*Journal of The Korea Institute of Information Security & Cryptology* ISSN 1598-3986(Print)
VOL.26, NO.2, Apr. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.2.377

# 악성코드 동적 분석을 위한 효율적인 다중실행경로 탐색방법*

황 호,[1†] 문 대 성,[1,2‡] 김 익 균[2]
[1]과학기술연합대학원대학교(UST), [2]한국전자통신연구원 네트워크보안연구실

# Efficient Exploring Multiple Execution Path for Dynamic Malware Analysis*

Ho Hwang,[1] Daesung Moon,[1,2‡] Ikkun Kim[2]
[1]Korea University of Science and Technology(UST),
[2]Electronics and Communications Research Institute(ETRI)
Network Security Research Team

## 요 약

악성코드의 수가 기하급수적으로 증가함에 따라 악성코드의 행위를 고속으로 분석하는 기술이 절실히 요구되고 있다. 또한, 정적 분석을 방해하는 실행압축과 가상화 같은 지능화된 코드 난독화 기법이 대부분의 악성코드에 적용되어 악성코드 동적 분석에 관한 연구가 다양하게 진행되고 있다. 그러나 동적 분석은 조건에 따라 다른 행위를 하는 악성코드를 분석하는 데 어려움이 있으며, 이를 해결하기 위한 기존의 연구들은 분석 속도가 느리거나 분석환경을 준비하는 데 많은 자원을 소모하는 문제를 가지고 있다. 본 논문은 단일 분석환경에서 악성코드의 다중실행경로를 고속으로 탐색하는 방법을 제안한다. 제안한 방법은 다중실행경로 분석이 병렬적으로 실행되도록 파이프라인화 하였고, 실험을 통해 2-코어 환경에서 29%, 4-코어 환경에서 70%의 성능향상과 지연노드에 영향 받지 않는 고속 탐색이 가능함을 보였다.

## ABSTRACT

As the number of malware has been increased, it is necessary to analyze malware rapidly against cyber attack. Additionally, Dynamic malware analysis has been widely studied to overcome the limitation of static analysis such as packing and obfuscation, but still has a problem of exploring multiple execution path. Previous works for exploring multiple execution path have several problems that it requires much time to analyze and resource for preparing analysis environment. In this paper, we proposed efficient exploring approach for multiple execution path in a single analysis environment by pipelining processes and showed the improvement of speed by 29% in 2-core and 70% in 4-core through experiment.

**Keywords:** Malware, Dynamic Malware Analysis, Multiple Execution

Received(01. 18. 2016), Modified(1st: 03. 21. 2016 ), Accepted(04. 11. 2015)

* 본 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.B0101-15-1293,다중소스 데이터의 Long-term History 분석기반 사이버 표적공격 인지 및 추적 기술개발)

† 주저자, kcats@etri.re.kr
‡ 교신저자, daesung@etri.re.kr(Corresponding author)

## I. 서 론

악성코드는 컴퓨터 또는 시스템에서 악의적인 행위를 수행하는 명령어 집합을 의미하며, 목적에 따라 바이러스, 웜, 스파이웨어, 트로잔 등으로 분류될 수 있다[1]. 악성코드 제작자가 제우스 빌더(Zeus Builder)나 스파이아이(SpyEye)와 같은 악성코드 생성도구를 이용하면서 생성되는 악성코드의 수가 폭발적으로 증가하고 있는 실정이다. Ahnlab ASEC REPORT에 따르면, 2015년 10월 한 달간 약 1천 4백만 건의 악성코드가 탐지되었고, 약 6백만 개의 악성코드가 수집된 것으로 집계되었다[2]. 이처럼 폭발적으로 증가하는 악성코드에 대응하기 위해 고속으로 악성코드를 분석하는 기술이 점점 더 요구되어지고 있다.

악성코드 분석은 크게 정적 분석과 동적 분석으로 나눌 수 있다. 정적 분석은 디스어셈블러(disassembler)나 디컴파일러(decompiler)를 이용하여 악성코드의 실행 없이 코드영역을 역으로 분석하는 역공학(Reverse Engineering)으로, 악성코드의 실행경로와 구조를 파악할 수 있으며 악성코드를 직접 실행하지 않기 때문에 안전하게 분석할 수 있는 장점이 있다. 하지만 실행압축, 가상화와 같은 난독화 기법이 적용된 악성코드를 분석하는데 한계가 있다. 동적 분석은 디버거를 이용하거나 가상 머신과 같은 분석환경에서 직접 악성코드를 실행하면서 발생되는 행위를 토대로 분석하는 방법으로, 악성코드가 실제 실행되는 동안의 변화를 분석할 수 있는 장점이 있지만 악성코드의 모든 실행경로 중 일부만 분석 가능한 한계가 있다[3]. 즉, 동적 분석은 가상환경 우회 악성코드(VM evasion)나 3.20 사이버테러의 악성코드처럼 특정 시점에만 동작하는 악성코드(time bomb)와 같은 방아쇠 기반 행위(trigger based bahavior)를 하는 악성코드를 분석하지 못하는 한계를 가진다[4].

오늘날 약 80%의 악성코드에 실행압축, 가상화와 같은 지능화된 난독화 기법이 적용됨에 따라[5], 동적 분석에 의한 악성코드 연구가 활발하게 진행되고 있다[6-7]. 특히, 악성코드 동적 분석의 한계로 지적되는 방아쇠 기반 행위가 적용된 악성코드를 분석하기 위해 모든 실행경로를 동적으로 분석하는 연구가 다양하게 보고되고 있다[8-11,13]. 그러나 효율적인 악성코드 동적 분석을 위한 기존의 다중실행경로 탐색 방법들은 대부분 오랜 분석시간, 다수의 분석환경 그리고 많은 컴퓨팅 자원을 요구하기 때문에, 기하급수적으로 증가하는 지능화된 악성코드에 실시간으로 대응할 수가 없다.

본 논문에서는 악성코드 동적 분석을 효율적으로 수행하기 위해 악성코드의 실행 경로를 병렬로 탐색하는 방법을 제안하고 프로토타입을 구현했다. 제안한 방법은 단일 분석환경에서 악성코드의 실행경로를 탐색하기 때문에 자원의 낭비를 최소화할 수 있다. 또한, 파이프화를 통해 초기 분석 지연에 영향을 받지 않는다.

제안한 방법의 타당성 및 성능을 평가하기 위해서 가상환경을 우회하는 악성코드를 제작하여 사용하였으며, 실험을 통해 제안한 방법이 악성코드의 다중실행경로를 고속으로 정확하게 추출하는 것을 확인하였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 동적 분석을 위한 악성코드의 다중실행경로 탐색에 관한 기존 연구를 기술하고 문제점을 분석한다. 3장에서는 본 논문에서 제안한 방법의 프로토타입을 제시하고 구성요소의 기능과 역할을 정의한다. 4장에서는 실험에 대해 서술하고 5장에서 결과 및 향후 연구방향을 제시한다.

## II. 관련 연구

악성코드 동적 분석은 악성코드를 직접 실행하여 나타나는 행위를 분석하기 때문에, 악성코드에 적용되는 다양한 코드 난독화 기법에 상관없이 악성행위를 정확하게 분석할 수 있는 장점이 있다. 하지만 특정 조건에서만 악성행위를 하는 악성코드의 경우, 조건을 만족하지 않으면 악성행위를 하지 않기 때문에 반복 실행하더라도 동일한 실행경로만 중복으로 분석하는 단점을 가진다. 예를 들어 Fig. 1.과 같이 VMware 환경에서의 실행여부를 확인하는 악성코드를

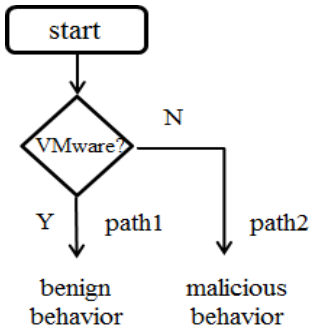



Fig. 1. VM Evasion Malware

VMware 환경에서 실행한다면, 경로 1(path 1)의 정상행위만 중복 분석하게 된다.

이처럼 악성코드가 조건에 따라 다른 행위를 나타낼 때, 동일한 실행경로만 반복 분석하는 문제를 해결하기 위해 악성코드의 모든 실행경로를 탐색할 수 있는 방법이 필요하다. 본 논문에서는 악성코드의 다중 실행경로 탐색에 대한 기존 연구를 분석환경의 개수와 테인트 분석(taint analysis)의 사용 여부를 기준으로 아래와 같이 3가지로 분류하였다.

- 다수 분석환경 기반 탐색
- 테인트 분석 기반 탐색
- 테인트 분석 기반 동적 분석환경 탐색

### 2.1 다수 분석환경 기반 탐색

Kirat[8]은 가상화(virtualization), 에뮬레이션(emulation), 하이퍼바이저(hypervisor), 그리고 실제 PC(bare-metal)와 같이 서로 다른 4가지 악성코드 분석환경에서 악성코드를 실행했다. bare-metal은 분석환경 내부에 감시도구를 설치하지 않는 것을 의미하므로 악성코드의 행위를 추출하기 위해 disk level과 network level의 행위 프로파일을 생성했다. disk level의 행위는 원격의 저장 공간에서 악성코드의 실행 전/후의 변화를 의미하고 network level의 행위는 악성코드가 보내는 패킷을 의미한다.

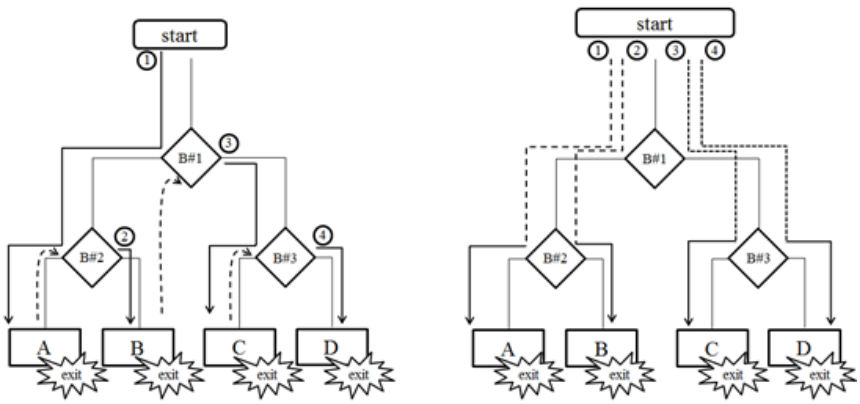
Lindorfer[9]는 모니터링 방법과 운영체제 이미지를 조합하여 4개의 분석환경에서 악성코드를 실행했다. 모니터링 방법은 anubis와 driver로 나눴고, 운영체제 이미지는 언어팩, jdk버전 등과 같은 소프트웨어를 다르게 설치하여 나눴다. 각 분석환경에서 악성코드가 사용되는 시스템콜을 모니터링하여 행위 프로파일(behavior profile)을 생성했다.

실행환경에 따라 다른 행위를 하는 악성코드를 분석하기 위해 Kirat[8]과 Lindorfer[9]는 다수의 분석환경을 준비한 후, 각 환경에서 악성코드를 실행했다.

### 2.2 테인트 분석 기반 탐색

Moser[10]와 Peng[11]은 단일 분석환경에서 테인트 분석을 통해 조건문에서 분기되는 모든 실행경로를 추적하는 방법을 제안했다. 테인트 분석은 정보의 흐름을 추적하는 방법이며, 두 논문에서는 탐색 경로의 수가 폭발적으로 증가하는 경로 폭발(path explosion)을 막는 용도로 사용되었다[12].

Fig.2.는 테인트 분석 기반 탐색에 관한 기존 연구를 설명한다. Fig.2.의 실행파일에서 3개의 분기(B#1 ~ B#3)가 존재하기 때문에 A부터 D까지 4가지의 실행경로를 가지며, ①부터 ④는 탐색 순서를 의미한다. Moser[10]는 분기를 만나면 스냅샷(snapshot)기능을 이용하여 분기에서 리소스들의 상태(CPU, 메모리 등)를 저장하고, 하나의 실행경로에 대한 수행이 끝나면 직전 분기로 복귀하여 실행하지 않은 경로를 탐색하는 방법을 제안했다. 즉, 실행경로를 탐색하다가 프로그램의 종료 명령(exit)을 만나게 되면 Fig.2.(a)의 점선과 같이 직전 스냅샷 정보를 복구(rollback)하여 다른 실행경로를 탐색하는




(a) Moser's Approach [10] (b) Peng's Approach [11]

Fig. 2. Sequential Exploring Multiple Exeuction Path

방법이다. 또한, 이 논문은 조건문에 의해 나뉜 두 경로를 모두 탐색하기 위해 solver를 사용하여 조건식의 참 값과 거짓 값을 찾았다. solver는 조건을 입력하면 해당 조건의 만족여부를 출력하는 프로시저로, 기호 실행(symbolic execution)에 주로 사용된다[12].

Peng[11]은 실행경로를 탐색하면서 만나는 분기의 위치를 저장하고 모든 실행경로를 탐색할 때까지 재실행하는 방법을 제안했다. 즉, Fig.2.(b) 와 같이 서로 다른 실행경로 A, B, C, D를 탐색하기 위해 4번 실행하는 방법이다. ①은 실행경로 A을 탐색하면서 만나는 분기 B#1과 분기 B#2의 위치를 저장하고 종료한다. 재실행되는 ②는 분기 B#2에서 ①과 다른 방향인 실행경로 B를 탐색하고 종료한다. ③은 분기 B#1에서 ①과 다른 방향인 실행경로 C을 탐색하면서 새롭게 만나는 분기 B#3을 저장하고 종료하며, ④가 실행경로 D를 탐색하면서 모든 실행경로 탐색을 마친다. 특히, Peng[11]은 분기처리를 solver로 계산하지 않고 Eflags 레지스터 값을 변경하는 플래그 변조(flag modification)를 하고, 메모리 충돌(memory crash)에 의한 비정상적인 종료를 해결하기 위해서 접근 위반(access violation)이 발생하면 즉각적으로 복구하였다.

하지만 Peng의 방법은 동일한 경로를 중복으로 분석하여 불필요한 시간낭비가 존재한다. 예를 들어, A와 B를 탐색하기 위해 Fig.2.(b)에서 ①과 ②이 시작 지점부터 B#2까지 점선으로 나타난 경로를 중복 실행하고, C와 D를 탐색하기 위해 ③과 ④이 시작 지점부터 B#3까지 점선으로 나타난 경로를 중복 실행한다.

Moser와 Peng의 방법은 solver의 사용에 따른 이율배반적(trade off) 관계에 있다. Moser는 solver를 사용하므로 현재 경로에 도착하기 위한 정보의 변화와 관계를 계산하여 정보를 정확하게 추적하고 관리할 수 있지만, 추가적인 저장공간이 필요하고 속도가 느리다. Feng의 Eflag레지스터 변경 방법은 추가적인 정보를 저장하지 않고 속도가 빠르지만 잘못된 실행경로에 도달 할 수 있는 문제가 있어 정확도가 떨어진다.

### 2.3 테인트 분석 기반 동적 분석환경 탐색

Xu[13]는 실행흐름을 테인트 분석으로 추적하며 악성코드에서 악성행위와 관련 있는 분석환경을 동적으로 생성하는 탐색방법을 제안했다. 예를 들어 동아시아의 사용자를 공격 목표로 하는 악성코드는 한국, 중국, 일본식 키보드의 사용여부에 따라 악성행위를 수행할 수 있다. 이와 같은 악성코드를 분석하기 위해 동아시아 국가(예, 한국, 중국, 일본)의 키보드를 사용하는 분석환경을 동적으로 생성하였다. 다수의 분석환경을 동적으로 생성하는 방법은 많은 자원이 필요하기 때문에, 이를 해결하기 위해 저장 공간의 임계치(Threshold)를 설정하여 조절하였다.

Table 1. limitation of Previous Researchs

| | # of analysis system | taint anaylsis | limitation |
|---|---|---|---|
| [8] | 4 | N | can't prepare analysis environment to trigger all malware |
| [9] | 4 | N | |
| [10] | 1 | Y | slow due to sequential exploring |
| [11] | 1 | Y | |
| [13] | N | Y | too many cost |

이상과 같이 다중실행경로 탐색은 다양한 연구가 진행되었으나 Table 1.과 같은 문제점을 가지고 있다. 다수 분석환경기반 탐색[8-9]은 모든 환경을 준비할 수 없기 때문에 방아쇠 기반 악성행위를 하는 악성코드에 완벽하게 대응할 수 없는 단점이 있다. 테인트 분석 기반 탐색[10-11]은 실행경로를 순차적으로 탐색하기 때문에 저속이다. 즉, Fig.2.(a)와 Fig.2.(b)의 실행경로 A의 탐색시간이 오래 걸릴수록 실행경로 B, C, D의 탐색시간이 지연되고, 이는 전체 탐색시간을 지연시킨다. 또한, 테인트 분석은 대상 프로그램의 모든 명령어를 추적하므로 대상 프로그램이 호출하는 API나 시스템 콜만을 후킹하는 일반적인 동적 분석보다 상대적으로 느리다. 마지막으로, 테인트 분석 기반 동적 분석환경 탐색[13]은 악성코드 분석에 필요한 실행환경을 지속적으로 만들어 많은 자원이 필요하다는 문제를 가진다. 이러한 기존 연구의 문제를 극복하기 위해 단일 분석환경에서 악성코드의 다중실행경로를 빠르게 탐색하는 기법이 필요하다.

## III. 병렬화를 통한 효율적인 다중실행경로 탐색 방법

본 논문에서 제안하는 다중실행경로 탐색방법은 단일 분석환경에서 다수의 실행경로를 병렬로 탐색하는 방법으로, 방아쇠 기반 악성코드의 실행경로를 단일

환경에서 고속으로 추출하는 것을 목적으로 한다.

방아쇠 기반 악성행위는 실행환경, 사용자 입력 등과 같은 외부 입력을 확인한 후, 조건을 만족하면 실행되는 악성행위를 의미한다. 외부 입력이 저장되는 변수의 변화를 테인트 분석을 통해 관찰하여, 해당 조건문이 외부입력과 연관 있는 지를 확인할 수 있다.

제안한 방법은 모든 방아쇠 기반 악성행위와 관련된 실행경로를 추적하기 위해 Table 2. 와 같이 악성코드가 외부 입력을 확인하는 명령어를 3가지로 분류했다. 이는 Péchoux[15]가 프로세스와 커널사이의 외부 상호작용을 시스템 콜(system call)로 정의한 것을 확장하여 명령어(instruction)를 기준으로 분류하였다. call, int 2E, sysenter과 같은 호출 명령어, FS세그먼트에 mov 등의 이동 명령어 그리고 in, out 등 기타 명령어로 나눈다.

외부 입력(external input)을 확인할 때, Brumley[16]와 같이 API를 후킹(hooking)하는 경우, API를 악성코드에 직접 삽입하는 인라인(inline) 처리 방식을 외부 입력으로 판단하지 못하는 단점이 있다. 예를 들어 isdebuggerpresent라는 API 함수는 PEB(Process Environment Block)의 BeingDebugged 변수를 확인하여 디버깅 여부를 판단한다. 만약 악성코드가 API 함수의 호출 없이 직접 BeingDebugged 변수에 접근하면 isdebugger-present 함수를 후킹하는 방법으로는 악성코드의 행위를 올바르게 분석할 수 없다. 이러한 문제점을 해결하기 위해, 본 논문에서는 Table 2.에서 분류한 것과 같이 호출 명령어를 추적하여 API의 사용을 확인할 뿐만 아니라 MOV 명령어를 추적하여 PEB에 직접 접근하는 행위도 확인할 수 있다.

본 논문에서 제안하는 다중실행경로 탐색 방법은 탐색기(Explorer)와 관리기(Manager)로 구성된다. 탐색기는 본 논문에서 정의한 외부 입력의 흐름을 추적하는 디버거다. 또한, 탐색기는 외부 입력에 영향을 받은 분기를 만나면, 해당 분기의 주소와 Eflags 레지스터의 값을 관리기에게 전송한다. 관리기는 탐색기로부터 받은 정보를 기반으로 새로운 실행

Table 2. External Input in Instructions Level

| category | instruction example |
|---|---|
| call | call, int 2E, sysenter |
| mov | mov with fs |
| etc | in, out, rdtsc, etc |

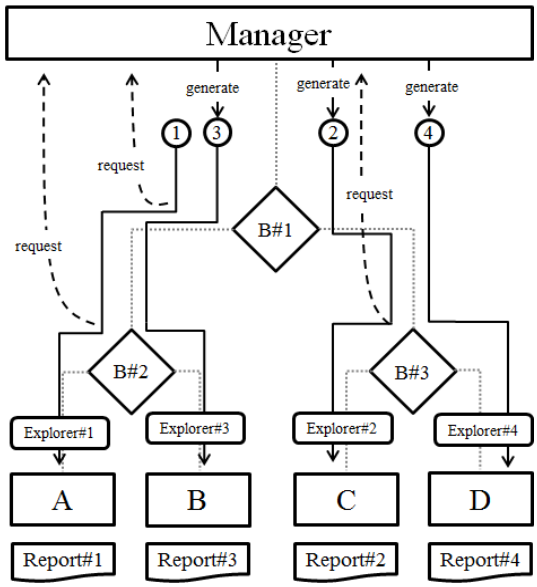



Fig. 3. Overview of Proposed Exploring

경로를 추적하는 탐색기를 생성한다. 이를 통해 두 개의 탐색기가 한 분기의 실행경로를 각각 병렬로 탐색할 수 있게 된다. 탐색이 완료되면 탐색기는 실행경로의 API 순서도 등 악성코드가 실행되는 동안에 발생하는 행위를 출력한다.

Fig.3. 은 본 논문에서 제안한 다중실행경로 탐색 방법을 나타내며, A부터 D까지의 네 가지 실행경로를 ①부터 ④의 순서로 탐색하는 것을 의미한다. 먼저, 탐색기#1(Explorer#1)이 실행경로 A를 탐색하면서 외부 입력에 영향을 받은 분기#1(B#1)을 만난다. 이때, Explorer#1은 관리기로 다양한 정보(분기의 위치, Eflags레지스터의 값)를 보내고, 관리기는 외부 입력에 영향을 받은 B#1에서 다른 실행경로 C를 탐색하는 새로운 탐색기를 생성한다. 이는 Explorer#1이 종료되기 전에 Explorer#2가 시작되어 동시에 탐색한다는 것을 의미한다. 나아가, Explorer#1은 B#2에서, Explorer#2는 B#3에서 관리기에게 새로운 탐색기 생성을 요청하여 실행경로 B, D를 Explorer#3과 Explorer#4가 탐색하도록 한다.

### 3.1 탐색기(Explorer)

탐색기는 외부 입력과 관련된 악성코드의 실행경로를 추적하면서 조건문을 만나면 새로운 탐색기의 생성을 관리기에게 요청하고, 해당 실행경로가 종료

될 때까지 발생되는 API나 system call의 호출 정보를 보고한다. 호출 정보는 순차적으로 발생되는 시스템 콜(system call)과 호출 명령어(call instruction) 그리고 이 둘의 주소로 구성되어 있으며, API의 경우 해당 함수의 이름을 나타낸다. 탐색기는 하나의 실행경로를 올바르게 탐색하기 위해서 외부 입력 판단, 테인트 분석, 분기처리의 3가지 기능을 수행한다.

먼저, 외부 입력 판단은 악성코드가 수행하는 명령어 중에서 외부 입력 명령어를 확인하는 과정이다. 앞서 언급된 Table 2.에서 정의한 것처럼, 악성코드가 호출 명령어를 사용하지 않고 MOV명령어로 FS 세그먼트에 접근하여 프로세스의 상태를 확인하는 경우에도 추적 가능하도록 하였다.

흐름추적은 테인트 분석을 활용하여 입력받은 정보의 흐름을 추적한다. 이를 통해 탐색기는 외부 입력과 연관있는 조건문을 판단할 수 있다. 또한, 탐색기는 변수의 오염여부를 저장한다. 예를 들어, B = A + 1 의 경우 B는 A보다 1이 크다는 관계를 저장하지 않고 A와 B가 외부 입력에 영향을 받았다는 정보만 저장한다.

마지막으로, 탐색기의 분기처리는 실행흐름을 제어하기 위한 방법이다. 실행흐름의 변경을 빠르게 하기 위해 solver로 계산하지 않고 Peng[11]과 같이 플래그 변조를 통해 실행흐름을 변경한다. 마이크로프로세서는 CF(Carry Flag), PF(Parity flag), ZF(Zero flag), SF(Sign Flag)의 Set(1)/Clear(0) 여부로 분기에서 실행흐름을 결정하고 이 값들을 가지고 있는 Eflags레지스터 값의 변경으로 실행흐름을 조작할 수 있다. 플래그 변조은 메모리 충돌이 발생할 수 있으며, 본 논문에서는 Peng[11]이 제안한 복구방법을 사용하여 해결하였다.

### 3.2 관리기(Manager)

관리기는 탐색기로부터 전달받은 분기위치 등의 다양한 정보를 토대로 새로운 탐색기를 생성한다. 외부 입력에 의한 분기에서 탐색기가 관리기에게 새로운 탐색기의 생성을 요청하면, 관리기는 요청한 탐색기를 복제하여 다른 실행경로를 탐색하는 새로운 탐색기를 생성하는 것이 이상적이다. 마치 유닉스 시스템에서 프로세스를 복제하는 포크(fork)와 같다. 하지만 본 논문의 대상이 되는 운영체제인 MS Windows는 내부적으로 프로세스 복제를 지원하지

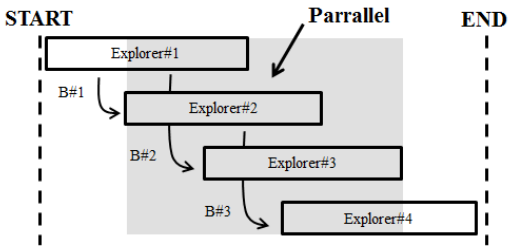



Fig. 4. Pipelining of Explorer for Parallel

않으므로 Fig. 4과 같이 파이프라인 방식으로 탐색하는 방법으로 구현하였다.

Fig.4.는 Fig.3.의 관리기가 분기 발생 위치를 받는 시점(B#1, B#2, B#3)에서 각 탐색기가 실행경로를 병렬적으로 탐색하는 것을 나타내며, 이를 통해 탐색완료 시간을 단축시킬 수 있다.

단일 코어 CPU에서 여러 탐색기들을 동시에 실행하면, 잦은 문맥교환(context switching)이 발생하여 순차적으로 탐색하는 방법보다 더 느려진다. 하지만 현재 대부분의 마이크로프로세서는 여러 개의 코어를 사용하고 있으므로 다중 마이크로프로세서 환경을 쉽게 구축하여 다중실행경로를 병렬로 탐색할 수 있다[14]. 따라서, 본 논문에서는 많은 실행경로를 병렬로 탐색하여 순차적으로 실행되는 방법보다 고속으로 실행이 가능하다.

## IV. 실험 및 분석

본 논문에서 제안한 탐색기와 관리기는 파이썬(python)으로 구현하였다. 특히, 탐색기는 동적 분석이 가능한 PyDbg라는 순수 파이썬 디버거를 이용하여, 디버거의 이벤트 핸들러가 단일 실행 이벤트(single step event)를 추적하면서 명령어를 하나씩 분석할 수 있도록 했다. 관리기는 탐색기와 별개의 프로세스이므로 multiprocessing의 queue를 이용하여 프로세스 들이 정보를 통신할 수 있도록 IPC(Inter Processing Communication)형태로 구현하였다.

실험 환경은 일반적으로 악성코드 동적 분석에 자주 사용하는 가상환경인 VMware에서 실험하였으며, CPU i5-2500, RAM 2GB, Windows 7 32bit로 구성하였다.

Table 3. Detection of Proposed method

| | object | detection |
|---|---|---|
| string | process name | X |
| API | drivers | O |
| | class_name | O |
| | registry | O |
| instruction | cpu_core | O |
| | cpuid | O |
| | vmx | O |

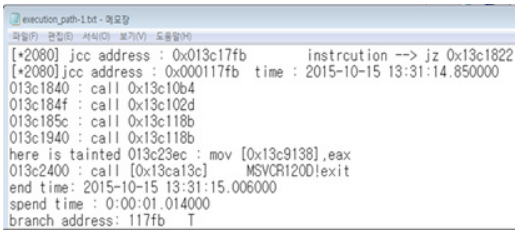
### 4.1 다중실행경로 탐색 기능

Table 3.은 다중실행경로 탐색여부를 확인하기 위해 Sudeep[17]이 정리한 8가지의 가상환경 우회방법 중 컴파일이 가능한 7개를 나타내며, 이를 본 논문에서는 크게 문자열 비교, API의 결과, 명령어 결과로 분류하였다. 테스트 파일은 Windows 7 Enterprise SP1에서 Visual Studio Express 2013으로 컴파일 되었다.

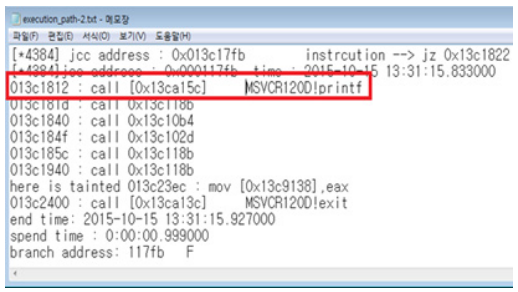
Table 3.의 결과를 통해 API와 명령어의 경우는 조건문의 반복횟수가 적어 탐색완료 시간이 짧지만, 문자열의 경우는 모든 프로세스의 이름을 문자 하나씩 비교하기 때문에 테스트 제한시간(10분) 내에 완료하지 못하는 문제가 있었다.

Fig.5.(a)는 앞서 언급한 7개의 우회방법 중 vmci.sys, vmxnet.sys 등과 같이 VMware와 VirtualBox의 guest OS에서 사용하는 드라이버의 존재여부를 판단하여 가상환경을 우회하는 코드이다.

Fig.5.(b)은 Fig.5.(a)의 IF문을 디버거(Ollydbg)를 이용하여 어셈블리 코드로 나타 낸 것이다. 왼쪽부터 순서대로 코드의 주소, opcode, 어셈블리어, 주석을 의미한다. 주소 0x13C17FB에 위치한 명령어 "JE SHORT 0x13C1822"는 드라이버




(a) Explorer#1




(b) Explorer#2

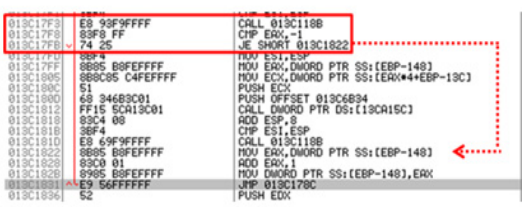
Fig. 6 Report of Explorers

의 존재 유무에 따라 다음 주소 0x13C17FD와 0x13C11822중 하나를 실행경로로 결정한다.

Fig.6.은 Fig.5.의 테스트 프로그램을 본 논문에서 제안한 방법으로 실행한 결과를 나타내며, Explorer#1과 Explorer#2의 호출 순서도를 보여준다. DLL내의 함수를 사용하는 경우에는 해당 DLL과 함수의 이름을 함께 보여주는 것을 확인할 수 있다. 각 결과의 마지막에 있는 branch address는 드라이버를 확인하는 함수에 의해 서로 다른 분기로 결과에 영향 받은 분기를 나타난다. T와 F를 통해 서로 다른 경로를 탐색했다는 것을 나타낸다.

Fig.6.(a)와 달리 Fig.6.(b)에는 printf함수가


(a) C code




(b) assembly code

Fig. 5. Detection Code of Guest OS's Drivers

나타난 것을 통해 제안한 방법이 테스트 프로그램의 방아쇠 기반 행위를 추출했다는 것을 알 수 있다. 두 결과를 비교했을 때 False의 경우만 0x013c1812에 printf 함수가 추가 된 것으로 Fig.5.(a)에서 드라이버의 존재를 확인했을 때 출력하는 부분의 추출을 확인할 수 있다.

Fig.6에서 나타난 결과를 통해 방아쇠 기반 행위를 하는 추출한 것을 확인한 것처럼 7개의 우회기법(process name, drivers, class_name, registry, cpu_core, cpuid, vmx)의 실행경로 탐색 여부를 Table 3.에 나타냈다. process_name을 제외하면 방아쇠 기반 행위가 적용된 악성코드의 다중실행경로를 탐색이 가능함을 보였다.

### 4.2 다중실행경로 탐색 속도

다중실행경로 탐색 속도는 조건문 처리방법과 실행경로 탐색방법에 영향을 받는다. Solver를 이용하여 조건문을 처리하는 방식은 Eflags 레지스터 값을 변경하는 방법보다 수행시간이 느리다. 이는 기하급수적으로 증가하는 악성코드에 대응하기 어려우며, Peng[11]은 Solver를 사용하지 않고 Eflags 레지스터 값을 변경하는 방법으로 조건문 처리방법에서 속도를 증가시켰다. 하지만 Peng[11]의 방법은 2.3절에서 언급한 것과 같이 실행경로를 순차적으로 실행하기 때문에 속도가 느리다. 제안한 방법은 실행경로 탐색을 속도 향상을 위해 파이프라인화 하였다. 실행경로 탐색 방법에 대해 기존연구와의 수행시간 비교실험에서 조건문 처리방법은 모두 Eflags 레지스터 값을 변경하는 방법으로 통일하였다.

Table 4.는 본 논문에서 제안한 다중실행경로 탐색 방법과 기존 연구의 수행시간을 결과이다. 실험에 사용된 프로그램은 64개의 노드를 가지는 포화 이진 트리로 구성되어 있으며, 각 실행경로의 길이는 동일하다. 즉, 이 실험은 Fig. 3. 에서 A, B, C, D와 같은 실행경로의 길이가 모두 동일한 대상 프로그램을 탐색하여, CPU 코어의 증가에 따른 실행경로 탐색방법의 속도를 비교했다.

Table 4. Efficiency of Proposed Exploring

| # of core | analysis time(sec) | | |
|---|---|---|---|
| | A. Moser | F. Peng | Proposed Method |
| 4 | 8.17 | 18.82 | 10.10 |
| 2 | 9.23 | 17.84 | 19.12 |
| 1 | 7.73 | 14.66 | 27.61 |

이 실험은 기존의 실행경로를 탐색하는 Moser[10]의 스냅샷 방식과 Peng[11]의 work list방식과 본 논문에서 제안한 방법을 pydbg로 구현 후 비교하였다. Table 4.에서 보는 바와 같이 코어가 1개인 분석환경에서는 제안한 방법이 27.60초로 가장 느리다. 하지만 2-코어에서는 19.12초로 성능이 약 29%증가했고, 4-코어에서는 10.1초로 성능이 약 64% 증가하여 Peng[11]의 방법보다 탐색 속도가 빠른 것을 알 수 있다. 8-코어에서는 제안한 방법뿐만 아니라 기존 방법에서도 속도 증가가 나타나지 않아 결과에서 제외하였다.

다양한 실험을 위해서, Table 4에서 수행한 64개의 포화이진 트리로 구성된 프로그램 이외에 다양한 상황을 가정한 추가실험을 수행하였다. Table 5는 이전 실험과 같이 64개의 노드를 가지는 프로그램에서 첫 노드(first node)와 마지막 노드(last node)에 각각 10초를 지연시키는 sleep함수를 추가하여 4-코어 환경에서 10번 실험한 결과의 평균이다. Table 5.에서 마지막 노드에 sleep함수가 추가된 경우는 이전 실험 결과와 동일하지만 첫 노드에 sleep 함수가 추가 된 경우에 제안한 방법이 약 1.71배 빠르다는 것을 알 수 있었다. 이는 기존 연구가 실행경로를 순차적으로 탐색하여 이전 분석의 종료를 기다리면서 발생되는 분석 지연이 전체 분석시간을 지연시키기 때문이다.

Table 5. Exploring time of different path

| position of sleep | analysis time(sec) | | |
|---|---|---|---|
| | A. Moser | F. Peng | Proposed Method |
| first node | **18.01** | 30.96 | **10.52** |
| last node | 17.16 | 31.04 | 19.91 |

## V. 결 론

본 논문에서는 악성코드의 효율적인 동적 분석을 위해 다중실행경로를 탐색하는 방법을 제안하고 프로토타입을 구현하였다. 제안한 방법은 악성코드의 외부 입력을 명령어 단위에서 추적하며, 병렬적인 다중

실행경로 탐색을 위해 파이프라인 방식을 사용하였다.

실험을 통해 제안한 방법이 API 호출뿐만 아니라 명령어와 FS레지스터를 외부 입력으로 보는 프로그램의 다중실행경로를 추출할 수 있음을 보였다. 또한, 본 논문에서 제안한 방법은 다수의 탐색기가 병렬로 실행되기 때문에 CPU의 코어 수가 증가함에 따라 경로의 탐색 속도가 빨라지는 것을 확인하였고, 병렬적으로 실행경로를 탐색하여 순차적인 기존 연구의 한계를 극복하였다.

본 논문에서 제안한 방법은 폭발적으로 증가하는 악성코드의 경로들을 단일 환경에서 병렬로 탐색하여 분석환경을 구성하는 자원의 소모가 적으며 속도가 빠르므로 실용적인 분석방법이 될 것이라 기대한다.

향후 연구에서는 제안한 방법을 문자열 비교방식과 중복된 경로를 실행하지 않도록 프로세스를 복제하는 방법을 구현하여 최적화 시킬 것이다. 또한, 정적분석이 완료된 실제 악성코드의 실행경로 분석을 통해 제안한 방법의 효율성을 실험할 계획이다.

## References


[1] E. Skoudis, L. Zeltser, *Malware: fighting malicious code*, Prentice Hall, Nov. 2003

[2] ASEC, "Asec report", vol. 70, Oct. 2015

[3] Boo-Joong Kang, Kyoung-Soo Han, Eul-Gyu Im, "Malicious code trends and detection technologies," communication of the korea I SCIENCE SOCIETY, 30(1), pp. 44-53, Jan. 2012

[4] NSHC, "3.20 South korea cyber attack, red alert research report", Mar. 2013

[5] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 40-45, Apr. 2007

[6] C. Willems,T Holz & F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Symposium on Security & Privacy*, vol. 5, no.2, pp 32-39, Mar. 2007

[7] K. Rieck, T. Holz, C. Willems, P. Düssel & P. Laskov, "Learning and classification of malware behavior", *In Detection of Intrusions and Malware and Vulnerability Assessment*, pp. 108-125, Jul. 2008

[8] D. Kirat, G Vigna and C Kruegel."Barecloud: bare-metal analysis-based evasive malware detection", *In Proceedings of the 23rd USENIX Security Symposium*, pp. 287-301, Aug. 2014

[9] M. Lindorfer, C Kolbitsch and P.M. Comparetti. "Detecting environment-sensitive malware," *In Recent Advances in Intrusion Detection*, pp. 338-357, Sep. 2011

[10] A. Moser, C. Kruegel and E Kirda, "Exploring multiple execution paths for malware analysis", *In Security and Privacy IEEE Symposium*, pp. 231-245, May. 2007.

[11] F. Peng, Z. Deng, X. Zhang, D. Xu, Z. Lin, Z and Z. Su, "X-force: Force-executing binary programs for security applications", *In Proceedings of the 2014 USENIX Security Symposium*, pp.829-844, Aug. 2014

[12] Byeong-ho Kang, Eul-Gyu Im. "Graph based Binary Code Execution Path Exploration Platform for Dynamic Symbolic Execution," Journal of THe Korea Institute of information Secuirty & Cryptology, 24(3), pp. 437-444, Jun. 2014

[13] Z. Xu, J. Zhang, G. Gu and Z. Lin "Goldeneye: efficiently and effectively unveiling malware's targeted environment" *In Research in Attacks, Intrusions and Defenses* , LNCS 8688, pp. 22-45, Sep. 2014

[14] D. Geer, Chip makers turn to multicore processors. *Computer*, vo. 38, no. 5, pp. 11-13. May. 2005

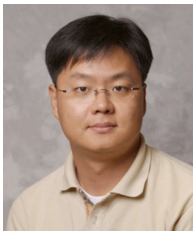
[15] R. Péchoux and T. D. Tam, "A Categorical Treatment of Malicious Behavioral Obfuscation," *In Theory and*

*Applications of Models of Computation*, LNCS 8402, pp. 280-299, Apr. 2014
〔16〕 D. Brumley,C. Hartwig,Z. Liang,J. Newsome,D. Song and H. Yin. "Towards automatically identifying trigger-based behavior in malware using symbolic execution and binary analysis," Technical Report CMU-CS-07-105, Carnegie Mellon University School of Computer Science, Jan. 2007
〔17〕 Sudeep Singh, "Breaking the Sandbox", Sep. 2014.

## 〈저자소개〉

황 호 (Ho Hwang) 학생회원
2014년 2월: 금오공과대학교 컴퓨터공학과 졸업
2014년 3월~현재: 과학기술연합대학원대학교 통합과정
〈관심분야〉 정보보호, 시스템보안, 네트워크보안

문 대 성 (Dae-Sung Moon) 정회원
1999년 2월: 인제대학교 전산학과 졸업
2001년 2월: 부산대학교 컴퓨터공학과 석사
2007년 2월: 고려대학교 전산학과 박사
2000년 12월~현재: 한국전자통신연구원 네트워크보안연구실 책임연구원
〈관심분야〉 네트워크 보안, 데이터마이닝, 영상처리, 지능형비디오감시, 바이오인식



김 익 균 (Kim, Ikkyun) 정회원
1994년 2월: 경북대학교 컴퓨터공학과 졸업(공학사)
1996년 2월: 경북대학교 컴퓨터공학과졸업(공학석사)
2009년 2월: 경북대학교 컴퓨터공학과 졸업(공학박사)
2004년~2005년 Purdue University 초빙 연구원.
1996년~현재 한국전자통신연구원 네트워크보안연구실 실장/책임연구원.
〈관심분야〉 네트워크 보안, 컴퓨터 네트워크, 클라우드보안, 빅데이터 분석