Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.6, Dec. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.6.1043

함수 단위 N-gram 비교를 통한 Spectre 공격 바이너리 식별 방법*

김 문 선,[1†] 양 희 동,[1] 김 광 준,[1] 이 만 희[2‡]
[1,2]한남대학교 (대학원생, 교수)

Detecting Spectre Malware Binary through Function Level N-gram Comparison*

Moon-Sun Kim,[1†] Hee-Dong Yang,[1] Kwang-Jun Kim,[1] Man-Hee Lee[2‡]
[1,2]Hanam University (Graduate student, Professor)

요 약

시그니처 기반 악성코드 탐지는 제로데이 취약점을 이용하거나 변형된 악성코드를 탐지하지 못하는 한계가 있다. 이를 극복하기 위해 N-gram을 이용하여 악성코드를 분류하는 연구들이 활발히 수행되고 있다. 기존 연구들은 높은 정확도로 악성코드를 분류할 수 있지만, Spectre와 같이 짧은 코드로 동작하는 악성코드는 식별하기 어렵다. 따라서 본 논문에서는 Spectre 공격 바이너리를 효과적으로 식별할 수 있도록 함수 단위 N-gram 비교 알고리즘을 제안한다. 본 알고리즘의 유효성을 판단하기 위해 165개의 정상 바이너리와 25개의 악성 바이너리에서 추출한 N-gram 데이터셋을 Random Forest 모델로 학습했다. 모델 성능 실험 결과, 25개의 Spectre 악성 함수의 바이너리를 99.99% 정확도로 식별했으며, f1-score는 92%로 나타났다.

ABSTRACT

Signature-based malicious code detection methods share a common limitation; it is very hard to detect modified malicious codes or new malware utilizing zero-day vulnerabilities. To overcome this limitation, many studies are actively carried out to classify malicious codes using N-gram. Although they can detect malicious codes with high accuracy, it is difficult to identify malicious codes that uses very short codes such as Spectre. We propose a function level N-gram comparison algorithm to effectively identify the Spectre binary. To test the validity of this algorithm, we built N-gram data sets from 165 normal binaries and 25 malignant binaries. When we used Random Forest models, the model performance experiments identified Spectre malicious functions with 99.99% accuracy and its f1-score was 92%.

Keywords: Spectre, Binary Analysis, Malware Detection, N-gram

I. 서 론

시그니처 기반 악성코드 탐지는 알려진 악성코드를 높은 정확도로 탐지할 수 있어 보편적으로 사용되고 있으나, 제로데이 취약점이나 변종과 같이 시그니처가 등록되지 않은 악성코드는 식별하기 어렵다. 또한, Spectre 공격과 같이 시그니처를 추출하기 어려운 악성코드가 등장하면서 시그니처 기반 탐지의 한계점이 더욱 부각되고 있다[1]. Spectre 공격 코드는 정상적인 프로그램과 유사한 형태의 알고리즘으

Received(10. 6. 2020), Modified(11. 18. 2020), Accepted(11. 18. 2020)
* 본 논문은 2020년도 한국정보보호학회 하계학술대회에 발표한 우수논문을 개선 및 확장한 것임
* 이 성과는 2020년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2018R1A4A1025632).
† 주저자, kmoonsun95@gmail.com
‡ 교신저자, manheelee@hnu.kr(Corresponding author)

로 구성되어 있으며, 프로세서의 추측 실행 메커니즘의 취약점을 악용하기 때문에 공격 중 시스템에 별다른 흔적을 남기지 않는다. 이에 따라, 현재 안전한 아키텍처를 적용할 수 없는 기존 시스템은 Spectre 공격을 방지하기 위해 이 공격이 악용하는 분기 예측 기능을 제한하는 명령어를 삽입하거나, 정적으로 취약할 수 있는 코드를 식별하여 완화하는 방법을 활용하고 있다. 하지만 이 방법은 많은 성능 저하를 유발한다[2, 3].

이 문제를 해결하기 위해 다양한 동적 분석 기반 Spectre 공격 탐지 연구들이 수행되었다[4, 5]. 이 연구들은 Spectre 공격 프로세스가 유발하는 고유한 하드웨어 이벤트들을 기반으로 공격 프로세스를 실시간 탐지를 한다. 혹은 기호 실행(symbolic execution)을 바탕으로 Spectre 공격에 악용될 여지가 있는 바이너리를 찾는다. 다만, 이 방법들은 이미 공격이 실행 중일 때 탐지를 하거나, 동적 분석을 기반으로 동작하기 때문에 대규모 파일에서 빠르게 Spectre 공격 파일을 탐색하는 것이 사실상 어렵다.

따라서 본 논문은 대규모 파일 검사 시스템 등에서 활용될 수 있는 N-gram[6] 기반 Spectre 악성 바이너리 정적 식별 모델을 제안한다. 이 모델은 N-gram을 통해 분석 대상 바이너리의 opcode 시퀀스를 추출한 뒤, Random Forest 알고리즘을 사용하여 악성 바이너리를 탐지한다. 또한, 탐지 정확도를 높이기 위해 함수 단위로 N-gram 특징을 비교한다.

함수 단위 N-gram 비교는 전체 바이너리를 분석하는 기존의 N-gram 기반 악성코드 분류 방식과 달리, 직접적으로 공격 행위가 구현된 악성 함수만 식별하는 데 중점을 둔다[7, 8, 9, 10, 11]. Spectre 악성 바이너리를 탐지하기 위한 바이너리 정적 분석 방법은 찾지 못했기 때문에 우리의 모델은 다른 Spectre 공격 방어 연구들이 적용되기 어려운 분야에 활용될 수 있을 것으로 판단된다.

이 방법의 유효성을 검증하기 위해 모델의 탐지 성능 실험을 진행했다. 실험 데이터는 Linux 시스템에서 사용하는 시스템 명령어 바이너리 165개와 변종을 포함한 Spectre 악성코드 25개를 사용했다. 탐지 실험 결과, 109,079개의 정상 함수와 25개의 Spectre 악성 함수를 99.99%의 정확도로 구분하였다.

II. 관련 연구

2.1 동적 분기 예측

추측 실행(speculative execution)은 프로세서가 향후 실행될 것으로 예상하는 명령어를 미리 실행하는 성능 향상 기술이다. 이 기술의 일종인 동적 분기 예측(dynamic branch prediction)은 분기 예측 유닛(branch prediction unit)에 위치한 BTB(Branch Target Buffer)에 저장된 명령어 실행 기록을 바탕으로 분기 명령어의 실행 방향을 예측한다[2]. 2-bit branch prediction 구조를 예로 들면, 한번 실행된 분기 명령어는 BTB에 자신의 주소와 분기하는 목적지 주소(branch target address), 분기 여부 비트(taken/not-taken bit)를 저장한다. 분기 여부 비트는 2^2개로 구성되며, 이진수로 각각 00(strong not-taken), 01(weakly not-taken), 10(weakly taken), 11(strong taken)이다.

Fig. 1.은 동적 분기 예측의 간단한 예시이다. 분기 명령어가 실행될 때 우선 BTB entry에 해당 분기 명령어의 주소가 있는지 조회한다. 만약 조회에 성공하면 분기 여부 비트를 확인한다. 이때, 분기 여부 비트가 10_2(weakly taken) 이상이면 이 분기 명령어의 다음 명령어가 분기 대상 주소인 것으로 판단하여, 분기 대상 주소로 jump, 즉 분기 대상 주소의 명령어를 읽어와 미리 실행한다. 반대로 10_2 미만이면 분기하지 않고 다음 명령어(PC)를 읽어와 실행한다. 분기 명령어가 BTB entry에 조회되지

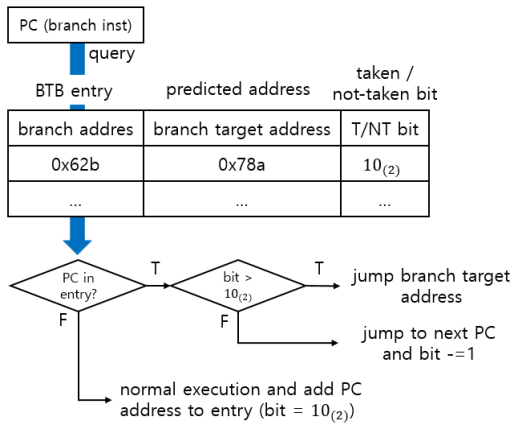


Fig. 1. Branch target buffer example

않는 경우에는 일반적인 경로로 실행한 결과를 토대로 해당 명령어의 주소와 분기 대상 주소를 BTB에 기록한다. 이때 분기 여부 비트는 10_2으로 기록된다. 이처럼 분기 예측은 조건 분기 명령어의 결과가 결정되는 동안 발생하는 파이프라인의 유휴상태를 최소화하여 프로세서의 전체적인 성능 향상에 기여한다[2, 12].

분기 예측은 때때로 예측에 실패하여 정상적인 경로를 벗어나는 코드를 실행할 수 있다. 따라서 프로세서는 분기 예측 결과로부터 실행된 코드를 즉시 commit, 즉 레지스터나 메모리를 변경하지 않은 상태로 분기 예측이 시작된 명령어의 결과를 확인한다. 만약, 이 분기 명령어가 분기를 하지 않는 것으로 결정되거나 분기 대상 주소가 틀린 경우에는, 추측 실행한 프로세서 내부 결과를 모두 폐기하고 올바른 경로로 다시 실행한다. 따라서 잘못 예측 실행된 결과는 commit 하지 않고 모두 폐기하므로, 추측 실행된 정보는 모두 사라지는 것으로 알려져 왔다.

2.2 Spectre 공격

P. Kocher 등은 Spectre 공격을 통해 의도적으로 잘못된 분기 예측을 유도한 뒤, 추측 실행된 결과를 캐시 부채널 공격(cache-based side-channel attack)을 통해 유출했다. 캐시 부채널 공격은 공격자가 L3 캐시와 같은 공유 캐시를 조작하여 간접적으로 다른 프로세스의 연산에 사용되고 있는 메모리 값을 유출하는 공격이다. Fig. 2.는 P. Kocher 등이 설명하는 Spectre 유형 1의 개념을 설명하기 위한 코드 예시이다.

Fig. 2.의 *spectre_v1_vulnerable_function* 함수는 일반적인 조건문을 포함하는 함수이다. line 15의 조건문은 *offset*이 *arr1*의 길이보다 작은 경우에만 *data*를 읽어온다. 이 조건을 만족하지 못하면 *data*에는 접근할 수 없다. 하지만 Spectre 공격은 프로세서의 추측 실행을 악용하여 *data* 배열의 크기(data->length)를 넘어서는 위치의 값을 읽을 수 있다. 이 과정은 다음과 같다.

1. 공격자는 공격 함수(*spectre_v1_vulnerable_function*) 안에 위치한 조건문(line 15)을 만족하는 *offset* 값을 반복적으로 제공한다. 이 과정에서 해당 조건문의 분기 명령어는 BTB에 strong taken 상태로 기록된다.

```
1  struct array {
2      unsigned long length;
3      unsigned char data[];
4  };
5
6  struct array *create_array_of_size(unsigned int size);
7
8  // The vulnerable function to spectre attack.
9  void spectre_v1_vulnerable_function(unsigned long offset) {
10     struct array *arr1 = create_array_of_size(5);
11     struct array *arr2 = create_array_of_size(4096);
12     unsigned char secret;
13
14     // bounds check bypass by spectre attack
15     if (offset < arr1->length) {
16         secret = arr1->data[offset];
17         // array2->data[secret] is cached in L1-Data-Cache
18         unsigned char temp = arr2->data[secret * 4096];
19     }
20 }
```

Fig. 2. Example code to explain spectre v1.

2. 조건문이 실행되기 전에 *offset*을 임의의 값으로 변경한다. 이후 clflush(Cache Line Flush) 명령어를 사용하여 *arr1->length*와 *arr2->data*를 모든 계층의 캐시 라인에서 제거한다.
3. line 15에서 조건 검사가 수행될 때, *offset*은 (2)과정에서 캐시 되었으므로 프로세서가 빠른 속도로 읽을 수 있다. 반면, *arr1->length*는 (2) 과정을 통해 캐시에서 제거되었기 때문에 프로세서는 메모리로부터 데이터를 요청한다. 이는 일반적으로 수백 사이클이 소요된다.
4. 프로세서는 수백 사이클을 대기하는 대신 BTB에 기록된 과거 실행 기록을 바탕으로 분기 명령어의 목적지를 추측한다. 이때, 해당 조건문(line 15)의 분기 대상 주소인 line 16의 시작 주소는 strong taken 상태로 기록되어 있다. 따라서 line 16과 18은 추측 실행된다.
5. 추측 실행 도중, line 18의 *temp = arr2->data[secret * 4096]*의 *arr2->data*는 캐시 미스가 발생한다. 따라서 메모리로부터 데이터를 읽어오며, 이 데이터는 L1 캐시에 적재된다.
6. 프로세서는 잘못된 분기 예측을 인지하고 *arr->data*를 비롯한 정보들을 제거하지만, 캐시에 적재된 *data*는 온전히 남는다. 따라서 공격자는 캐시 부채널 공격을 수행하여 이 데이터를 유출할 수 있다.

이처럼, Spectre 공격은 분기 예측의 취약점을 악용하면 배열의 경계를 넘는 비밀 값을 읽거나 공유 라이브러리 내부의 gadget을 이용하여 다른 프로세스의 메모리를 유출하는 공격이 가능하다[1]. 따라서 이 공격에 대응하기 위해 새로운 아키텍처가 설계되는 등 다양한 완화 패치들이 공개되었다[3, 13, 14]. 본 논문은 하드웨어적인 패치를 적용할 수 없는 기성 시스템에서의 완화 패치에 대해 중점적으로 설명한다.

2.3 기존 시스템에 대한 Spectre 공격 완화 패치

Microsoft Visual C/C++ 컴파일러는 Spectre 공격 코드로 의심되는 조건 분기에 lfence 명령어를 삽입하여 Spectre 유형 1을 완화한다. lfence 명령어는 병렬로 실행 중인 모든 명령어가 완료될 때까지 실행되지 않는다. 따라서 잘못된 분기 예측으로부터 발생할 수 있는 Spectre 공격을 방지할 수 있다[14]. Fig. 3.의 하단은 qspectre 옵션이 적용된 상태에서 컴파일된 Spectre 유형 1의 실행 코드이다. line 4에 lfence 명령어가 있기 때문에 line 3의 분기는 추측 실행되지 않는다.

다만, G. Wang 등의 연구에 의하면 qspectre 옵션은 P. Kocher가 제시한 15가지 Spectre 공격 유형 중 단 2개만 완화할 수 있다. 또한 lfence 명령어는 분기 예측의 이점을 포기하기 때문에 이 옵션이 적용된 프로그램은 많은 성능 저하가 우려된다[15, 16].

Linux 커널은 lfence 명령어를 커널의 주요 부분에 삽입하여 Spectre 유형 1을 완화했다. 또한 지속적으로 보고되고 있는 Spectre 변종 유형을 방지하기 위해 retpoline과 같은 대응 패치를 지속적으로 적용하고 있다[3, 13]. 하지만 이 패치들은 N. A. Simakov 등의 응용 프로그래밍 성능 평가 실험에서 평균 2~3%의 성능 오버헤드를 유발하는 것으로 측정되었으며, 일부 구간에서는 오버헤드가 74%에 달하는 것으로 나타났다[17].

spectre PoC

```
void victim_function(size_t x) {
    if (x < array_size) {
        temp &= array2[array[x] * page_size]
    }
}
```

qspectre

```
1. mov eax, DWORD PTR array1_size
2. cmp rcx, rax
3. jae SHORT @LN2@victim_fun
4. lfence // migration instruction
…
```

Fig. 3. Binary with qspectre option applied.

이러한 성능 오버헤드를 피하기 위해 Spectre 공격을 탐지하는 다양한 연구들이 수행되었다[4, 5, 16]. M. Mushtaq 등은 프로세서의 특수 목적 레지스터인 HPC(Hardware Performance Counter)를 이용하여 캐시 부채널 공격의 이벤트를 높은 정확도로 동적 탐지하는 도구인 WHISPER를 제안했다. Spectre 공격은 캐시 부채널 공격을 통해 최종적으로 데이터를 유출하기 때문에 WHISPER는 Spectre 공격을 99% 이상의 정확도로 탐지했다[4]. 다만, 동적 탐지는 효과적으로 Spectre 공격을 감지할 수 있지만, 실시간으로 모든 프로세스를 감시해야하기 때문에 오버헤드가 발생한다. 또한 오탐지의 위험 때문에 프로세스를 강제로 종료하는 등 강력한 조치를 취하기 어렵다는 단점이 있다.

M. Guarnieri 등은 기호 실행 기반 동적 바이너리 분석을 수행하여 잘못된 추측 실행을 유발할 수 있는 분기 명령어를 바이너리 수준에서 탐지하는 SPECTECTOR를 제안했다[5]. 이 도구는 qspectre에 비해 더 다양한 변종을 탐지할 수 있다. 하지만 Z3 기호 실행 엔진과 Ciao 언어를 기반으로 동작하기 때문에 분석 대상이 커질수록 분석이 완료되지 못할 가능성이 발생하며, 반드시 Ciao 언어를 위한 환경이 구축되어야 하는 단점이 있다.

G. Wang 등은 Spectre 바이너리를 안전하게 패치하는 도구인 oo7을 제안했다. 이 도구는 바이너리 분석을 통해 잠재적으로 Spectre 공격에 취약한 분기 명령어들을 찾은 뒤, 적절한 위치에 lfence 명령어를 삽입하여 5.7%의 오버헤드로 취약한 바이너리를 패치한다[16]. 하지만 이 도구는 일부 Spectre 변종만 식별할 수 있으며, 대상 바이너리를 직접 수정하기 때문에 프로그램의 무결성이 훼손되는 문제가 있다.

2.4 N-gram 기반 악성코드 분석

N-gram은 문자열의 유사도를 도출할 수 있는 알고리즘으로써 악성코드 식별 및 패밀리 분류 분야에서 활발히 사용되고 있다[7, 8, 9, 10, 11]. 관련

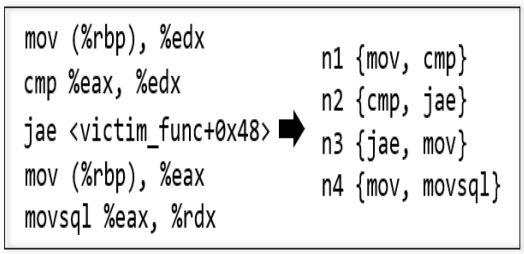


Fig. 4. Example of N-gram (N=2)

연구들은 Fig. 4와 같이 분석 대상 바이너리의 바이트 스트림에서 API 호출 순서, opcode, 문자열 정보 등의 특징 정보를 N-gram을 통해 추출한다. 이후 추출한 특징 정보를 바탕으로 양성 및 악성 여부를 식별하거나 악성코드 패밀리를 분류한다.

I. Santos 등은 기존의 악성코드들을 시그니처 기반으로 탐지했을 때, 변종이나 바이너리 난독화와 같은 우회 기법에 대응하기 어려운 점을 지적하며 N-gram 기반 악성코드 분류 기법을 제안했다[7]. S. Jain 등은 양성 및 악성 바이너리의 특징 정보를 N-gram으로 추출한 뒤, 랜덤 포레스트 모델을 사용하여 99%의 정확도로 악성 바이너리를 식별했다[8].

또한 E. Raff 등은 N-gram이 더 높은 정확도를 얻을 수 있도록 개선하는 다중 바이트 식별 기법을 고안하여 N-gram을 이용한 악성코드 분류의 정확도를 향상시켰다[9]. A. Pektas 등은 n의 크기가 3 또는 4일 때 가장 악성코드를 잘 분류할 수 있음을 실험적으로 보였으며, B. Kang 등은 악성 안드로이드 app을 N-gram과 머신러닝을 사용하여 98% 정확도로 분류할 수 있음을 보였다[10, 11].

III. 함수단위 N-gram 기반 Spectre 탐지 방안

3.1 기존 연구들의 한계점

N-gram 기반 정적 분석 도구들을 악성코드를 98% 이상의 높은 정확도로 분류했다. 하지만 Spectre 공격 코드는 기존 악성코드와 다른 특징을 보인다. Fig. 3.의 상단 C 코드는 Spectre 유형 1의 핵심 공격 함수이다. 2.2절에서 설명한 것과 같이 Spectre 공격은 이 함수의 내부 분기문 (*if x < array*)의 분기 여부 비트를 strong taken 상태로 조작한다. 이후 *x*의 값에 array의 크기를 초과하는 값을 전달하여 잘못된 추측 실행을 유도한 뒤, 캐시 부채널 공격을 수행하여 *array[x]의* 데이터를 유출한다.

이처럼, Spectre 공격 코드는 기존 악성코드들에게 나타나는 C&C(Command & Control) 서버와의 통신이나 수상한 파일을 쓰는 등의 패턴들이 나타나지 않는다. 따라서 지금까지 알려진 악성코드 분석 방법으로는 Spectre 공격 코드를 탐지하기 어렵다. 또한 핵심 공격 함수를 포함하여 전체적인 코드가 정상적인 프로그램과 유사한 형태의 알고리즘으로 구성되기 때문에 더욱 탐지하기 어렵다.

3.2 함수 단위 식별

Spectre 공격 코드의 핵심 공격 함수는 매우 짧은 바이너리로 구성된다. 우리가 조사한 Spectre 공격 코드의 경우, 전체 바이너리 대비 공격 함수 바이너리의 비율이 0.18%로 나타났다. 또한 바이너리는 동적으로 매핑되는 공유 라이브러리나 파일 형식을 유지하기 위한 정보들이 포함되기 때문에 프로그램의 성격과 다른 바이너리들이 다수 포함된다. 따라서 바이너리 전체를 비교하는 기존 연구들의 접근 방법은 길이가 짧은 Spectre 공격 함수의 바이너리를 찾기가 어렵다.

이 문제를 해결하기 위해 우리는 함수 단위 N-gram 비교 기법을 제안한다. 이 기법은 분석 대상의 바이너리를 함수 단위로 분리한 뒤, 각 함수를 하나의 분석 대상으로 취급한다. 이 방법은 공격 함수의 바이너리에서 추출한 N-gram 특징을 기억한 뒤, 다른 함수들과 비교하여 탐색하기 때문에 짧은 길이의 함수도 효과적으로 탐지할 수 있다.

N-gram 유사도를 이용하여 악성 함수를 분류하기 위해서는 임계값을 설정해야 한다. 하지만 임계값은 과적합 문제가 발생하여 실험에 사용한 데이터셋에서만 높은 정확도를 보일 수 있다. 이 문제를 해결하기 위해 우리는 머신러닝 알고리즘 중 하나인 Random Forest 모델을 사용했다. N-gram 특징은 머신러닝 모델이 악성 및 양성 여부를 판단하기 위한 근거 자료로 사용되며, 최종적인 유사성 판단은 Random Forest 모델이 담당한다. RandomForest 모델을 선택한 자세한 내용은 4장에서 설명한다.

3.3 학습 데이터 구축

모델 학습을 위해 수집한 양성 데이터셋은 Ubuntu 16.04 LTS에 기본적으로 설치되는 sbin 폴더의 165개 실행 파일이다. 이 실행 파일에서 정상 함수 109,079개를 추출했다. 악성 데이터셋은 GitHub 등에서 수집한 10개의 Spectre 유형 1, 유형 2 및 P. Kocher가 제시한 15개 Spectre 유형 1이다[15]. 이 중 식별된 악성 함수는 25개이다.

더 다양한 변종과 악성 바이너리를 수집하지 못한 이유는 제안하는 모델의 정확한 성능 검증을 위해 명확히 Spectre 공격 코드로 식별되는 데이터만 수집했기 때문이다. 즉, 악성코드 데이터베이스 등에서 공유되는 Spectre 바이너리는 유형 식별 및 오탐지 여부를 확인하기 어렵기 때문에 코드가 공개된 데이터만 수집했다. Table 1.은 양성 및 악성 데이터의 개수와 전체 데이터에서 차지하는 비율을 정리한 것이다.

악성 함수 식별 및 라벨 지정은 오픈소스 역공학 프레임워크인 radare2[18]를 사용했다. 이 도구는 다양한 파일 유형과 아키텍처의 바이너리에 대한 여러 가지 분석을 제공한다. 우리는 이 도구를 사용하여 심볼 테이블을 제거한 상태로 컴파일한 바이너리와 원본 코드를 대조하여 악성 함수를 특정하고 라벨을 지정했다.

라벨을 지정한 악성 함수는 radare2 스크립트를 사용하여 opcode N-gram 시그니처를 추출했다. radare2 스크립트는 radare2의 기능을 코드로 작성하여 활용할 수 있는 지원이다. 이후 추출된 시그니처를 분석하여 공통적으로 나타나는 상위 33개 N-gram 패턴을 피처로 선정했다. 33개를 선정한 이유는 이 패턴들이 모든 악성 함수에 대해 1개 이상 포함되었기 때문이며, 그 이상부터는 포함되지 않는 함수가 많아진다. 피처의 개수는 악성 함수의 특성에 따라 변경될 수 있다.

Table 1. Malicious and normal dataset

Class		Count	Rate
Malicious	variant 1	20	0.02%
	variant 2	5	
Normal		109,079	99.98%
Total		109,104	100%

3.4 제안하는 모델

Fig. 5.는 제안하는 탐지 모델을 구축하는 과정을 보여준다. 그 과정은 다음과 같다.

1. 분석할 파일에서 바이너리를 함수 단위로 추출한다. 이때 바이너리 난독화 문제가 해결된 상태라고 가정한다.
2. N-gram 알고리즘을 사용하여 각 함수의 opcode 패턴을 추출한다.
3. 정적 분석을 수행하여 악성 함수에 라벨을 부여한다.
4. 수집한 악성 라벨 함수의 N-gram 패턴 중 상위 33개의 패턴을 피처로 사용하여 정상 함수의 N-gram 데이터를 추출한다. 피처는 사용하는 데이터셋에 따라 유동적이다.
5. 수집한 데이터셋을 사용하여 Random Forest 모델을 학습한다.
6. 과정 (5)에서 학습한 모델을 사용하여 Spectre 공격 함수가 포함된 바이너리를 식별한다.

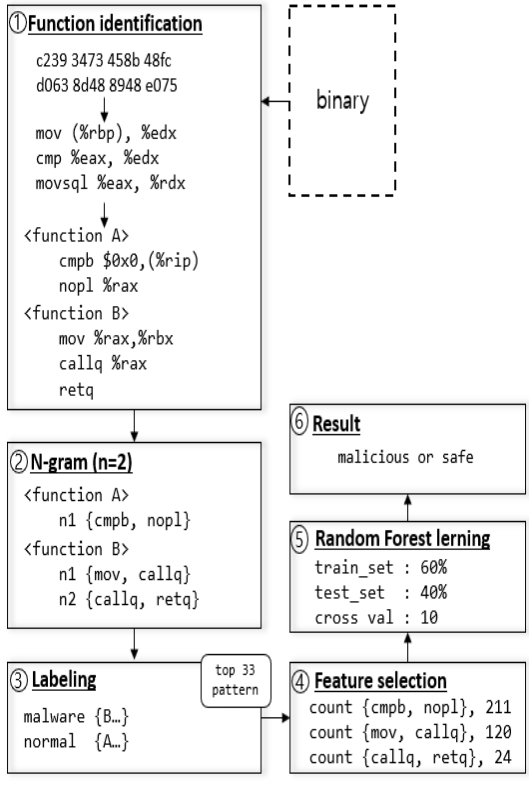


Fig. 5. Overview of the proposed system.

IV. 실 험

구성한 데이터셋을 바탕으로 모델을 학습하고 성능 평가를 수행했다. 이 과정에서 악성 함수 분류에 가장 적합한 모델을 찾고, 모델의 정확도를 높이기 위해 최적 n 선정 및 데이터 오버 샘플링 알고리즘을 적용했다. 모델 학습에 사용한 학습 데이터와 테스트 데이터의 비율은 6:4이다.

4.1 최적 모델 선택

최적 머신러닝 알고리즘을 선택하기 위해 SVM(Support Vector Machine), KNN(K-Nearest Neighbor), Random Forest 모델을 각각 학습한 뒤, 분류 성능 실험을 진행했다. 성능 실험 결과, Random Foreset 알고리즘을 사용한 모델이 99%의 정확도로 가장 좋은 성능을 보였다. 각 모델의 성능 지표 기반 AUC(Area Under Curve)는 Fig. 6.과 같다.

N-gram은 데이터셋의 특성과 n 값의 변화에 따라 많은 차이를 보인다[9]. 따라서 수집한 데이터셋에 적합한 n 값을 찾기 위해 n이 2, 3, 4인 경우에서 모델의 정확도를 측정했다. 이때 실험 데이터의 편중을 막기 위해 10번의 교차 검증을 수행했다. 실험 결과, n=2인 경우에서 precision과 recall이 각각 100%와 82%를 보이며 가장 좋은 결과를 보였다(Table 2). mov 및 add처럼 opcode는 짧은 sequence로 의미를 표현하기 때문에 낮은 n에서 우수한 성능을 보인 것으로 판단된다.

여기서 precision은 모델이 True라고 분류한 것 중에서 실제 라벨이 True인 비율이며, recall은 실

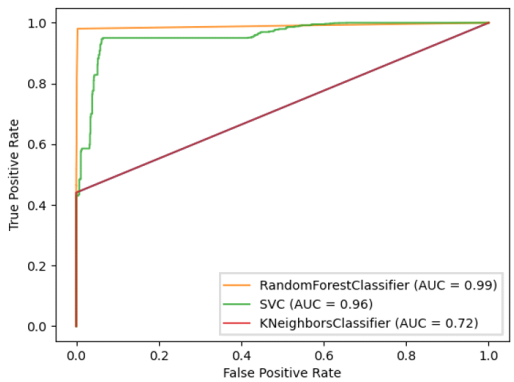


Fig. 6. Auc graph of models

Table 2. Confusion matrix according to n

N	Accuracy	Precision	Recall
n=2	99.98%	100%	82%
n=3	99.98%	89%	73%
n=4	99.98%	62%	45%

제 True 라벨들 중 모델이 True라고 예측한 비율이다. 이 결과는 n=2일 때, 테스트 데이터로 분류된 10개의 악성 함수 중 2개를 탐지하지 못했다는 것을 의미한다. 이러한 미탐지(false negative)가 발생하는 이유는 악성 함수의 개수가 부족하여 학습에 사용되지 못한 악성 함수가 테스트 데이터로 분류되었기 때문으로 보인다. 하지만 precision이 100%인 것으로 미루어 볼 때, 학습한 데이터에 대한 분류 정확도는 우수한 것으로 판단된다.

4.2 데이터 오버 샘플링

데이터셋의 불균형은 머신러닝 모델의 성능을 저하시킬 수 있다. 실험에 사용한 데이터 셋은 Table 1.과 같이 불균형 데이터이다. Random Forest 모델은 학습 과정에서 데이터의 클래스 간 분포를 고려하지 않기 때문에 데이터셋의 균형이 맞지 않으면 분류 성능에 악영향을 미칠 수 있다[19].

이 문제를 해결하기 위해 오버샘플링 알고리즘인 SMOTE(Synthetic Minority Over-sampling TEchnique)를 사용했다[20]. 오버샘플링은 데이터의 클래스간 불균형을 해결하기 위한 기법 중 하나이다. 불균형 데이터를 KNN 알고리즘을 통해 분석하고, 기존 클래스의 범위를 넘어서지 않는 새로운 데이터를 생성한다. 이 기법은 단순하게 데이터를 복제하는 것과 달리 생성된 데이터가 기존 클래스의 성향을 따르기 때문에 원본 데이터셋의 성질을 유지할 수 있다.

Table 3은 오버 샘플링을 적용하기 전과 후의 Random Forest 모델의 성능지표이다. 오버 샘플링을 적용한 후 Recall 지표가 10%만큼 크게 향상

Table 3. Confusion matrix after over-sampling

Over sampling	Accuracy	Recall	F1-score
X	99.98%	82%	90%
O	**99.99%**	**92%**	**96%**

되었으며, 그에 따라 F1-score도 6% 향상되었다. 즉, 10개의 악성 함수 중 1개만 탐지하지 못했다. 이에 따라, Accuracy도 0.1% 향상되었다.

V. 한계점

본 논문에서 제안하는 것은 동적 분석이 활용되기 어려운 파일 분석 분야에 활용될 수 있는 Spectre 바이너리 탐지 시스템이다. 이 시스템이 실질적으로 활용되기 위해서는 더 다양한 Spectre 변종을 탐지할 수 있어야 한다. 현재는 Spectre 유형 1과 2에 대해서만 탐지가 가능하다. 따라서 더 다양한 변종 코드를 학습시켜야 한다.

또한 일부 Spectre 코드는 핵심 공격 함수가 여러 함수로 분할되어 있는 경우가 있다. 이러한 경우에 분할된 각 함수를 악성으로 탐지할 것인지에 대한 연구가 필요하다. 마지막으로, 난독화된 바이너리에 대한 대응이 마련되어야 한다. 제안하는 시스템은 radare2 역공학 프레임워크를 기반으로 동작하기 때문에 난독화된 바이너리는 정확히 공격 함수를 추출하는 것이 어렵다. 따라서 이 문제를 해결하기 위한 추가적인 연구가 수행되어야 한다.

VI. 결 론

본 논문은 짧은 바이너리를 효과적으로 탐지할 수 있는 함수 단위 N-gram 비교 기법을 제안했다. 제안한 모델의 정확도를 109,079개의 정상 함수와 25개의 Spectre 악성 함수를 통해 검증한 결과, 악성 함수의 바이너리를 99.99%의 정확도로 탐지할 수 있었다. 따라서 기존 AV 엔진과 같은 대규모 바이너리 검사 시스템과에 우리의 모델을 적용하면, Spectre 공격으로부터 시스템을 더 안전하게 보호할 수 있을 것으로 기대한다.

다만, 현재 모델은 바이너리 난독화를 비롯한 정적 분석 방지 기술이 적용된 Spectre 악성코드는 식별할 수 없다. 또한 지금까지 밝혀진 모든 Spectre 변종을 탐지하지는 못한다. 따라서 향후 연구로는 위 문제를 해결할 수 있는 새로운 기법을 개발하고, Spectre 공격뿐만 아니라 백도어나 overflow 공격 등에 취약한 바이너리도 탐지할 수 있도록 확장하고자 한다.

References

[1] P. Kocher, D. Genkin, D. Gruss, W. Hass, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz and Y. Yarom, "Spectre attacks: Exploiting speculative execution," 2019 IEEE Symposium on Security and Privacy (SP), pp. 1-19, May. 2019.

[2] Intel, "Intel® 64 and IA-32 Architectures Software Developer's Manual: Volume 3," https://www.intel.co.kr/

[3] J. Corbet, "Meltdown/Spectre mitigation for 4.15 and beyond," LWN.net, https://lwn.net/Articles/744287/

[4] M. Mushtaq, J. Bricq, M.K. Bhatti, A. Akram, V. Lapotre, G. Gogniat and P. Benoit, "WHISPER: A Tool for Run-Time Detection of Side-Channel Attacks," IEEE Access 8, pp. 83871-83900, May. 2020.

[5] G. Marco, B. Kopf, J.F. Morales, J. Reineke and A. Sanchez, "SPECTECTOR: Principled detection of speculative information flows," 2020 IEEE Symposium on Security and Privacy (SP), pp. 1-19, May. 2020.

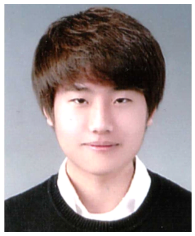
[6] P. F. Brown, V.J.D. Pietra, P.V. Desouza, J.C. Lai and R.L. Mercer, "Class-based n-gram models of natural language," Computational linguistics 18(4), pp. 467-480, Dec. 1992.

[7] I. Santos, Y.K. Penya, J. Devesa and P.G. Bringas, "N-grams-based File Signatures for Malware Detection," Proceedings of the 11th International Conference on Enterprise Information Systems(ICEIS), pp. 317-320, May. 2009.

[8] S. Jain and Y. K. Meena, "Byte level

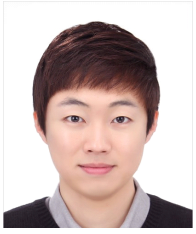
n-gram analysis for malware detection," nternational Conference on Information Processing, pp. 51-59, Aug. 2011.
[9] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. Mclean and C. Nicholas, "n investigation of byte n-gram features for malware classification," Journal of Computer Virology and Hacking Techniques 14.1, pp. 1-20, Sep. 2018.
[10] A. Pektas, M. Eris and T. Acarman, "Proposal of n-gram based algorithm for malware classification," The Fifth International Conference on Emerging Security Information, Systems and Technologies, pp. 7-13, Aug. 2011.
[11] B. Kang, S.Y. Yerima, K. Mclaughlin and S. Sezer, "N-opcode analysis for android malware classification and categorization," 2016 International conference on cyber security and protection of digital services (cyber security), pp. 1-7, Jun. 2016.
[12] A. Fog, "The Microarchitecture of Intel, AMD and VIA CPUs," May. 2017.
[13] Turner, Paul. "Retpoline: a software construct for preventing branch target injection," https://support. google. com/faqs/answer/7625886, 2018.
[14] Microsoft Visual C/C++ complier, "Qspectre," https://docs.microsoft.com/ko-kr/cpp/build/reference/qspectre?view=vs-2019
[15] P. Kocher, "Spectre Mitigations in Microsoft's C/C++ Compiler," https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html
[16] G. Wang, S. Chattopadhyay, I. Gotovchits, T. Mitra and A. Roychoudhury, "oo7: Low-overhead Defense against Spectre attacks via Program Analysis," IEEE Transactions on Software Engineering, pp. 1-1, Nov. 2019.
[17] N.A. Simakov, M.D. Innus, M.D. Jones, J.P. White, S.M. Gallo, R.L. Deleon and T.R. Furlani, "Effect of meltdown and spectre patches on the performance of HPC applications," arXiv preprint arXiv:1801.04329, Jan. 2018.
[18] Radare2, "radare2," https://rada.re/n/
[19] S. Ertekin, J. Huang, L. Bottou and C.L. Giles, "Learning on the border: active learning in imbalanced data classification," Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, pp. 127-136, Nov. 2007.
[20] N.V. Chawla, K.W. Bowyer, L.O. Hall and W.P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," Journal of artificial intelligence research 16, pp. 321-357, Jun. 2002.

〈저 자 소 개〉



김 문 선 (Moon-sun Kim) 학생회원
2020년 2월: 한남대학교 컴퓨터통신무인기술학과 졸업
2020년 3월~현재: 한남대학교 컴퓨터공학과 석사과정
〈관심분야〉 시스템 보안, 취약점 분석, 역공학, 악성코드 탐지

양 희 동 (Hee-Dong Yang) 학생회원
2019년 2월: 한남대학교 컴퓨터통신무인기술학과 졸업
2019년 3월~현재: 한남대학교 컴퓨터공학과 석사과정
〈관심분야〉 악성코드 탐지, 머신러닝, 모바일 보안, ARM TrustZone



김 광 준 (Kwang-Jun Kim) 학생회원
2017년 2월: 한남대학교 컴퓨터공학과 졸업
2019년 2월: 한남대학교 컴퓨터공학과 석사
2019년 3월~현재: 한남대학교 컴퓨터공학과 박사과정
〈관심분야〉 정보보호, 침입 탐지, 네트워크/시스템/공급망 보안

이 만 희 (Man-hee Lee) 종신회원
1995년 2월: 경북대학교 컴퓨터공학과 졸업
1997년 2월: 경북대학교 공학석사
2008년 8월: Texas A&M 대학교 컴퓨터공학과 공학박사
1997년~2003년: 한국과학기술정보연구원 연구원
2008년~2009년: Cisco Systems, San Jose
2010년~2012년: 국가보안기술연구소 선임연구원
2012년~현재: 한남대학교 부교수
〈관심분야〉 네트워크/시스템/스마트폰 보안, 고성능 시스템, 컴퓨터교육