 때 취약점이 발생하는지 소프트웨어 보안 검사 도구를 이용하여 확인 후 값을 저장하였다.

학습 과정에 이용할 바이너리의 취약 여부 데이터가 존재한다면 소프트웨어 보안 검사 도구를 이용하는 과정을 생략하여도 되지만 직접 데이터를 모아서 실험하기 위해 검사 과정을 추가하였으며 검사 도구는 zzuf[15] 퍼저를 이용하였다. 덧붙여 본 논문의 목적은 바이너리가 취약한지 안 한 지 빠르게 예측하는 것이기 때문에 단순히 퍼징에서 크래시(crash)가

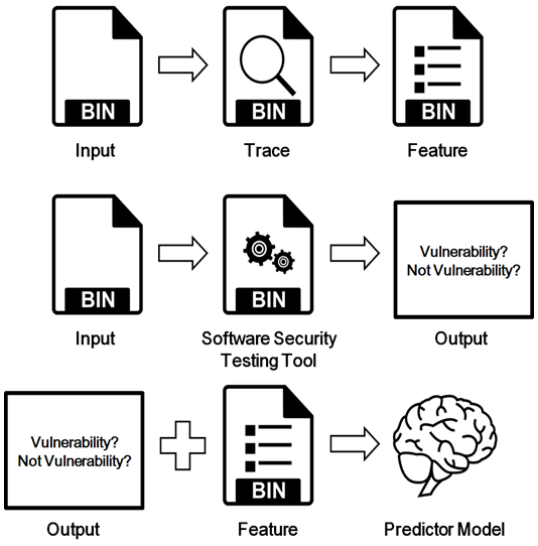



Fig. 1. Learning Process

발생할 경우 크래시에 대해 따로 분석하지 않고 취약하다고 값을 저장하였다. 만약 정보누설(information leak)취약점 이나 흐름 제어(control flow)취약점 등의 취약 정보를 수집하여 학습한다면 해당 취약점에 대한 예측 확률이 높아질 수 있다.

위 방법으로 생성된 바이너리가 실행될 때의 특징 14,886개와 바이너리 취약 여부 14,886개를 1:1매핑 한 14,886개의 데이터로 예측 모델을 생성하였으며 과정을 요약하면 Fig.1.과 같다.

### 3.2 예측 방법

예측은 학습 방법에서 14,886개의 데이터로 생성한 예측 모델을 통해 새로운 바이너리 취약 여부를 예측하게 된다. 우선 14,886개 데이터의 자연어 처리를 위해 널리 사용되는 전처리 기술인 Bag-of-words 알고리즘[16]으로 데이터를 벡터 공간에 위치시킨다. 그리고 예측할 바이너리의 특징을 ptrace로 추출해서 기존의 벡터 공간에 위치시킨다. 이때 예측할 바이너리의 특징이 취약한지 취약하지 않은지 판별하기 위해 기계학습 알고리즘 중 지도학습 알고리즘 모델을 이용하여 벡터 공간에 위치한 데이터를 분류하고 만약 취약하지 않은 영역에 새롭게 예측할 바이너리의 특징이 있다면 새롭게 예측할 바이너리는 취약하지 않다고 예측하며 예를 들면 Fig.2.와 같

다. 또한 실험에 사용된 모델을 살펴보면 Table 1.과 같고 예측과정을 정리하면 Fig.3.과 같으며 VulPredictor를 실행하게 되면 Fig.4. 결과는 Fig.5.와 같이 출력된다.

본 논문에서 지도학습 알고리즘을 이용한 이유는 벡터 공간에 위치한 데이터 분류가 가능하고, 정답이 포함된 학습 데이터를 주고 테스트 데이터를 주면 어느 부분에 속하는지 분류할 수 있기 때문에 지도학습 알고리즘 모델을 사용하였다.

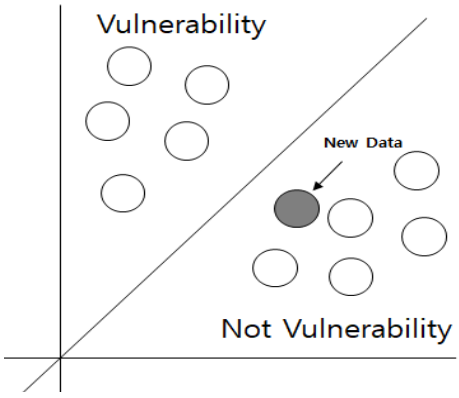



Fig. 2. Classifier Diagram

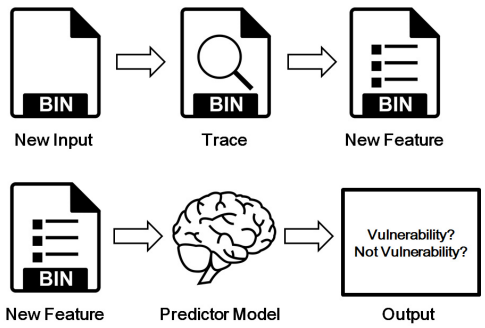



Fig. 3. Prediction Process

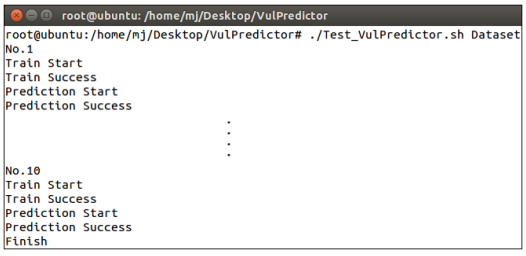



Fig. 4. Execution Screen of VulPredictor

Table 1. Supervised Learning Algorithm Model[17]

| Model | Summary |
|---|---|
| Random Forest | A method that does not require data scale adjustment. |
| Logistic Regression | A way to learn large data and high dimensional data. |
| Support Vector Machine | Data scaling adjustment is required and sensitive to parameters. |
| k-Nearest Neighbors | This is the basic method and is suitable if the data is small. |

```
Number 1
              precision    recall  f1-score   support

           0       0.72      0.74      0.73       102
           1       0.50      0.48      0.49        56

avg / total       0.64      0.65      0.64       158

Accuracy per class: 0.74 0.48
Average accuracy: 0.61
                         .
                         .
                         .
                         .

Number 10
              precision    recall  f1-score   support

           0       0.87      0.70      0.78       115
           1       0.48      0.72      0.57        43

avg / total       0.76      0.71      0.72       158

Accuracy per class: 0.7 0.72
Average accuracy: 0.71
```

Fig. 5. Result Screen of VulPredictor

## IV. 성능평가

성능평가는 우분투 14.04 32bit에서 진행하였으며 기계학습 알고리즘 중 지도학습 알고리즘을 사용할 수 있는 오픈소스 도구인 Scikit-learn[18] 도구를 이용하여 데이터를 분류하였다.

실험은 과대적합(overfitting)을 줄이기 위해 교차검증[19][20]을 시행하였다. 교차검증은 학습 과정을 통해 생성된 데이터 중 무작위 25% 데이터를 예측 시험 대상으로 이용하였고 나머지 75% 데이터를 예측 모델로 만들어 예측 시험 대상의 취약 여부를 얼마나 잘 예측하는지 실험하였다. 또한, 알고리즘 당 위 과정을 총 10번 반복하여 평균값을 측정하였으며 1번 반복 시간은 약 15분 소요되었다. 평균

값은 알고리즘 성능을 직접적으로 평가하기 위해 오차 행렬(confusion matrix)[21]을 이용하여 Precision, Recall, F-measure, Accuracy를 계산하였으며 요약하면 Table 2.와 같고 실험 결과는 Table 3., Fig.6.과 같다.

실험 결과의 평가 항목 중 Precision, Recall, F-measure, Accuracy에 대해 설명하면, 먼저

Precision은 1이라고 예측한 것 중 실제 1이라고 예측한 것을 비율로 나타낸 것으로 다음과 같다.

$$Precision = \frac{TP}{TP+FP}$$

Recall은 실제 값이 1인 것 중 1이라고 예측한 것을 비율로 나타낸 것으로 다음과 같다.

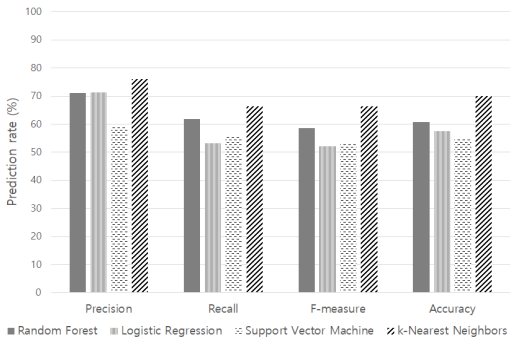



Fig. 6. Experiment Result

Table 2. Confusion Matrix Summary

| | | Actually | |
|---|---|---|---|
| | | Yes | No |
| Predicted | Yes | True Positive (TP) | False Positive (FP) |
| | No | False Negative (FN) | True Negative (TN) |

$$Recall = \frac{TP}{TP+FN}$$

F-measure는 Precision과 Recall의 조화평균을 나타낸 것으로 다음과 같다

$$F-measure = 2 \times \frac{Recall \times \mathrm{Precision}}{Recall + \mathrm{Precision}}$$

Accuracy는 전체에서 올바르게 예측한 것이 몇 개인지를 비율로 나타낸 것으로 다음과 같다.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

위 식에서 TP는 True Positive를 뜻하며 1이라고 예측했는데 실제 1일 경우이고, TN은 True Negative를 뜻하며 1이라고 예측했는데 실제는 0인 경우이다. 반대로 FP는 False Positive를 뜻하며 0이라고 예측했는데 실제 1일 경우이고, FN은 False Negative를 뜻하며 0으로 예측해야 하는데 1로 예측한 경우이다.

실험 결과 Grieco 등[13]의 실험에서 가장 높은 예측률을 보인 Random Forest 알고리즘보다 VulPredictor에서 이용된 k-Nearest Neighbors 알고리즘으로 바이너리 취약 여부를 예측할 때 Precision 항목 4.9%, Recall 항목 4.5% F-measure 항목 7.9, Accuracy 항목 9.3% 향상되었다. 이는 본 실험에서 사용된 ptrace로 추출한 System Call 데이터가 취약 여부 예측에 가장 적합한 특성을 가졌기 때문에 k-Nearest Neighbors의 예측률이 가장 높게 나온 것이라고 판단된다.

## V. 결 론

컴퓨터의 보급화에 따라 다양한 소프트웨어가 개

Table 3. Experiment Result

| Algorithm | Precision | Recall | F-measure | Accuracy |
|---|---|---|---|---|
| Random Forest | 71.1% | 61.9% | 58.5% | 60.8% |
| Logistic Regression | 71.2% | 53.1% | 52.2% | 57.5% |
| Support Vector Machine | 59.5% | 55.7% | 52.9% | 55.0% |
| k-Nearest Neighbors | 76.0% | 66.4% | 66.4% | 70.1% |

발, 사용되고 있고 소프트웨어에는 다수의 취약점이 존재하는데 해당 취약점으로 인해 개인 및 기업에 막대한 손실을 줄 수 있다. 막대한 손실을 가져올 수 있는 취약점을 찾기 위해서는 비용과 시간이 소요되는데, 이러한 손실을 줄이기 위해 본 논문에서는 기계학습 알고리즘을 이용해 소프트웨어 취약 여부를 예측하였고 더 나아가 기계학습 알고리즘 중 지도학습 알고리즘들의 예측률을 비교하였다. 실험 결과 기존 트레이스 기반 취약 여부 예측 방법[13] 연구에서 가장 뛰어난 예측률을 보인 Random Forest 알고리즘 보다 본 논문에서 제안한 VulPredictor에서 이용된 k-Nearest Neighbors을 사용할 시 전체적인 예측 수치가 향상되었다. 향후 VulPredictor의 예측률을 향상시키기 위해 트레이스 기반이 아닌 다른 방식 및 기계학습의 다양한 알고리즘을 연구하여 적용할 예정이다.

## References


[1] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," Proceedings of the 14th ACM conference on Computer and communications security, pp. 529 - 540, Oct. 2007.

[2] Catal, Cagatay, and Banu Diri, "A systematic review of software fault prediction studies," Expert systems with applications, vol. 36, no. 4, pp. 7346-7354, May. 2009.

[3] Sangsu Kim, and Dongsu Kang, "Software Vulnerability Analysis using File Fuzzing," Conference of The Korean Society Of Computer And Information, 25(2), pp. 29-32, Jul. 2017.

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

[5] Sunnyeo Park, Wonchan Oh, Hyeonkee Cho and Eunsun Cho, "Automated Vulnerability Analysis Tool for Binary Codes based on Symbolic Execution," Journal of Scientific Conference of Korean Institute of Information Scientists and Engineers, pp. 1917-1919, Jun. 2017.

[6] 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.

[7] MinSik Shin, JungBeen Yu, and Taekyoung Kwon, "A Study of File Format-Aware Fuzzing against Smartphone Media Server Daemons," Journal of the Korea Institute of Information Security, 27(3), pp. 541-548, Jun. 2017.

[8] Jegyeong Jo, and Jaecheol Ryou, "Method of Fuzzing Document Application Based on Android Devices," Journal of the Korea Institute of Information Security, 25(1), pp. 31-37, Feb. 2015.

[9] 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, pp. 1-8, Sep. 2010.

[10] 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, pp. 315-317, Oct. 2008.

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

[12] 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 2017 Symposium, pp. 245-249, Dec. 2017.

[13] 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, pp. 85-96, Mar. 2016.

[14] V. Stinner, python-ptrace, http://python-ptrace.readthedocs.org/

[15] Hocevar, S. zzuf-multi-purposefuzzer, http://caca.zoy.org/wiki/zzuf/

[16] Witten, I.H., Frank, E., Hall, M.A. and Pal, C.J., Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, Oct. 2016.

[17] Müller, Andreas C. and Sarah Guido, Introduction to machine learning with Python: a guide for data scientists, O'Reilly Media, Inc., Oct. 2016.

[18] Pedregosa, Fabian and et al., "Scikit-learn: machine learning in python," Journal of Machine Learning Research, pp. 2825-2830, Oct. 2011.

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

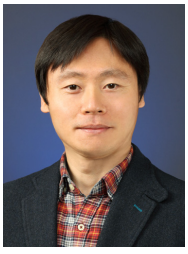
[20] Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin, "A practical guide to support vector classification," Department of Computer Science, National Taiwan University, Jul. 2003.

[21] 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.

## 〈저자소개〉

최 민 준 (Minjun Choi) 학생회원
2014년 2월: 인천대학교 메카트로닉스공학과, 컴퓨터공학부 복수전공 학사
2016년 9월~현재: 세종대학교 일반대학원 정보보호학과 석사과정
〈관심분야〉 네트워크 보안, 임베디드 보안, 시스템 보안

김 주 환 (Juhwan Kim) 학생회원
2016년 2월: 학점은행제 정보보호학 전공 학사
2017년 3월~현재: 세종대학교 일반대학원 정보보호학과 석사과정
〈관심분야〉 악성코드 분석, 시스템 보안, 소프트웨어 취약점



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 조교수
〈관심분야〉 네트워크 보안, 시스템 보안, 클라우드 컴퓨팅 보안