Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.34, NO.3, Jun. 2024 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2024.34.3.403

코드패치 및 하이브리드 분석 환경을 활용한 악성코드 데이터셋 추출 프레임워크 설계*

최 기 상,[1†] 최 상 훈,[2] 박 기 웅[3‡]
[1,2,3]세종대학교 (학생, 박사후 연구원, 교수)

Framework Design for Malware Dataset Extraction Using Code Patches in a Hybrid Analysis Environment*

Ki-Sang Choi,[1†] Sang-Hoon Choi,[2] Ki-Woong Park[3‡]
[1,2,3]Dept. of Computer and Information Security, Sejong University
(Undergraduate student, Researcher, Professor)

요 약

악성코드는 금전적인 목적에 의하여 서비스의 한 형태로 블랙마켓에 판매되고 있다. 판매에 따른 수요가 증가함에 따라 악성코드를 통한 공격이 확장되었다. 이에 대응하기 위해 인공지능을 활용한 탐지 및 분류 연구들이 등장하였지만, 공격자들은 분석을 방지하고자 다양한 안티 분석기술을 악성코드에 적용하고 있다. 본 논문에서는 안티 분석 기술이 적용된 악성코드들로부터 데이터셋을 확보하기 위해 하이브리드형 바이너리 분석 프레임워크 Malware Analysis with Dynamic Extraction(MADE)을 제안한다. MADE 프레임워크는 Anti-VM, Anti-Debugging이 적재된 바이너리를 포함하여 자동화된 동적 분석을 수행할 수 있다. MADE 프레임워크는 Anti-Analysis 기술이 적용된 다양한 악성코드들에 대해 90% 이상 우회가 가능하며, API 호출 정보를 포함한 데이터셋 추출이 가능함을 실험을 통해 검증하였다.

ABSTRACT

Malware is being commercialized and sold on the black market, primarily driven by financial incentives. With the increasing demand driven by these sales, the scope of attacks via malware has expanded. In response, there has been a surge in research efforts leveraging artificial intelligence for detection and classification. However, adversaries are integrating various anti-analysis techniques into their malware to thwart analytical efforts. In this study, we introduce the "Malware Analysis with Dynamic Extraction (MADE)" framework, a hybrid binary analysis tool devised to procure datasets from advanced malware incorporating Anti-Analysis techniques. The MADE framework has the proficiency to autonomously execute dynamic analysis on binaries, encompassing those laden with Anti-VM and Anti-Debugging defenses. Experimental results substantiate that the MADE framework can effectively circumvent over 90% of diverse malware implementations using Anti-Analysis techniques and can adeptly extract relevant datasets.

Keywords: Malware, Hypervisor, SandBox, Binanry Analysis, AI

Received(09. 13. 2023), Modified(1st: 01. 22. 2024, 2nd: 03. 14. 2024), Accepted(03. 28. 2024)

* 본 논문은 2023년도 한국정보보호학회 하계학술대회에서 발표한 우수논문을 개선 및 확장한 것임.

* 본 논문은 과학기술정보통신부의 재원으로 정보통신기획평가원(IITP)의 정보통신방송기술 국제공동연구(Project No. RS-2022-00165794, 40%), 국방ICT융합연구(Project No. 2022-0-00701, 20%), 실감콘텐츠핵심기술개발(ProjectNo. RS-2023-00228996, 10%), 정보통신방송혁신인재양성사업(Project No. 2021-0-01816, 10%) 및 한국연구재단(NRF) 중견후속연구사업(Project No. RS-2023-00208460, 20%)의 지원을 받아 수행된 연구임.

† 주저자, whaimpoonise@sju.ac.kr
‡ 교신저자, woongbak@sju.ac.kr(Corresponding author)

I. 서 론

정보화 시대에 따라 전자기기가 필수적으로 요구되면서 악성코드에 따른 피해사례가 잇따라 존재해왔다. 뿐만 아니라 악성코드 개발자들은 악성코드를 통하여 금전적인 목적을 달성하고자 블랙마켓에 판매해왔고 이는 하나의 서비스로 정착해나가면서 공격자의 범위가 전문가부터 일반인들까지 증가해 나갔다[1]. 공격자 범위가 증가함에 따라 고도화되고 다양화되고 있는 악성코드를 대응하고자 악성 분석 연구자들은 정적 분석과 동적 분석을 활용하여 악성코드에 대한 분석을 진행했다[2]. 악성 행위 정보들을 바탕으로 머신러닝이나 딥러닝와 같은 인공지능에 활용하여 악성코드 분류 및 탐지하는 방안에 사용되었다[3,4]. 하지만 이를 인지한 악성코드 개발자는 인공지능을 통한 탐지 및 분류를 저지하고 악성코드 분석가들의 분석 속도를 늦추고자 다수의 안티 디버깅을 악성코드에 첨부하였다[5]. 첨부된 안티 디버깅으로 인해 정적 분석 도구를 통해 추출한 데이터는 악성 행위 외의 데이터가 추출되었고, 동적 분석 도구를 통해 추출한 데이터 역시 샌드박스 환경임에 따라 악성 행위가 추출되지 않았다. 안티 디버깅에 따른 잘못된 구성은 추후 악성코드 탐지 및 분류 연구에 있어 성능을 저해할 것으로 사료되기에 무력화할 방안이 필요하다. 본 논문에서는 안티 디버깅 요소를 무력화하고 최대한 많은 악성 행위 정보들을 추출하고자 하이브리드 기반의 안티디버깅 우회 프레임워크를 설계하였다.

본 논문에서 제시한 프레임워크는 코드 난수화를 우회하기 위해 두 개의 샌드박스 환경과 하나의 베어메탈 환경으로 구성한다. 각각의 환경임에 따라 우회할 수 있는 요소들은 크게 샌드박스 환경을 탐지할 수 있는 Anti-VM과 디버깅 기술을 탐지하는 Anti-Debugging이 존재한다. 각각의 요소들을 우회하고자 커스텀된 OS 환경, Xen 기반의 가상화 기술 그리고 베어 메탈 환경을 사용한다. Anti-VM을 우회하는 샌드박스에서 우선적으로 동작하며, 만일 충분치 않은 데이터가 추출되지 않는다면 Xen 기반의 가상환경 또는 베어메탈 환경에 재배치하여 진행 및 데이터를 추출한다.

본 논문의 구성은 다음과 같다. 2장에서는 샌드박스 기반의 안티 디버깅 우회 프레임워크를 구성하기 위해 요구되는 Anti-VM과 Anti-Debugging 기술 및 우회 방안 관련 연구에 대해 기술한다. 3장에서는 우회 프레임워크를 구축하기 위해 Anti-VM과 Anti-Debugging을 분석하였으며, 이에 따른 회피 방안에 대해 설명한다. 4장에서는 본 논문에서 제안하는 하이브리드 기반의 안티 디버깅 우회 프레임워크를 소개하면서 환경에 대한 구성 방안을 설명한다. 5장에서는 설계한 프레임워크를 기반으로 실험한 결과, 각각의 환경에 따른 분석 가능 여부 및 시간에 대해 나열한다. 마지막으로 6장에서는 결론 및 향후 연구 계획에 대해 제시한다.

II. 관련 연구

정적분석 기술과 달리, 동적분석 기술은 샌드박스 환경 내부에서 악성코드를 실행하여 아티팩트 요소들을 수집하는 기술이다. 실질적으로 수행함에 따라 생성되는 아티팩트들을 여러 측면으로 수집할 수 있기에 샌드박스 환경 기반의 데이터셋 추출 프레임워크를 구축하였다. 하지만 Cuckoo Sandbox, Joe Sandbox와 같이 샌드박스 환경 기반의 동적분석 기술은 이전부터 활용되었고, 이를 저지하기 위한 Anti-VM 기술과 Anti-Debugging 기술이 등장하였다 [6-8]. 각각의 기술은 데이터 추출을 방해하는 요소이기에 무력화 대상이 되며, 본 장에서는 이를 달성하기 위한 관련 연구를 서술하고자 한다.

2.1 Anti-VM 기술에 대한 우회 방안 연구

본 절에서는 Anti-VM 기술 우회 방안 관련 연구에 대해 서술한다. Anti-VM 기술은 악성코드 제작자가 샌드박스 환경 내에서 분석을 방해하기 위한 목적으로 도입한 기술이다. 위와 같은 기술을 첨부하여 악성코드에 대한 정보들을 획득하기 어렵게 하며, 대처 및 분석 시간을 길게 한다. 이러한 불편함을 해소하고자 다음과 같이 Anti-VM 기술을 우회하기 위한 연구들이 등장하였다.

2.1.1 DBI 대상의 Anti-VM 기술 우회

악성코드 제작자들은 샌드박스 기반의 분석을 막고자 상용화 패킹 도구를 사용하여 Anti-VM 기술을 적용하였다. Anti-VM 기술을 회피하고 악성 행위를 추출하기 위해 참고 논문에서는 Themida, VMProtect, Enigma 외의 2개의 상용화 패킹 도구들 대상으로 분석을 진행한다. 상용화 도구들은 인

스트럭션과 API를 활용하여 아티팩트를 확인하였고 이를 토대로 샌드박스 환경임을 인지하였다 [9].

인스트럭션을 통해 탐지하는 방안으로는 IN 인스트럭션과 CPUID 인스트럭션이 있다. IN 인스트럭션은 통신 관련 정보를 반환한다. 이때 EAX 레지스터를 0x564D5868, DX 레지스터를 0x5658로 설정하게 된다면 샌드박스 환경에서는 EBX 레지스터를 0x564D5868 설정되고 관련 포트가 존재하지 않는 베어 메탈 환경에서는 예외 처리가 발생한다. 이와 같은 기술은 Themida의 상용화 도구에서 발견할 수 있었으며 비교 대상 문자열을 다른 값으로 설정하여 우회할 수 있었다. CPUID 인스트럭션은 하나의 물리적인 영역에서 여러 샌드박스 환경을 실행하는 데 필요한 하이퍼바이저를 탐지하기 위해 사용된다. EAX가 1일 때, EBX 레지스터의 31번째 비트가 1로 설정되어 있다면 하이퍼바이저 사용 중임을, 그렇지 않으면 하이퍼바이저가 사용 중에 있지 않음을 나타낸다. 이는 EBX 레지스터의 31번째 비트를 0으로 설정하여 우회할 수 있다. 레지스트리를 통해 샌드박스 환경임을 탐지하는 방안은 RegOpenKeyExA API를 이용하여 레지스트리 값들을 확인하는 것이다. 가상화 도구에 따라 첨부되는 레지스트리 키와 값이 달랐으며 메모리에서 탐색 대상의 문자열을 다른 값으로 치환하여 우회하였다.

그 외로 ENIGMA에서 process32next API를 이용하여 프로세스 목록들을 탐지하는 방안과 VMProtect에서 GetSystemFirmwareTable API를 활용하여 하드웨어 목록들을 탐지하는 방안이 있다. 탐지 대상 목록에서 가상화 도구 관련 문자열을 다른 값으로 치환하여 우회하였다.

2.1.2 Bare-Metal 기반의 Anti-VM 기술 우회

실제 하드웨어에서 파생된 샌드박스 환경은 Host 환경과의 통신 포트, 가상 CPU, 가상 메모리와 같은 기능을 운용하기 위해 가상의 디스크, 프로세스 등을 생성하거나 수행한다. Anti-VM이 첨부된 악성코드는 환경 설정 도중에 설정된 기타 환경 값들을 감지하고 악성 행위에 대한 분석을 저지한다. 환경 탐지 기반의 Anti-VM을 우회하기 위해 물리적 하드웨어 기반인 Bare-Metal 기반의 악성코드 분석 환경 설정 방안이 제시되었다 [10].

샌드박스 환경이나 에뮬레이션의 도움 없이 Bare-Metal 환경만으로 악성코드를 분석하기 위해 분석할 때마다 초기 설정값으로 복원해야 하는 도전적인 요소와 악성 행위 목록들에 대한 모니터링이 필수적으로 요구되었다. 참고 논문에서는 추가적인 디스크나 운영체제를 구현하여 하드디스크의 복원 및 메모리의 복원을 구현하고 SSDT(System Service Descriptor Table) 후킹을 통해 API 관련 정보들을 추출하였다.

2.2 Anti-Debugging 기술에 대한 우회

본 절에서는 Anti-Debugging 기술 우회 방안 관련 연구에 대해 서술한다. Anti-Debugging 기술은 실행파일에 대한 디버깅을 저지하기 위한 목적으로 도입한 기술이다. 프로세스를 디버거로부터 숨기거나 디버깅 과정을 탐지하여 일반적인 디버깅 방식으로부터의 추적을 방지한다. 분석 어려움에 따른 분석가의 피로도를 해소하고 지연 없이 진행하기 위해 Anti-Debugging 기술 탐색 및 우회하기 위한 연구들이 등장하였다.

2.2.1 코드 패치 및 후킹을 통한 Anti-Debugging 기술 우회

Anti-Debugging 기술은 프로그램을 보호하기 위한 목적으로 도입되었으나 악성코드 제작자들이 이를 악용하여 악성 행위에 따른 대응 및 대처를 늦추고자 악성코드에 적용하였다. Anti-Debugging 기술을 무력화하고자 여러 시도가 진행해왔고 참고 논문에서는 이와 같은 문제를 해결하고자 Anti-Debugging 기술 설명과 각 기술에 대한 우회 방안에 대해 나열하였다[11].

디버깅을 방해하는 API 함수로는 크게 IsDebuggerPresent, NtQueryInformationProcess, NtSetInformationThread 그리고 GetTickCount와 같은 여럿 함수가 존재한다. IsDebuggerPresent 함수는 PEB(Process Environment Block) 구조체로부터 Being Debugged 요소를 수집하여 판별한다. 해당 flag 요소가 1일 시 디버깅 중임을, 0일 시 디버깅을 진행하지 않음을 나타낸다. 이를 활용하여 해당 flag 값을 0으로 설정하여 우회한다. NtQueryInformationProcess 함수는 첫 번째 인자(ProcessHandle)에 지정된 프로세스에 대해 정보를 검색한다. 탐색하고자 하는 정보를 두 번째 인자(ProcessInformationClass)를 통해 지정할 수

있으며, 만일 0x7(ProcessDebugPort), 0x1E(ProcessDebugObjectHandle), 0x1F(ProcessDebugFlags)를 기입하게 된다면 디버깅 관련 정보를 산출한다. 각각의 프로세스들이 디버깅되고 있지 않으면 0을 산출한다는 특징을 활용하여 반환한 값을 0으로 설정하여 우회한다. NtSetInformationThread 함수는 주로 스레드의 우선순위를 설정할 때 사용된다. 두 번째 인자 내의 주어진 설정 값 중 0x11(ThreadHideFromDebugger)라는 값이 존재하는데, 해당 요소를 설정함에 따라 디버거는 해당 스레드로부터 이벤트 통지를 받을 수 없게 된다. 해당 함수는 후킹 혹은 코드 패치를 통해 두 번째 인자의 값을 0으로 설정하여 우회할 수 있다. GetTickCount 함수는 시스템이 시작된 후 경과된 시간을 반환한다. 시간 관련 함수들을 통해 비정상적인 시간 간격이 탐지된다면 디버깅되고 있음을 나타낸다. 이는 시간 함수 호출 과정들을 nop으로 설정하거나 시간 관련 함수들의 반환값 변경을 통해 우회하는 방안이 존재한다.

이 외로 프로세스에서 ollydbg, x64dbg와 같이 알려진 디버거 프로세스 명칭을 찾거나 메모리 상에서의 0xCC(INT 3)을 탐색함에 따라 디버깅 요소를 발견한다. 이들은 각각 잘 알려진 명칭을 다른 명칭으로 변경하거나 하드웨어 브레이크포인트를 설정하여 우회할 수 있다.

2.2.2 Virtual Machine Introspection을 통한 Anti-Debugging 기술 우회

VMI(Virtual Machine Instropection)는 샌드박스 환경에 대해 외부 모니터링를 진행하는 도구로써 디스크, 메모리, 네트워크와 같은 샌드박스 환경의 상태를 수집한다. 모니터링을 통해 수집한 데이터는 샌드박스 환경 내의 프로세스에서 이루어지는 행위를 분석할 수 있기에 악성코드 분석 영역에서도 활용할 수 있다 [12, 13]. VMI를 통한 디버깅이 샌드박스 환경에 영향을 미칠 경우, 악성 행위가 이를 탐지하고 실행되지 않지만 참고 논문은 샌드박스 환경에 영향을 미치지 않고 System Call과 메모리에서 추출할 수 있는 데이터를 수집하는 방안에 대해 제시했다 [14].

참고 논문에서 제시한 VMI는 Xen 기반으로 Monitoring VM이라고 명칭한 Dom0 가상환경과 Production VM이라고 명칭한 가상환경을 통해 이루어진다. Monitoring VM 내에서 추적 대상의 함수에 소프트웨 중단점(INT 3)을 삽입하여 중단점에 따라 발생하는 트랩마다 Production VM의 상태를 분석한다. 분석 대상으로는 System Call, 메모리 그리고 레지스터를 수집하여 유의미한 데이터로 재구성한다. 이후 중단점(INT 3)을 원래의 명령어로 실행하도록 구성함에 따라 재개한다. Monitoring VM을 통한 메모리 접근만이 이루어지기에 Production VM에 직접적인 영향을 행사하지 않은 채 분석할 수 있다. 이는 Production VM이 호출한 System Call에 대하여 순차적으로 수집할 수 있지만 트랩 갯수에 비례하여 분석시간이 증가하는 한계점을 가진다.

III. Anti-VM, Anti-Debugging에 대한 설명 및 우회 방법론

본 장에서는 Anti-VM과 Anti-Debugging에 대한 설명과 우회 방안을 설명한다. 본 논문에서 제안하는 프레임워크을 통해 샌드박스 환경과 베어 메탈 환경에서 추출할 수 있는 악성 행위 관련 정보들을 최대한 수집하는 것이 목적이다. 이와 같은 목적을 달성하기 위해 샌드박스 환경을 탐지하고 우회하는 기술인 Anti-VM 기술과 API 정보 수집을 방해하는 Anti-Debugging 기술을 회피해야 할 필요가 있었다. 두 가지의 우회 기술에 대한 분석 및 회피 방안은 크게 다음과 같다.

3.1 Anti-VM 기술 및 회피 방안

본 절에서는 Anti-VM 기술들에 대한 세부 항목과 항목에 따른 우회 방안에 대해 설명한다. 샌드박스 환경 내에서 악성코드를 실행하여 얻을 수 있는 정보들은 추후 같은 악성코드에 대하여 예방 및 변이에 대한 대처가 가능해진다. 하지만 악성코드 제작자들은 이러한 순작용을 방해하고자 Anti-VM 기술을 도입하여 저지하였다. 이와 같은 Anti-VM 기술을 분석한 결과, 크게 샌드박스 환경 요소 탐지, 사용자와의 상호 작용 그리고 분석시간 초과와 같은 기술들이 제시된다.

3.1.1 샌드박스 환경 요소 탐지

샌드박스 환경 요소 탐지는 실행시킨 파일에 대하

여 환경을 분석하고 실행을 저지하는 기법이다. Virtual Box, Vmware 그리고 Qemu와 같은 가상화 도구를 사용하여 샌드박스 환경을 이용할 시 가상화 도구 관련 아티팩트들이 드라이브, 레지스터리 등과 같은 경로에 남게 된다. Anti-VM이 첨부된 악성코드들은 여러 경로들을 순회하며 가상화 도구 관련하여 아티팩트 요소들을 탐지하는데, 이는 대표적으로 하드웨어 정보, 프로세스 정보 그리고 사용자와의 상호작용 정보가 있다.

3.1.2 하드웨어 정보

하드웨어 기반으로 동작하는 운영체제는 부팅 과정 중에 하드웨어 사양 정보를 레지스트리와 WMI(Window Management Instruction) 프로그램에 업데이트한다. 레지스트리는 윈도우 운영체제에서 시스템을 구성하는 데 필요한 정보를 저장하는 중앙 계층형 데이터베이스이고 WMI는 네트워크를 통해 관리정보를 확인하여 시스템이나 하드웨어 요소들을 저장하는 Microsoft 도구이다. Anti-VM 기술은 특정 경로의 레지스터리 값이나 WMI 프로그램에 제조업체를 조회하는 쿼리를 전송하여 샌드박스 환경임을 탐지하고 실행을 저지한다. 탐지 대상 요소로는 크게 펌웨어 정보, 디스크 드라이브 그리고 MAC 주소로 분류할 수 있다.

펌웨어 정보는 하드웨어에 내장된 소프트웨어 프로그램으로, 하드웨어의 동작을 제어하고 관리하기 위해 사용된다. 펌웨어 정보들은 컴퓨터 시스템의 하드웨어 및 소프트웨어 정보를 제공하는 SMBIOS(Standard Management BIOS)를 통해 확인할 수 있다. 레지스트리 경로 중 \\HKEY_LOCAL_MACHINE\\HARDWARE\\DESCRIPTION\\System\\BIOS 경로 내부 또는 \\HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\services\\mssmbios\\Data 경로의 SMBiosData 의 데이터를 통해 SMBIOS 정보들을 확인할 수 있다. 레지스트리 외의 관찰할 수 있는 방안으로는 WMI를 활용하는 방안이 있으며 이는 커맨드 창에 'WMIC PATH Win32_ComputerSystem Get * /Value'로 펌웨어 정보들을 확인할 수 있다. 샌드박스 환경을 활용하게 된다면 SMBIOS 하드웨어 정보에 가상화 도구 관련 명칭이 설정된다. 해당 명칭은 가상 머신을 관리하기 위한 명령줄 인터페이스 도구인 virsh를 활용하여 Fig. 1.과 같이 SMBIOS 설정 값을 Host 환경으로 반영하도록 변경한다면 우회할 수 있다.

```
<os>
  <type arch='x86_64' machine='pc-q35-4.2'>hvm</type>
  <boot dev='hd'/>
</os>
```

```
<os>
  <type arch='x86_64' machine='pc-q35-4.2'>hvm</type>
  <boot dev='hd'/>
  <smbios mode="host"/>
</os>
```

Fig. 1. Bypass SMBIOS by reflecting Host

디스크 드라이브는 사용 대상의 운영체제를 부팅하고 저장 공간으로 활용하기 위해 요구된다. 디스크 드라이브 정보는 운영체제 초기 세팅 과정 중에 HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Enum\\ 경로 중 하위 폴더 하에 키와 값이 설정된다. 추후 관련 아티팩트 삭제 목적으로 드라이브를 해제하여도 레지스트리 경로에 키와 값이 잔존하다. 이와 같은 특징으로 운영체제 초기 부팅 과정 이전에 가상화 도구 명이 담긴 드라이브를 제거하거나 드라이브의 디스크 버스를 SCSI로 설정하고 Fig. 2.와 같이 드라이브 명칭을 가상화 도구 외의 명칭으로 설정하여 우회한다.

MAC 주소는 네트워크 인터페이스에 할당된 고유 식별자이다. 장치 식별, 네트워크 관리 외의 여러 기능을 수행하기 위해 필수적으로 요구된다. 가상화 도구 별로 부여하는 명칭이 동일하듯이 MAC 주소 또한 지원하는 가상화 도구별로 고정적이다. Anti-VM이 첨부된 악성코드는 가상화 도구 관련 MAC 주소를 탐지하고 실행을 저지하여 정보들이 추출되지 않게 설정한다. 가상화 도구별로 지원하는 MAC 주소는 Table 1. 과 같으며 MAC 주소 기반의 Anti-VM을 우회하기 위해 가상화 도구들이 사용하

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2'/>
  <source file='/home/unweather/Desktop/qemu/img/win7_test.img'/>
  <target dev='sdc' bus='scsi'/>
  <boot order='2'/>
  <address type='drive' controller='0' bus='0' target='0' unit='2'/>
</disk>
```

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2'/>
  <source file='/home/unweather/Desktop/qemu/img/win7_test.img'/>
  <target dev='sdc' bus='scsi'/>
  <serial>WT1239468</serial>
  <vendor>WEATR</vendor>
  <product>WEATHER</product>
  <boot order='2'/>
  <address type='drive' controller='0' bus='0' target='0' unit='2'/>
</disk>
```

Fig. 2. Bypass virtualization-related Disk Name

Table 1. MAC address according to Virtualized Environment

Environment	MAC
Qemu	52:54:00:XX:XX:XX

지 않은 MAC 주소로 변경하면 우회할 수 있다.

3.1.3 프로세스 정보 및 가상화 도구 지원 서비스

가상화 도구들은 사용자에게 추가적인 편리성을 제공하기 위해 추가적인 프로세스를 샌드박스 환경 내부에서 수행한다. Anti-VM이 첨부된 악성코드들은 가상화 도구에 의해 수행되는 프로세스를 탐지하여 악성 행위를 중단하거나 정보들을 추출하지 못하도록 설정한다. 위와 같은 요소들은 대표적으로 vmtools.exe, 하이퍼바이저 그리고 백도어 포트와 같은 요소들이 있다.

vmtools.exe은 사용자에게 편리성을 제공하기 위해 제공하는 프로세스 중 하나이다. Host 환경과 Guest 환경 간의 통신, 화면 맞춤과 같은 추가적인 기능을 제공한다. 하지만 샌드박스 환경임을 나타내는 대표적인 아티팩트이기에 프로세스 항목에서의 탐색 대상이 된다. 하이퍼바이저는 하드웨어에서 시스템의 운영 체제와 리소스를 분리해 VM에 할당하여 가상 머신을 생성 관리 할 수 있도록 도와주는 소프트웨어이다. 이는 EAX 레지스터 값이 0x1 또는 0x40000000일 때 x86 아키텍처에서 지원하는 CPUID 인스트럭션을 통해 확인할 수 있다. CPUID 인스트럭션은 EAX 레지스터 값에 따라 프로세서에 대한 정보 또는 특징 정보를 반환하는데 0x1일 때 반환 값인 EBX 레지스터 값의 31번째 비트 값을 확인함으로써 유추할 수 있다. 만일 해당 비트 값이 1로 설정되어 있다면 해당 CPU가 하이퍼바이저를 사용하고 있음을 나타내고 0으로 설정되어 있다면 해당 CPU가 하이퍼바이저를 사용하고 있지 않음을 나타낸다. EAX 레지스터가 0x40000000일 때는 하이퍼바이저의 제조업체를 EBX, EDX 그리고 EBX에 저장하고 가상화 도구명과 비교함으로써 탐지할 수 있다. 단순 하이퍼바이저의 사용 상태를 해제함으로써 우회할 수 있지만 오베헤드에 따른 속도 저하와 같은 제약 조건이 발생한다.

Backdoor Port는 Host 환경과 Guest 환경 간의 통신을 위해 존재하는 포트로, x86 아키텍처에서 사용되는 IN 인스트럭션을 통해 확인할 수 있다. 샌드박스 환경임을 탐지하기 위한 조건으로 I/O 디바이스의 목적지인 EAX 레지스터 값에 0x564D5868을, I/O 디바이스의 포트 번호를 의미하는 DX 레지스터 값에 0x5658을 기입한 채로 in eax,dx 어셈블리어 코드를 수행한다. 만일 샌드박스 환경에서 수행하게 된다면 백도어 포트가 존재하여 수행하게 되고 ECX 레지스터 값에 샌드박스 환경의 버전 정보 그리고 EBX 레지스터 값에 매직 넘버인 0x564D5868 이 남게 된다. 만일 베어 메탈 환경이라면 포트로부터의 응답이 없어 예외를 수행함에 따라 정상적인 행위를 실행한다.

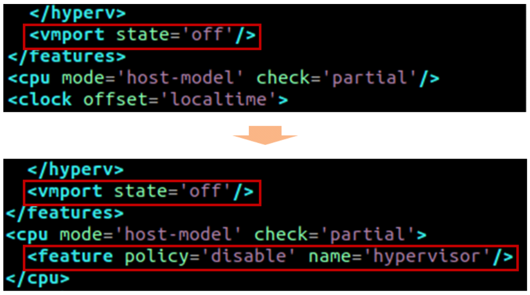

Fig. 3. Bypass Backdoor Port and Hypervisor

백도어 포트를 우회하기 위한 방안으로는 악성코드 내부의 EBX 레지스터 비교 값을 다른 값으로 변경하여 우회할 수 있다. 이와 같이 바이너리에 대해 코드 패치를 진행함에 따라 우회할 수 있지만 코드 패치에 따른 무결성 침해와 코드 패치 자동화의 어려움이 존재한다. 이에 악성코드를 수행하는 샌드박스 환경에서 지원하는 백도어 포트와 하이퍼바이저 지원 옵션을 Fig. 3.과 같이 해제하여 우회할 수 있다.

3.1.4 Reverse Turing Test

샌드박스 기술을 활용하여 샌드박스 환경 내부에서 악성코드를 자동적으로 분석하게 된다면 분석 대상의 악성코드는 사용자와 상호작용 없이 악성코드를 실행하게 된다. 만일 해당 악성코드가 마지막 입력값을 받아내는 API인 GetLastInputInfo를 활용하여 사용자로부터의 클릭이 있는지의 유무와 커서 위치 정보를 반환하는 API인 GetCursorPos를 활용하여 커서의 이동이 없는지 확인하는 Anti-VM 기술이 첨부되어 있다면 악성 행위가 추출되지 않을 것이다. 이를 우회하기 위해 랜덤하게 커서를 이동 및 클릭을 하도록 설정하였다.

3.2 Anti-Debugging 기술 및 우회 방안

본 절에서는 악성 행위의 추적을 방지하는 Anti-Debugging 기술을 설명한다. 악성코드의 악성 행위를 추적하기 위해 네트워크, 프로세스, API 그리고 메모리 등을 수집한다. 해당 정보들은 디버거 외로 DLL Injection, SSDT 후킹을 통해 획득할 수 있었으나 Anti-Debugging 기술을 첨부함에 따라 이를 회피하였다. Hooking 및 DLL Injection을 방지하기 위한 방안과 이에 따른 우회 방안은 다음과 같다.

3.2.1 DLL 로드 경로 확인

API 함수는 DLL과 같은 라이브러리 파일 안에서 호출되어 실행된다. 시스템 관련 API를 호출하는 DLL(Dynamic Link Library) 파일들은 C:\WINDOWS\system32와 같은 시스템 디렉토리나 \Device\HarddiskVolume\Windows\System32와 같은 시스템 디바이스 경로에 존재한다. 이와 같은 특성을 이용하여 DLL 파일들이 시스템 경로나 시스템 디바이스 경로에 존재하지 않으면 DLL Injection을 탐지한다.

3.2.2 API 내부 조각 확인

악성코드는 악성 행위를 수행하고자 할 때 DLL 파일 하에 존재하는 API 함수들을 호출하여 수행한다. 수행 대상의 API 함수들이 담긴 DLL 파일들은 메모리에 로드되는데 이를 Injection이나 hooking 기법을 사용하여 테이블 주소를 다른 주소로 변경하거나 내부 실행 코드를 다른 값으로 변경하여 API 함수들을 수집할 수 있다. 악성코드 개발자는 이러한 행위를 방지하고자 LoadLibraryA와 GetProcAddress를 사용하여 해당 API 주소에 접근한 다음 메모리 상의 올라온 API 내부 실행 코드가 실제 수행 대상의 코드와 동일한 지를 판단함으로써 후킹 및 Injection을 탐지한다.

3.2.3 Anti-Debugging에 대한 우회 방안

일반적으로 바이너리 패치 또는 메모리 패치를 통해 Anti-Debugging 기술을 우회할 수 있다. 하지만 새롭게 등장하는 Anti-Debugging 기술들에 대해 코드 패치하는 행위는 장기적인 관점에서 많은 시간과 노력이 요구된다. 이에 우리는 Xen 하이퍼바이저를 활용하여 API system call에 대해 추적하고자 한다. Xen 하이퍼바이저 기술은 전체 하드웨어를 완전히 가상화하는 전가상화와 달리 Fig. 4.와 같이 반가상화로 HyperCall을 호출하여 하이퍼바이저가 실행하도록 하는 기술이다. 이와 같은 호출 과정에서 Dom 0 이라는 특수한 가상머신을 통해 하드웨어에게 요청 및 수신하게 된다.

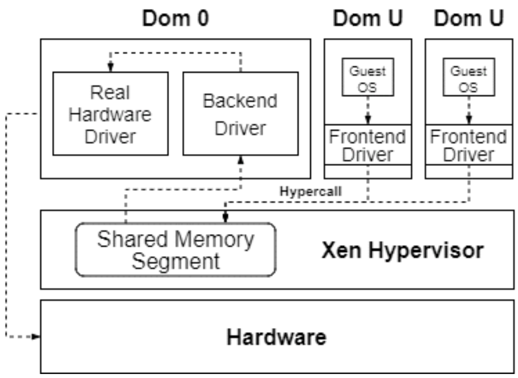

Fig. 4. Architecture of Xen

Guest OS가 동작하는 Dom U 외의 영역인 Dom 0을 모니터링하여 HyperCall을 통해 하드웨어에게 요청하는 행위만을 수집하게 된다면 해당 Guest OS 내에서 수행되는 System Call을 차례대로 수집할 수 있다. 이는 Windows System Call 영역에 소프트웨어 브레이크 포인트(INT 3)을 삽입함에 따라 달성할 수 있으며, 트랩 발생마다 Guest OS의 메모리, 레지스터를 분석함에 따라 여러 정보들을 추출할 수 있다. 해당 과정들은 하이퍼바이저 하에 이루어지기에 디버거 탐지와 변조된 dll 탐지로 구성된 Anti-Debugging 기술을 우회한다.

IV. Malware Analysis with Dynamic Extraction(MADE)

4.1 안티 디버깅 우회 프레임워크 구조

본 논문에서는 악성코드의 데이터셋을 확보하기 위해 하이브리드형 악성코드 분석 프레임워크인 Malware Analysis with Dynamic Extraction(MADE)을 제안한다.

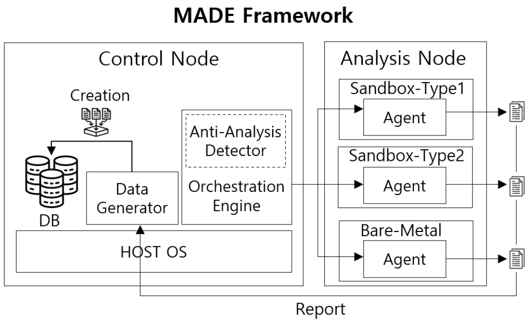

Fig. 5. Structure of the MADE Framework

MADE 프레임워크는 일반 바이너리뿐만 아니라, Anti-VM, Anti-Debugging이 적재된 바이너리를 포함하여 자동화된 동적 분석을 수행할 수 있다. 우리가 제안하는 MADE 프레임워크는 Fig. 5. 와 같다. 우리의 프레임워크는 고도화된 최신 악성코드로부터 효과적으로 데이터셋을 추출하기 위해 2개의 샌드박스 머신과 베어메탈 머신을 활용한 하이브리드 환경을 사용한다. MADE의 구조는 Control Node와 Analysis Node로 구성된다. Control Node는 악성코드 분석의 환경을 제어하기 위한 노드이다. Control Node는 Orchestration Engine을 통해 분석대상 환경을 동적으로 제어한다. 최초의 악성코드 분석은 Sandbox-Type1(S1)에서 수행되며, Anti-Analysis Detector는 S1의 분석결과를 분석하여 Anti-Analysis 요소를 판단한다. Data Generator는 Analysis Node로부터 전달받은 데이터를 분석에 활용 가능한 데이터로 재가공하여 DB에 저장하는 역할을 수행한다. Analysis Node는 악성코드를 동적으로 분석하기 위한 환경이다. 각 Analysis Node에는 Agent가 존재하며, Agent는 Control Node의 Orchestration Engine으로부터 악성코드 및 제어요소를 전달받고, Host OS에 분석결과를 전송하는 역할을 수행한다. Analysis Node에 대한 자세한 설명은 다음 장에서 설명한다.

4.2 Analysis Node (SandBox, Bare-Metal)

Analysis Node는 2개의 샌드박스 환경과 1개의 베어메탈 환경으로 구성된다. 샌드박스 환경은 Fig. 6.과 같다. S1의 경우, 악성코드의 API 호출 정보를 수집하기 위해 DLL Injection을 통한 API 후킹을 사용한다.

S1은 QEMU-KVM 환경에서 구동되며, Cuckoo Sandbox와 같이 Guest OS에 Agent 설치하여 악성코드를 분석한다 〔15〕. MADE는 Agent를 통해 악성코드를 실행하고, 해당 PID에 Hook DLL을 삽입시켜 API 호출 정보를 수집한다. 따라서, 대상 바이너리에서 호출하는 API를 집중적으로 수집할 수 있으므로 빠르고 정확하게 API 후킹이 가능하다. 하지만, Anti-Debugging 기법이 적용된 일부 악성코드의 경우 DLL Injection, Debugging Mode 등을 탐지하기 때문에 API 후킹이 불가능하다. 본 절에서 해당 요소를 고도화된 Anti-Debugging으로 판별하였고 관련 함수를 호출하는 악성코드를 대상으로 Sandbox-Type 2(S2) 환경에서 분석된다. 즉, S1 환경에서 분석이 실패한 악성코드의 경우 S2를 통해 분석이 수행된다.

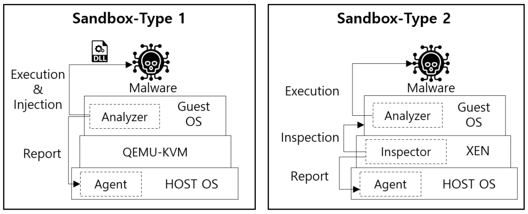

Fig. 6. Sandbox Environment for Dynamic Analysis of Malware

S2의 경우 VT-x 와 EPT를 지원하는 인텔 CPU를 활용하여 Guest OS 내에 구동되는 모든 프로세스에 대한 Introspection을 수행한다. 따라서, S1과 다르게 표적을 지정해두고 분석하는 것이 어려우므로 API 수집에 많은 시간이 소요되며, 후처리 연산이 필요하다. 하지만 Anti-Debugging 기법이 적용된 악성코드의 모든 API 호출 정보를 미시적으로 수집할 수 있다. S2의 악성코드를 분석하기 위한 구조는 다음과 같다. S2의 경우 Xen 환경에서 분석하기 때문에 Virtual Machine Monitor(VMM)을 통해 Guest OS에 대한 메모리 접근이 자유롭다는 특징이 있다. 따라서, Windows syscall 영역에 software breakpoint 트랩(INT 3)을 사용하여 syscall 호출 여부를 추적할 수 있다. Windows7의 syscall에 대한 심볼정보는 Volatility 프레임워크의 Windows 7 Profile 정보 확인을 통해 파악할 수 있다 〔16〕. S2의 경우 Guest OS 외부에서 Syscall을 추적하기 때문에 User-level에서 구동되는 Anti-Debugging 기법을 우회할 수 있다. 이와 같은 syscall 후킹은 커널모드에서 동작하는 루트킷

또한 syscall 추적이 가능하다. 다만, 모든 Windows에서 호출되는 모든 syscall이 트랩을 거쳐서 수행되기 때문에 분석 시간이 증가한다는 단점이 있다.

결과적으로, S1은 DLL Injection 기반의 API 후킹, S2는 하이퍼바이저 레벨에서의 API 후킹을 수행한다. Fig. 7.은 S1과 S2 환경에서 WannaCry를 분석하여 수집한 데이터셋의 일부이다. S1과 S2의 가상화 환경 및 API 후킹 방법이 상이하지만, 구조적 특성으로 인한 속성값을 제외한 Method, Process Name, ObejctAttributes, PID 등 모든 값이 정상적으로 추출되는 것을 확인할 수 있다. 우리가 설계한 S1, S2의 경우 커스텀된 OS를 사용하기 때문에 대부분의 Anti-VM에 대한 우회가 가능하다. 우리의 Sandbox 환경에서 우회할 수 있는 Anti-VM 기법은 3장 1절에 표기하였다.

MADE 프레임워크는 다양한 Anti-VM을 우회할 수 있지만, 고도화된 Anti-VM 기법을 우회할 수 없다. 본 절에서 말하는 고도화된 Anti-VM 기법은 전반적인 하드웨어 사양을 담고 있는 ACPI Table 내의 OEM ID와 같은 요소를 GetSystemFirmwareTable API를 통해 가상화 사용 여부를 판단하는 방식 등이 되겠다. 해당의 경우 베어메탈 머신을 통해 분석을 수행한다.

Bare-Metal 분석 환경은 하이퍼바이저를 사용하지 않기 때문에 모든 Anti-VM 기법에 대한 우회가 가능하다. Bare-Metal 기반의 분석환경은 Fig. 8.과 같다. Bare-Metal 환경에서는 DLL Injection을 통해 API 호출 정보를 수집한다.

수집 한 정보는 실시간으로 Control Node의 DB 서버에 전송된다. Sandbox 환경의 경우 Host OS로 API 호출 정보를 전송하기 때문에 데이터 유실에 대한 위험이 없지만, Bare-Metal의 경우 격리되지 않은 공간에서 악성코드를 실행시키기 때문에 데이터 유실을 방지하기 위해 외부 DB에 분석 데이터를 전송한다. Bare-Metal 환경의 한계점은 고도화된 Anti-Debugging 기법이 적용된 악성코드로부터 데이터 추출이 불가능하다는 것이다. Bare-Metal의 경우 PXE(Preboot Execution Environment)를 활용한다. PXE를 활용하면 네트워크를 통해 Bare-Metal 머신에 OS를 구동시킬 수 있다. 우리는 Bare-Metal 노드를 제어/관리하기 위해 OpenStack의 ironic의 iPXE 부팅 기능을 활용한다 [17, 18]. 악성코드 분석이 수행될 때마다 새롭게 OS를 로드하여 분석을 수행한다. 즉, Anti-VM 기법이 적용된 악성코드를 분석하기 위해 네트워크 부팅이 필요하므로 SandBox와 비교하여 분석에 많은 시간이 소요된다. 최근 서비스 동향의 경우 대부분 서비스가 클라우드 구동되고 있으므로, 공격자들은 표적을 클라우드 환경으로 확장할 것이다. 따라서, 향후 악성코드가 클라우드 환경에서도 구동되기 위해서는 Anti-VM에 대한 적용이 불가능할 것이다. 최근에 발견되는 대부분의 악성코드는 Anti-VM 기법이 적용되어있지 않다. 이와 같은 동향은 향후 S1, S2의 활용에 대한 가능성을 보여준다.

Fig. 7. Compare Analytics Data collected in S1 and S2 Environments

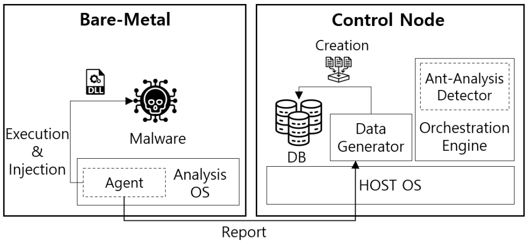

Fig. 8. Bare-Metal Environments for Bypassing Anti-VM

V. Evaluation

5.1 악성코드 분석을 위한 데이터 수집

MADE 프레임워크 대상으로 실험하기 위해 임의의 데이터셋을 수집한다. 비교적 육안으로 확인하기 쉬운 랜섬웨어를 대상으로 데이터셋을 수집하였으며, 중복을 방지하기 위해 데이터셋 간의 패밀리를 다르게 구성하였다. 실험한 악성코드 바이너리는 MalwareBazaar를 통해 수집하였다 [19]. 우리가

Table 2. Information of Malware

Malware Family	Sha256 Hash	MITRE-ATT&CK Technique	Comment
WannaCry	ed01ebfbc9eb5bbea545af4d01bf5f107166 1840480439c6e5babe8e080e41aa	T1497.003 (Anti-VM)	Employ various time-based methods to detect analysis environments.
Dharma	6b1f4df924fb0e5067df18dfc5063d409f3bf 2ee0d14b381b3f583e0d0da3ae5	T1497.003 (Anti-VM)	Employ various time-based methods to detect analysis environments.
TeslaCrypt	9b462800f1bef019d7ec00098682d3ea7fc6 0e6721555f616399228e4e3ad122	T1569 (Anti-VM)	Designed to be performed by ransomware in a win7 environment
DoejoCrypt	10bce0ff6597f347c3cca8363b7c81a8bff52 d2ff81245cd1e66a6e11aeb25da	T1569.002 (Anti-VM)	Designed to be performed by ransomware in a win7 environment
Jigsaw	3ae96f73d805e1d3995253db4d910300d8 442ea603737a1428b613061e7f61e7	T1497.003 (Anti-VM)	Employ various time-based methods to detect analysis environments.
Locky	bc98c8b22461a2c2631b2feec399208fdc4e cd1cd2229066c2f385caa958daa3	T1497.003 (Anti-VM)	Employ various time-based methods to detect analysis environments.
BlackBasta	17205c43189c22dfcb278f5cc45c2562f622 b0b6280dcd43cc1d3c274095eb90	T1497.001 (Anti-VM)	Check system flags and API's to detect code emulation or sandboxing
BlackMatter	99d3003cc577c3635bcbb883634037469b3 5c3b1a31109dc145600bb1e683d4b	T1622 (Anti-Debugging)	Designed to hide Threads From Debugger
Satana	683a09da219918258c58a7f61f7dc4161a3 a7a377cf82a31b840baabfb9a4a96	T1542.003 (Anti-VM)	Infect the MBR area to proceed with encryption on reboot
Petya	4c1dc737915d76b7ce579abddaba74ead6 fdb5b519a1ea45308b8c49b950655c	T1542.003 (Anti-VM)	Infected and rebooted MBR area to proceed with encryption
Thanos	58bfb9fa8889550d13f42473956dc2a7ec4f 3abb18fd3faeaa38089d513c171f	T1562.009 (Anti-Debugging)	Reboot to Safe Mode to disable debugging

Table 3. Experimental Results of Extracting Datasets by Malware

Malware Family	MADE Framework			Malware Family	MADE Framework		
	S1 (Time)	S2 (Time)	BM (Time)		S1 (Time)	S2 (Time)	BM (Time)
WannaCry	O (182s)	O (607s)	O (183s)	BlackBasta	X	O (606s)	X
Dharma	O (181s)	O (608s)	O (179s)	BlackMatter	X	O (607s)	X
TeslaCrypt	O (143s)	O (605s)	O (188s)	Satana	X	O (606s)	X
DoejoCrypt	△ (152s)	O (607s)	△ (161s)	Petya	X	△ (606s)	X
Jigsaw	△ (166s)	O (607s)	△ (127s)	Thanos	X	X	X
Locky	X	O (608s)	X				

수집하고 실험한 악성코드의 MITRE ATT&CK 기술 맵핑 정보는 Table 2.와 같다 〔20〕.

5.2 실험 환경

MADE 프레임워크의 성능을 평가한 환경은 Table 4.와 같다. 악성코드 분석 환경은 모두 Intel 기반의 CPU에서 수행되었으며, 샌드박스 환경의 경우 QEMU-KVM과 XEN을 활용하였다 〔21-23〕. 추가적으로 분석이 수행되는 OS는 모두 Windows7-x64를 사용하였다. 우리가 Windows7을 사용한 이유는 메모리 레이아웃이 분석되어 있어 API 후킹 적용이 가능하기 때문이다.

Table 4. Malware Analysis Performance Evaluation Environment

	Spec
CPU	Intel(R) Core(TM) i7-9700
MEM	64GB
HOST OS	Ubuntu 20.04 (5.15.0-83-generic)
HYPERVISOR	XEN (4.17.1)
	QEMU-KVM (4.2.1)
GUEST OS	Windows 7 x64 SP 1
GUEST CPU	2 Core
GUEST MEM	4GB

5.3 악성코드 분석을 통한 데이터셋 추출 평가

본 절에서는 우리가 제안한 MADE 프레임워크를 통해 분석한 악성코드의 범위에 대한 결과를 설명한다. DLL Injection을 활용하여 추출하는 S1과 S2의 경우 기본 분석 시간을 3분으로 선정하였고, Xen 하이퍼바이저를 활용하여 추출하는 방안은 10분으로 선정하여 분석하였다. 평가 방식으로는 분석 시간 동안 데이터셋 추출에 성공한 것은 'O', 비정상적인 중단으로 부분적으로 성공한 것은 '△' 그리고 암호화 행위 또는 패밀리에 해당하는 악성행위가 아닐 시 'X'로 판단하였다. 이와 같이 판단한 결과, Table 3.와 같은 결과가 나왔다. 총 11개의 악성코드 중 9개의 악성코드로부터 데이터셋을 추출하는데 성공하였고 1개의 Petya 악성코드는 부분적으로 데이터셋 수집이 가능하였다.

부분적으로 분석을 완료한 악성코드에 대해 이유를 분석한 결과, Petya는 MBR(Master Boot Record)을 공격하고 종료하는 악성코드이기에 종료 이전까지 분석이 가능하였고 DoejoCrypt와 Jigsaw는 각각 윈도우 버전에 따른 프로그램 부재와 시간적인 속임수로 인해 암호화 이후의 악성행위가 관찰되지 않았다. 분석에 실패한 악성코드는 Thanos로 Windows 7 환경에서의 안전모드로 재부팅하기에 분석이 불가능하였다.

5.4 Sandbox 분석 시간 비교 (S1 vs. S2)

본 절에서는 S1과 S2 환경으로부터 데이터셋을 추출하는데 소요되는 시간을 비교 측정한다. S1는 QEMU-KVM 기반의 샌드박스이며 DLL-Injection을 통한 API 후킹을 지원하는 분석 환경이다. S2는 XEN 기반의 샌드박스이며 Trap을 통한 API 후킹을 수행하는 분석 환경이다. 우리는 각각 100번 파일을 쓰는 실행파일과 1000번 파일을 쓰는 실행파일을 제작하였고, 동적 분석이 완료되기까지의 시간을 측정하였다. 우리가 분석 대상을 제작한 이유는 파일 쓰는 API 개수에 따라 소요되는 시간을 측정하기 위함이다. 실험결과는 Fig. 9.와 같다. 100번 쓰는 실행파일의 경우 S2가 S1에 비해 약 1.5배로 시간 지연이 발생하였고 1000번 쓰는 실행파일의 경우 S2가 S1에 비해 25배로 시간 지연이 발생하는 것을 확인할 수 있다. S2에서 분석 및 데이터 추출 시간 지연이 발생하는 이유는 하이퍼바이저 레벨에서 API 후킹을 수행하기 때문이다. S1

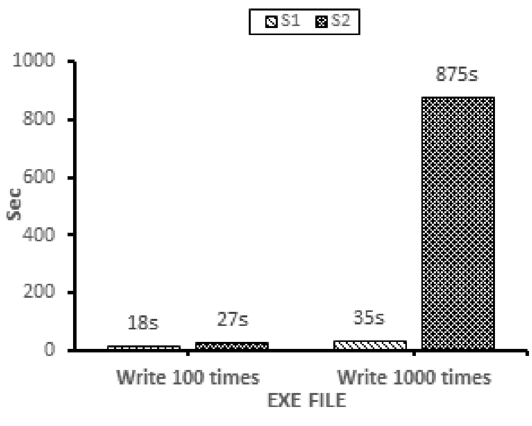

Fig. 9. Comparison of Analysis Time by Sandbox Environment

의 경우 DLL Injection을 통해 대상 프로세스를 지정하여, 집중적으로 API 후킹을 수행할 수 있는 반면에, S2의 경우 Guest OS에서 발생하는 모든 API를 트랩하기 때문에 분석 성능 지연 현상이 발생한다. 이러한 특징은 악성코드의 로직의 복잡성과 비례하여 분석시간이 증대하는 단점이 있다. 하지만, 테스트케이스 대상으로 MBR를 조작하거나 특정 환경에서만 수행하는 행위를 제외한 Anti-Debugging이 적재된 악성코드들을 S2 환경에서는 모두 우회가 가능하여 필수적이다. 따라서, 분석대상 악성코드의 Anti-Debugging 여부에 따라 분석 환경이 달라진다. Anti-Debugging이 없는 경우 대부분 S1 환경에서 분석할 수 있다. 우리가 제안한 MADE 구조는 악성코드 Anti-Analysis 요소에 따라 효율적으로 분석하기 위해 Orchestration Engine이 동적으로 분석 스케줄링을 관리한다.

VI. 결 론

본 논문에서는 Anti-Analysis 기술이 적용된 고도화된 악성코드들로부터 데이터셋을 확보하기 위해 하이브리드형 바이너리 분석 프레임워크 Malware Analysis with Dynamic Extraction(MADE)을 제안하였다. 우리는 Anti-Analysis 기술을 분류하고 안티 디버깅 우회 연구 사례를 분석하여 안티 디버깅 우회 프레임워크를 설계하였다. MADE 프레임워크는 Anti-VM, Anti-Debugging이 적재된 바이너리를 포함하여 자동화된 동적 분석을 수행할 수 있다.

우리는 MADE의 성능평가를 위해 일반 악성코드를 포함하여, Anti-Analysis가 적용된 11개의 악성코드를 분석하였다. 결과적으로, MADE 프레임워크를 통해 10개의 악성코드로부터 API 호출 정보를 포함한 다양한 데이터셋 정보를 추출하는 데 성공하였다. 우리가 제안한 MADE 프레임워크는 Windows 7에서 동작하는 악성코드만을 분석할 수 있다는 한계점이 있다. 게다가 MADE의 S2 분석 환경은 OS의 모든 API를 트랩하기 때문에 연산적으로 효율적이지 못하다는 단점이 존재한다.

향후 연구에서는 Windows 10을 포함하여, XEN 환경에서 악성코드들의 API 호출 정보를 연산 효율적으로 수집하여 데이터셋을 생성할 수 있는 연구를 수행할 예정이다.

References

〔1〕 Ki-woon Moon and Jong-Hyouk Lee, "The latest ransomware trends and developments," Journal of the Korea Institute of Information Security & Cryptology, 32(3), pp. 33-39, June. 2022.

〔2〕 Singh, Jagsir, and Jaswinder Singh, "A survey on machine learning-based malware detection in executable files," Journal of Systems Architecture, vol. 112, no.- , pp. 101861, Jan. 2021.

〔3〕 Pfeffer, Avi, and et al, "Artificial intelligence based malware analysis," arXiv preprint arXiv:1704.08716, vol. - , no. - ,pp. -, Apr. 2017.

〔4〕 Boydell, B. Mac, and M. Scanlon, "Deep learning at the shallow end :Malware classification for non-domain experts," Digital Investigation, vol. 26, no. -, pp. S118-S126, July. 2018.

〔5〕 Muralidharan, Trivikram, and et al, "File Packing from the Malware Perspective: Techniques, Analysis Approaches, and Directions for Enhancements," ACM Computing Surveys, vol. 55, no. 5, pp. 1-45, Dec. 2022.

〔6〕 Cuckoo Sandbox, https://www.cuckoosandbox.org, (accessed Sep. 11, 2023)

〔7〕 Joe Sandbox, https://www.joesandbox.com, (accessed Sep. 11, 2023)

〔8〕 Branco, Rodrigo Rubira and et el, "Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies," Black Hat, vol. 1, no.-, pp. 1-27, July. 2012.

〔9〕 Young Bi Lee, Jae Hyuk Suk, and Dong Hoon Lee. "Bypassing anti-analysis of commercial protector methods using DBI tools," IEEE

Access, vol.9, no.-, pp. 7655-7673, Jan. 2021.

〔10〕 Kirat, Dhilung, and et al, "Barebox: efficient malware analysis on bare-metal," Proceedings of the 27th Annual Computer Security Applications, pp. 403-412, Dec. 2011.

〔11〕 Saad, Muhammad, and Muhammad Taseer, "The Study of the Anti-Debugging Techniques and their Mitigations," International Journal for Electronic Crime Investigation, vol.6, no.3, pp. 33-44, Sep. 2022.

〔12〕 Rakotondravony, Noëlle, and Hans P. Reiser, "Visualizing and controlling vmi-based malware analysis in iaas cloud," IEEE 35th Symposium on Reliable Distributed Systems, pp.211-212, Sep. 2016.

〔13〕 Taubmann, Benjamin, and Hans P. Reiser. "Secure Architecture for VMI-based Dynamic Malware Analysis in the Cloud," Fast Abstract in the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. -, May. 2016.

〔14〕 Taubmann, Benjamin, and Hans P. Reiser. "Towards hypervisor support for enhancing the performance of virtual machine introspection," Distributed Applications and Interoperable Systems: 20th IFIP WG 6.1 International Conference, pp. 41-54, June. 2020.

〔15〕 Oktavianto, Digit, and Iqbal Muhardianto, Cuckoo malware analysis, Packt Publishing Ltd, Oct. 2013.

〔16〕 Volatility, https://www.volatilityfoundationorg, 2007, (accessed Sep. 11, 2023)

〔17〕 OpenStack, https://www.openstack.org, 2012, (accessed Sep. 11, 2023)

〔18〕 Kominos, Charalampos Gavriil and et al, "Bare-metal, virtual machines and containers in OpenStack," 2017 20th conference on innovations in clouds, internet and networks, pp. 36-43, Mar. 2017.

〔19〕 Malwarebazaar, https://bazaar.abuse.ch, (accessed Sep. 11, 2023)

〔20〕 MITRE ATT&CK®, https://attack.mitre.org/(accessed Mar. 1, 2024)

〔21〕 Barham, Paul, et al, "Xen and the art of virtualization," ACM SIGOPS operating systems review, vol. 37, no. 5, pp. 164-177, Oct. 2003

〔22〕 Kivity, Avi, and et al, "kvm: the Linux virtual machine monitor, " Proceedings of the Linux symposium, Vol. 1, No. 8, pp. 225-230, Aug. 2006.

〔23〕 Bellard and Fabrice. "QEMU, a fast and portable dynamic translator," USENIX annual technical conference, pp 10-5555, Apr. 2005.

〈저자소개〉

최 기 상 (Ki-Sang Choi) 학생회원
2021년 3월~현재: 세종대학교 정보보호학과 학사과정
〈관심분야〉 악성코드 분석, 시스템 보안

최 상 훈 (Sang-Hoon Choi) 정회원
2014년 3월: 대전대학교 정보보안학과 학사
2016년 3월: 대전대학교 전산정보보안학과 석사
2023년 3월: 세종대학교 정보보호학과 박사
2024년 3월~현재: 세종대학교 정보보호학과 연구교수
〈관심분야〉 하이퍼바이저, 시스템 모니터링, 시스템 메모리, 악성코드 분석, 딥러닝

박 기 웅 (Ki-Woong Park) 종신회원
연세대학교 Computer Science 학사
KAIST Electrical Engineering 석사
KAIST Electrical Engineering 박사
2009년 10월: Microsoft Research, Graduate Research Fellow
2012년 8월: 국가보안기술연구소 연구원
2016년 8월: 대전대학교 정보보안학과 교수
2016년 9월~현재: 세종대학교 정보보호학과 교수
〈관심분야〉 클라우드 시스템 보안, 초고속 보안 시스템, 시스템 인스펙션, 디지털포렌식