情報保護學會論文誌
第16卷 第3號, 2006. 6

우회기법을 이용하는 악성코드 행위기반 탐지 방법*

박 남 열†, 김 용 민, 노 봉 남‡
전남대학교

A Behavior based Detection for Malicious Code Using Obfuscation Technique*

Nam-Youl Park,† Yong-Min Kim, Bong-Nam Noh‡
Chonnam National University

요 약

우회기법을 사용하는 변종 악성코드의 출현은 바이러스 백신에 의한 탐지를 우회하여 확산을 가속화시키고 있다. 만일 보안 취약점 패치가 되지 않고, 바이러스 백신 패턴에 포함되지 않았을 경우에 신종 웜은 수분 내에 단위 네트워크상의 시스템을 감염 시킬 수 있으며, 다른 지역 네트워크로의 확산도 가능하다. 따라서 변종 및 신종 악성코드에 대한 기존의 패턴기반 탐지 및 치료 방식에는 한계가 있다. 본 논문에서는 실행 압축의 우회기법을 사용한 악성코드에 대하여 행위기반의 정적 및 동적 분석에 의한 탐지패턴을 생성하고, 동적 탐지에 의한 변종 및 신종에 대한 탐지 방법을 제안한다. 또한 동적 탐지에서의 유사도 비교를 위한 방법을 제안하여 시스템의 중요자원에 접근하는 실행압축 기법을 사용하는 바이러스의 탐지가 가능함을 보인다.

ABSTRACT

The appearance of variant malicious codes using obfuscation techniques is accelerating the spread of malicious codes around the detection by a vaccine. If a system does not patch detection patterns for vulnerabilities and worms to the vaccine, it can be infected by the worms and malicious codes can be spreaded rapidly to other systems and networks in a few minute. Moreover, It is limited to the conventional pattern based detection and treatment for variants or new malicious codes. In this paper, we propose a method of behavior based detection by the static analysis, the dynamic analysis and the dynamic monitoring to detect a malicious code using obfuscation techniques with the PE compression. Also we show that dynamic monitoring can detect worms with the PE compression which accesses to important resources such as a registry, a cpu, a memory and files with the proposed method for similarity.

Keywords : 악성코드, 행위기반, 우회기법, 유사도 비교

Ⅰ. 서 론

악성코드의 위협은 최근 인터넷 보안에서 가장 큰 위협 중 하나로 평가되고 있다. 초기 악성코드의 유일한 형태는 바이러스였지만, 최근에는 네트워크로 확산되는 웜, 트로이 목마, DDOS 에이전트, IRC

접수일: 2005년 12월 23일; 채택일: 2006년 6월 9일
* 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성지원 사업의 결과로 수행되었음.
† 주저자, hodumaru@lsrc.jnu.ac.kr
‡ 교신저자, bongnam@jnu.ac.kr

기반의 봇(bot), 스파이웨어의 다양한 형태를 보이고 있다. 또한 악성코드는 메일, 운영체제 취약점, 알려진 취약점 공격도구(exploit), P2P 네트워크를 이용하는 등 확산 방법도 다양해졌다. 정상적인 인터넷 사용자들은 온라인상에서 악성코드와 만나게 될 가능성이 높아졌고, 사용자도 인지할 수 없도록 다양한 방법으로 감염을 유도하고 있다. 대부분의 악성코드는 안티 바이러스 소프트웨어와 스파이웨어 제거 도구 등으로 치료하지만, 최근의 악성코드는 분석을 어렵게 하기 위해 실행압축으로 바이러스를 변조하여 안티 바이러스 패턴 제작을 지연시켜 전파 시간을 늘리는데 이용하고 있다. 변조된 바이러스는 기존 안티 바이러스 패턴으로는 탐지 및 치료가 되지 않으며, 새로운 패턴으로 추가되어야 한다.

이러한 추세를 개선하기 위한 방법으로 실행압축 등의 우회기법을 사용하는 악성코드의 탐지패턴 생성 및 탐지 방법에 대한 연구가 필요하다. 본 논문에서는 정적 및 동적 분석을 통해 실행압축으로 변조된 바이러스에 대해 패턴을 생성하고 추출한 주요자원에 접근하는 행위기반 동적 탐지와 WIN32 API간의 패턴 유사도 비교를 통하여 정확도를 향상시키는 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구에 대하여 기술하고, 3장은 정적 및 동적 분석을 이용하여 악성코드 탐지를 위한 패턴생성 및 탐지과정에 대해 설명한다. 4장에서는 악성코드 유형별 유사도 비교에 대한 실험결과를 보이고, 5장에서는 결론 및 향후 연구 방향에 대해 기술하였다.

II. 관련 연구

2.1. 악성코드 탐지 방법

웜이나 바이러스가 실행압축을 사용하는 이유는 네트워크를 통하여 많은 곳으로 전염될 수 있도록 백신제작자들로 하여금 악성코드를 발견하거나 분석하기 어렵게 하기 위함이다. 코드를 분석하는 동안 악성코드 전파 시간을 벌 수 있기 때문이다. 실행 압축은 ZIP, RAR과 같이 데이터를 하나로 묶어 놓은 압축과 달리 대상이 실행할 수 있는 파일을 압축하는 것으로 압축을 해제하는 과정 없이 바로 프로그램을 실행할 수 있다[1,2]. PE(Portable Executable) 파일은 Win32에서 사용하고 있는 파일 구조로 Win32 실행파일이 적재되는 가상주소, Import 함수 목록, Export 함수 목록, 데이터, 코드 등의 정보를 관리하기 위해 파일의 첫 부분에 여러 가지 구조체 묶음으로 되어 있다. PE 파일 포맷은 COFF (Common Object File Format)라는 포맷을 계승한 파일 포맷으로서 COFF의 확장이다[3].

악성코드의 탐지방법은 일반적으로 알려진 악성코드에 대한 탐지방법과 알려지지 않은 악성코드에 대한 탐지방법이 있다. 알려진 악성코드에 대한 탐지방법은 패턴매칭에 의한 방식으로 진행된다. 먼저 악성코드가 확산되면 이들에 대한 정보를 수집 및 분석하여 시그니처(signature)를 생성한 뒤, 의심되는 파일과 비교하여 탐지하는 방식이다[4]. 실행압축 되지 않은 파일 상태에서는 매우 효율적인 탐지방법이고, 현재 대부분의 안티바이러스 제품군에서 채택되고 있는 방식이다. 이러한 방식은 침입탐지 범위에서 오용행위 탐지와 마찬가지로 패턴이 있는 경우에 정확한 탐지가 가능하지만, 알려지지 않은 패턴에 대해서는 탐지가 불가능하다. 따라서 신규로 발생하는 악성코드나 변종을 탐지하지 못하여 네트워크로 급속히 확산되는 악성코드를 대처하는데 한계가 있다고 할 수 있다.

알려지지 않은 악성코드에 대한 탐지에는 바이너리에 대한 패턴이 아닌 어셈블리 수준 및 API 호출의 순서를 이용한 탐지 방법을 사용한다. Wisconsin 대학의 SAFE(Static Analyzer For Executables)[5]는 변조 바이러스를 어셈블리 수준에서 탐지하기 위한 방법을 제시하고 있다. SAFE에서 사용된 방법은 바이너리에서 어셈블리 코드를 추출하고, 추출한 코드에서 변조 부분을 주석자(annotator)에 의해 우회된 코드를 추상화한 후, 악성코드에서 보이는 패턴이 있는지 비교하여 탐지하는 방식이다. 이 방식은 바이러스 검출을 어렵게 하는 nop 코드 삽입, 데이터 수정, 제어흐름 수정, 데이터와 제어흐름 수정, 포인터 별명 사용 등의 단순한 변조 방식을 이용한 우회 기법에 대한 탐지는 가능하지만 실행압축에 의해 변조된 방식은 탐지가 불가능하다. New Mexico 대학에서 진행하는 SAVE (Static Analyzer of Vicious Executables)[6]는 알려진 바이러스의 API 순서와 PE 실행파일에서 추출한 API 순서 간의 유사성을 측정하는 방식으로 바이러스를 탐지한다. 유사성을 측정하기 위해 코사인(cosine) 방식, 확장된 자카드(extended jaccard) 방식, 피어슨 상관관계(Pearson's correlation) 방식을 혼용하여 사용한다. 유사도 비교 시, 세 가지

방식에 대해 모두 계산한 뒤에 임계치를 부여하여 임계치를 초과한 경우 바이러스로 탐지하게 된다. SAVE는 PE 파일의 실행압축을 해제한 뒤에 유사성을 이용하여 변조된 바이러스에 대한 탐지방법을 제시한다. 그러나 실행압축에 대한 알고리즘을 모두 보유하고 있지 않을 경우에는 특정 바이러스에 대한 탐지만 가능하다. 그리고 유사도 비교 방식 중 피어슨 상관관계 방식과 코사인 방식은 정확히 일치하지 않더라도 일정한 간격으로 패턴이 반복될 경우에 높은 유사도 수치로 계산되어 바이러스가 아닌 경우에도 바이러스로 탐지될 수 있다[7].

2.2. 정적 분석과 동적 분석

악성코드 탐지를 위한 주요한 접근방법으로 정적 분석(static analysis)과 동적 분석(dynamic analysis)이 있다. 정적 분석은 프로그램을 실행하지 않고 이 프로그램의 동적인 실행의 특성을 확인하기 위하여 프로그램의 코드를 검사하는 것이다. 이 기법은 다양한 분석과 코드의 최적화를 위해 컴파일러 개발자들에 의해 널리 사용되었고, 프로그램의 이해와 소프트웨어 시스템의 역공학을 위해 사용된다.

정적 분석의 결과는 프로그램에서 사용하는 중요 문자열, 삽입된 DLL(Dynamic Link Library), 함수 호출, 제어문 등을 식별할 수 있다. 그렇지만 제어문은 단지 변수를 어떻게 처리할 것인지에 관한 명령이며, 함수 호출에 있어서 입출력 변수들은 확인하는데 어려움이 많기 때문에 프로그램을 실행하지 않은 상태에서 그 동작을 정확히 알기는 어렵다[8].

정적 분석은 프로그램의 특정한 실행 조건에 제한되지 않고, 프로그램의 모든 실행에 대해 분석할 수 있다는 점에서 철저한 분석이 가능하다. 그리고 악성코드의 위험성을 실행하기 전에 검토하여 판단할 수 있으며, 실행에 있어서 과부하가 없다는 장점이 있다.

반면 동적 분석은 주로 악성 행위를 발견하기 위하여 프로그램의 실행을 감시하고, 추적하는 것이 주목적이다. 동적 분석은 정적 분석에 비해 다음의 이점을 가진다[9,10].

첫째, 동적 분석은 PE 파일의 변경에 영향을 받지 않는다. 백신을 우회하기 위해 기본적으로 사용되는 nop 삽입, 데이터 변경, 제어흐름 변경, 데이터와 제어흐름 변경, 포인터 별명의 사용과 수십여 가지의 실행압축을 사용한 방법은 정적 분석을 어렵게 하는 요인이 되고 있다. 특히 실행압축에 의한 변경은 해당 실행압축을 해제하는 알고리즘을 가지고 있지 않은 이상 정적 분석이 불가능하다는 문제가 있다.

둘째, 동적 분석은 자원별 접근형태를 정확히 확인할 수 있다. 정적 분석은 API 호출정보를 토대로 자원에 대한 접근 정보를 반복 추적해야 하는 부담이 있고, 문자열에 대해 암호화를 사용할 경우에는 별도의 복호화 프로그램을 사용해야 하는 등, 정확한 정보획득에 어려움이 많다.

본 논문에서는 정적 및 동적 분석으로 악성코드의 행위패턴과 순차패턴을 생성하고, 생성된 행위패턴을 근거로 동적 탐지를 통하여 행위기반 탐지를 수행하며, 순차패턴과 API 호출 순서 간 유사도 비교로 탐지의 정확성을 높이는 방법을 제안한다.

Ⅲ. 악성코드 패턴생성 및 탐지

3.1. 악성코드와 API간 인과성 분석결과

악성코드의 특징을 추출하기 위해서는 악성코드와 WIN32 API의 인과성(causality)을 찾을 필요가 있다. 전체 API를 대상으로 악성코드를 탐지할 경우에 발생하는 부하 문제가 있을 뿐만 아니라, 악성코드의 행위 특성상 악성코드가 시스템에 영향을 지속적으로 미치기 위해 주요자원에 접근하여 자신의 정보를 남기기 위한 행위를 수행하려 하므로, 이러한 인과성을 분석함으로써 악성코드의 행위를 설명할 수 있고, 악성코드를 효과적으로 탐지하기 위한 API를 선택할 수 있다. 대상 API는 중요자원에 접근하려 하는 행위를 수행하는 함수를 대상으로 하였으며, 악성코드가 접근하여 시스템에 피해를 입힐 수 있는 자원은 파일 시스템, 메모리, 프로세스, 훅킹, 레지스트리, 서비스, 네트워크를 대상으로 설정하였다. 선택된 API와 API의 순서들의 조합으로 변종 또는 새로운 악성코드를 탐지하거나, 이와 같은 것을 탐지하기 위한 패턴 생성에도 효과적이다. 또한 대량의 API 호출 데이터 중에서 관련 API만을 감시함으로써 작업의 부하를 줄일 수 있고, 실시간으로 악성코드를 탐지할 수 있다.

표 1은 netsky, sasser, raleka, sdboter, agobot의 악성 프로그램과 정상 프로그램인 메모장과 아웃룩 익스프레스를 포함하여 호출되는 API를 분석한 결과이다. 전체 WIN32 API 중 주요자원에 접근하는 API를 대상으로 분석한 결과, 악성코드가

실행될 때 호출되는 30개의 API를 대상으로 주요자원에 접근하려는 API는 ○로 표시하고, 호출은 되었지만 주요자원에 접근하지 않는 API는 △로 표시하였으며, 호출되지 않은 경우에는 표시하지 않았다. 결과에서 나타나듯이 정상적인 프로그램은 해당 API를 사용하지만 주요 자원에 접근하려는 행위는 없었고, 악성코드는 API를 사용함과 동시에 주요 자원에 접근하여 정보를 기록하려 하는 행위가 빈번하게 발생하였다. 대부분의 악성코드에서 사용되는 공통된 API를 선별하고, 선별된 API를 대상으로 탐지패턴을 생성하여 탐지할 경우에 부하를 최소화하고 탐지 효율을 극대화할 수 있다.

3.2. 악성코드 탐지패턴 생성 구조

본 논문에서 제안하는 악성코드 탐지패턴 생성 구조는 그림 1과 같이 정적 및 동적 분석을 통한 악성코드의 행위 및 순차패턴 생성과 이 두 가지 패턴의 통합과정을 통해 시그니처 DB를 생성하고, 이 시그니처 DB는 동적 탐지를 위해 사용되어진다.

행위패턴(behavior pattern)은 동적 분석을 통해 생성되고, 분석 시에 확산 방지를 위해 PE 실행파일 분석을 위한 환경이 구축되어 있는 안전한 조건하에 실행한 뒤[10], API의 상세한 정보를 추적한다. 생성된 API의 상세정보에서 악성코드와 인과성이 높은 주요 API를 대상으로 하여 정보를 걸러내는 API 필터링 과정과 주요자원에 접근하려는 인자에 대한 내용만을 추출하기 위한 인자 필터링과정을 거쳐 행위 기반 패턴을 생성해낸다.

순차패턴(sequence pattern) 생성은 동적 API 추적정보를 기초로 정적 분석 시에 추출된 PE 헤더 구조를 분석하여 IAT(Import Address Table)에

표 1. 악성코드와 API간의 인과성 분석결과. 1: Outlook, 2: Notepad, 3: Bagle, 4: Netsky, 5: Sasser, 6: Raleka, 7: Sdboter, 8: Agobot

No	자원	API	1	2	3	4	5	6	7	8
1	File System	CreateFile	△	△		○		○	○	○
2		CopyFile			○	○	○		○	○
3		MoveFile								
4		FindFirstFile	△	△	○	○		○	○	
5		FindNextFile	△	△	○	○			○	
6		DeleteFile			○					○
7	Memory	WriteProcessMemory	△	△	△	△	△	△	△	△
8		ReadProcessMemory	△	△	△	△	△	△	△	△
9	Process	CreateProcess			○				○	○
10		OpenProcess			○	○				○
11		TerminateProcess			○	○				○
12		GetStartupInfo				△	△			△
13		AbortSystemShutdown					○	○		
14		CreateThread					○	○	○	
15		CreateRemoteThread								△
16	Hook	SetWindowsHook			○	○		○		○
17		UnhookWindowsHook			○	○				○
19	Registry	RegCreateKey	△	△	○	○		○	○	○
20		RegOpenKey	△	△	○	○	○	○	○	○
21		RegSetValue	△	△		○	○	○	○	○
22		RegDeleteKey			○	○				
23		RegDeleteValue			○	○		○	○	○
24	Service	CreateService				○		○	○	○
25		DeleteService								○
26	Network	inet_addr	△		○	○	○	○	○	○
27		htons	○			○	○	○	○	○
28		gethostbyname			○	○				
29		Bind				○	○	○	○	○
30		Connect	△		○	○	○	○	○	○

서 중요한 API 정보를 분류한 뒤, 각각의 순서번호를 부여한 API ID와 행위패턴 생성 시에 추출된 호출되는 API의 순서를 일치시켜 API 순차패턴을 생성한다.

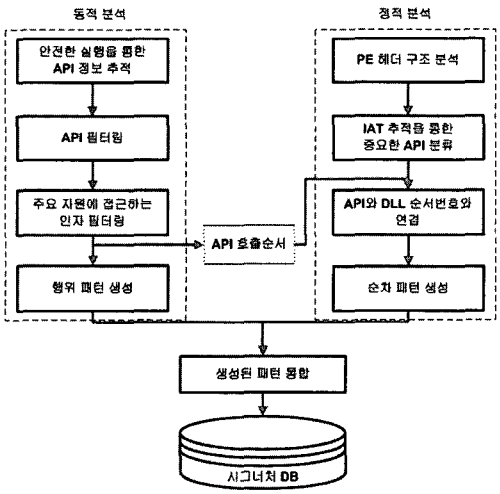


그림 1. 악성코드 패턴 생성

API ID는 표 2와 같이 DLL에 번호를 부여하고 API 별로 유일하게 식별 가능한 순서번호(ordinal number)를 연결하여 만들어 진다.

표 2. DLL, API를 이용한 API 순차패턴 생성

포함DLL	API Name	DLL No	Ord No	API ID
Kernel32	CreateFile	1	84	1.084
	CopyFile		68	1.068
	MoveFile		623	1.623
	...		...	...
	OpenProcess		647	1.647
	TerminateProcess		863	1.863
	GetStartupInfo		440	1.440
	CreateRemoteThread		107	1.107

행위패턴은 종류와 유형으로 구분되고, 종류(class)는 메일형, 취약점형, 메신저형 등 악성코드의 큰 분류를 위해 사용된다. 유형(type)은 각 종류에서 탐지의 효율을 높이기 위해 포트 등의 특성화할 수 있는 내용으로 작성된다.

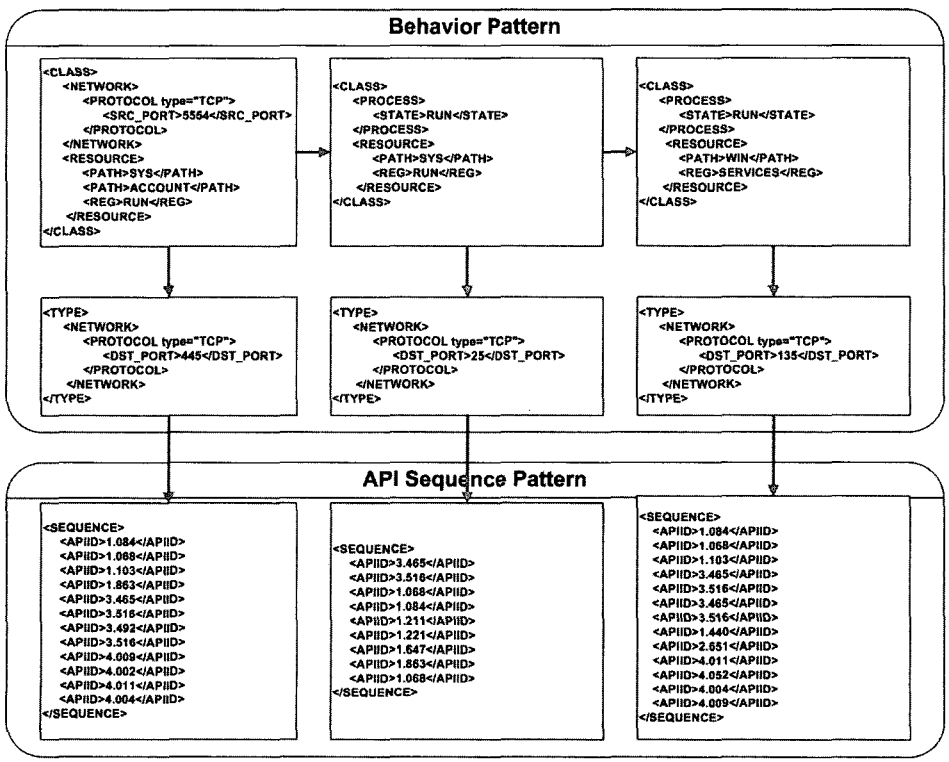


그림 2. 악성코드 탐지패턴 형식

〈Exporting DLL〉:〈API name〉:〈DLL〉.〈Ordinal〉

순차패턴은 행위패턴에서 탐지된 결과 중 오탐을 줄이기 위하여 API 호출에 대해 유사도 비교를 할 수 있도록 생성된다. 그림 2의 패턴구조에서 가장 좌측의 패턴을 살펴볼 경우, 이 패턴은 종류를 구분하기 위한 패턴으로 소스 포트로 5554번을 사용하고, System32와 사용자의 계정에 해당되는 디렉토리에 파일을 생성하며, 레지스트리 위치 중에서 실행 가능한 Run 위치에 정보를 생성하는 것을 패턴으로 생성한다. 유형을 구분하기 위한 패턴으로 프로토콜은 TCP를 사용하고, 목적지 포트를 445번을 사용하는 패턴을 생성한다. 또한 정확도 향상을 위한 순차패턴으로 12개의 API 호출에 대한 패턴을 생성한다.

표 3. 주요 자원별 행위 현황

Action / Resource	열기	닫기	읽기	쓰기	만듦	제거
Reg	O	O	O	O	O	O
File	O	O	O	O	O	O
NetIP	O	O	O	O		
NetPort	O	O	O	O		
Process					O	O
Mem			O	O		
Hook					O	O
Thread					O	O

3.3. 동적 탐지

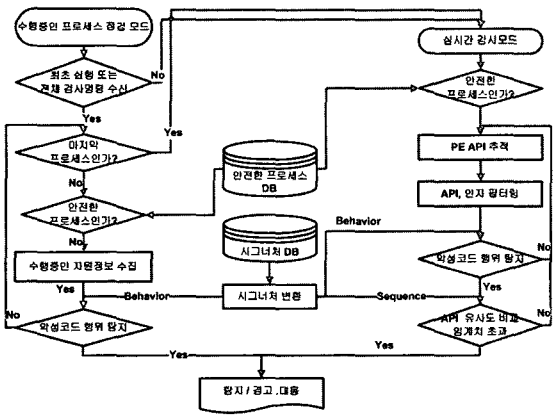
3.3.1. 동적 탐지 절차

동적 탐지모듈은 행위기반 탐지를 수행한다. 그림 3과 같이 정적 및 동적 분석에서 생성된 패턴은 동적 탐지에 맞는 시그니처로 변환된 후에 메모리에 적재된다. 이미 실행되고 있는 PE 파일에 대해서는 현재 자원에서 수집 가능한 프로세스, 네트워크, 레지스트리, 파일의 정보를 수집하여 시그니처에 맞는 행위가 발견될 경우에 악성코드로 탐지하여 위험을 경고한다. PE 파일이 실행될 때, API를 추적하고 해당 내용을 감시하여 탐지를 수행하며, 악성코드로 의심되는 행위가 발견될 경우에는 유사도 비교과정을 거쳐 최종적으로 악성코드 여부를 진단한다.

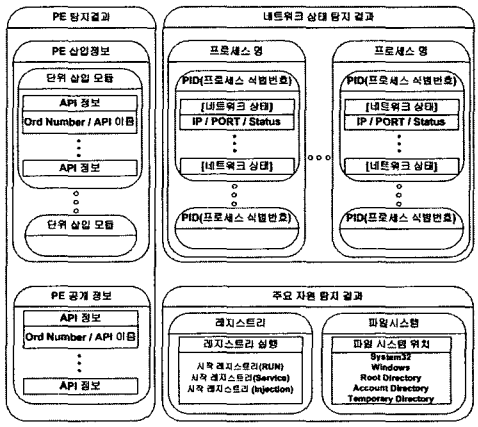
3.3.2. 행위기반의 동적 탐지

동적 탐지를 위해 사용되는 패턴은 표 3과 같은 주요 자원별 행위로 구성된 형태로 구분 하였다. 정적 분석에서 생성된 패턴 정보는 동적 탐지를 위하여 다음과 같이 자원과 동작의 쌍으로 구성된 패턴으로 변환한다.

악성코드에 대한 행위기반 탐지구조는 그림 4와 같다. 행위기반 탐지는 프로세스 실행 시에 프로세스의 네트워크 자원에 대한 점유정보, 파일시스템, 레지스트리에 접근하는 정보를 토대로 탐지를 수행하고, 부가적으로 PE에 대해 추출된 정보를 구조화하여 정보를 유지한다.



(그림 3) 동적 탐지 구조



(그림 4) 행위기반 동적 탐지 구조

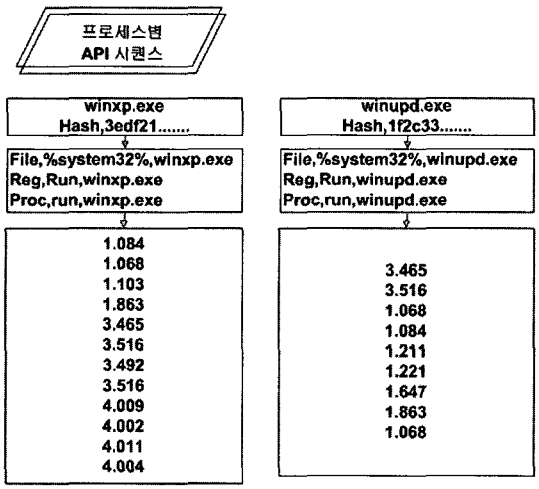


그림 5. API 유사도 비교 절차

3.4. 악성코드 동적 탐지의 유사도 계산

3.4.1. 유사도 비교 절차

제안된 악성코드의 탐지방법은 침입탐지 시스템의 오용탐지 기법과 유사하다. 즉, 피해 시스템에서 발견된 실행파일의 Win32 API 호출 순서 특성이 미리 저장되어 있는 악성코드의 API 순서와 비교하여 유사도를 계산하고, 그 값이 일정 수준을 넘는다면 악성코드 프로그램으로 간주한다.

API 유사도 비교를 위하여 프로세스가 실행되는 순간부터 API 추적을 시작하여 각각의 프로세스마다 그림 5와 같은 프로세스의 상세 정보와 접근 자원정보, API 호출 순서를 저장한다. 프로세스의 상세정보는 프로세스가 생성하는 파일, 레지스트리, 프로세스 등이고, 탐지 결과에 따라 삭제 및 종료에 사용되는 정보이다.

3.4.2. 유사도 비교 알고리즘

테스트 프로그램에 악성코드와 유사한 부분이 있는지 판단하기 위해서는 각 악성코드의 단위 자원들의 API 호출이 유사한지 판단해야 한다. 단위 자원별 API간 유사도 계산식은 수식 1과 같다.

$$Sim(S_i,S_j)=\frac{|S_i \cap S_j|}{|S_i \cup S_j|} \quad (1)$$

순차패턴은 하나의 집합으로 표현될 수 있으며, 이때의 집합에서 각 원소들은 ',' 기호로 구분되는 각각의 API를 의미한다. 순차패턴의 교집합은 단순히 서로 중복되는 원소들의 개수로만 얻는 것이 아니고, 순서에 대한 처리가 포함되어야 한다. 그리고 합집합은 주어진 유사도 측정치의 일반화를 위하여 사용된다. 이산적인 데이터들 사이의 유사도는 그들의 공통된 원소들, 즉 교집합의 크기 정도를 의미하지만, 그 값은 단지 절대적인 수치이며 비교되는 데이터들의 길이에 비례하여 커지기 쉽다. 이는 근본적으로 유사도 비교를 위한 임계값 결정을 불가능하게 한다. 따라서 유사도 값을 공통된 범위 내에 위치시켜 여러 유사도 값들 사이에서도 상대적인 크기 비교가 가능토록 정규화 시킬 필요가 있다. 그러기 위하여 정규화 계수(normalization factor)인 합집합으로 나누어 주었으며, 이때 유사도 값이 항상 0과 1 사이의 범위 안으로 얻어질 수 있도록 하였다. 본 논문에서 유사도 비교 알고리즘은 동적 프로그래밍 기법을 응용하여 제안하였다.

그림 6과 같이 교집합 계산을 위하여 먼저 가중행렬(Weight Matrix)을 구성하였다. 순차패턴 S_i = {A, B, C, C, A, B, B} 과 순차패턴 S_j = {D, A, B, D, C, A, A, C, B}의 교집합을 구하기 위하여 가중 행렬을 생성하고 동적 프로그래밍 기법을 적용하여 각 교차점의 가중치를 계산하는 방법을 보였다. 최종 교집합의 개수는 이들 가중치들로 구할 수 있을 것이다.

수식 2는 좌표 (x, y)에서의 가중치(W)를 구하기 위한 함수이다. x와 y 좌표 상에 놓인 두 순차의 공통된 요소를 교집합의 원소로 정하고, 서로 교차된 지점에서의 가중치는 바로 이전에 두 순차의 요소가 교차된 지점으로부터의 거리 값의 역수로 환산한다.

$$W_{(x,y)}=\frac{1}{(x-x_{prev})(y-y_{prev})} \quad (2)$$

교차점 (x, y)가 이전의 교차점 (x_{prev}, y_{prev})로부터 떨어진 거리는 두 교차점 사이의 빈 블록의 개수에 비례하고, 만일 바로 이전의 교차점이 존재하지 않는다면 초기 교차점은 좌표(0,0)이며 이 좌표의 가중치는 0 값을 가진다.

앞에서 구했던 가중치를 이용하여 최종 교집합의 수를 계산하는 식은 수식 3과 같다.

$$|S_i \cap S_j|=\Sigma_{s\in M} W_s \quad (3)$$

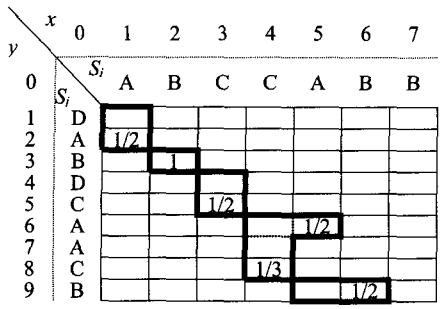


그림 6. 가중 행렬

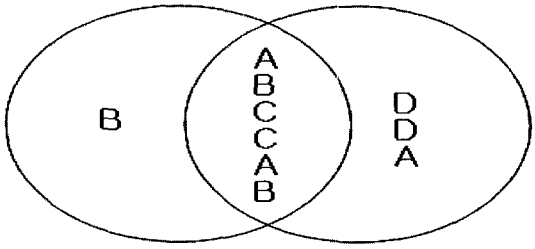


그림 7. 유사도 비교 알고리즘의 합집합

수식 3에서 M은 모든 일치된 교차점의 좌표들이고, Ws는 교집합의 원소 s가 가지는 모든 가중치들의 합을 의미한다.

최종적으로 구해진 모든 교차점들에서의 가중치들을 이용하여 교집합을 구해보면 다음과 같다.

$$|S_i \cap S_j| = \text{A가중치합} + \text{B가중치합} + \text{C가중치합}$$
$$= (1/2+1/2)+(1+1/2)+(1/2+1/3)$$
$$= 3.333$$

표 4. 각종 유사도함수의 비교결과

구분	(V_s), (V_u)	유클리드 거리법	코사인 계수법	피어슨 상관관계	제안된 방법
(1)	(1,9,1,9,1,9) (9,1,9,1,9,1)	19.595	0.219	1.000	0.833
(2)	(1,9,1,9,1,9) (5,5,5,5,5,5)	9.798	0.787	0	0
(3)	(1,2,3,4,5,6) (1,2,3,4,9,6)	5.000	0.931	0.863	0.607
(4)	(1,2,1,2,1,2) (8,9,8,9,8,9)	17.146	0.965	1.000	0
(5)	(1,1,1,1,1,2) (1.1.1.1.1.100)	98.000	0.683	1.000	0.714

합집합의 수치는 그림 7과 같이, 집합의 각 요소들의 중복을 허용하는 다중집합(multi-set 또는 bag) 내의 모든 원소 개수를 의미한다.

합집합의 개수는 위와 같이 10개로 얻어질 수 있다. 따라서 구하려던 유사도 수치는 3.333 / 10 = 0.3333 이다.

탐지 시 API 순차패턴 간의 유사도 비교는 이 함수를 사용한다.

3.4.3. 타 알고리즘 비교 결과

기존 방식의 유사성 척도와의 비교를 위해, 표 4와 같은 상황을 제시하였다. 유사도 비교 예를 서열정렬이 되지 않은 상태로 SAVE에서 사용한 여러 가지 알고리즘과 제안된 방법을 비교한 내용이다.

본 논문에서 제안된 방법은 서열 정렬(sequence alignment)을 거치지 않고, 가중 행렬을 이용하여 유사도를 비교한 방법으로 SAVE에서 시간이 많이 소요되는 서열정렬 과정을 생략할 수 있으므로, 빠른 탐지 성능을 발휘할 수 있다. 유클리드 거리 법은 서열 정렬이 되어 있지 않은 관계로, 오히려 (1)이 더 유사하다고 표시되는 문제가 있으며, 코사인 계수법도 유클리드 거리법과 유사한 결과를 보인다. (1)에서 피어슨 상관관계법은 완전히 일치하는 1의 값을 표현하여 신뢰도에 문제가 발생한다.

제안된 방법에서는 (1)의 유사도가 0.8333으로 완벽하게 일치하지 않음을 보이지만 상당히 유사한 패턴임을 보여준다. (2)에서는 일치하는 숫자가 단 한 개도 없으므로 유사도를 0으로 산출한다. (3), (4), (5)는 SAVE에서 제시한 유사도 알고리즘의 문제를 유발시키는 경우이며, 특히 (4)와 같이 일정 패턴이 반복되지만, 전혀 다른 두 패턴을 코사인 계수법에서는 0.9 이상을 피어슨 상관관계법은 1.0으로 유사도를 계산하여 신뢰도에 문제가 발생한다. 하지만, 제안된 알고리즘에서는 0이라는 유사도를 산출해 낸다.

Ⅳ. 실험 결과

4.1. 탐지패턴별 탐지 결과

4.1.1. 메일형 웜

표 5는 Bagle 웜에 대한 API 순서 추출결과이

다. 최초 원형과 변종에 대해 식별하였고, 기능상으로 크게 변화가 없는 유사한 변종은 그룹화하여 표시하였다.

표 5. Bagle 웜 API 순서 추출 결과

Seq	원형(a)	b~n	o~z	Bagle.af	bagle.al~au
1	3.465	3.465	3.465	3.465	3.465
2	3.516	3.516	3.473	3.473	3.473
3	1.068	1.068	3.465	3.465	3.465
4	1.084	1.084	3.516	3.516	3.516
5	1.211	1.211	1.068	1.068	1.084
6	1.221	1.221	1.211	1.084	1.068
7	1.647	1.647	1.221	1.211	1.211
8	1.863	1.863	1.647	1.221	1.221
9	1.068	1.068	1.863	1.647	1.647
10	0.861	0.861	1.068	1.863	1.863
11			0.861	1.068	1.068
12				0.861	0.861
유사도	1.000	1.000	0.633	0.783	0.658

Bagle.o~n까지의 변종과 al~au까지의 변종은 파일 감염 및 백신 관련 제품 삭제에 대한 패턴이 추가되어 초기의 파일 감염능력이 없고, 백신 관련제품의 삭제 기능이 없는 웜을 기준으로 생성된 순차패턴과의 비교 시에 낮은 유사도를 보인다. 그러나 유사도 비교는 행위패턴을 통한 탐지 이후에 정확도 향상을 위한 추가 탐지의 형식으로 낮은 수치의 유사도일지라도 악성코드의 행위를 수행하였다는 근거가 될 수 있다.

그림 8은 Bagle 웜에 대한 API 및 인자 필터링 전후의 탐지 결과이다. API 필터링은 주요 관심 API에 대한 요청이 발생할 경우 필터링 하는 것이고, 인자 필터링은 주요 자원에 대한 요청이 있을 경우, 필터링을 하는 것으로 탐지의 정확성을 높이고 평준화시키는 결과를 보인다. API 및 인자 필터링 전 결과는 각 호출하는 인자가 다양함에 따라 유사도 비교 시에 탐지가 되는 않는 문제가 발생하였으나, 인자 필터링 후의 결과는 웜의 특징에 맞추어 평준화 되었음을 알 수 있다.

탐지패턴의 생성에 관련하여 최초 원형 바이러스로 패턴을 생성하는 것이 얼마나 다양한 변종까지를 탐지할 수 있는지의 문제가 발생한다. 최초 원형 바이러스가 나온 뒤 변종은 패턴을 변경하거나 기능을 추가할 경우에 발생하는 API 순서가 조금씩 다른 모습을 보인다. 그림 9는 최초 원형인 Bagle.a와 Bagle.o로 패턴을 생성하여 다른 변종 웜을 탐지하는 유사도를 그린 도표이다. 표시된 결과처럼 임계치의 설정이 중요한 문제로 대두되며, 패턴의 생성 시기는 탐지를 수행하는데 있어서 큰 영향이 없음을 알 수 있다.

4.1.2. 취약점 형

표 6은 Sasser 웜에 대한 API 순서 추출 결과이다. Sasser 15872.C에서부터 새롭게 특정 파일을 생성하고자 하는 동작이 발생하지 않아, API 순서에서 삭제됨으로써 유사도에 영향을 미치고 있다.

표 6. Sasser Worm API 순서 추출 결과

No	원형 (15872)	15872.B	15872.C	15872.D	16384	74752
1	1.084	1.084	1.068	1.068	1.068	1.068
2	1.068	1.068	1.103	1.103	1.103	1.103
3	1.103	1.103	1.863	1.863	1.863	1.863
4	1.863	1.863	3.465	3.465	3.465	3.465
5	3.465	3.465	3.516	3.516	3.516	3.516
6	3.516	3.516	3.492	3.492	3.492	3.492
7	3.492	3.492	3.516	3.516	3.516	3.516
8	3.516	3.516	4.009	4.009	4.009	4.009
9	4.009	4.009	4.002	4.002	4.002	4.002
10	4.002	4.002	4.011	4.011	4.011	4.011
11	4.011	4.011	4.004	4.004	4.004	4.004
12	4.004	4.004				
유사도	1.000	1.000	0.950	0.950	0.950	0.950

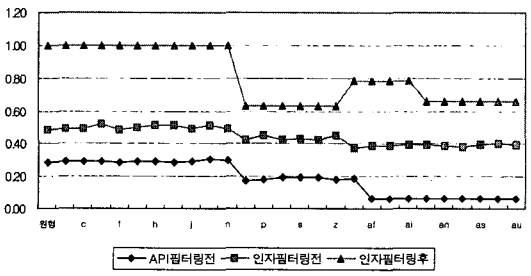


그림 8. 메일형 웜 필터링 전 후 탐지결과

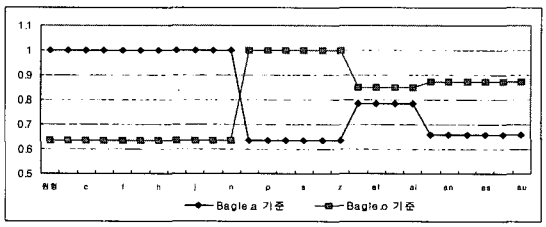


그림 9. 탐지패턴 생성 시기 비교

표 7. Agobot 웜 API 순서 추출 결과

No	원형	64000.D	441344	138752.B	197120.M	99328	98417
1	1.068	1.068	1.068	1.068	1.068	1.068	1.068
2	1.103	1.103	1.103	1.103	1.103	1.103	1.103
3	3.465	3.465	3.465	3.465	3.465	3.465	3.465
4	3.516	3.516	3.516	3.516	3.516	3.516	3.516
5	1.440	1.440	3.465	1.440	3.465	3.465	1.440
6	1.084	2.651	3.473	1.084	3.473	3.516	1.084
7	2.651	1.647	1.440	2.651	1.440	3.465	2.651
8	1.647	1.863	2.651	1.647	1.084	3.473	4.011
9	1.863	4.011	1.647	1.863	2.651	1.440	4.052
10	4.011	4.052	1.863	4.011	1.647	1.084	4.004
11	4.052	4.004	4.011	4.052	1.863	2.651	4.009
12	4.004	4.009	4.052	4.004	4.011	1.647	
13	4.009		4.004	4.009	4.052	1.863	
14			4.009		4.004	4.011	
15					4.009	4.052	
16						4.004	
17						4.009	
유사도	1	0.885	0.738	1	0.845	0.789	0.795

그러나 취약점 형 웜이 일반적으로 수행하는 동작을 그대로 유지하여 0.9 이상의 유사도로 변종 Sasser에 대한 탐지가 가능함을 알 수 있다.

4.1.3. 봇 형

표 7은 봇 형 웜의 대표적인 Agobot 웜에 대한 API 순서 추출 결과이다.

변종은 원형에 비해 새롭게 특정 파일을 생성하고자 하는 동작이 발생하지 않거나, 레지스트리에 백신 및 보안제품의 레지스트리 정보를 삭제하는 순서가 발생하였거나, 보안 관련 프로세스 종료, 레지스트리 기본 실행 및 서비스 시작 위치에 등록 등의 여러 가지 변수에 의해 API 순서에서 삭제 및 추가되어 유사도에 영향을 미치고 있다. 봇 형 웜도 마찬가지로 원형에 비교하여 0.7 이상의 유사도를 보이고 있어 탐지가 가능함을 알 수 있다.

4.1.4. 메신저 형

표 8은 메신저 형 웜의 대표적인 Bropia, Sumom에 대한 API 순서 추출 결과이다. 메신저 형 웜은 웜의 특징보다는 웜을 확산시키기 위한 경로로 사용되는 Dropper의 특징이 강하기 때문에 순차패턴이 단순하다. 동일한 Bropia 변종은 유사도가 1로 정확히 탐지가 가능하며, 유사종인 Sumom은 임계치로 설정된 0.6 범위 안에 포함되어 탐지가 가능하다.

표 8. Bropia, Sumom 웜 API 순서 추출 결과

No	B.159744	B.188928	S.17429	S.23476	S.17429
1	1.068	1.068	1.068	1.068	1.068
2	1.103	1.103	1.103	1.103	1.103
3	3.465	3.465	3.465	3.465	3.465
4	3.516	3.516	3.516	3.516	3.516
5	1.084	1.084	1.647	1.647	1.647
6			1.863	1.863	1.863
7			1.084	1.084	1.084
유사도	1	1	0.619	0.619	0.619

4.2. 유사도 비교 결과 분석

그림 10과 같이 유사도의 임계치가 취약점 형은 0.9 이상, 메일 형과 메신저 형은 0.6이상, 봇 형은 0.7이상일 때에 탐지율 100%를 보인다. 그림 10과 표 9는 유사도 수치에 따른 탐지율을 비교한 것이다.

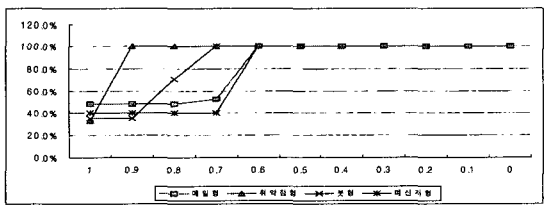


그림 10. 유사도 수치에 따른 탐지율

표 9. 유형별 유사도 임계치에 따른 탐지율/미탐율

(단위: %)

구분 / 임계치	메일형		취약점형		봇형		메신저형	
	탐지	미탐	탐지	미탐	탐지	미탐	탐지	미탐
1	48	52	33	67	35	65	40	60
0.9	48	52	100	0	35	65	40	60
0.8	48	52	100	0	71	29	40	60
0.7	52	48	100	0	100	0	40	60
0.6	100	0	100	0	100	0	100	0
0.5	100	0	100	0	100	0	100	0

V. 결 론

실행압축을 통해 변조된 바이러스 변종의 출현은 특정 위치의 문자열을 검색하여 바이러스로 탐지하는 안티 바이러스 소프트웨어로 하여금 많은 수의 패턴을 유지하게 하는 주원인이 되고 있다. 이런 변조 기법을 사용하는 바이러스는 정적 분석 시에도 IAT의 변경 및 각종 문자열의 제거 등을 이용하기 때문에, 분석에 있어서 많은 한계를 가져온다. 따라서 실행압축 시에도 탐지가 가능한 정적 탐지 및 행위기반의 동적 탐지가 필요하다.

본 논문에서는 실행압축 변조를 통한 우회기법을 사용하는 악성코드에 대한 패턴 생성 및 탐지방법을 제안하였다. 제안된 방식은 정적 및 동적 분석을 통한 패턴생성과 동적 탐지를 통하여 실행압축에 의한 바이러스에 대해 탐지 및 대응이 가능하고, 실시간 행위기반의 동적 탐지는 여러 변종 바이러스에 대하여 주요 자원별 접근 여부에 따라 바이러스 여부를 판단하여 그에 대한 능동적 대응이 가능하다. 또한 정확도를 향상시키기 위한 방법으로 행위패턴의 탐지결과를 순차패턴의 유사도 비교결과를 거쳐 정확도를 향상 시킬 수 있었으며, SAVE에서 제시된 세 가지 경우의 문제도 해결이 가능함을 보였다.

향후 실행압축 상태의 원에 대한 분석 및 탐지 가능한 시그니처를 생성하고, 시작 위치에 등록되는 정상적인 프로그램이 원으로 판단되는 부분에 대해 유사도 비교의 정확성을 높이는 연구와 정적 분석 시에 패턴 생성의 자동화와 생성된 패턴의 재활용 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] 이호동, *Windows 시스템 실행파일의 구조와 원리*, 한빛미디어, 2005.

[2] M. Pietrek, "Inside Windows: An In-Depth Look into the Win32 Portable Executable File Format Part I, II," *MSDN Magazine*, March 2002.

[3] Microsoft Corporation, "Portable Executable Formats," *Formats specification for Windows*.

[4] Jose Nazario, "Defense and Detection Strategies against Internet Worms," *Artech House*, 2004.

[5] Mihai Christodorescu, Somesh Jha, "Static Analysis of Executables to Detect Malicious Patterns," *12th USENIX Security Symposium*, pp. 169-186, 2003.

[6] A. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static Analyzer for Vicious Executables (SAVE)," *20th Annual Computer Security Applications Conference*, pp. 326-334, Dec. 2004.

[7] J-Y. Xu, A. H. Sung, P. Chavez, S. Mukkamala, "Polymorphic Malicious Executable Scanner by API Sequence Analysis," *4th International Conference on Hybrid Intelligent Systems*, pp. 378-383, Dec. 2004.

[8] Cullen Linn, Saumya Debray, "Obfuscation of Executable Code to Improve Resistance to Static Disassembly," *In Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pp 290-299, October 2003.

[9] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie and N. Tawbi, "Static Detection of Malicious Code in Executable Programs," *SREIS '01*, 2001.

[10] S. G. Masood, Malware analysis for administrators, http://www.securityfocus.com/ infocus/1780, 2004.

〈著者紹介〉

박 남 열 (Nam-Youl Park) 정회원
e-mail : hodumaru@lsrc.jnu.ac.kr
1999년 2월 : 광주대학교 컴퓨터학과(공학사)
2001년 8월 : 전남대학교 전산학과(이학석사)
2006년 2월 : 전남대학교 정보보호협동과정(이학박사)
2006년 3월 ~ 현재 : 전남대학교 리눅스 보안 연구센터 Post-doc.

김 용 민 (Yong-Min Kim) 종신회원
1989년 : 전남대학교 전산통계학과(이학사)
1991년 : 전남대학교 전산통계학과(이학석사)
2002년 : 전남대학교 전산통계학과(이학박사)
2003년 6월 ~ 2004년 2월 : 전남대학교 리눅스 보안 연구센터 Post-doc.
2004년 3월 ~ 2006년 2월 : 여수대학교 정보기술학부 전자상거래전공
2006년 3월 ~ 현재 : 전남대학교 전자상거래전공 조교수

노 봉 남 (Bong-Nam Noh) 종신회원
1978년 2월 : 전남대학교 수학교육과 졸업(학사)
1982년 2월 : KAIST 대학원 전산학과 졸업(석사)
1994년 2월 : 전북대학교 대학원 전산과 졸업(박사)
1983년 ~ 현재 전남대학교 전자컴퓨터정보통신공학부 교수
2000년 ~ 리눅스 보안 연구센터 소장