Intel VT 기술을 이용한 Xen 기반 동적 악성코드 분석 시스템 구현 및 평가

(Development and Analyses of Xen based Dynamic Binary Instrumentation using Intel VT)

김 태 형 † 김 인 혁 † 엄 영 익 †† 김 원 호 †††
(Taehyoung Kim) (Inhyuk Kim) (Young Ik Eom) (Won Ho Kim)

요 약 악성코드를 분석하기 위한 기법에는 다양한 방법들이 존재한다. 하지만 기존의 악성코드 분석 기법으로는 악성코드들의 동작들을 정확하게 분석하는 것이 점점 어려워지고 있다. 특히, 분석 시스템들이 악성코드의 안티-디버깅 기술에 의해 감지되기 쉽고, 실행속도 등 여러 가지 한계점을 보임에 따라 이를 해결할 수 있는 분석 기법이 요구되고 있다. 본 논문에서는 동적 코드 분석을 위한 기본 요구사항인 명령어 단위 분석 및 메모리 접근 추적 기능을 제공하는 동적 코드 분석 시스템을 설계 및 구현한다. 그리고 DLL 로딩 추적을 통한 API 호출 정보를 추출하여, 다양한 실행 코드들을 분석 할 수 있는 기반 환경을 구축한다. 제안 시스템은 Intel의 VT 기술을 이용하여 Xen 기반으로 전가상화 환경을 구축하였으며, 게스트에서는 윈도우즈 XP가 동작할 수 있도록 하였다. 제안 시스템을 이용하여 대표적인 악성코드들을 분석해 봄으로써 제안 시스템 각각의 기능들의 활용을 살펴보고, 제안 시스템이 악성코드들을 정확하게 분석 및 탐지함을 보여준다.

키워드 : 하드웨어 지원 가상화 기술, 악성코드 분석, 동적 코드 분석 시스템, 안티-디버깅 기술

Abstract There are several methods for malware analyses. However, it is difficult to detect malware exactly with existing detection methods. Especially, malware with strong anti-debugging facilities can detect analyzer and disturb their analyses. Furthermore, it takes too much time to analyze malware. In order to resolve these problems of current analyzers, more improved analysis scheme is required. This paper suggests a dynamic binary instrumentation which supports the instruction analysis and the memory access tracing. Addtionally, by supporting the API call tracing with the DLL loading analysis, our system establishes the foundation for analyzing various executable codes. Based on Xen, full-virtualization environment is built using Intel's VT technology. Windows XP can be used as a guest. We analyze representative malware using several functions of our system, and show the accuracy and efficiency enhancements in binary analyses capability of our system.

Key words : Hardware-assisted virtualization, Malware analysis, Dynamic binary instrumentation, Anti-debugging

· 본 연구는 ETRI 부설연구소의 국가핵심기술보호를 위한 사이버위협 대응 기술 개발의 연구비지원(B551179-09-01-00)으로 수행되었습니다.
· 이 논문은 2010 동계 워크샵에서 'Intel VT 기술을 이용한 Xen 기반 동적 코드 분석 시스템'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 성균관대학교 전자전기컴퓨터공학과
kim15m@ece.skku.ac.kr
kkojiband@ece.skku.ac.kr
†† 종신회원 : 성균관대학교 컴퓨터공학과 교수
yieom@ece.skku.ac.kr
(Corresponding author임)
††† 정 회 원 : 한국전자통신연구원 부설연구소 사이버기술개발본부 연구원
kwh@ensec.re.kr

논문접수 : 2010년 4월 14일
심사완료 : 2010년 7월 20일

Copyright©2010 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 시스템 및 이론 제37권 제5호(2010.10)

1. 서 론

인터넷 기술이 발달함에 따라 이에 따른 보안성 향상에 대한 요구사항도 높아지고 있다. 특히, 악의적인 의도로 개발된 악성코드는 개인 혹은 기업의 비밀 정보를 노출시키고, 시스템을 혼란에 빠뜨린다. 이를 위해서 악성코드는 운영체제, 시스템 소프트웨어의 불완전성, 관리자의 실수 등을 이용하여 다양한 방식으로 동작한다. 따라서 이러한 악성코드를 분석하고 탐지하는 것이 현대 정보 보호 분야의 핵심 역할 중에 하나로 손꼽히고 있다. 최근에는 악성코드의 수많은 변종들이 생성 되고 은닉 기술이 발달함에 따라 기존의 분석 기법들이 제 역할을 다하지 못하는 경우가 많아지고 있다. 이에 따라 시스템 보안을 위하여 더욱 정확하고 빠르게 악성코드 분석하고 탐지하는 기술이 요구되고 있다. 악성코드를 분석하기 위한 필수 요구 조건으로는, 악성코드를 고립된 환경에서 실행하여 현재 시스템에는 영향을 주지 않는 sandbox 기능, 악성코드를 시스템콜, 명령어 단위로 추적할 수 있는 기능, 악성코드가 동적 분석을 인식하여 분석을 방해하는 은닉 기술을 역으로 무력화할 수 있는 기능 등을 들 수 있다.

이러한 요구사항을 충족시키기 위해서 최근 들어 몇 년 사이에 win32 API hooking, emulator(Qemu), full-virtualization(VMWare), hardware-assisted virtualization(Intel VT-x) 등 다양한 가상화 기술들을 이용하여 악성코드를 분석하는 기법들이 소개되었다. 하지만 이들 기법들 또한 이러한 요구 조건을 모두 만족시키지는 못하고 있다. win32 API hooking의 경우, 시스템콜 추적에는 탁월한 성능을 보이지만 명령어 단위 및 메모리 접근 추적은 불가능하다. 프로세서, 메모리, 디스크 등 하드웨어 기능들을 소프트웨어적으로 구현하는 emulator(Qemu)는 명령어 단위 추적 및 메모리 접근 추적 등이 용이하지만, 실행 속도가 매우 느리고 안티-디버깅에 취약하다. VMWare와 같이 binary translation 기법을 이용하는 full-virtualization의 경우에는 emulator보다 실행 속도는 빠르지만 안티-디버깅에는 여전히 취약하다.

최근 Intel과 AMD의 프로세서들이 제공하는 하드웨어 가상화 지원 기능을 이용할 경우에는 Qemu와 같이 하드웨어를 소프트웨어적으로 구현할 필요도 없고, VMWare처럼 binary translation 기법을 이용할 필요 없이 full-virtualization이 가능하기 때문에 실행 속도와 안티-디버깅 문제를 대부분 해결할 수 있다. 이에 본 논문에서는 악성코드 분석을 위하여 하드웨어 가상화 지원 기능을 활용하는 새로운 동적 코드 분석 시스템을 제안한다.

2. 관련 연구

동적 코드 분석 기법은 프로세스의 동작을 동적으로 분석하거나 애플리케이션 개발 과정에 활용되는 등 다양한 목적으로 활용된다. 본 논문에서는 악성코드 분석을 위한 목적으로 동적 코드 분석 시스템을 설계 및 구현하였으며, 이를 위해 악성코드, 동적 코드 분석 기법 구현을 위해 이용한 가상화 기술 및 동적 코드 분석 기법 등에 대하여 살펴본다.

2.1 악성코드의 위협

악성코드란 공격자의 의도에 따라 해당 시스템에 영향을 미치는 명령어들의 집합을 총칭하여 말한다. 최근 악성코드의 수많은 변종이 생성되고, 바이러스를 비롯 대부분의 악성코드들이 초창기처럼 단순히 감염만 시키는 모습을 모두 벗어버리고 빠른 확산력과 파괴력을 가지기 위해 다른 형태로 진화하고 있다. 컴퓨팅 환경이 인간의 생활과 밀접해 짐에 따라 이러한 악성코드들은 우리에게 많은 피해를 입히며, 이에 대한 대책이 시급히 요구되고 있다[1-3]. 특히, 악성코드 분석 및 대응을 회피하기 위하여 다양한 Anti-Unpacking 기술 또한 다양해 짐에 따라, 이에 대한 대비책 또한 필요하다. Anti-Unpacking 기술은 분석 방식에 따라 분석 회피기법(Anti-analyzing), 디버깅 회피기법(Anti-debugging), 가상화 회피기법(Anti-virtualization) 세 가지로 구분할 수 있다[4-9].

2.2 가상화 기술

가상화 기술은 한정된 물리적 자원을 하나의 서비스 혹은 여러 개의 서비스로 만들어 제공할 수 있는 중재자 역할을 수행하는 기술이다. 서버, 데스크탑, 운영체제, 애플리케이션, 스토리지 및 네트워크 등 다양한 분야에서 활용되고 있는 가상화 기술은 기존의 복잡했던 환경을 단순화시키며 작업처리의 분산 및 관리수단의 통합을 가능하게 하여 비용 절감과 관리 능률 향상 등 긍정적 효과를 창출하고 있다.

시스템 가상화 기술은 전가상화, 반가상화, OS-level 가상화의 3가지 형태로 분류할 수 있었다[10,11]. 하지만 최근 들어 하드웨어 지원 가상화 기술이 새롭게 등장함에 따라 기존의 전가상화/반가상화 기법의 한계를 극복할 수 있게 되었다[12]. 기존의 세 가지 유형들은 가상머신 마다 독립된 환경을 마련해주기 위해서 소프트웨어만을 사용하여 가상화를 구현하였다면, 하드웨어 지원 가상화는 하드웨어의 도움을 받아서 가상화를 구현하는 기법이다. 최근 Intel과 AMD에서는 서로 유사한 형태의 하드웨어 가상화 지원 기술들을 발표하고 있다. Intel에서 제공하는 VT-x라고 불리는 프로세서 가상화 기술은 현재 Intel 프로세서가 제공하는 실행 권한 4단계 위

에 VMX root/non-root 모드를 추가하여 호스트와 게스트가 독립적으로 기존의 실행 권한 4단계를 모두 사용할 수 있게 하였다. 그리고 VMX non-root 모드에서 동작하는 게스트가 특수 권한이 필요한 명령어를 수행하면 자동으로 예외를 발생시켜 VMX root 모드에서 동작 중인 호스트로 전환되어 필요한 작업들을 적절히 수행할 수 있게 된다. 하드웨어 가상화 지원 기술의 장점은 가상 머신의 구현이 쉽고, 여러 가지 하드웨어 가상화 지원 기술들이 접목되어 성능 저하가 거의 없이 게스트를 가상화 할 수 있다는 점이다.

2.3 동적 코드 분석 기법

2.3.1 에뮬레이터

대표적인 동적 코드 분석 기법으로 에뮬레이터를 사용하는 방식이 존재한다. 에뮬레이터 환경은 실제 시스템과 독립적으로 운영되기 때문에 프로그램을 실행시켜도 실제 시스템에는 영향을 미치지 않기에 이러한 장점을 살려서 다양한 목적으로 널리 사용되고 있다. 에뮬레이터는 하드웨어와 독립적으로 소프트웨어를 통한 가상의 컴퓨터 시스템을 제공하며, 이를 통해서 코드 수정 없이 리눅스와 다른 운영체제를 실행할 수 있다. 대표적인 에뮬레이터로는 QEMU과 Bochs가 있다. 이를 활용한 동적 코드 분석 기법으로는 BitBlaze[13], Renovo[14], VMScope[15], TTAnalyze[16] 등이 존재한다. 하지만 실행 속도가 매우 느린 단점이 있어서 이를 활용하는 데에 한계가 있다.

2.3.2 유저레벨에서 동작하는 동적 코드 분석기

또 다른 동적 코드 분석 기법으로는 유저레벨에서 프로그램의 실행과정을 모니터링하는 동적 코드 분석기가 존재한다. 일반적인 운영체제에서 사용자 모드로 동작하는 프로그램의 경우, 범용 레지스터와 사용자 영역의 주소 공간만을 접근이 가능하며 나머지 자원을 사용하기 위해서는 시스템 콜을 통해 처리한다. 반면 유저레벨 동적 코드 분석기를 통한 모니터링에서 사용자 컨텍스트는 동적 코드 분석기에 의해 새도우 레지스터와 메모리 영역을 통하여 관리하고, 시스템 콜 호출의 경우는 동적 코드 분석기의 리다이렉션을 통하여 커널에 의해 처리된다. 이를 통해 커널 안에서의 시스템 콜 처리를 제외한 응용 프로그램의 모든 코드를 동적으로 모니터링이 가능하다.

이러한 동적 코드 분석기에는 대표적으로 Valgrind와 Pin이 있다[17-19]. Valgrind는 프로그램의 메모리 누수와 잘못된 메모리 접근 등을 탐지하는 오픈 소스 프로그램이다. 다양한 플러그인을 통해 다른 기능을 추가가 가능하다. Pin은 인텔에서 개발한 동적 코드 분석기로써, JIT 컴파일러 기반으로 동작한다.

유저레벨 동적 코드 분석기를 이용한 악성 코드의 분석은 에뮬레이터를 이용한 분석에 비해 빠르며, 응용 프로그램을 대상으로 하기 때문에 프로세스 별 추적이 용이하다. 그러나 이를 위해서는 코드 변환이 필요하고, 이에 따른 악성코드의 디버깅 회피 기법에 노출되기 쉬운 단점이 있다.

2.3.3 머신레벨에서 동작하는 동적 코드 분석기

머신레벨에서 프로그램의 실행과정을 분석하는 동적 코드 분석 기법에는 대표적으로 Ether가 존재 한다. Georgia Institute of Technology에서 논문[20]으로 발표한 Ether의 경우, 본 논문에서의 접근방식과 동일한 방식으로 하드웨어 가상화 지원 기능을 사용하는 Xen을 이용한 동적 코드 분석 시스템이다. 가상 메모리 및 TSS 이용한 VAMPiRE의 은닉 브레이크 포인트를 활용하여, 메모리 추적, 시스템콜 추적, 프로세스 제한적 추적 등 기본적인 동적 코드 분석 시스템의 기능과 실행 압축, 안티-디버깅 기술에 대한 강점을 실험을 통해 보여주고 있다.

그림 1은 Ether의 시스템 아키텍처를 보여준다. Ether는 크게 Dom0에서 동작하는 유저 공간 컴포넌트와 Xen 하이퍼바이저 내에서 동작하는 컴포넌트로 나뉜다. Xen 하이퍼바이저 컴포넌트는 하드웨어 지원 가상화를 이용하여 DomU 윈도우즈 게스트들의 동작을 감시하며, 모든 실행 코드에 대한 정보를 로깅한다. Dom0의 유저 공간 컴포넌트는 로깅된 정보를 읽고 악성 코드 여부를 판단한다. 하지만 Ether에서는 DLL 및 API 호출 정보까지 분석해 내지 못하였다.

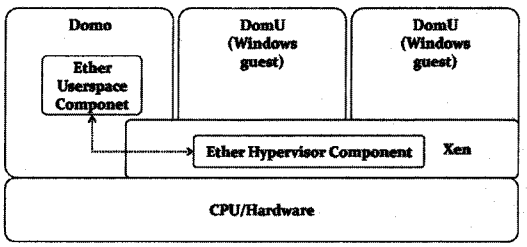

그림 1 Ether의 시스템 구조

3. 동적 악성코드 분석 시스템 설계 및 구현

가상 머신에서 동작하는 특정 프로세스를 분석하기 위한 동적 코드 분석기의 기본적인 요구사항으로는 크게 다음과 같은 내용들을 들 수 있다.

- 명령어 단위 분석
- 메모리 접근 추적
- API 호출 정보 추적

먼저, 특정 프로세스를 분석하기 위하여 명령어 단위 분석이 가능해야 한다. 싱글 스텝으로 프로세스의 동작

을 주시하며, 어떠한 동작을 하는지 모니터링 할 수 있어야 한다. 이처럼 모든 명령어 수행 과정을 분석할 수 있게 되면, 각각의 동작에 대응하여 필요한 작업을 할 수 있게 된다. 그리고 명령어 단위 분석과 더불어 메모리 접근 추적이 가능해야 한다. 예를 들어 메모리 접근 추적 기능을 이용하여 실행압축을 수행하는 악성코드에서 실행 압축이 풀리는 시점을 찾아내거나, GDT(Global descriptor table) 값을 확인 및 변경하는 등 여러 기능을 수행하기 위해서 메모리 접근 추적 기능이 필요하다. 또한, 앞서 두 가지 요구 사항과 더불어 프로세스에서 호출하는 API 호출 정보도 추출할 수 있어야 한다. DLL 인젝션을 수행하는 악성코드를 분석하거나 실행 코드에서 호출하는 다양한 함수들을 추적함으로써 프로세스 동작에 대한 정확한 분석 및 대응이 가능하게 된다.

3.1 동적 악성코드 분석 시스템의 구조

본 논문에서 제안하는 동적 악성코드 분석 시스템의 구조는 다음 그림 2와 같다.

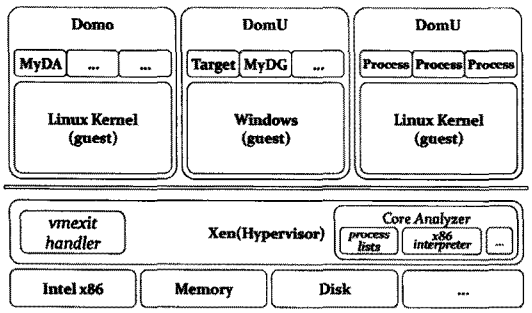

그림 2 제안 시스템의 구조

제안 시스템은 인텔 x86 프로세서에서 지원하는 VT 기술을 이용하여 Xen 기반으로 전가상화 기법을 구현하였다. 이를 기반으로 동적 코드 분석을 위해서는 3 개의 컴포넌트가 필요하다.

- MyDA
- Core Analyzer
- MyDG

먼저, MyDA는 Dom0에서 동작하며, 동적 악성코드 분석 시스템의 모니터링 역할을 하는 유틸리티 프로그램이다. 프로세스의 추가/삭제 등 동적 악성코드 분석 시스템의 동작들을 제어하며, 더불어 동적 악성코드 분석 시스템에서 추출한 로깅 정보들을 저장하는 기능도 한다. 그리고 또 다른 컴포넌트인 Core Analyzer는 Xen 하이퍼바이저 내부에서 동작하며, 게스트의 동작들을 싱글스텝을 통해 명령어 단위로 분석하여준다. 또한, 메모리 내용을 확인하고 메모리 값이 새로이 변경될 경우 이를 추적하는 기능도 제공하며, API 호출 정보 또한 분석하여 제공한다. 마지막으로 남은 하나의 컴포넌트인 MyDG는 게스트 안에서 윈도우즈 시스템 기반으로 동작하는 디버거 프로그램이다. 분석하고자 하는 특정 프로세스를 디버깅 모드로 실행하여 모듈 적재 내역을 추적한다.

3.2 동적 악성코드 분석을 위한 기본 기능

제안 시스템에서는 기본적으로 게스트에서 동작하는 특정 프로세스를 구분하여 분석 할 수 있으며, 각각의 프로세스들을 싱글스텝으로 명령어 수행과정을 분석한다. 또한, 임의의 메모리 내용 확인 및 변경 내용 추적 기능을 제공한다. 각각의 세부 동작 과정은 아래와 같다.

3.2.1 특정 프로세스 분석 기능

특정 프로세스를 분석하기 위한 시스템 구성은 그림 3과 같다.

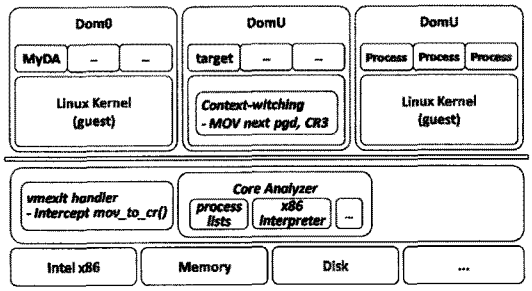

그림 3 특정 프로세스 분석을 위한 시스템 구성

우선, DomU (게스트)에서 동작하는 특정 프로세스를 분석하기 위해서, Dom0에서 MyDA를 실행시킨다. MyDA는 실행과 동시에 타겟 프로세스의 이름을 CoreAnalyzer에 있는 프로세스 목록에 추가한다. CoreAnalyzer는 DomU에서 동작하는 모든 프로세스를 관장하며, 그 가운데 프로세스 목록에 존재하는 프로세스만 상세히 분석한다. 이를 가능하게 하는 원리는 다음과 같다. 기본적으로 DomU에서 동작하는 프로세스들은 윈도우의 Context-switching을 통하여 실행 권한을 얻게 된다. Context-switching 과정에서 페이지 디렉토리를 가리키는 CR3 레지스터 값을 교체하는 명령어가 수행되게 되는데, 이때 DomU에서는 예외(exception)가 발생하며, Xen 하이퍼바이저에 있는 해당 예외(exception)에 대한 핸들러가 동작한다. 이 핸들러를 vmexit handler라고 지칭하며, vmexit handler는 mov_to_cr() 함수를 호출한다. mov_to_cr()는 Core Analyzer에 존재하는 프로세스 목록을 보고, 목록에 실행 권한을 얻게 될 프로세스의 이름이 존재하는지 여부를 확인한다. 존재한다면 DomU의 CPU eflags 레지스터에 트랩플래그를 설정하여, 해당 프로세스가 명령어를 수행할 때마다 디버그 예외를 발생시키도록 한다. 이를 통해서 Core Analyzer는

명령어 단위로 해당 프로세스를 분석하게 된다. 이후, vmexit handler에서는 vmresume 명령어를 수행하여 DomU의 Context로 전환시키며 실행권한을 DomU로 넘겨준다.

3.2.2 싱글 스텝

앞에서 간단하게 언급하였던 바와 같이, 게스트에서 동작하는 프로세스들을 명령어 단위로 분석하는 싱글 스텝은 게스트의 CPU eflags 레지스터의 트랩 필드를 이용한다. 게스트의 CPU에서 명령어를 실행시킬 때마다 디버그 예외가 발생하게 되면서, 하이퍼바이저에서 이를 분석할 수 있도록 한다. 싱글 스텝의 동작 과정에서의 시스템 구성은 그림 4와 같다.

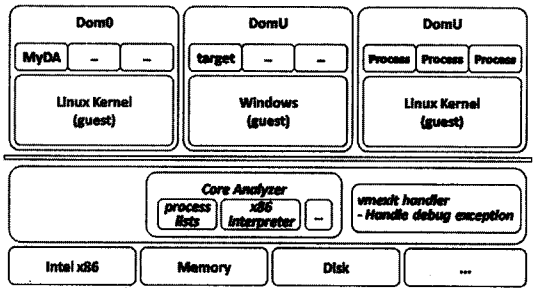

그림 4 싱글 스텝 동작 과정에서의 시스템 구성

이 기능은 다음과 같이 동작한다. DomU의 프로세스가 앞서 설명한 프로세스 분석 기능에 의해 분류되어 분석 대상 프로세스라는 가정 하에, vmexit handler에서는 DomU 의 CPU eflags 레지스터에 트랩 플래그를 설정한 후, vmresume 명령어를 수행한다. 이에 따라 실행 권한이 vmexit handler에서 DomU로 넘어가며 저장되어 있던 DomU의 context가 재실행하게 된다. Context-switching 과정에 있었던 DomU가 재실행하게 되면서 분석 대상이 되는 프로세스가 동작하게 된다. 이 때에는 프로세스의 명령어 실행 시마다 eflags 레지스터의 트랩 플래그 설정에 의해 디버그 예외가 발생하고, 이때 발생한 예외를 하이퍼바이저에 있는 vmexit handler가 받아서 처리한다. Core Analyzer에서는 vmexit handler로부터 필요한 정보를 받아서 해당 명령어를 분석하고, 분석 결과를 Dom0의 MyDA에게 전달하여 준다. 이후, vmexit handler는 다시 DomU의 CPU eflags 레지스터에 트랩 플래그를 설정하고 vmresume 명령어를 수행한다.

3.2.3 메모리 내용 확인 및 변경 내용 추적

게스트의 임의의 주소에 기록되어 있는 메모리 내용을 확인하고, 메모리 변경을 추적하기 위한 동작 과정은 다음과 같다. 먼저, DomU의 임의의 주소 내용을 확인하기 위하여 DomU의 페이지 디렉토리와 섀도우 페이지 디렉토리를 이용한다. 제안 시스템은 전가상화 기법으로 가상환경을 구축하여서 DomU의 메모리를 섀도우 페이징으로 관리하므로, DomU의 페이지를 관리하는 페이지 디렉토리와 게스트들의 메모리를 맵핑하여 관리하는 하이퍼바이저의 섀도우 페이지 디렉토리를 통해서 필요로 하는 실제 물리 메모리 내용을 확인 할 수 있다.

그리고 DomU의 메모리 변경 내용 추적하는 기능은 DomU에서 실행되는 메모리 읽기/쓰기 명령어를 분석하여 동작한다. DomU에서 동작하는 프로세스가 프로세스 분석 기능에 의해 분류되어 분석 대상 프로세스라는 가정 하에, 해당 프로세스에서 실행되는 모든 명령어는 디버그 예외를 발생시키며 Core Analyzer에 의해 분석된다. 디버그 예외가 발생하면 vmexit handler가 이에 대응하여 필요한 정보를 Core Analyzer에게 전해주게 되는데, 이때 실행된 명령어가 메모리 읽기/쓰기 명령어일 경우에는 Core Analyzer가 현재 메모리 값과 수정될 메모리 값을 분석하여 분석 결과를 MyDA로 전달해 준다. 메모리 변경 내용 추적 기능 또한 기본적으로 싱글 스텝 동작 과정 가운데 기능하므로, 분석된 내용을 MyDA로 전달한 후에는 vmresume을 실행하며 DomU로 전환된다.

3.3 동적 악성코드 분석을 위한 세부 기능

제안 시스템에서 제공하는 기본 기능들을 기반으로 프로세스 레벨 디버거를 이용하여 DLL 로딩 추적 및 API 호출 정보를 추출하여 준다. 지금까지 가상 머신에서 프로세스를 분석했던 기법들의 경우, 시스템마다 DLL EXPORT TABLE 위치가 상이하여 프로세스가 IMORT하는 DLL의 정보를 분석하는데 어려움이 있었다. 이러한 문제를 제안시스템에서는 MyDG를 이용하여 해결하며, 프로세스가 적재하는 DLL의 정보를 정확하게 분석하게 된다.

DLL 로딩 추적 및 API 호출 정보 추출 기능은 프로세스 레벨 디버거인 MyDG를 이용하여 수행하며, 기본 원리는 다음과 같다. Dom0에서 동작하는 MyDA와 DomU에서 동작하는 MyDG는 서로 연동하며 동작한다. 먼저, MyDG는 분석 대상이 되는 DLL 파일들을 읽어 들여 DLL 구성 정보를 획득한다. 이를 통해서 DLL 내부에서 각각의 API가 위치한 오프셋 값을 분석해 내고, 이후에 획득하게 되는 타겟 프로세스가 로딩한 DLL 로딩 주소 값을 기반으로 타겟 프로세스에서 호출하는 API 정보를 추출할 수 있게 된다. 다음은 MyDG를 통해 DLL 로딩 주소값을 추출하는 과정이다.

MyDG는 디버거 모드로 분석 대상 프로그램을 실행시키며, 해당 프로세스의 시작 주소에 소프트웨어 브레이크포인트를 설정한다. 디버거 모드로 프로그램 실행을 지속하여 분석할 경우 다양한 Anti-debugging 기법에

노출되므로, MyDG는 DLL 로딩 정보만 획득한 후 디버거 역할을 끝내도록 한다. 이와 같은 순서에 따라 디버깅 모드로 프로그램이 실행되면 DLL을 로딩할 때마다 이벤트가 발생하며 MyDG로 전달된다. 그리고 MyDG는 해당 이벤트를 분석하여 DLL의 이름과 메모리 주소를 MyDA로 전달함으로써 DLL 정보를 획득하게 된다. 모든 DLL 로딩이 완료된 후, 시작 주소에 설정해 놓은 소프트웨어 브레이크포인트로 인해 디버그 예외가 발생하게 된다. MyDG는 소프트웨어 브레이크포인트를 제거하고 기존 명령어를 복구해 준다. 그리고 MyDA는 Core Analyzer에 디버깅 중이었던 프로그램을 분석 대상 프로그램으로 등록하여, 해당 프로세스를 싱글 스텝으로 분석하도록 한다. 프로그램 등록 이후에는 싱글 스텝 과정에 따라 프로세스의 동작과정을 분석한다.

3.4 동적 악성코드 분석 시스템의 구현

제안 시스템은 Intel VT 기술이 지원되는 x86 프로세서 기반으로 VT 기술을 활용한 Xen 하이퍼바이저를 구축하고, 게스트에서 마이크로소프트사의 윈도우즈 XP가 동작할 수 있도록 구현하였다.

제안 시스템의 구현 알고리즘 흐름도는 다음 그림 5와 같다. 기본적으로 제안 시스템은 vmexit/vmresume의 호출 사이에서 동작한다. 우선, vmexit 발생 원인에 따라 해당 핸들러가 동작하게 되는데, context switching일 때에는 특정 프로세스 분석 기능에 해당하는 핸들러가 동작한다. 그리고 vmexit가 debug exception에 의해 발생했을 때에는 싱글 스텝을 처리하는 핸들러가 동작하며, 메모리 접근이나 API 호출일 때에는 추가적으로 이에 대한 핸들러가 동작하도록 구현하였다.

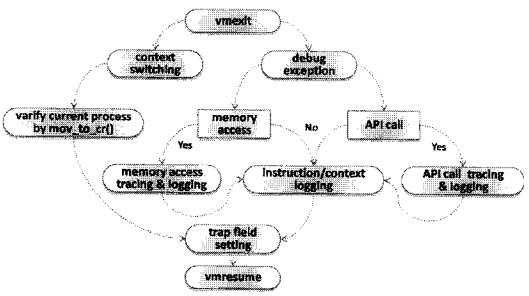

그림 5 제안 시스템의 구현 알고리즘 흐름도

4. 대표적 악성코드 분석 및 탐지를 통한 동적 악성코드 분석 시스템 검증

동적 분석 시스템에서 제공하는 기능들을 이용하여 대표적인 악성코드들을 분석 및 탐지해 봄으로써 본 제안 시스템의 성능을 검증한다. 테스트할 악성코드 유형으로는 실행압축을 사용하는 악성코드, DLL 인젝션 동작을 수행하는 악성코드, 그리고 Time Stamp Counter를 이용하여 Anti-Virtualization을 사용하는 악성코드를 선정하였다. 이를 통해 악성코드에서 수행하는 전형적인 행동을 제안 시스템에서는 어떻게 탐지하는지 살펴보며 동적 악성코드 분석 시스템으로써의 기능을 평가해 본다.

4.1 실행 압축 악성코드 분석

실행 압축된 프로그램의 경우, 압축을 푸는 과정 없이 바로 프로그램을 실행시킬 수 있다. 따라서 웹 환경에서는 실행 압축을 활용하여 빠르게 프로그램을 실행시킬 수 있게 되었고, 시스템에서도 이를 활용하여 메모리 공간을 효율적으로 사용할 수 있게 되었다. 하지만 악성코드들은 실행 압축된 프로그램들의 경우 분석하기 어렵다는 특징을 악용하여, 자신의 실행 코드를 실행 압축시켜서 분석 하는데 시간이 걸리도록 구현하기도 한다. 이러한 악성 코드의 실행 압축을 빨리 해제하여 분석할 수 있다면, 악성 코드의 전파를 최소화하고 피해 규모를 현저히 줄일 수 있게 된다. 이에 따라 실행 압축 악성코드를 제안 시스템을 이용하여 분석하여 본다.

제안 시스템에서 실행압축 프로그램을 분석하는 방법은 실행 압축 프로그램의 기본 동작 과정의 특성을 활용한다. 기본적으로 실행 압축 프로그램은 실행 압축 전의 프로그램을 복원시키기 위해서 특정 위치에 원래의 코드를 복사한다. 그리고 복원이 끝나면 복구된 코드를 OEP(Original Entry Point)부터 실행하게 되는데, 이 과정을 간단히 정리하면 Write 명령어를 통해 메모리에 새로운 값이 기록되고, 기록된 영역에서 명령어가 수행되는 시점에서 실행압축이 해제된다는 것이다. 실행 압축 악성코드가 이와 같이 동작한다는 점에 착안하여 제안 시스템에서는 분석 대상 프로세스가 특정 메모리 영역에 Write를 수행하는지 메모리 변경 추적 기법을 통해 모니터링하고, Write가 시행될 때마다 해당 주소 영역 정보를 로깅한다. 그리고 싱글 스텝으로 모든 명령어를 분석하면서 로깅된 영역에서 명령어가 수행되는지 모니터링 하고, 특정 시점에서 로깅된 영역에서의 명령어가 실행된다면 실행압축이 해제되고 원래의 실행 코드가 동작하는 것으로 판별할 수 있게 된다. 이처럼 코드를 복사 후, 해당 코드를 실행한다는 실행압축의 기본 동작 과정을 근간으로 제안 시스템에서는 실행압축의 동작을 분석해낸다.

실행 압축 검증 대상 악성코드는 www.offensivecomputing.net에서 다운로드 받았으며, 해당 코드 이름은 Trojan.Downloader.Zlob_r.DQ.EXE이다.

제안 시스템에서 실행 압축을 정상적으로 분석하는지 확인하기 위해서, 프로그램 디버깅을 위해 일반적으로 주로 사용되는 프로세스 레벨 디버거 Ollydbg를 이용하

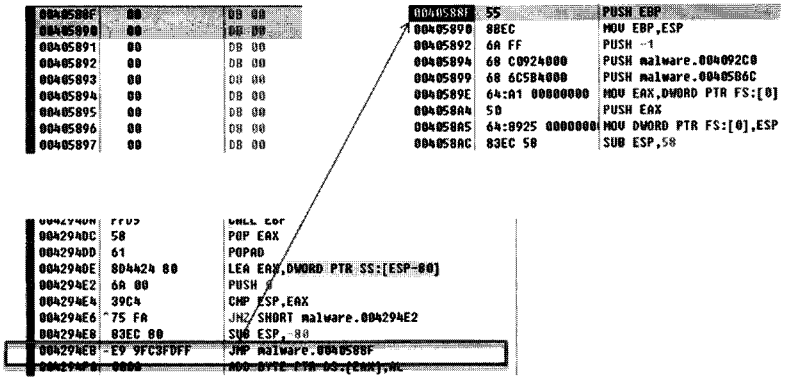

그림 6 프로세스 레벨 디버거 ollydbg를 이용한 실행압축 악성코드의 압축 해제 과정 분석

여 실행 압축 악성코드을 분석하였다. 타겟 악성코드는 UPX에 의해 실행 압축 되었으며 OEP(Original Entry Point)는 0x0040588F이다. 실행 초기 단계에서는 그림 6에서와 같이 OEP 위치 메모리에 0으로 초기화 되어있는 것을 확인할 수 있다. 그리고 실행압축이 풀리면서 해당 위치에는 본래의 코드가 복원되고, 0x004284EB에서 JMP 0040588F 명령어가 실행되면서 OEP로 이동할 시점에는 이미 모든 코드가 복원되어 있음을 확인할 수 있다.

제안 시스템에서는 앞서 설명한 바와 같이, 프로세스가 Write 하는 영역을 로깅하고 해당 영역에서 명령어가 실행되는지를 모니터링 한다. 싱글스텝으로 명령어 단위로 분석하면서 메모리 변경 내용 추적 기능을 이용하여, 표 1과 같이 실행압축이 해제되고 OEP부터 실행되는 명령어들을 정확하게 추출하였다.

표 1 실행 압축 분석 결과

```
<insn eip="0x40588f">push  %ebp</insn>
<insn eip="0x405890">mov   %esp, %ebp</insn>
<insn eip="0x405892">push  $0xFF</insn>
<insn eip="0x405894">push  $0x004092C0</insn>
<insn eip="0x405899">push  $0x00405B6C
</insn>
<insn eip="0x40589e">movl  %fs:0x00000000,
%eax</insn>
<insn eip="0x4058a4">push  %eax</insn>
<insn eip="0x4058a5">movl  %esp,
%fs:0x0</insn>
```

4.2 DLL 인젝션 악성코드 분석

제안 시스템에서는 MyDA를 통해 획득한 API 호출 정보를 이용하여, 해당 API의 인자 값 및 반환 값 등을 추출할 수 있다. DLL 인젝션의 경우, 감염대상 프로세스의 PID를 구한 이 후 CreateRemoteThread()와 같은 API를 통하여 원하는 DLL을 감염대상 프로세스에 로딩시키게 된다. 따라서 DLL 인젝션을 수행하는 악성코드가 감염 대상 프로세스의 PID를 구하기 위해 GetWindowThreadProcessId() 함수를 호출할 때, 제안 시스템에서는 반환 값에 해당하는 PID값을 탐지하여 감염대상 프로세스를 알아낼 수 있다. 이처럼 DLL 인젝션 수행과정에서 호출되는 API를 분석함으로써, DLL 인젝션의 발생 여부, DLL 인젝션의 동작과정 및 DLL 인젝션된 타겟 프로세스의 동작까지 분석할 수 있게 된다.

DLL 인젝션 수행을 검증하기 위한 대상 악성코드는 www.offensivecomputing.net에서 다운로드 받았으며, 해당 코드 이름은 Trojan.Injector.AQ4.EXE이다.

실행 압축에서와 마찬가지로 프로세스 레벨 디버거인 ollydbg를 이용하여 DLL 인젝션 악성코드를 분석하였다. 그림 7에서 보는 바와 같이 0x00604B28에서 GetwindowThreadProcessID() 함수를 통해 Exeplore.exe 프로세스의 PID를 구하는 것을 확인할 수 있다. 그리고 Explore.exe 프로세스의 가상메모리 공간을 확보하고, kernel32.dll을 확보된 공간에 적재하기 위해서 OpenProcess(), GetProcAddress(), WriteProcessMemory() 등을 호출하는 것을 확인할 수 있다. 마지막으로 Exeplore.exe 프로세스의 주소 영역에서 thread를 생성하기 위해 CreateRemoteThread()가 호출되는 것을 확인할 수 있다.

제안 시스템에서 테스트 대상 악성코드를 분석한 결과, 표 2와 같이 악성코드에서 호출된 API 정보를 구할 수 있었으며, 이를 통해 DLL 인젝션 된 프로세스의 PID 값이 1324임을 알아내었다. 이 후, PID 1324 프로세스를 분석대상 목록에 추가함으로써, 표 3과 같이 DLL 인젝션된 이후 PID 1324 프로세스의 명령어 수행 과정을 추출할 수 있다.

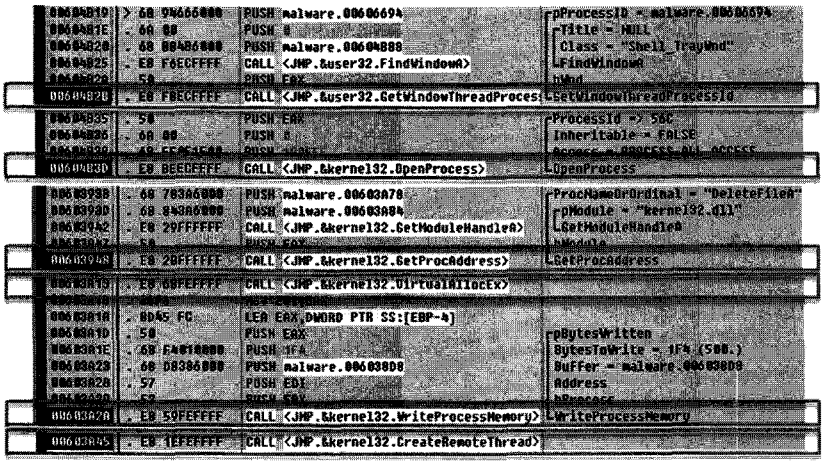

그림 7 테스트 대상 프로그램에서 DLL 인젝션을 위해 호출하는 API 탐색 결과

표 2 악성코드에서 호출된 IsDebuggerPresent(), GetWindowThreadProcessId() 인자 정보

```
<?xml version="1.0" encoding="utf-8"?>
<myda>
<check pid="1888" name="IsDebuggerPresent"
present="0"></check>
<check pid="1888"
name="GetWindowThreadProcessId"
target_pid="1324"></check>
</myda>
```

표 3 DLL 인젝션 수행된 이후 타겟 프로세스의 명령어 수행 과정 분석 결과

```
<insn eip="0x40588f">push  %ebp</insn>
<insn eip="0x405890">mov  %esp, %ebp</insn>
<insn eip="0x405892">push  $0xFF</insn>
<insn eip="0x405894">push  $0x004092C0</insn>
<insn eip="0x405899">push  $0x00405B6C</insn>
<insn eip="0x40589e">movl  %fs:0x00000000,%eax
</insn>
<insn eip="0x4058a4">push  %eax</insn>
<insn eip="0x4058a5">movl  %esp,%fs:0x0</insn>
<insn eip="0x4058ac">sub  $0x58, %esp</insn>
<insn eip="0x4058af">push  %ebx</insn>
<insn eip="0x4058b0">push  %esi</insn>
```

4.3 RDTSC를 이용한 가상머신 탐지 악성코드 분석

최근 가상머신 관련 기술이 발달하여 단일 시스템과 비교될 정도로 성능이 향상되었지만, 아직까지는 특정 명령어 수행 및 전체적인 성능 면에 있어서 단일 시스템에 비해 부족한게 사실이다. 이러한 특성을 활용하여 어떤 악성코드들은 특정 명령 수행 시간을 점검하여 자신이 가상머신 위에서 동작하는지 확인 후, 가상머신에서 동작할 경우 원래의 실행코드가 아닌 다른 루틴을 수행한다. 일반적으로 가상환경에서 악성코드를 분석 및 탐지하므로, 악성코드의 이러한 의도를 탐지해 내지 못할 경우 악성코드의 함정에 빠지게 된다. 이러한 목적으로 악성코드들은 주로 RDTSC 명령어를 이용한다. RDTSC 명령어는 프로그램을 수행하는 과정에서 런타임 레벨에서 시간을 측정하고자 할 때 사용된다.

제안 시스템에서는 RDTSC 명령어 수행 시, 반환 값을 조정하여 가상 머신 탐지 기술을 회피한다. 일반적으로 RDTSC 명령어를 수행할 경우, EDX:EAX 레지스터에 time stamp 값을 저장한다. 이를 이용하여 가상머신 탐지 기술은 두 번의 RDTSC 명령어 수행하여 반환 값을 구한 후, 반환 값의 차이를 기준으로 가상 머신 존재 유무를 탐지한다. 이를 역이용하여 제안 시스템에서는 첫 번째 RDTSC 명령어 수행은 정상적으로 동작하도록 한 이후, 두 번째 RDTSC 명령어가 수행되면 첫 번째 반환 값과 근소한 값의 차이가 나도록 값을 조정하여 반환하도록 제어한다. 이는 제안 시스템의 싱글스텝 분석과정에서 하이퍼바이저에 존재하는 CoreAnalyzer를 통해 간단하게 처리할 수 있다. 따라서 제안 시스템에서는 가상 머신의 동작과 관계없이 가상 머신 탐지 기법을 회피할 수 있게 된다.

프로세스 레벨 디버거 Ollydbg에 의한 분석 결과는 그림 8과 같다. 0x00402493에서 RDTSC를 수행한 이후, 가상 머신에서 처리 시간이 상대적으로 오래 걸리는 CPUID를 수행하도록 하여 0x00402499에서 RDTSC 수행 했을 때 시간 차를 크게 함을 확인할 수 있다. 두 번의 RDTSC 수행 결과에 따라, 시간 차가 클 경우에는

그림 8 ollydbg를 이용한 가상머신 탐지 악성코드 분석 결과

표 4 RDTSC 회피한 시점의 실행 명령어 추출 내용

```
<insn pid="1504" tid="1500" eip="0x4024a2"
esp="0x12ffa4">ja  0xFFFFFFF1</insn>
<regs pid="1504" eax="0x1" ebx="0x0" ecx="0x0"
edx="0x1d" edi="0x0" esi="0xc957edbb"></regs>
<insn pid="1504" tid="1500" eip="0x4024a4"
esp="0x12ffc4">popa  </insn>
<regs pid="1504" eax="0x0" ebx="0x7ffd5000"
ecx="0x12ffb0" edx="0x7c93eb94" edi="0x0"
esi="0x9"></regs>
<insn pid="1504" tid="1500" eip="0x4024a5"
esp="0x12ffc4">ljmp  0xFFFFFF84</insn>
<regs pid="1504" eax="0x0" ebx="0x7ffd5000"
ecx="0x12ffb0" edx="0x7c93eb94" edi="0x0"
esi="0x9"></regs>
```

가상 머신 임을 탐지하여 0x004024A2의 JA 명령어를 통해서 0x00402493으로 점프하고, 시간 차가 작을 경우에는 0x004024A4의 POPAD 명령어가 수행되게 된다.

제안 시스템에서 테스트 대상 악성코드를 분석한 결과, 표 4와 같이 EIP="0x4024a4" 의 POPAD 명령어가 수행됐음을 확인할 수 있다. 즉, 악성코드는 가상 머신이 존재하는 것을 탐지하지 못하였음을 알 수 있다.

5. 결 론

본 논문에서는 악성코드에서 쉽게 탐지할 수 없는 가상 머신을 구축하여, 악성코드들을 동적 분석 및 탐지할 수 있는 시스템을 구현하였다. 가상 머신의 구현상 한계를 극복하고자 최근 프로세서 밴더들에서 제공하는 하드웨어 가상화 지원 기술을 이용하여 가상 머신을 구축하였다. 이를 기반으로 제안 시스템에서는 가상 머신에서 동작하는 프로세스의 명령어 수행 및 메모리 접근 정보를 추적하고 분석해 준다. 또한, 가상머신에서 실행되고 있는 프로세스의 각종 API 호출 정보도 제공해 준다.

제안 시스템의 성능평가를 위해 실행압축, DLL 인젝션, RDTSC를 활용한 가상머신 탐지 등 대표적인 악성코드 유형들에 대하여 실험해 보았다. 실험 결과 제안 시스템의 악성코드 분석 기능들이 모두 정확하게 동작함을 확인할 수 있었으며, 이를 통해 제안 시스템이 악성코드 분석을 위한 필요 기능들을 충분히 구비했음을 확인할 수 있었다. 실행압축 수행하는 악성코드에 대해서는 실행 압축 해제되는 시점을 분석하여, 압축 해제된 이후 수행된 명령어 및 실행 정보들을 추출해 준다. 그리고 DLL 인젝션 수행하는 악성코드가 동작할 때는, 인젝션된 타겟 프로세스의 동작을 분석하여 프로세스 실행 정보를 보여준다. RDTSC를 이용한 가상머신 존재 유무를 탐지하는 악성코드에 대해서도 안티-언패킹 회피기술을 반영하여 악성코드가 제안시스템을 탐지하지 못함을 보여주었다.

이처럼 제안 시스템에서는 안티-언팩킹 회피 기술을 반영한 자동화된 악성코드 분석 및 탐지 기능을 제공하며 기존 악성코드 분석 기법의 한계점을 개선하였다. 또한, 기존의 가상 머신 기반 악성코드 분석 기법들과는 달리 실행 중인 프로세스의 다양한 API 호출 정보를 추출할 수 있게 됨에 따라, 안전한 환경에서 악성코드들을 보다 정확하고 빠르게 분석하고 탐지할 수 있게 되었다.

참 고 문 헌

[1] N. Idika, and A. P. Mathur, "A Survey of Malware Detection Techniques," *Research, Dept. of Computer Science*, Purdue Univ., 2007.

[2] H. Carvey, "Malware analysis for windows administrators," *Digital Investigation*, vol.2, pp.19-22, 2005.

[3] C. P. Pfleeger, and S. L. Pfleeger, security in Computing, Prentice hall, 2003.

[4] T. Garfinkel, K. Adams, A. Warfield, J. Franklin, "Compatibility is Not Transparency: VMM Detection Myths and Realities," *Proc. 11th Workshop on Hot Topics in Operating Systems*, 2007.

[5] P. Ferrie, "Anti-unpacker tricks," CARO Workshop, 2008.

[6] P. Ferrie, "Attacks on Virtual Machines," AVAR Conf., pp.128-143, 2006.

[7] T. Listion, and E. Skoudis, "On the Cutting Edge: Thwarting Virtual Machine Detection," SANS Internet Storm Center, 2006.

[8] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Morden Malware," *DSN2008*, pp.117-186, 2008.

[9] M. Xu, V. Malyugin, J. Sheldon, G. Venkitachalam, and B. Weissman, "ReTrace: Collecting Execution Trace with Virtual Machine Deterministic Replay," *Proc. of 2007 Workshop on Modeling, Benchmarking and Simulation*, 2007.

[10] VMware, Inc. "Understanding Full Virtualization, Paravirtualization, and Hardware Assist," http://www.vmare.com, 2007.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A.

Warfield, "Xen and the Art of Virtualization," *Proc. of the 19th ACM Symposium on SOSP*, 2003.
[12] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization," *Intel Technology Journal*, pp.167-177, 2006.
[13] BitBlaze Binary Analysis Platform. http://bitblaze.cs.berkeley.edu.
[14] M. G. Kang, P. Poosankam, and H. Yin, "Renovo: A Hidden Code Extractor for Packed Executables," *Proc. of WORM*, 2007.
[15] X. Jiang, X. Wang, and D. Xu, "Stealthy Malware Detection Through VMM-Based "Out-of-the-Box" Semantic View Reconstruction," *Proc. of CCS*, pp. 128-138, 2007.
[16] U. Bayer, C. Kruegel, and E. Kirda, "TTanalyze: A Tool for Analyzing Malware," *Proc. of EICAR*, pp.180-192, 2006.
[17] Instrumentation Framework for building dynamic analysis tools, http://valgrind.org.
[18] A Dynamic Binary Instrumentation Tool, http://pintool.org.
[19] A. Vasudevan, R. Yerraballi, "SPiKE: engineering malware analysis tools using unobtrusive binary-instrumentation," *Proc. of the 29th ACM International Conference*, vol.171, 2006.
[20] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware Analysis via Hardware Virtualization Extensions," *Proc. of ACM CCS*, 2008.

김 태 형
2007년 성균관대학교 컴퓨터공학과 졸업(학사). 2009년 성균관대학교 전자전기컴퓨터공학과 졸업(석사). 2009년~현재 성균관대학교 전자전기컴퓨터공학과 박사과정. 관심분야는 시스템보안, 가상화, 운영체제

김 인 혁
2006년 성균관대학교 전자전기컴퓨터공학과 졸업(학사). 2010년 성균관대학교 전자전기컴퓨터공학과 졸업(석사). 2010년~현재 성균관대학교 전자전기컴퓨터공학과 박사과정. 관심분야는 시스템보안, 가상화, 운영체제

엄 영 익
정보과학회논문지 : 시스템 및 이론
제 37 권 제 4 호 참조

김 원 호
2002년 한동대학교 전산전자공학부 졸업(학사). 2004년 포항공과대학교 전자컴퓨터공학부 졸업(석사). 2004년~현재 한국전자통신연구원 부설연구소 선임연구원
관심분야는 시스템보안, 역공학, 가상화