알려지지 않은 실행파일의 악의적인 특징들을 분석하기 위한 행위추적 프로그램

김대원*, 김익균*, 오진태*, 장종수*
*한국전자통신연구원
e-mail : {dwkim77, ikkim21, showme, jsjang}@etri.re.kr

Behavior Tracing Program to Analyze Malicious Features of Unknown Execution File

Daewon Kim*, Ikkyun Kim*, Jintae Oh*, and Jongsoo Jang*
* Knowledge-based Information Security and Safety Research Department,
Electronics and Telecommunications Research Institute

요 약

컴퓨팅 환경에서 각종 보안 위협들의 핵심에는 악성 실행파일들이 있다. 전통적인 시그니처 기반의 보안 시스템들은 악의적인 실행파일들 중에서 알려지지 않은 것들에 대해서는 런타임 탐지에 어려움이 있다. 그러한 이유로 런타임 탐지를 위해 시그니처가 필요 없는 정적, 동적 분석 방법들이 다각도로 연구되어 왔으며, 특히 악성 실행파일을 실제 실행한 후 그 동작상태를 모니터링 하는 행위기반 동적 분석방법들이 많은 발전을 이루어왔다. 그러나 대부분의 행위기반 분석방법들은 단순히 몇 가지 행위나 비순차적인 분석정보를 제공하기 때문에, 차후 악성여부를 최종 판단하는 방법론에 적용하기에는 그 분석정보가 충분하지 않다. 본 논문에서는 악성 실행파일이 실행되는 동안 발생할 수 있는 행위들을 분류하고, 이를 모니터링 하는 프로토타입 프로그램을 구현하였다. 또한, 악성 실행파일을 직접 실행하는 것은 제한된 컴퓨팅 환경에서 이루어지기 때문에, 실제 악성 실행파일을 모니터링 한 결과를 토대로 행위기반 모니터링 방법이 극복해야 될 이슈들에 대해서도 언급하고 있다.

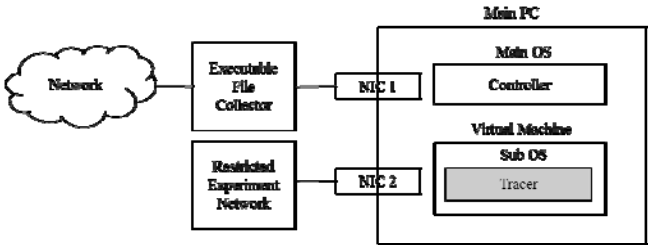
1. 서론

기존 보안 시스템들이 알려지지 않은 새로운 위협에 대처하기 어려운 근본 원인은 시그니처 기반 탐지가 이루어지기 때문이다. 기존 시스템들은 새로운 위협이 발생될 때마다 이를 탐지하기 위한 시그니처들을 계속 추가해야 하지만, 그 시그니처의 수는 기하급수적으로 늘어나고 있어[1], 시그니처 추가는 점점 그 한계에 도달하고 있다. 또한 시그니처 추가가 주로 수작업을 통해서 이루어지기 때문에 런타임(run-time) 탐지가 어려우며, 이로 인해 이미 대규모 피해가 발생한 이후에 대처가 가능하다는 점도 큰 문제이다.

컴퓨팅 환경에 대한 이런 위협들의 핵심에는 악성 실행파일이 있다. 악성 실행파일은 원격지 컴퓨터에 침투하기 위한 익스플로잇 코드(exploit code)를 발생시키며, 침투한 컴퓨터에서는 각종 시스템 리소스를 차지하여 운영에 피해를 줄뿐 아니라, 중요 사이트에 분산 서비스 거부 공격(DDoS attack, Distributed Denial of Service attack) 을 유발시키는 좀비(zombie) 호스트[2]를 만들기도 한다. 이와 같은 악성 실행파일들이 자동화된 공격 변형 툴들을 통해 쉽고 다양한 형태로 만들어지기 때문에, 기존의 시그니처 기반의 탐지 방법을 통한 런타임 대응은 점점 그 한계에 가까워지고 있는 것이 사실이다.

악성 실행파일의 런타임 탐지를 위해 이미 많은 연구들이 진행되어 왔으며, 이들을 크게 정적 분석과 동적 분석의 두 가지 방법으로 나눌 수 있다. 정적 분석 방법들[3, 4, 5]은 악성 실행파일을 직접 실행하지 않고, 일반적으로 역어셈블을 수행하여 컨트롤 플로우나 데이터 플로우에 대한 의심스러운 패턴을 찾아내게 된다. 동적 분석 방법들[6][7][8][9][10]은 제한된 실험 환경을 구성하여 악성 실행파일을 직접 실행한 후, 실제 실행되는 행위들을 모니터링하여 악성여부를 판단하게 된다. 정적 분석 방법은 동적 분석 방법보다 일반적으로 처리속도가 빠르다는 장점을 가지고 있지만, 분석된 패턴 정보와 실행시의 동작과는 의미론적인 차이가 있어 오탐율과 미탐율이 높다는 단점이 있다. 따라서 최근에는 이 문제를 극복할 수 있는 방안인 동적 분석 방법이 훨씬 활발히 연구되고 있다.

동적 분석 방법은 악성 실행파일을 직접 실행하여 실제 악의적인 행위의 발생여부를 탐지하는 것으로서, 가상화 방식을 통해 제한된 실험 환경을 구축하여 악의적인 행위로부터 실험 시스템을 보호하고 공격이 외부로 유출되지 않도록 한다. 가상화[11]는 크게 가상 머신(Virtual Machine) 방식과 에뮬레이터(Emulator) 방식이 있으며, 이 두 방식은 서로 장단점이 있다. 가상 머신 방식은 속도가 빠르고 악성 실행파일 실행 후 시스템 복구가 간편하다는 장점이 있는 반면에, OS 가 기반 하드웨어를 지원해야 하므로 다양한 OS 용 악성 파일을 분석하기 위해서는 해당 하드웨어들도 종류별로 있어야 하는 단점이 있다. 이와는 반대로, 에뮬레이터 방식은 하드웨어 자체를 에뮬레이션 하기 때문에 하나의 하드웨어에 다양한 OS 를 설치할 수 있지만, 속도가 느리다는 단점이 있다. 가상 머신의 대표적인 프로그램은 VMware[12]이고, 에뮬레이터의 대표적인 프로그램은 QEMU[13]이다. 현재 두 방식 모두 가상화 여부가 악성 코



(그림 1) 운영 환경

드에 의해 탐지될 수 있다는 단점들이 알려져 있지만[14], 여전히 대다수의 악성 실행파일들은 가상화 여부를 판단하지 않고 동작한다.

본 논문은 윈도우 계열의 운영체제에서 알려지지 않은 악성 실행파일에 대한 행위 기반 추적 프로그램의 구현 결과 및 실험 결과에 대해 설명을 할 것이다. 본 논문에서 구현한 사항들은 기존 논문들이나 기존 구현 테크닉들과 비교해서 새로운 내용은 없다. 그러나 기존 분석 프로그램들이 도출하는 개별 결과만으로는 차후 악성여부를 자동적으로 판단하는 방법론을 적용하기에는 분석정보의 범위 및 내용이 충분치 않다. 따라서 우리는 기존 개별 분석 결과들을 통합할 수 있는 프로그램의 필요성을 느끼게 되었고, 그 프로토타입 프로그램을 구현하였다.

2. 행위 추적 프로그램

2.1 운영 환경

그림 1 은 본 논문의 행위 추적 프로그램이 운영되는 환경을 보여주고 있으며, 행위 추적 프로그램은 가상 머신(Virtual Machine) Sub OS 의 Tracer 에 해당한다.

실행 파일 수집기(Executable File Collector)는 외부 네트워크로부터 샘플파일을 수집하고, 수집된 샘플파일을 NIC1(Network Inter-face Card 1)을 통해 관리 프로그램(Controller)으로 전송한다. NIC1 은 Main OS 에서만 사용이 가능하도록 설정하여, 샘플파일이 실행될 Sub OS 와 외부 네트워크를 완전히 분리한다. Controller 는 샘플파일의 주요 정보들을 기록한 후, 가상화 네트워크 인터페이스를 통해 Sub OS 의 Tracer 로 샘플파일을 전송한다. 샘플파일이 실행될 Sub OS 는 NIC2 를 통해 제한된 실험 네트워크(Restricted Experiment Network)에 연결되어 있다. 제한된 실험 네트워크는 Tracer 의 분석품질에 관련되어 있기 때문에, 가능한 한 실제 네트워크에 가깝도록 구축하여야 한다. 단적인 예로 NIC 이 살아있지 않으면 실행된 후 주요 증상 없이 바로 종료되는 악성 실행파일들도 존재한다. Tracer 는 샘플파일을 직접 실행하여 행위들에 대한 로깅 작업을 진행하고, 그 결과를 Controller 로 전송한다. 로깅 작업이 종료되면, Controller 는 현재의 가상머신을 종료하고, 새 가상화 이미지로 가상머신을 구동함으로써, 새로운 샘플파일이 분석될 환경을 복구하게 된다.

2.2 악의적인 행위들을 추적하기 위한 감시 항목

본 프로그램에서 로깅 하는 행위 내용들은 악성 실행파일의 실행 시에 발생되는 비정상적인 시스템 상태를 추적하기 위한 내용들이다. 비정상적인 시스템 상태란 사용자가 의도하지 않은 행위들이 발생되는 상태를 의미하며, 이런 행위들은 응용 계층에서 커널 계층까지 시스템의 모든 구성 요소들에서 발생될 수 있다. 따라서, 몇 개의 시스템 구성 요소만 모니터링 하거나, 응용 계층을 위주로 한 분석 정보들로는 자동화된 악성여부 판단에는 충분한 자료를 제공하지 못한다.

<표 1> 윈도우 계열에서의 시스템 구성 요소 및 대표적인 비정상 동작/상태들

Monitoring Components	Abnormal Statuses
CPU	• Excessive cpu usage
MEMORY	• Excessive memory usage
FILE	• Arbitrary file accessing in the System Folder • 'host' file accessing • E-mail address file accessing • Registry file accessing • Arbitrary file accessing with execut-ing/copying/modifying/creating/deleting • Important DLL file accessing
REGISTRY	• Important key value accessing with read-ing/creating-g/deleting/modifying
NETWORK	• Increasing of connection trial IP and port • Listening port creating • Excessive traffic usage • Arbitrary IP address connection
INTERFACE	• Key logging • Arbitrary information (advertisement etc.) pop-up • Operation disturbance of important programs • Input devices (keyboard/mouse etc.) disturbance
PROCESS	• Termination trial of important service programs (security programs/RPC service etc.) • Termination trial of system • Multiple processes creation • Termination trial of normal service programs

<표 1>에서는 7 가지 주요 시스템 구성 요소와 각 구성 요소에서 발생할 수 있는 대표적인 비정상 행위들을 나열하고 있다. 본 프로토타입 프로그램의 최종목표는 샘플 실행파일들을 실행 한 후, 각각의 구성 요소들에서 발생된 시스템 행위들을 기록하고, 해당 실행파일이 악의적인지의 여부를 최종 결정하기 위한 방법론들에 충분한 로깅 결과를 제공하는데 있다.

2.3 Tracer 의 동작 과정

Tracer 는 실행된 파일의 악의적인 동작들을 추적하기 위한 복잡한 규칙 같은 것들이 없다. 단지 실행된 파일의 Process ID (PID) 정보만을 이용할 뿐이다. Tracer 는 대상 PID 와 대상 PID 로부터 파생된 PID 들이 2.2 절과 관련된 동작을 하는지를 모니터링 하기 위해 멀티쓰레드로 동작된다. 샘플파일이 Tracer 에 전송되면, Tracer 는 CreateProcess()를 통해 중지모드(SUSPENDED_MODE)로 샘플파일을 실행한다. 이 때, CreateToolHelp32Snapshot()을 통해 생성된 프로세스의 PID, 프로세스 명, 부모 프로세스 ID, 실행파일의 경로 등에 관한 정보를 얻는다. 수집된 프로세스 명 및 PID 등은 각 감시항목별 탐지 기능에 정책으로 적용한다. 이는 샘플파일과 관련 없는 불필요한 정보를 로깅 하지 않고, 성능 부하도 줄이기 위함이다.

각 감시항목에 해당하는 행위들을 독립적으로 로깅 하기 위해, 각 항목별 로깅 기능들을 AfxBeginThread()를 통해 쓰레드로 구동한다. 앞서 설정된 중지모드를 해제하여, 샘플파일을 실제 구동한다. 각 항목별 로깅 기능들은 적용된 정책에 해당하는 프로세스에서 발생된 동작들을 시간 순으로 로깅 하게 된다. 처음 생성된 프로세스에서 다른 프로세스들을 생성하는 경우, 프로세스간의 관계를 부모-자식 관계의 트리로 구성하고 새로 생성된 프로세스에 관한 정보들도 로깅을 위한 정책으로 적용하여 로깅이 되도록 한다. 이는 악성 실행파일에 의해 처음 생성된 프로세스가 직접 악

<표 2> 감시 항목에 대한 정보 습득 구현 방법

Monitoring Components	The methods of information acquisition
CPU	• PdhGetFormattedCounterValue()
MEMORY	• GetProcessMemoryInfo()
FILE	• Kernel-level file filter device driver such as Filemon
REGISTRY	• Kernel-level registry filter device driver such as Regmon
NETWORK	• Session tracing function like Netstat • Kernel-level packet capture device driver such as Winpcap
INTERFACE	• CALLBACK DebugProc(), etc.
PROCESS	• CreateToolhelp32Snapshot()

성 행위를 하지 않고 다른 프로세스들을 생성하여 악의적인 행위를 할 수도 있기 때문이다.

실행된 프로세스들을 로깅 하는 지속시간은, 로깅 되고 있는 모든 프로세스가 종료되거나 최대 60 초간 이루어진다. 그러나 악성 실행파일들 중 행위 기반 분석 기법들을 회피하기 위해 특정 시간 동안 액션을 취하지 않는 것들이 있다. 따라서, 본 프로그램은 프로세스가 실행된 후 3 초간 행위가 없을 때마다 1 시간, 1 일, 1 달, 1 년 이렇게 Sub OS 의 내부 시간을 조정하게 된다. 1 년이 지난 설정을 적용한 경우에도 3 초간 행위가 없을 때는 로깅을 종료한다.

2.4 Tracer 의 구현

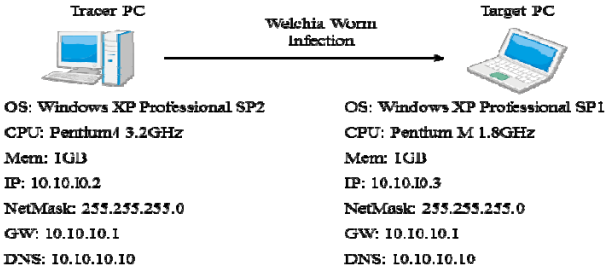
Tracer 는 각 감시항목별 실행파일의 행위를 로깅 하기 위해 윈도우가 제공하는 API 및 기존 유명 프로그램들에서 사용된 기법들을 사용한다. 따라서 Tracer 그 자체적으로 새로운 기법들이 사용되는 것은 아니다. <표 2>는 감시할 구성 요소와 정보를 습득하는 방법에 관해서 본 프로그램에서 이용중인 대표적인 구현기법들을 나타낸 것이다. CPU, 메모리, 프로세스, 인터페이스 관련 정보들은 윈도우가 제공하는 API 를 주로 이용하여 중요 정보들을 얻을 수 있고, 그 외 정보들은 필터 디바이스 드라이버[16][17][18] 제작과 같은 별도의 구현이 필요하다.

3. 실험 결과

3.1 실험 환경

이번 테스트에 사용된 Tracer 는 아직 구현이 모두 이루어지지 않은 프로토타입 버전이며, Visual Studio 2008 Professional Edition 에서 제작되었다. 현재 본 실험이 이루어진 Tracer 에 구현되어 있는 행위 추적 기능들은 프로세스, 네트워크, 파일, 레지스트리 등이다.

<그림 2>는 본 실험이 이루어진 환경을 보여주고 있다. Tracer PC 는 행위 추적을 위한 Tracer 프로그램이 설치되어 있으며, 웰치아 웜 (V3 진단명 Win32/Welchia.wor-m.10240, Kaspersky 진단명 Net-Worm.Win32.Welchia.a)을 실행하게 된다. 현재는 Tracer 만 개발이 되어 있는 상태이므로, 이번 테스트에서는 2.1 절과 같은 가상 머신(VMware)을 이용하지는 않는다. 본 논문은 그림 1 과 같은 전체 분석 시스템을 구축하는 것이 목적이 아니기 때문에, 가상화 적용여부는 본 논문의 우수성을 입증하기 위한 프로토타입의 실험결과에 중요한 부분이 아니다. Target PC 는 Tracer 의 네트워크를 인식시키고 웰치아 웜의 스캐닝 대상으로 사용할 목적으로



(그림 2) 실험 환경

<표 3> 웰치아 웜을 수정하지 않은 상태의 실험결과

Time	Monitoring Components	Abnormal Statuses
2010-12-17 11:21.16	FILE	*w.mal:237 C:\WINDOWS\SYSTEM32\NTDLL.DLL IRP_MJ_CREATE
2010-12-17 11:21.16	FILE	*w.mal:237 C:\WINDOWS\SYSTEM32\KERNEL32.DLL IRP_MJ_CREATE
2010-12-17 11:21.16	FILE	*w.mal:237 C:\WINDOWS\system32\dllcache\tftpd.exe IRP_MJ_CREATE
2010-12-17 11:21.16	FILE	*w.mal:237 C:\WINDOWS\system32\conime.exe IRP_MJ_CREATE
2010-12-17 11:21.16	PROCESS	*w.mal:237 => conime.exe:786
2010-12-17 11:21.17	NETWORK	*w.mal:237 ARP SRC=10.10.10.2 DST=10.10.10.10 ……

<표 4> 수정된 웰치아 웜의 실험결과

Time	Monitoring Components	Abnormal Statuses
2010-12-20 13:11.17	NETWORK	*w.mal:495 TCP SRC=xxx.xxx.62.110:1453 DST=204.203.18.152:80 ESTABLISHED
2010-12-20 13:11.21	PROCESS	*w.mal:495 => RpcServicePack.exe:8496
2010-12-20 13:11.22	PROCESS	*RpcServicePack.exe:8496 => xpsp1hfm.exe:1175
2010-12-20 13:11.23	PROCESS	*xpsp1hfm.exe:1175 => update.exe:1277
2010-12-20 13:11.44	NETWORK	*w.mal:495 ICMP SRC=xxx.xxx.62.110 DST=xxx.xxx.0.1 *w.mal:495 ICMP SRC=xxx.xxx.62.110 DST=xxx.xxx.0.2 ……
2010-12-20 13:11.04	NETWORK	*w.mal:495 TCP SRC=xxx.xxx.62.110:1483 DST=xxx.xxx.2.47:135 SYN_SENT *w.mal:495 TCP SRC=xxx.xxx.62.110:1488 DST=xxx.xxx.2.53:135 SYN_SENT ……

연결되었다. 실제 Tracer 가 운영되는 환경은 다양한 웜을 대상으로 할 것이며, 웜 마다 감염을 시킬 수 있는 Target PC 의 상태가 다르기 때문에 실제 감염을 시키기 위해서는 정밀한 Target PC 망을 구축할 필요가 있다.

3.2 로그 결과

<표 3>은 프로세스 명이 w.mal 인 웰치아 웜이 프로세스 ID 237 로 실행되는 모습들을 시간 순으로 보여주고 있다. w.mal 은 윈도우 시스템 디렉터리의 주요 DLL 들과 악성코드 확산을 위해 사용할 tftpd 프로그램을 액세스 함을 알 수 있다. w.mal 은 악의적인 작업을 사용자가 인지하지 못하도록 하기 위해 콘솔 창에서 실행되었으며(conime.exe 는 콘솔 창의 다국어 입력 지원 프로그램), DNS 서버를 찾기 위한 ARP 를 보내는 작업을 반복하게 된다. 실제 위 로그는 위와 같은 형태로 613 개가 출력이 되었다.

<표 4>는 외부 인터넷으로 접속이 가능한 환경을 구축해 놓고, 웰치아 웜의 소스 코드를 변경하여 제한된 영역만 스캐닝 및 확산을 시도하도록 한 상태의 주요 실험 결과이다. w.mal 이 마이크로소프트 업데이트 사이트(204.203.18.152)에 접속 성공한 후, 실제 악성코드 확산을 위한 작업들이 일어나는 모습을 보여주고 있다. 일반 호스트에서 웰치아 웜이 실행된다면, <표 3>에서 ARP 에 의해 DNS 가 검색되면 그 다음 과정들로 <표 4>와 같은 작업들이 수행되는 것이다. 업데이트 프로세스들이 단계적으로 실행되는 모습과 스캐닝을 위해 ICMP 로 PING 을 날린 후 해당 Target 호스트로 접속을 시도(SYN_SENT)하는 모습을 볼 수 있다.

4. 동적 분석 방법의 극복 과제

악성실행 파일들을 실행한 후, 발생되는 행위들을 추적하였지만 공격 여부를 판단하기에는 충분하지 못한 정보들을 얻게 된다면 결국 해당 악성실행 파일의 탐지에는 실패한 것이 된다. 최근의 악성실행 파일들은 자신이 실행되는 환경을 감지한 후, 악의적인 행위의 동작 여부를 결정하는 형태들이 점점 많아지고 있다.

- 외부 인터넷과의 접속 가능 여부 확인

동적 분석 방법들은 실행된 공격이 외부로 나가지 않도록 하기 위해 제한된 네트워크를 구성하게 된다. 그러나 이런 분석 환경은, 본 논문의 실험에 사용된 웰치아 웜과 같이 자신이 실행된 환경에서 외부 인터넷으로 접속이 가능한지의 여부를 감지하는 악성실행 파일들의 행위 분석에는 한계를 가지게 된다. 외부 인터넷과 연결이 되지 않는 환경이라면, 악의적인 행위를 진행하지 않거나 해당 실행을 종료시켜 버리기 때문이다.

- 가상 환경 내의 실행 여부 확인

행위 추적 방법들은 악성실행 파일을 실행하고 분석을 종료한 후, 또 다른 분석을 즉시 할 수 있도록 원래의 상태로 빠른 복구를 위해 주로 가상 머신 혹은 에뮬레이터 환경을 기반으로 한다. 따라서, 악성실행 파일들 중 자신이 실행되고 있는 가상 환경을 탐지하는 것들이 있다면, 악의적인 행위를 진행하지 않거나 해당 실행을 종료시켜 버릴 수도 있다.

- 악의적인 행위를 위한 미션 타임

악의적인 행위를 실행하기 위한 미션 타임이 구체적이고 정확한 시간으로 결정되어 있다면, 동적 분석 방법들은 행위를 오랜 시간 기다려야 하거나, 그 시간을 놓쳐버릴 수도 있다.

- 로깅 성능 문제

CWSandBox[8]와 같이 DLL 인젝션 기법을 사용하여 로깅이 이루어지는 경우는 주요 API 들이 실행될 때 마다 DLL 에 인젝션 된 로깅 함수들이 동작하게 된다. 그러나, DLL 인젝션 기법의 경우에는 응용계층 수준으로 처리되는 API 들은 커버가 가능하지만, 커널계층 수준으로 처리되는 naïve API 들은 로깅이 힘들다. 따라서, naïve API 로 구성된 악성실행 파일들의 경우는 악의적인 행위 추적이 어려울 수 있다. 커널계층의 상태를 추적하기 위해서는 커널 메시지 후킹 기법을 사용해야 하지만, 커널 메시지 후킹 기법의 경우는 처리 성능 상의 문제로 모든 커널 메시지들이 후킹된다고 장담할 수 없게 된다. 예를 들어, 같은 악성실행 파일에 대해 커널 메시지 후킹 기법을 사용하여 로그를 남기게 되면, 매번 다른 로그가 생성될 수 있다는 것이다. 따라서, 악성실행 파일을 판단하는 중요 이벤트를 놓칠 수도 있게 된다.

5. 결론

개별 분석 결과만을 도출하는 기존의 행위 분석 방법들은 차후 악성 여부를 최종 판단하는 방법론들에 대한 연구에 적용하기에는 그 정보의 양이 부족한 것이 사실이다. 따라서, 본 논문에서는 악성 실행파일들이 실행되는 동안 발생할 수 있는 악의적인 행위들을 분류하고, 그 분류에 해당하는 영역들을 종합적으로 모니터링하여, 해당 실행파일이 실행되는 동안 발생되는 모든 행위를 추적하기 위한 프로그램의 프로토타입을 구현하였다. 또한 실험 결과를 통해 행위 분석 기반 방법들이 극복해야 할 중요한 이슈들도 논의를 하였다.

참고문헌

[1] Symantec's internet security threat report, Apr. 2009. http://eval.symantec.com/mktginfo/enterprise/white_papers/bwhitepaper_internet_security_threat_report_xiv_04-2009.en-us.pdf

[2] Symantec's monthly state of spam report, Oct. 2008. http://eval.symantec.com/mktginfo/enterprise/other_resources/bstate_of_spam_report_10-2008.en-us.pdf

[3] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," Proc. of the ACM USENIX Security Symposium, 2003.

[4] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant, "Semantics-aware Malware Detection," Proc. of the IEEE Symposium on Security and Privacy, 2005.

[5] C. Kruegel, W. Robertson, and G. Vigna, "Detecting Kernel Level Rootkits through Binary Analysis," Proc. of the Annual Computer Security Application Conference, 2004.

[6] S. Sidiroglou, J. Ioannidis, A. Keromtis, and S. Stolfo, "An Email Worm Vaccine Architecture," Proc. of the First Information Security Practice and Experience, 2005.

[7] Norman SandBox, http://www.norman.com/technology/norman_sandbox/

[8] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," Proc. of the IEEE Symposium on Security and Privacy, 2007.

[9] U. Bayer, C. Kruegel, and E. Kirda, "TTanalyze: A Tool for Analyzing Malware," Proc. of the 15th Annual Conference European Inst. for Computer Antivirus Research, 2006.

[10] LitterBox, http://www.wiul.org

[11] IBM Developer Works. Virtual Linux, http://www.ibm.com/developerworks/library/l-linuxvirt/

[12] VMware: Vitrual Machine Software, http://www.vmware.com

[13] F. Bellard, "Qemu, A Fast and Portable Dynamic Translator," Proc. of the USENIX Security Symposium, 2005.

[14] J. Rutkowska, "Red pill... or How to Detect VMM Using (almost) One CPU Instruction," http://invisiblethings.org/papers/redpill.html

[15] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware Analysis via Hardware Virtualization Extensions," Proc. of the ACM Conference on Computer and Communications Security, 2008.

[16] Microsoft TechNet. FileMon: File Monitor, http://technet.microsoft.com/en-us/sysinternals/bb896642.aspx

[17] Microsoft TechNet. RegMon: Registry Monitor, http://technet.microsoft.com/en-us/sysinternals/bb896652.aspx

[18] WinPCap: The Windows Packet Capture Library, http://www.winpcap.org