프로젝트 최종보고서

악성코드 내 Anti-VM 행위 무력화

K-Shield Jr. 보안사고 분석대응 과정

ㅇㅇ팀

ㅇㅇㅇ

ㅇㅇㅇ

유희영

ㅇㅇㅇ

ㅇㅇㅇ

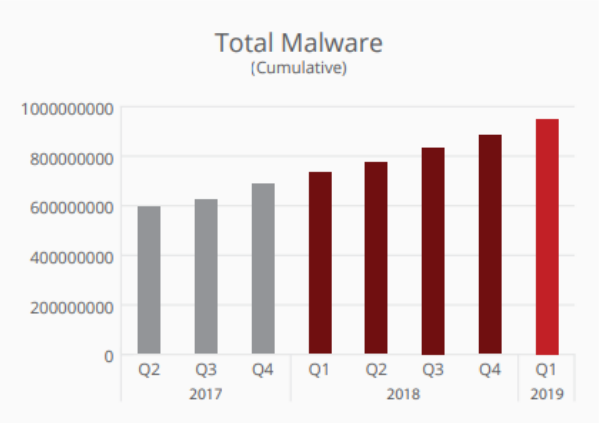
ㅇㅇㅇ

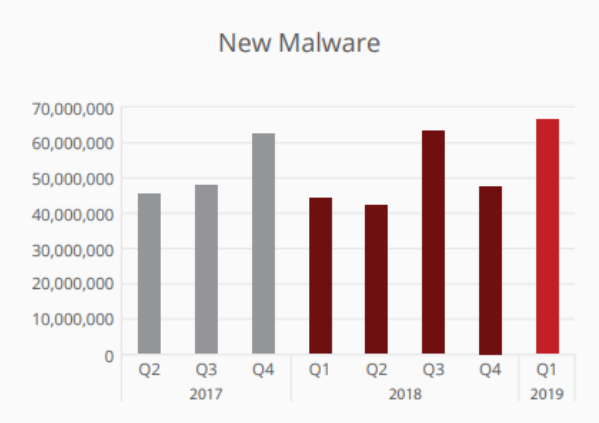
[여기에 입력][여기에 입력][여기에 입력]

1. 프로젝트 개요 및 필요성

1.1 개발 목적 및 필요성

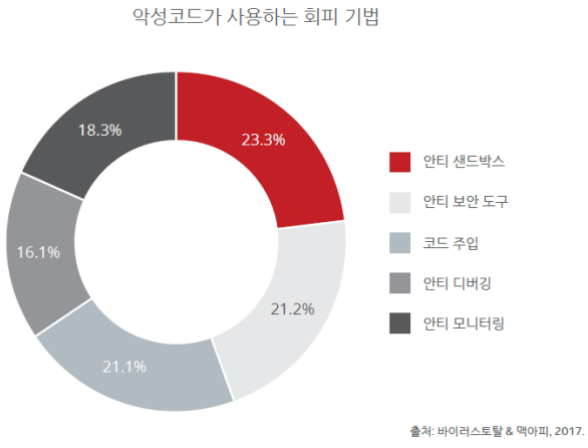
IT기술이 발전함에 따라 매년 악성코드는 증가한다. [그래프1-1]을 보면 총 악성코드 양 뿐만 아니라 새로운 악성 코드가 증가함을 알 수 있다.





[그래프 1-1] 악성코드의 증가량 및 신규 악성코드 생성량

이에 따라 악성코드 분석의 중요성이 높아졌다. 악성코드 분석은 정적분석과 동적분석으로 나뉜다. 정적분석은 악성 프로그램의 기능 파악을 위해 실행시키지 않고 코드나 프로그램의 구조를 분석한다. 악성코드는 정적분석에 대해 실행 압축기법이나 난독화 및 암호화 기술로 우회할 수 있다. 이를 보완하고자 악성코드를 직접 실행시켜 행위를 분석하는 동적 분석이 등장하였다.



[그래프 1-2] 악성코드가 사용하는 분석 회피 기법

가상 환경을 구축하여 동적분석하는 샌드박스 분석 툴을 사용하게 되면서, 가상환경을 우회하는 기능을 추가한 악성코드들이 등장하였다. McAfee와 VirusTotal이 발표한 [그래프 1-2]를 참조하면, 악성코드가 사용하는 회피기법 중 가장 비율이 높은 것은 Anti-Sandbox인 것을 볼 수 있다. Anti-Sandbox 활

용 비율이 높은 만큼, 가상 환경을 탐지 및 우회하여 동적분석을 방해하는 악성코드를 막을 필요성이 있다고 생각했다. 악성코드가 가상환경 탐지에 이용하는 가상환경과 관련된 요소를 확인하여 Anti-VM을 무력화하는 프로그램을 개발하고자 한다.

1.2 선행연구

가상환경을 우회하는 악성코드의 유형에는 크게 레지스트리 값 검색, 특정 명령어 사용, IDT(Interrupt Descriptor Table)의 메모리 위치, 메모리 문자열 검색으로 나눌 수 있다.

레지스트리 값 검색의 경우, 해당 환경의 레지스트리에 저장되어있는 키를 확인한다. 시스템/ 하드웨어/소프트웨어 레지스트리의 하위키에 저장되어 있는 키 값을 가상머신 식별에 사용한다. 가상환경을 특정하는 값이 있다면, 가상환경으로 인식한다.

특정 명령어 사용의 경우 2가지 방법이 존재한다. 첫번째는 인텔의 커널에서만 사용되는 명령어를 사용하는 방법으로 명령어에 대한 로컬환경의 반환값과 가상환경의 반환값이 다르다는 점을 이용한다. 두번째는 명령어의 실행 후 소요 시간을 비교하는 방법이다. 가상환경은 로컬환경보다 처리속도가 느리다는 점을 이용한다. 명령어 실행 소요시간이 일정 시간보다 길어지면 해당 환경을 가상환경으로 판단한다.

IDT(Interrupt Descriptor Table) 메모리는 인터럽트가 발생했을 때 인터럽트 핸들러의 벡터와 벡터정보를 저장하는 구조체이다. IDT 메모리는 가상환경과 로컬환경의 주소 위치가 다르다. 악성코드는 메모리의 주소가 특정 값보다 크면 가상환경으로, 작으면 로컬환경으로 판단한다.

메모리 문자열 검색의 경우, 가상머신의 메모리에서 가상환경과 관련된 문자열을 검색하고 해당 문자열이 있는지 확인하는 방법이다. 메모리에서 관련된 문자열이 확인되면 해당 환경을 가상환경이라고 판단한다.

1.3 유사 기술에 대한 차별성 및 독창성

악성코드의 가상환경 탐지와 관련된 오픈소스에는 pafish와 sems가 있다. 해당 오픈소스들은 실제 악성코드들이 가상환경을 탐지하기 위해서 사용하는 API나 레지스트리 값을 바탕으로 해당 환경을 검사하여 어떤 요소가 가상환경으로 판단되는 내용인지 나타낸다. 이 오픈소스들은 가상환경을 탐지한 결과를 보여줄 뿐, 이를 무력화 하지는 않는다.

본 프로젝트는 pafish나 sems와 유사하게 악성코드가 검사하는 레지스트리나 API들을 확인하고, 더 나아가 악성코드들이 해당 값을 검사할 때 가상 환경임을 인지할 수 없도록 악성코드의 가상환경 탐지를 무력화 하여 악성코드가 로컬환경이라고 판단하게 만든다.

2. 프로젝트 조직

2.1 프로젝트 팀원 구성 및 역할 분담

공통업무
API Hooking, 악성코드 수집, 악성코드 분류

	이름	역할	담당 업무
1	ㅇㅇㅇ	PM	프로젝트 총괄, 악성코드 분석, 발표, 보고서
2	ㅇㅇㅇ	팀원	QA/유지보수, Global Hooking, PPT
3	ㅇㅇㅇ	팀원	QA/유지보수, Global Hooking, 악성코드 분석
4	유희영	팀원	프로젝트 개발 총괄, Global Hooking
5	ㅇㅇㅇ	팀원	악성코드 분석, Cuckoo Agent 분석, PPT, 보고서
6	ㅇㅇㅇ	팀원	Cuckoo Sandbox 환경 구축, Global Hooking

[표 2-1] 프로젝트 팀원 구성

2.2 프로젝트 진행과정

단계		5월 4주차	6월 1주차	6월 2주차	6월 3주차	6월 4주차	7월 1주차
계획 및 분석							
설계	VM환경						
	프로세스 탐지						
	레지스트리 탐지						
	vm관련파일탐지						
	악성코드 분석						
개발	VM환경 변경						
	프로세스 탐지						
	레지스트리 탐지						

	vm관련파일탐지						
테스트	단위테스트						
	통합테스트						
	디버깅						
완료	개발 완료 보고서						
	최종보고서						

[표 2-2] 프로젝트 진행과정

2.3 프로젝트 환경

- 사용언어 : C, C++

구분	버전	비고
Host Vmware	Vmware Workstation 15.0.0	
Host OS	Ubuntu 18.04.4	
Guest Vmware	VirtualBox 5.2.42	
Guest OS	Windows7	
Sandbox	Cuckoo 2.0.7	
Python	Python 2.7.17	Cuckoo sandbox
Mongo DB	Mongo DB 2.7	Cuckoo sandbox
Tcpdump	Tcpdump 4.9.2-3	Cuckoo sandbox
Volatility	Volatility Framework 2.6	Cuckoo sandbox
M2Crypto	M2Crypto 0.30.1,typing 3.6.6	Cuckoo sandbox

[표 2-3] 프로젝트 환경 구성

3. 프로젝트 개발 진행 내용

3.1 악성코드 분석

악성코드에서 어떤 방식으로 가상환경을 탐지하는지 확인하기 위해 악성코드 분석을 진행하였다. [그림 3-1]에서 분석한 악성코드는 특정 프로세스를 찾기 위해서 CreateToolhelp32Snapshot 함수를 사용한다. 이 함수는 시스템의 어느 한 시점에 대한 스냅샷을 찍는다. 해당 시점의 프로세스 리스트에서 프로세스들의 이름을 하나씩 비교하여 가상환경과 관련된 프로세스가 있는지 확인한다. 해당 악성코드는 VboxService.exe 프로세스가 존재하는지 확인한다. 만일 해당 프로세스가 존재하면, 악성코드는 가상환경이라고 판단한다. 이러한 방식을 통해서 해당 환경에 가상환경에 존재하는 프로세스를 검사한다.

[그림 3-1] 프로세스 분석 결과 (Hash : 0c2f6aee0453d7b54ee713fae2b1befb)

또한, 메모리 상에 가상환경과 관련된 문자열들이 있는지 확인해보았다. 악성코드에서 가상환경을 탐지하기 위해서 메모리에 가상머신과 관련된 문자열을 올린다는 것을 활용했다. [그림 3-2]에서 분석한 악성코드에서는 'vmsrvc.exe', 'vmusrvc.exe', 'vboxservice.exe', 'vmwaretray.exe' 등의 문자열을 가상환경 탐지에 사용하고 있는 것을 볼 수 있었다.

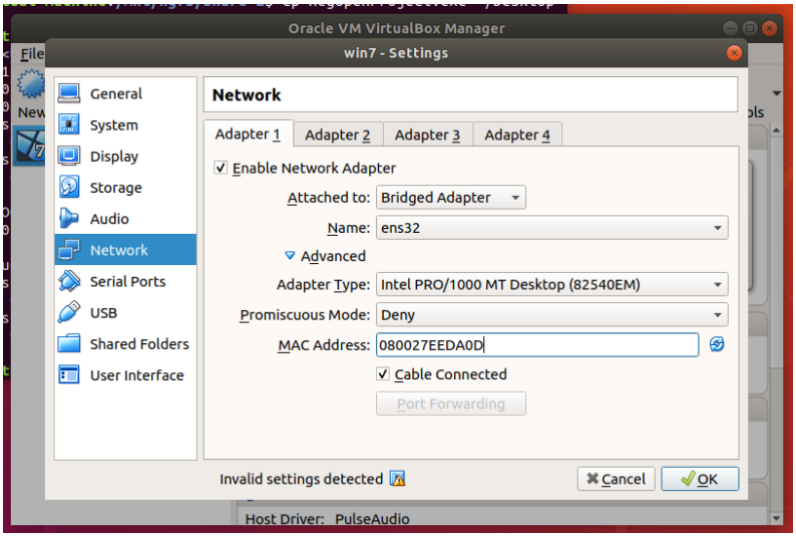
```
.rdata:00404C66                 align 4
.rdata:00404C68 aVmsrvcExe      db 'vmsrvc.exe',0       ; DATA XREF: sub_402EED+22↑o
.rdata:00404C73                 align 4
.rdata:00404C74 aVmusrvcExe     db 'vmusrvc.exe',0      ; DATA XREF: sub_402EED+29↑o
.rdata:00404C80 aXenserviceExe  db 'xenservice.exe',0   ; DATA XREF: sub_402EED+30↑o
.rdata:00404C8F                 align 10h
.rdata:00404C90 aVboxserviceExe db 'vboxservice.exe',0  ; DATA XREF: sub_402EED+37↑o
.rdata:00404CA0 aVboxtrayExe    db 'vboxtray.exe',0     ; DATA XREF: sub_402EED+3E↑o
.rdata:00404CAD                 align 10h
.rdata:00404CB0 aVboxcontrolExe db 'vboxcontrol.exe',0  ; DATA XREF: sub_402EED+45↑o
.rdata:00404CC0 aVmwareserviceE db 'vmwareservice.exe',0
.rdata:00404CC0                                         ; DATA XREF: sub_402EED+4C↑o
.rdata:00404CD2                 align 4
.rdata:00404CD4 aVmwaretrayExe  db 'vmwaretray.exe',0   ; DATA XREF: sub_402EED+53↑o
.rdata:00404CE3                 align 4
.rdata:00404CE4 aTpautoconnsvcE db 'tpautoconnsvc.exe',0
.rdata:00404CE4                                         ; DATA XREF: sub_402EED+5A↑o
.rdata:00404CF6                 align 4
.rdata:00404CF8 aVmtoolsdExe    db 'vmtoolsd.exe',0     ; DATA XREF: sub_402EED+61↑o
.rdata:00404D05                 align 4
.rdata:00404D08 aVmwareuserExe  db 'vmwareuser.exe',0   ; DATA XREF: sub_402EED+68↑o
.rdata:00404D17                 align 4
```

[그림 3-2] 메모리 문자열 분석 결과, (Hash : 4f6969b237a911d9be440baf21a90e56)

3.2 가상환경 탐지 요소

3.2.1 MAC

1. 대부분의 Virtual Machine 소프트웨어는 게스트 시스템에 고유한 MAC 주소를 할당한다. Virtual Box의 경우 일반적으로 08-00-27, VMWare는 00-50-56, 00-0C-29, 00-05-69로 시작하는 OUI(조직고유식별자)를 사용한다. 악성코드는 해당 MAC 주소인지를 확인하여 가상환경을 탐지할 수 있다.

2. 

[그림 3-3] 가상환경 MAC주소 변경되기 전

3.2.2 레지스트리

3. 시스템 / 하드웨어 / 소프트웨어 레지스트리 키의 하위키에 가상머신을 식별할 수 있는 정보가 남는다. 악성코드는 이 정보를 확인하여 가상환경임을 탐지하고 우회한다. 레지스트리 중 HKLM은 윈도우에서 사용하는 파일 시스템, 하드웨어 드라이버, 윈도우의 커널이 사용하는 정보 등 윈도우 하부 시스템에 관련된 다양한 설정 내용에 대한 정보가 저장된다.

시스템 정보는 HKEY_LOCAL_MACHINE₩₩SYSTEM 경로에 존재하며 시스템이 부팅될 때의 환경 설정 정보를 가지고 있다. 시스템이 정상적으로 부팅되었을 때 복사되고 시스템이 비정상적으로 종료되었을 때 복사해 둔 정보를 바탕으로 부팅할 수 있는 옵션을 사용자에게 제공하는 역할을 한다. 가상환경에는 다음과 같은 하위키에 가상머신을 식별할 수 있는 정보가 남는다.

레지스트리 HKLM₩₩SYSTEM	
1	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩VBoxGuest(VBOX)
2	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩VBoxMouse(VBOX)
3	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩VBoxService(VBOX)
4	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩VBoxSF(VBOX)
5	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩VBoxVideo(VBOX)
6	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Control₩₩Class₩₩ {4D36E968-E325-11CE-BFC1-08002BE10318}₩₩0000₩₩DriverDesc

7	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Control₩₩Class₩₩ {4D36E968-E325-11CE-BFC1- 08002BE10318}₩₩0000₩₩ProviderName(VBOX)
8	HKLM₩₩SYSTEM₩₩CurrentControlSet₩₩Services₩₩SbieDrv
9	HKLM₩₩SYSTEM₩₩CurrentControlSet₩₩Enum₩₩PCI₩₩VEN_80EE*
10	HKLM₩₩SYSTEM₩₩CurrentControlSet₩₩Enum₩₩PCI₩₩VEN_5333*
11	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩vpcbus
12	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩vpc-s3
13	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩vpcuhub
14	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩msvmmouf
15	HKLM₩₩SYSTEM₩₩ControlSet001₩₩Services₩₩Disk₩₩Enum₩₩ (DeviceDesc, FriendlyName)(VBOX)
16	HKLM₩₩SYSTEM₩₩ControlSet002₩₩Services₩₩Disk₩₩Enum₩₩ (DeviceDesc, FriendlyName)(VBOX)
17	HKLM₩₩SYSTEM₩₩ControlSet003₩₩Services₩₩Disk₩₩Enum₩₩ (DeviceDesc, FriendlyName)(VBOX)
18	HKLM₩₩SYSTEM₩₩CurrentControlSet₩₩Control₩₩SystemInformation₩₩ SystemProductName(VIRTUAL, VIRTUALBOX)
19	HKLM₩₩SYSTEM₩₩CurrentControlSet₩₩Enum₩₩IDE(VBOX)

[표 3-1] 가상환경을 식별하는 HKLM₩₩SYSTEM 레지스트리 목록

하드웨어 정보는 HKEY_LOCAL_MACHINE₩₩HARDWARE 경로에 존재하며 부팅시 감지된 모든 하드웨어와 그 하드웨어 장치의 드라이버 맵핑 정보로 메모리에 휘발성 정보로 존재한다. 다음과 같은 하위키에 가상머신을 식별할 수 있는 정보가 남는다.

레지스트리 HKLM₩₩HARDWARE	
1	HKLM₩₩HARDWARE₩₩DEVICEMAP₩₩Scsi₩₩Scsi Port 0₩₩Scsi Bus 0₩₩ Target Id 0₩₩Logical Unit Id 0 (Identifier) (VBOX)
2	HKLM₩₩HARDWARE₩₩Description₩₩System (SystemBiosVersion) (VBOX)
3	HKLM₩₩HARDWARE₩₩Description₩₩System (VideoBiosVersion) (VIRTUALBOX)
4	HKLM₩₩HARDWARE₩₩Description₩₩System (SystemBiosDate) (06/23/99)
5	HKLM₩₩HARDWARE₩₩ACPI₩₩DSDT₩₩VBOX__ (VBOX)
6	HKLM₩₩HARDWARE₩₩ACPI₩₩FADT₩₩VBOX__ (VBOX)

7	HKLM₩₩HARDWARE₩₩ACPI₩₩RSDT₩₩VBOX__ (VBOX)
8	HKLM₩₩HARDWARE₩₩Description₩₩System₩₩BIOS (SystemProductName)(VIRTUAL)

[표 3-2] 가상환경을 식별하는 HKLM₩₩HARDWARE 레지스트리 목록

소프트웨어 정보는 HKEY_LOCAL_MACHINE₩₩SOFTWARE 경로에 존재하며, 시스템 부팅에 필요 없는 시스템 전역 구성 정보(소프트웨어 정보)를 담고 있으며, 다음과 같은 하위키에 가상머신을 식별할 수 있는 정보가 남는다.

레지스트리 HKLM₩₩SOFTWARE	
1	HKLM₩₩SOFTWARE₩₩Oracle₩₩VirtualBox Guest Additions (VBOX)
2	HKLM₩₩SOFTWARE₩₩Microsoft₩₩Virtual Machine₩₩Guest₩₩Parameters (HYPER-V)
3	HKLM₩₩SOFTWARE₩₩MICROSOFT₩₩CRYPTOGRAPHY; Key:MACHINEGUID
4	HKLM₩₩SOFTWARE₩₩Classes₩₩Folder₩₩shell₩₩sandbox
5	HKLM₩₩SOFTWARE₩₩Microsoft₩₩Windows₩₩CurrentVersion₩₩Uninstall₩₩Sandboxie

[표 3-3] 가상환경을 식별하는 HKLM₩₩SOFTWARE 레지스트리 목록

3.2.3 프로세스

가상머신에서 동작하고 있는 프로세스 중 가상환경을 식별할 수 있는 프로세스 목록이다.

프로세스	
1	VBoxService.exe
2	VBoxTray.exe

[표 3-4] 가상환경을 식별하는 프로세스 목록

3.2.4 파일

가상머신이 정상적으로 동작할 수 있도록 하는 SYS파일, Dll파일, exe파일이다. 해당 파일들이 존재하지 않는 경우, 가상환경이 정상적으로 동작하지 않는다.

	.SYS	.DLL	.EXE
1	VBoxMouse.sys	vboxoglarrayspu.dll	vboxservice.exe
2	VBoxGuest.sys	vboxoglcrutil.dll	vboxtray.exe
3	VBoxSF.sys	vboxoglerrorspu.dll	VBoxControl.exe
4	VBoxVideo.sys	vboxoglfeedbackspu.dll	
5	vmmouse.sys	vboxoglpackspu.dll	

6	vmmemctl.sys	vboxoglpassthroughspu.dll	
7	vmmouse.sys	vboxdisp.dll	
8	vmrawdsk.sys	vboxhook.dll	
9		vboxmrxnp.dll	
10		vboxogl.dll	
11		vmcheck.dll	

[표 3-5] 가상환경을 식별하는 파일 목록

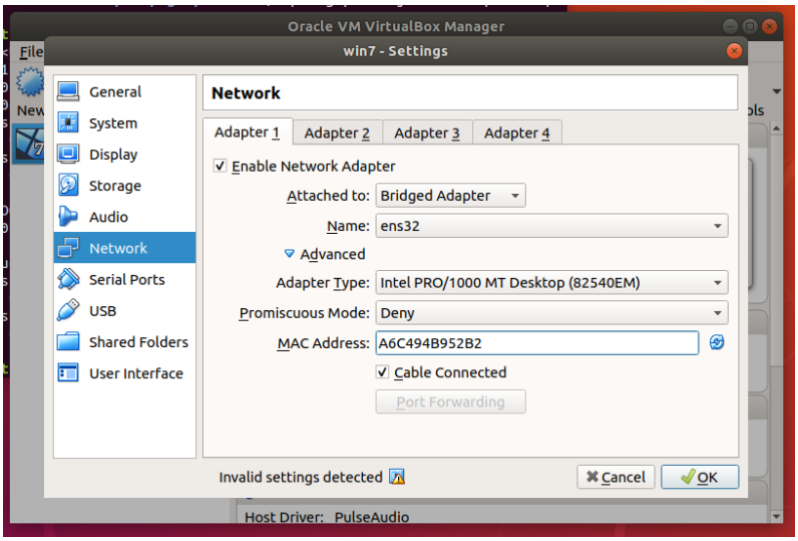
4.

5.

3.3 우회방안

3.3.1 Mac Address 변경

6. MAC 주소를 통한 가상머신 탐지를 회피하기 위해 가상머신의 MAC 주소를 수동으로 할당한다. 일반적으로 탐지에 사용하는 MAC 주소 OUI는 해당 장치를 제작한 업체의 고유 번호가 할당되므로 Host PC에서 사용하는 MAC 주소를 활용하여 새로운 MAC 주소를 할당한다. 이때 내부 네트워크 안에서 중복되는 MAC 주소를 사용하게 되면 충돌이 발생하므로 주의해야한다.

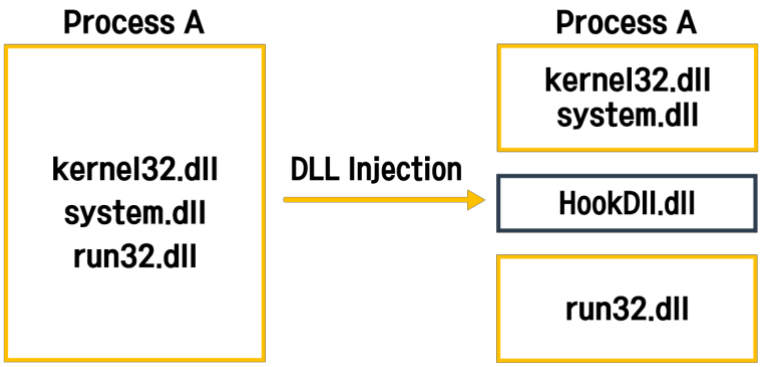


[그림 3-4] 가상환경에서 MAC주소 변경한 후

3.3.2 DLL Injection

DLL Injection은 다른 프로세스에 DLL을 강제적으로 삽입하는 기술이다. DLL이 프로세스에 삽입되면 DllMain()이 실행되며 프로세스의 실행을 변조할 수 있다. 이 기법은 다른 프로세스에서 LoadLibrary()를 이용하여 특정 DLL이 로드될 수 있도록 한다. 아래 [그림 3-5]는 다른 프로세스에서 Process A에 HookDll.dll을 DLL Injection한다. 그 결과 Process A에 HookDll.dll이 삽입된다.

7.



[그림 3-5] DLL Injection 전후 관계

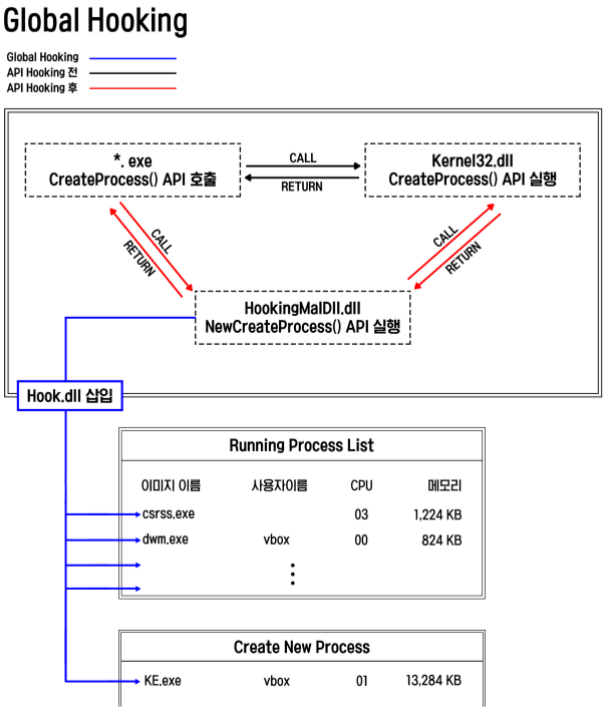
3.3.3 Hooking

8. 리버싱에서 Hooking은 정보를 가로채고, 실행 흐름을 변경하고, 원래와는 다른 기능을 제공하게 하는 기술이다. Hooking 종류에는 API Hooking과 Global Hooking 등이 있다.

API Hooking이란 Win32 API 호출을 중간에서 가로챈 후 제어권을 얻어 낸다. 이를 이용하여 API 호출 전/후에 Hook 코드를 실행시킬 수 있다. 또한, API에 넘어온 파라미터 혹은 API 함수의 리턴 값을 엿보거나 조작할 수 있다.

Global Hooking은 현재 실행 중인 프로세스와 앞으로 실행될 모든 프로세스에 대해서 API Hooking을 시키는 것을 의미한다. 아래 [그림 3-6]은 API Hooking과 Global Hooking의 진행 과정이다. 기존에 실행되고 있던 프로세스들은 Dll이 Injection되는 시점에 API Hooking이 진행된다. Global Hooking이 실행 중인 환경에서는 새 프로세스가 생성될 때마다 Kernel32.dll의 CreateProcess API가 아닌 HookingMalDll.dll의 NewCreateProcess API가 실행되어 모든 프로세스에 HookingMalDll.dll이 삽입될 수 있도록 한다.

9.



[그림 3-6] API Hooking과 Global Hooking 진행 과정

3.4 개발 산출물

프로젝트 산출물로는 Hooking을 진행하는 AntiAntiVMDll.dll과 해당 Dll을 프로세스에 삽입해주는 AntiAntiVM.exe가 있다.

3.4.1 AntiAntiVM.exe

AntiAntiVM.exe는 해당 환경에 존재하는 모든 프로세스들에 Dll Injection을 진행한다. DLL 주입을 위해서는 권한 상승이 필요하기 때문에 SetPrivilege() 함수를 사용한다. 권한 상승 후, 해당 환경에 존재하는 프로세스에 API Hooking을 하기 위해서 AntiAntiVMDll.dll을 주입시킬 함수가 필요하다. 이를 위해 InjectDll() 함수를 사용했으며, 이를 이용하여 가상 환경에 존재하는 프로세스에 Hooking을 진행한다.

3.4.2 AntiAntiVMDll.dll

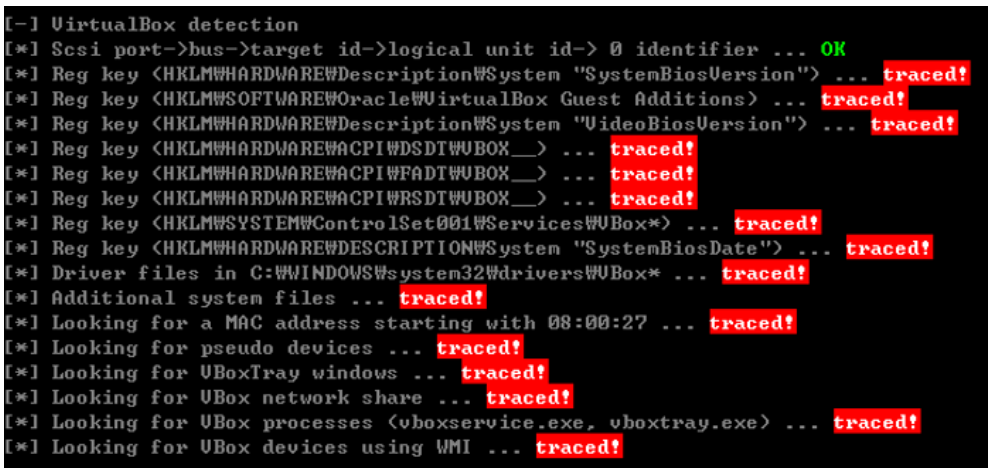
AntiAntiVMDll.dll은 기존에 존재하는 API를 후킹한다. 악성코드들이 가상환경을 탐지하는 API를 후킹하여 악성코드가 로컬환경이라고 인식하도록 변경한다. 예를 들어, CreateFileA()를 호출하면 API Hooking을 통해 NewCreateFileA()가 호출된다. NewCreateFileA()에서는 unhook_by_code()를 통해 hooking을 해제하여 기존의 CreateFileA()를 이용할 수 있도록 한다. 이후 기존의 함수 결과를 이용하여 가상환경을 로컬환경으로 인식할 수 있도록 바꿔준다. hook_by_code()를 통해 이후 CreateFileA()가 다시 호출되었을 때 NewCreateFileA()가 호출될 수 있도록 후킹해준다.

4. 최종결과물

4.1 Sandbox 하드닝 툴

10. Pafish는 샌드박스 하드닝을 위해 만들어진 오픈소스 도구이다. 해당 프로그램을 이용하면 어떤 요소들을 통해 가상환경으로 판단되는지 알아볼 수 있다. 이번 프로젝트의 경우는 Virtual Box를 사용했으므로 VirtualBox에서 탐지될 수 있는 다양한 요소들에 대한 탐지를 진행했다.

11.

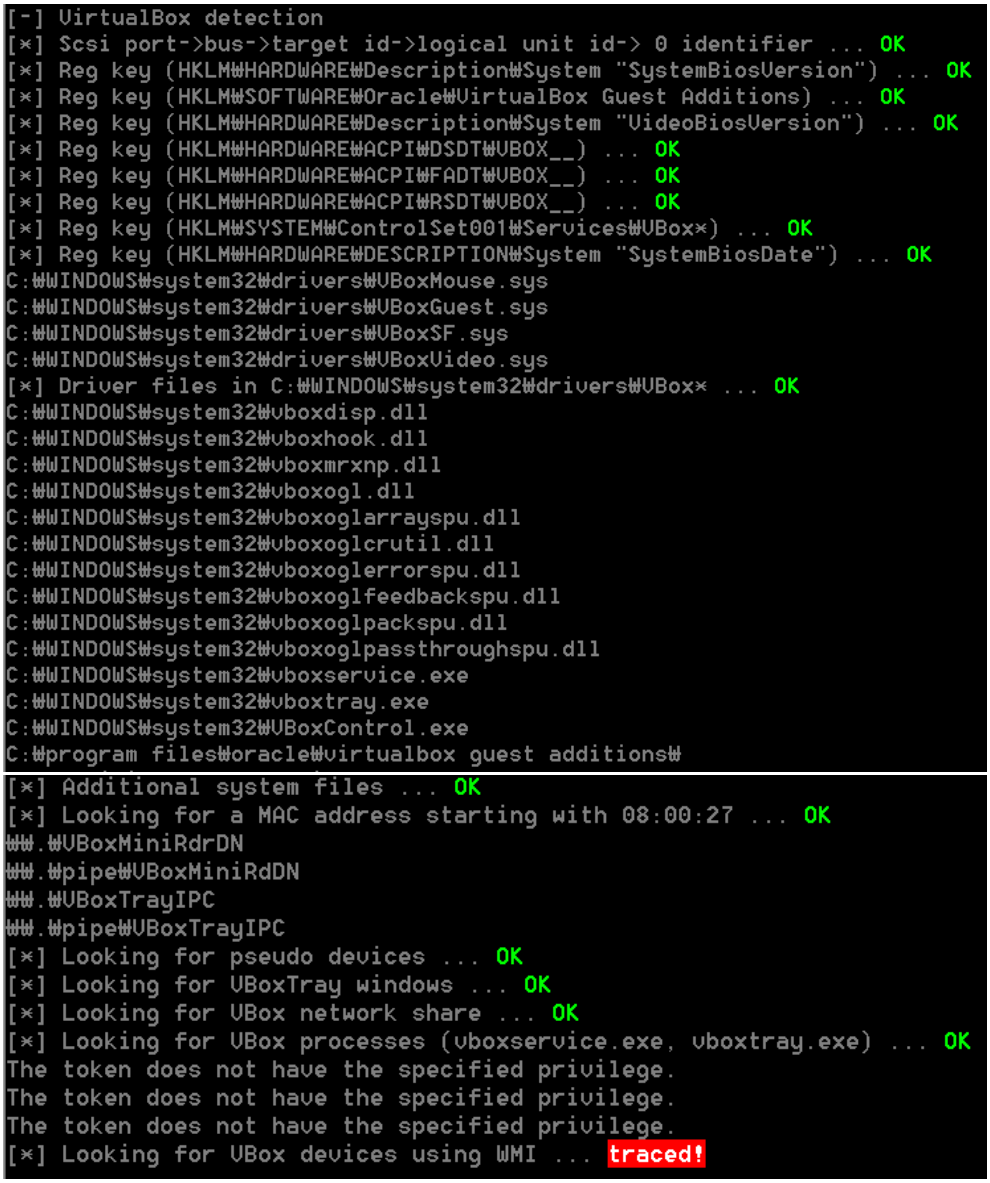


[그림 4-1] Pafish로 확인한 후킹 전 가상환경의 상태

제작한 AntiAntiVM.exe와 AntiAntiVMDll.dll을 이용하여 Global Hooking을 진행했다.

[그림 4-2] API Hooking을 위한 Dll Injection

다시 Pafish를 가상환경에서 실행시키면 이전과는 다른 결과가 나타나는 것을 알 수 있다. 이전에는 가상환경으로 판단하도록 한 기준이 모두 Hooking되어 해당 값을 제대로 인지하지 못하는 것이다. 다만 이번 프로젝트 기간 동안에는 커널 후킹을 하지 않아서 커널을 사용하는 일부 요소들에 대한 후킹은 이뤄지지 않아 일부 부분에서 traced!라는 결과가 나타났다.

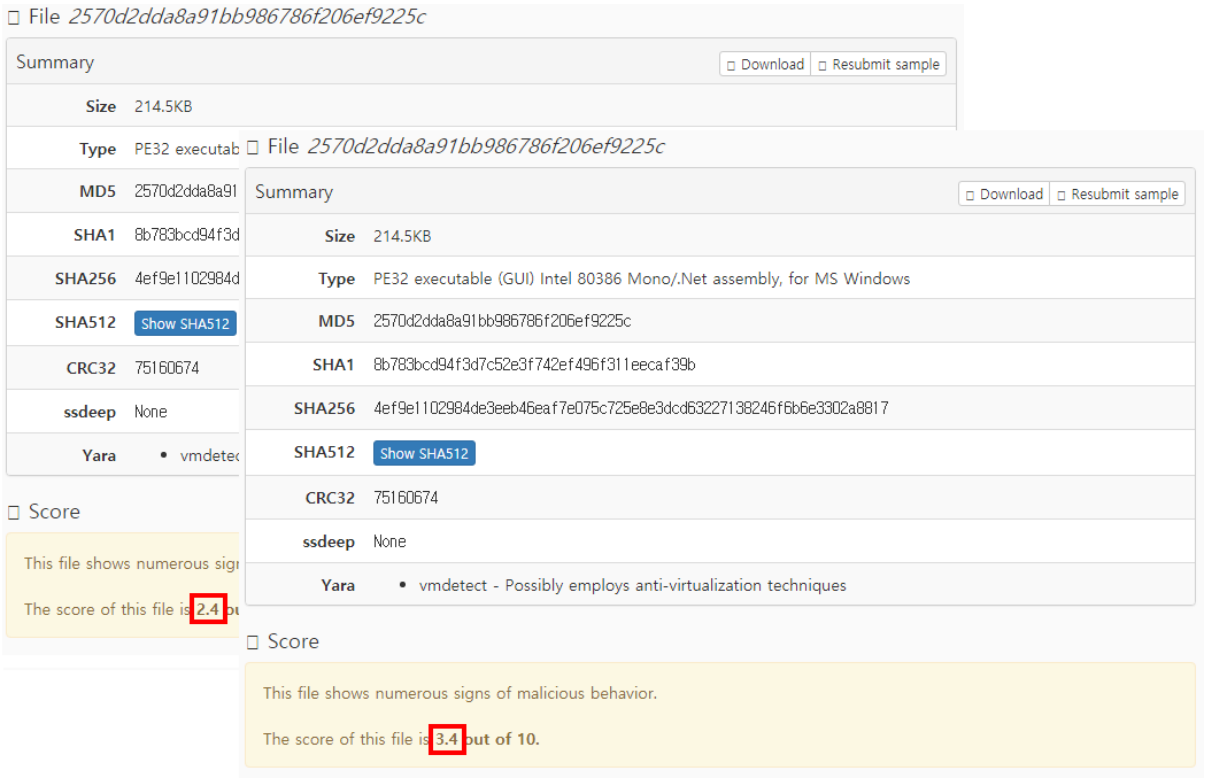


[그림 4-3] Pafish로 확인한 후킹 후 가상환경 상태

4.2 Cuckoo Sandbox

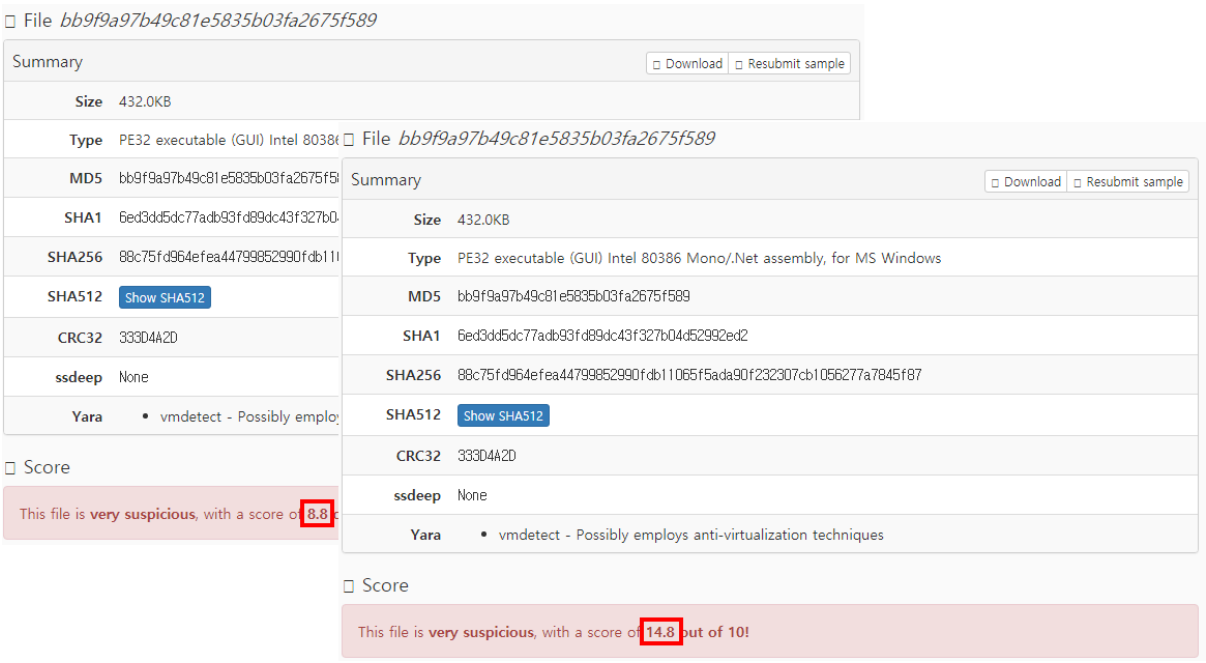
12. Cuckoo SandBox에서 AntiAntiVM.exe와 AntiAntiVMDll.dll을 실행시키기 전과 후의 동적분석 결과를 비교했다. 아래는 일부 검사의 결과값이다. [그림 4-4]의 위험도 값과 [그림 4-5]의 위험도 값을 비교해보면 악성코드의 위험 정도가 상승한 것을 볼 수 있었다. 이후 다른 악성코드들도 모두 확인해본 결과, Cuckoo Sandbox에서 판단한 위험도의 값이 변화한 경우가 70%, 변화하지 않은 경우가 30%였다. 다만, 변화한 위험도 값이 상승한 경우도 있었지만 값이 하락한 경우도 있었다. 이는 악성코드 파일이 완전하지 않아 제대로 된 동작을 하지 않았기 때문이라고 판

단된다.



[그림 4-4] Cuckoo sandbox에 적용 전/후 악성코드 검사 리포트

(Hash: 2570d2dda8a91bb986786f206ef9225c)



13.

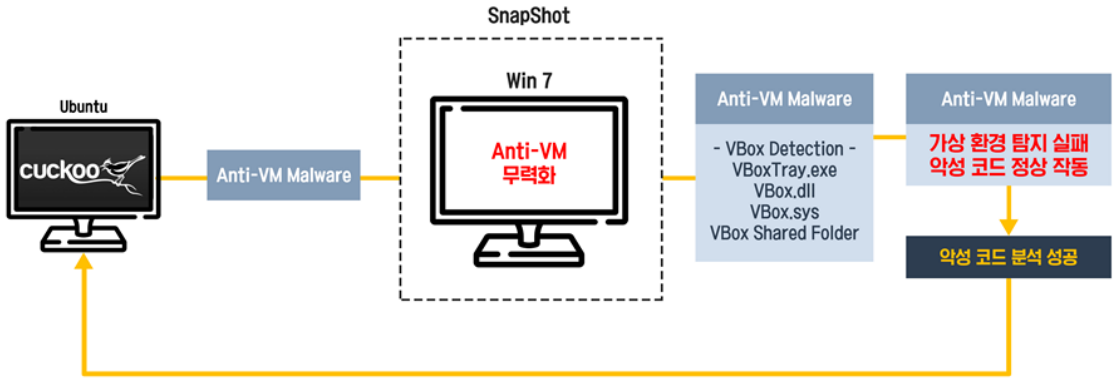
[그림 4-5] Cuckoo sandbox에 적용 전/후 악성코드 검사 리포트

(Hash: bb9f9a97b49c81e5835b03fa2675f589)

5. 결론

5.1 기대효과

본 프로젝트는 가상 환경을 탐지 및 우회하여 동적분석을 방해하는 악성코드의 무력화 필요성에 의해 만들어졌다. 이에 악성코드가 가상 환경 탐지에 사용하는 API를 Hooking하는 exe와 dll을 제작하였다. 이 파일을 Sandbox가 사용하는 가상환경인 Virtualbox내의 Win7 OS 환경에 삽입하여 실행시킨 후 스냅샷을 찍는다. 이후 해당 상태에서 악성코드 동적분석을 수행한다면 Anti-vm 기능이 있는 악성코드가 가상 환경에서 정상적으로 작동되어, 악성코드의 분석이 보다 원활하게 이루어질 수 있을 것이다.



5.2 한계점

이번 프로젝트의 한계점으로는 악성코드가 vpcext 명령어처럼 커널 구간에서 함수를 실행시키는 경우에는 무력화가 불가능하다는 점이다. 해당 함수는 인텔 커널에서 실행되기 때문에 일반적으로 메모리에 문자열이 있는지 확인하기 어렵다. 현재 시도한 후킹방식은 Global hooking으로 윈도우 유저모드에서만 작동하여 윈도우의 커널을 건드리지 않는다. 악성코드가 본 프로젝트에서 후킹한 API를 사용하지 않거나, 커널에서 가상환경인지 탐지하는 경우에는 악성코드가 가상환경임을 탐지할 수 있다. 또한 악성코드가 완전하지 않은 경우가 많아 악성코드 분석이 정확하지 않았고, 샌드박스 내에서도 동적분석이 명확히 되지 않은 문제가 존재했다.