情報保護學會論文誌
第 22 卷 第 3 號, 2012. 6

웹 브라우저 기반 악성행위 탐지 시스템(WMDS) 설계 및 구현*

이 영 욱,[1†] 정 동 재,[1] 전 상 훈,[2‡] 임 채 호[1,2]
[1]KAIST 정보보호대학원, [2]KAIST 사이버보안연구센터

Design and Implementation of Web-browser based Malicious behavior Detection System(WMDS)*

Youngwook Lee,[1†] Dongjae Jung,[1] Sang-hun Jeon,[2‡] Chae-ho Lim[1,2]
[1]KAIST Graduate School of Information Security,
[2]KAIST Cyber Security Research Center

요 약

악성코드 유포자들은 웹 어플리케이션 취약점 공격을 이용해 주로 악성코드를 유포한다. 이러한 공격들은 주로 악성링크를 통해 이루어지며, 이를 탐지하고 분석하는 연구가 활발히 이루어지고 있다. 하지만, 현재의 악성링크 탐지 시스템은 대부분 시그니처 기반이어서 난독화 된 악성링크는 탐지가 거의 불가능하고 알려진 취약점은 백신을 통해 공격을 사전에 방지 할 수 있지만 알려지지 않은 취약점 공격은 사전 방지가 불가능한 실정이다. 이러한 한계점을 극복하기 위해 기존의 시그니처 기반 탐지 방법을 지양하고 행위기반 탐지 시스템에 관한 연구가 이루어지고 있다. 하지만 현재 개발된 탐지 시스템은 현실적으로 제약사항이 많아 실제로 활용하기에는 한계가 있다. 본 논문에서는 이와 같은 한계를 극복하고 탐지 효율을 높일 수 있는 새로운 웹 브라우저 기반 악성행위 탐지 시스템인 WMDS (Web-browser based Malicious behavior Detection System)를 소개 하고자 한다.

ABSTRACT

Vulnerable web applications have been the primary method used by the attackers to spread their malware to a large number of victims. Such attacks commonly make use of malicious links to remotely execute a rather advanced malicious code. The attackers often deploy malwares that utilizes unknown vulnerabilities so-called “zero-day vulnerabilities.” The existing computer vaccines are mostly signature-based and thus are effective only against known attack patterns, but not capable of detecting zero-days attacks. To mitigate such limitations of the current solutions, there have been a numerous works that takes a behavior-based approach to improve detection against unknown malwares. However, behavior-based solutions arbitrarily introduced a several limitations that made them unsuitable for real-life situations. This paper proposes an advanced web browser based malicious behavior detection system that solves the problems and limitations of the previous approaches.

Keywords: malicious link, web application vulnerability, client honeypot, dynamic analysis

접수일(2012년 4월 16일), 수정일(2012년 6월 20일), 게재확정일(2012년 6월 20일)
* 이 연구는 대한민국 지식경제부 정보통신진흥기금으로 수행되었으며, 정보통신산업진흥원(NIPA)의 관리로 진행된 사이버보안연구센터 지원사업임. (과제코드 NIPA-H0701-12-1001)
† 주저자, black83@kaist.ac.kr
‡ 교신저자, p4ssion@gmail.com

I. 서 론

최근 웹 어플리케이션을 통해 인터넷 뱅킹, 커뮤니케이션, 게임 등 수많은 일들을 할 수 있게 되었다. 이로 인해 웹 어플리케이션의 사용 빈도가 증가하게 되었고 웹 어플리케이션의 기능 또한 풍부해졌다 [7]. 풍부해진 기능으로 인해 웹 어플리케이션의 취약점이 많이 발견 되었는데 해커들은 이를 놓치지 않고 수많은 악성링크를 통해 웹 어플리케이션 취약점을 이용하여 최종 악성코드를 다운로드 받고 실행 시키는 drive by download 공격을 감행하여 악성코드를 유포시켰다.

해커들의 주된 공격 대상은 기관이나 회사의 서버였으나 보안에 대한 인식이 높아짐에 따라 비교적 보안의식이 낮고 관리가 허술한 클라이언트를 공격하기 시작하였는데 웹 어플리케이션 취약점은 해커들에게 이상적인 공격 도구가 되었다.

해커들의 공격 대상 변화로 인해 공격 기술이 변하였고 방어하는 입장에서의 기술 또한 변하게 되었다. 허니팟을 하나의 예로 들면, 주로 서버에서 네트워크 패킷을 감시하던 허니팟에서 클라이언트 시스템에서 여러 악성행위를 탐지하는 클라이언트 허니팟으로 변화하게 되었다. 클라이언트 허니팟은 저 상호작용 클라이언트 허니팟과 고 상호작용 클라이언트 허니팟으로 나뉘게 되는데 저 상호작용 클라이언트 허니팟은 OS 및 어플리케이션의 핵심 기능만을 구현하여 상호작용을 하게 하는 것이다. 비교적 가볍고 빠른 처리능력이 장점이지만 OS나 어플리케이션의 모든 기능을 제공하는 것이 아니므로 모든 공격에 대한 탐지가 불가능하다. 고 상호작용 클라이언트 허니팟은 실제 OS 및 어플리케이션을 사용함으로서 제약사항이 없지만 저 상호작용 클라이언트 허니팟에 비해 속도가 느린 것이 단점이다.

웹 어플리케이션 취약점 공격은 주로 인터넷 익스플로러와 인터넷 익스플로러의 플러그인인 Adobe Flash Player, Windows Media Player 그리고 Java Applet의 취약점을 통해 이루어진다. 이러한 공격들은 주로 악성링크를 통해 이루어지는데 최초 악성링크 접속 시 웹 브라우저의 버전을 체크하고 이어서 각종 웹 브라우저 플러그인들의 버전을 체크하여 가장 취약한 부분을 공격하는 고도화 된 형태를 띤다. 또한 주말에만 공격하고 평일에는 흔적을 감추기에 탐지 및 대응에 어려움이 따른다. 더군다나 웹 어플리케이션 취약점 공격을 위한 악성링크는 대부분 난독화 되어 있어서 시그니처 탐지 기반의 백신이나 악성링크 탐지 시스템은 현존하는 악성링크를 찾아내기엔 역부족이다. JSunpack [1]이나 JSbeautifier [2]와 같은 웹 상에서 난독화 되어있는 코드를 정적으로 풀어주는 툴들이 존재 하지만 난독화 된 코드 중 일부분만 풀어주어 실용성이 높지 않다. 또한 알려진 취약점을 이용한 공격은 백신에서 탐지가 가능하지만 알려지지 않은 취약점을 이용한 공격은 탐지가 불가능하다. 이러한 이유 때문에 시그니처 기반의 탐지 시스템의 한계를 극복하고자 행위 기반 탐지 시스템에 대한 많은 연구가 이루어지고 여러 시스템이 개발되었다. 하지만 아직까지 실제로 사용하기에는 부족한 점이 많고 더 많은 연구가 필요하다.

본 논문에서는 기존 시스템의 제약사항을 극복 할 수 있는 새로운 웹 브라우저 기반 악성행위 탐지 시스템을 제안하고자 한다. 제안 하고자 하는 시스템의 특징은 첫 번째로 최초 악성링크 방문부터 어떤 웹 어플리케이션의 취약점 공격을 시도하는지에 대한 추적이 가능한 것이고, 두 번째로는 취약점 공격 성공 후 쉘코드 실행 시 어떤 식으로 drive by download 이벤트를 생성하는 지 알 수 있고, 마지막 세 번째로는 스크립트 언어를 포함한 drive by download되는 모든 악성코드에 대해서 추적이 가능하고 자동으로 동적분석이 가능한 것이다.

본 논문의 2장을 통해 악성링크의 동향을 알아보고 각각의 웹 어플리케이션 취약점으로 인해 실행되는 쉘코드를 분석한 후 마지막으로 쉘 코드에 의해 drive by download 되는 최종 악성코드에 대해 분석하여 악성링크 방문부터 최종 악성코드가 실행되기까지의 일련의 과정을 알아보고자 한다.

이후 3장에서는 기존에 구현된 웹 기반 악성링크 탐지 시스템인 Anubis [3], Capture-HPC [4], Strider HoneyMonkey [5], BLADE [6], PhoneyC [7]에 대해서 알아보고 각 시스템의 한계점에 대해서 알아보도록 하겠다. 4장에서는 기존에 구현된 시스템의 단점을 극복할 수 있는 새로운 웹 브라우저 기반 악성행위 탐지 시스템인 WMDS에 대해서 설명하고 성능을 증명하고자 한다. 5장에서는 이러한 연구의 결과를 정리하고, 6장에서는 향후 연구에 대해서 알아보고자 한다.

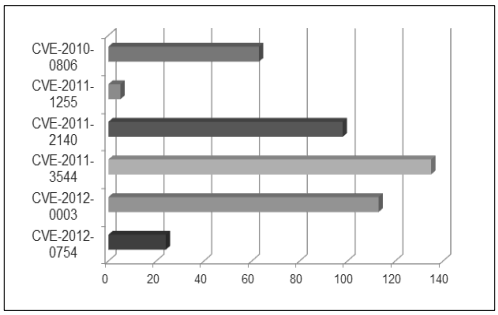
II. 악성링크를 통한 웹 어플리케이션 취약점 공격 분석

이 장에서는 최근 발생하고 있는 웹 어플리케이션 취약점 공격을 목적으로 하는 악성링크의 동향에 대해서 알아보겠다. 분석에 사용된 악성링크들은 Google의 Safe Browsing [16]을 통해 수집하였다.

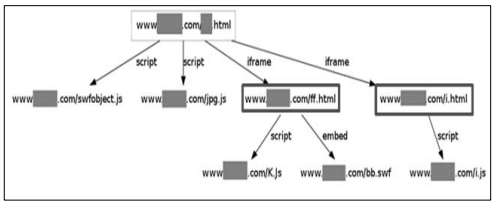
2.1 악성링크 동향

〔그림 1〕은 최근 두 달 간의 악성링크의 유포에 사용된 취약점 통계를 나타낸다. 개별적으로 보면 CVE-2011-3544(Java Applet 취약점) [8] 135건(30.8%), CVE-2012-0003(MIDI 취약점) [9] 113건(25.7%), CVE-2011-2140(Flash 취약점) [10] 98건(22.3%), CVE-2010-0806(MS IE 취약점) [11] 63건(14.3%), CVE-2012-0754(Flash 취약점) [12] 24건(5.4%), CVE-2011-1255(MS IE 취약점) [13] 5건(1.1%) 순으로 나타났다. 이 취약점들은 개별적으로 쓰이기보다는 확률을 높이기 위해서 몇 개가 같이 사용된다. 가장 흔하게 나타나는 링크의 구조는 〔그림 2〕와 같다.

주말이 되면 정상링크에 IFRAME 및 IMG와 같은 HTML 태그를 추가하여 감염된 링크에 접속하는



〔그림 1〕 최근 두 달간의 취약점 통계



〔그림 2〕 악성링크 구조

```
<script type="text/javascript" src="swfobject.js"></script>
<script src=jpg.js></script>
<script type="text/javascript">
function hRtfW7(){
    var Cazl6=deconcept.SWFObjectUtil.getPlayerVersion();
    if(Cazl6['major']==10&&Cazl6['minor']<=3&&Cazl6['rev']<183)
    {
        document.writeln("<iframe src=ff.html><\/iframe>");
    }
    else {
        var teTn4 = window.navigator.userAgent.toLowerCase();
        if ((teTn4.indexOf('msie 6.0') > -1)
            || (teTn4.indexOf('msie 7.0') > -1)) {
        document.writeln("<iframe src=i.html><\/iframe>");
        }
    }
}
if(document.cookie.indexOf("dadong=")==-1)
{
var expires=new Date();
expires.setTime(expires.getTime()+24*60*60*1000);
document.cookie="dadong=Yes;path=/;expires="
                +expires.toGMTString();
if(navigator.userAgent.toLowerCase().indexOf("msie")!=-1)
```

〔그림 3〕 악성링크 소스코드

일반 사용자들을 악성링크의 루트페이지로 연결시킨다. 악성링크의 루트페이지의 소스를 보게 되면 〔그림 3〕과 같이 되어있는 것을 볼 수 있는데 악성링크 접속 시 감염 성공률을 최대화하기 위해 클라이언트의 운영체제, 어플리케이션의 버전 등의 정보를 체크하여 취약한 부분을 찾아서 분기하는 방식을 사용한다. 최근에는 CVE-2012-0003 [9], CVE-2011-3544 [8], CVE-2011-2140 [10] 취약점 세트가 가장 빈번하게 나타나고 있는데, 각 취약점을 간략히 살펴보면 아래와 같다.

2.1.1 CVE-2012-0003 (MIDI Remote Code Execution Vulnerability)

CVE-2012-0003 취약점은 Windows Media Player(WMP)가 Windows Multimedia Library(winmm.dll)를 통해 MIDI 파일을 파싱(parsing)하는 과정에서 발생한다. 공격자는 임의로 조작한 MIDI 파일을 통해 힙(heap)에 할당된 특정 버퍼크기 이후의 데이터를 변경시킬 수 있다. 해당 취약점은 Internet Explorer에 모듈로 로딩 될 수 있으며 이를 힙 스프레이 공격 기법과 함께 이용하면 공격자는 자신이 원하는 코드를 실행시킬 수 있다.

2.1.2 CVE-2011-3544 (Oracle Java Applet Rhino Script Engine Remote Code Execution)

본 취약점은 JDK/JRE 7 버전과 JDK/JRE 6 Update 28버전 이하에서 동작하며, 힙 스프레이와

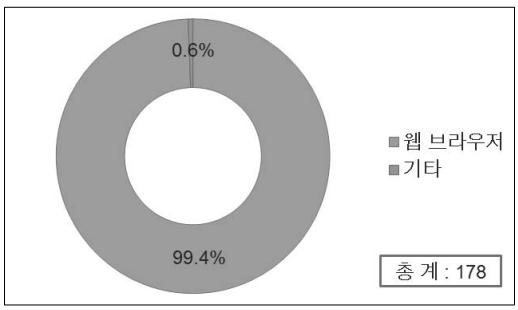
같은 부가적인 공격 기법을 필요로 하지 않아 공격자들이 선호하는 취약점이라고 할 수 있다. 더욱이 일반적인 사용자가 Java 업데이트를 잘 수행하지 않는다는 점을 고려한다면 그 파급효과는 더욱 크다고 할 수 있다.

2.1.3 CVE-2011-2140 (Adobe Flash Player MP4 sequenceParameterSetNALUnit Vulnerability)

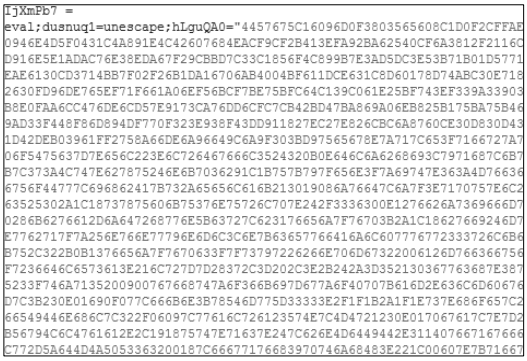
본 취약점은 2011년 5월 최초 보고되었으며 Adobe Flash Player 10.3.183.5이상에서는 취약점이 발생되지 않는다. 해당 취약점은 SWF파일 내에 미디어 파일을 가진 또 다른 SWF파일을 통해 원격코드를 실행 시킬 수 있다.

2.2 쉘 코드 분석

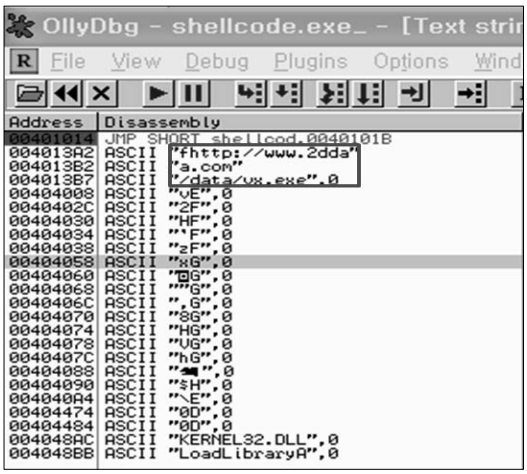
공격자는 클라이언트를 악성링크로 유도하여 취약한 웹 어플리케이션을 공격하고 쉘 코드가 실행 되게 한다. 대부분의 쉘 코드는 다운로더 역할을 하는데 URLDownloadToFile API를 이용하여 최종 악성코드를 다운받고 실행한다. 이 경우의 악성코드를 실행하는 주체는 웹 어플리케이션이 된다. 쉘 코드가 웹 어플리케이션의 프로세스에서 실행되기 때문이다. 하지만 극소수는 이와는 다른 양상을 나타내기도 하였는데, 어떤 쉘 코드는 CreateRemoteThread API를 사용하여 이미 동작중인 explorer.exe 프로세스의 내부에 원격 쓰레드를 생성하여 감염시킨 후 감염된 explorer.exe가 악성코드를 다운받고 실행하게 하는 방식이었다. 샘플을 통해 직접 통계를 낸 결과는 〔그림 4〕에서 볼 수 있듯이 악성링크를 통해 실행되는 쉘 코드의 99.4%가 웹 어플리케이션에서 drive by download 이벤트를 발생시키고 0.6%의 쉘 코드만



〔그림 4〕 drive by download 이벤트 생성 객체



〔그림 5〕 난독화 코드



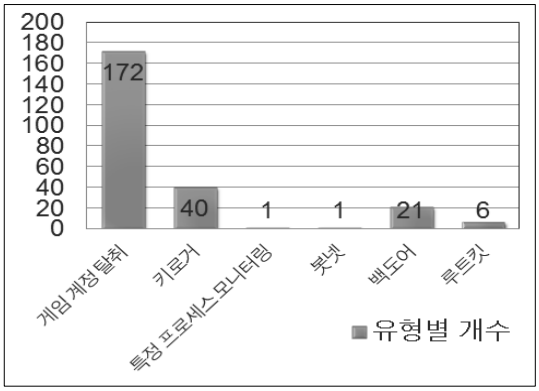
〔그림 6〕 쉘 코드 예제

이 explorer.exe를 감염시켜 drive by download 이벤트를 발생시켰다.

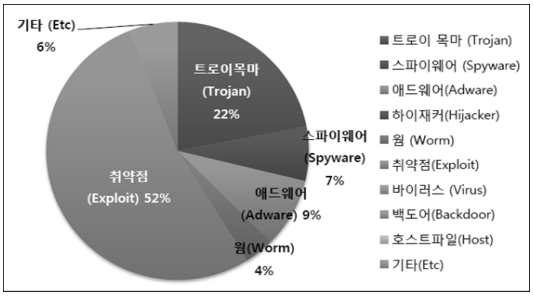
악성링크의 소스코드는 탐지와 분석을 어렵게 하기 위해서 난독화 기법을 이용한다. 분석을 위해서는 먼저 난독화된 소스코드를 푸는 과정을 거쳐야 한다. 〔그림 5〕와 같이 난독화된 소스코드를 풀어서 분석하면 쉘 코드가 나오게 되며 쉘 코드에는 〔그림 6〕과 같이 최종 다운받을 악성파일의 경로가 나타나 있는 것을 볼 수 있다.

2.3 최종 악성코드 분석

공격자의 궁극적인 목표는 여러 가지 웹 어플리케이션 취약점을 이용해 클라이언트에서 악성코드를 다운로드 받고 실행하여 사용자의 정보를 빼내는 것이다. 여기에 사용되는 최종 악성코드는 게임 계정 탈취, 키로거, 특정 프로세스 모니터링, 봇넷, 백도어,



〔그림 7〕 악성코드 유형



〔그림 8〕 카테고리 별 악성코드 유형

루트킷 등의 유형으로 분류할 수 있으며 최근에는 주로 게임 계정을 탈취하는 악성코드가 주를 이루고 있다. 샘플을 통해 직접 통계를 낸 결과는 〔그림 7〕에서 볼 수 있듯이 지난 두 달간의 최종 악성코드는 게임 계정 탈취 172개, 키로거 40개, 특정 프로세스 모니터링 1개, 봇넷 1개, 백도어 21개, 루트킷 6개로 측정되었다.

III. 관련 연구

1장과 2장을 통해 웹 어플리케이션을 통한 취약점 공격에 대해 알아보았다. 실제로 Alyac Report 에서는 〔그림 8〕에서 볼 수 있듯이 전체 악성코드 중 과반수 이상이 취약점을 이용한 악성코드라는 것을 알려주었다[14]. 실제로 웹 어플리케이션 취약점 공격을 노린 악성링크의 증가로 인해 웹 브라우저 기반 악성행위 탐지 시스템의 중요성은 더더욱 부각되고 있고 현재까지 여러 웹 브라우저 기반 악성행위 탐지 시스템이 구현 되었다. 이번 장에서는 기존 웹 브라우저 기반의 탐지 시스템에 대해 알아보겠다.

3.1 PhoneyC

PhoneyC[7]는 저 상호작용 클라이언트 허니팟으로서 HTTP 프로토콜을 포함한 웹 어플리케이션의 핵심 기능만을 구현하여 의심이 가는 웹 페이지의 JavaScript와 ActiveX add-on을 이용한 취약점 공격을 탐지할 수 있고 난독화 된 JavaScript코드 및 Visual Basic Script코드를 풀어주고 재분석 하는 기능을 갖고 있다. 그러나 저 상호작용이기 때문에 OS의 기능을 충분히 제공해주지 못하고 알려지지 않은 취약점 및 웹 브라우저의 여러 플러그인에 대한 취약점은 판별해 낼 수 없다.

3.2 Anubis

Anubis[3]는 알려지지 않은 바이너리를 자동으로 분석해주는 자체적으로 구현된 샌드박스로서 웹 페이지를 통해 의심이 가는 웹페이지 URL 및 바이너리를 입력해주면 자동적으로 분석해서 파일, 레지스트리, 프로세스 및 네트워크 행위에 대한 분석을 해준다. Anubis는 바이너리에 대한 분석은 잘 이루어지는 편이나 웹 브라우저나 웹 브라우저 플러그인 취약점을 이용한 공격은 시스템의 특정 조건에 만족해야 이루어지는 것이므로 Anubis의 단일화 된 환경에서는 취약점을 이용한 공격을 탐지 할 수 없다.

3.3 Capture-HPC

Capture-HPC[4]는 고 상호작용 클라이언트 허니팟으로서 가상 환경에서 웹 브라우저를 통해 발생하는 모든 악성행위를 탐지 할 수 있다 또한 사용자가 시스템 환경을 임의대로 바꿀 수 있어 여러 취약점 공격이 이루어지도록 유도 할 수 있다. Capture-HPC의 기본 원리는 시스템에서 발생하는 모든 파일, 레지스트리, 프로세스에 대해서 이벤트가 발생하면 사용자가 만들어 놓은 예외 리스트를 참조하여 예외 리스트에 기록해 놓지 않은 곳에서 발생한 이벤트면 악성으로 간주하고 로그를 출력시켜주는 것이다. Capture-HPC의 단점은 사용자가 예외 리스트를 지속적으로 관리해야 하고 만약 예외 리스트에 있는 폴더에서만 악성행위를 한다면 악성행위를 탐지 할 수 없다는 데 있다.

3.4 Strider HoneyMonkey

Strider HoneyMonkey [5]는 고 상호작용 클라이언트 허니팟으로서 웹 브라우저를 통해 발생하는 악성행위의 여부를 통해 악성링크를 찾는데 목적을 갖고 있다. 웹 페이지에 방문 전 메모리, 프로세스, 레지스트리의 상태를 저장해 놓고 웹 페이지 방문 후 저장해 놓은 상태에 변화가 생기면 변화가 생긴 곳을 참조하여 악성인지 아닌지 판단하는 시스템이다. 이러한 시스템은 모든 메모리, 프로세스, 레지스트리의 상태를 저장하고 바뀐 곳을 비교하여야 하므로 속도가 느린 것이 단점이다.

3.5 BLADE

BLADE [6]는 웹 브라우저 기반 침입 탐지 시스템으로 웹 브라우저로부터 사용자가 허락하지 않은 실행파일의 실행을 악성행위로 구분해 침입을 방지하는 시스템이다. 웹 브라우저를 통해서 생성되는 실행파일은 먼저 BLADE에서 제어하는 안전지대에 옮겨두고 해당 실행파일의 실행 시 사용자의 허락이 입증된 것이면 실행시키고 만약 사용자의 허락이 입증되지 않은 것이면 실행시키지 않고 격리시킨다. BLADE의 탐지율은 완벽하다고 주장하고 있으나 실행파일이 아닌 스크립트로 실행되는 파일은 탐지가 불가능하다는 것이 한계점이다.

IV. WMDS

기존 시스템의 한계를 몇 가지로 요약해 보면 첫 번째로 시스템에서 생성되는 모든 파일, 레지스트리, 프로세스, 네트워크 등의 이벤트 중 웹 어플리케이션 취약점 공격에 의해 파생되는 프로세스 및 쓰레드의 이벤트만을 구분해 내기에는 기준점이 모호 하다는 것이고, 두 번째는 일반적인 텍스트 형식과 같은 스크립트 언어를 사용하는 악성코드는 탐지가 불가능하다는 것이다. 마지막 세 번째는 악성링크 방문 전 모든 메모리, 프로세스, 레지스트리의 상태를 저장해 놓고 악성링크 방문 후 변화를 비교하는 방식은 너무 오랜 시간이 걸린다는 것이다. 이러한 한계를 개선하기 위해 본 장에서는 웹 브라우저부터 시작해 웹 어플리케이션 취약점 공격에 의해 파생되는 모든 프로세스 및 쓰레드 주체의 이벤트를 탐지 해주는 모듈인 Observer를 적용한 새로운 웹 브라우저 기반 악성행위 탐지 시스템인 WMDS를 제안하고자 한다.

4.1 Observer

Observer의 동작 방식은 본 논문 2장인 악성링크를 통한 웹 어플리케이션 취약점 공격 분석을 바탕으로 설계되었다.

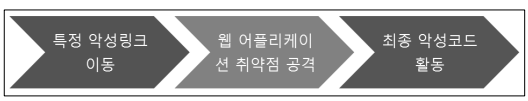
웹 어플리케이션 취약점 공격의 전체적인 과정은 〔그림 9〕와 같이 총 3단계로 나뉜다. 첫 번째 과정은 최초 악성링크 방문 후 웹 페이지에서 방문자의 웹 브라우저 버전과 각종 웹 어플리케이션들의 버전을 확인하고 가장 취약한 웹 어플리케이션을 공격하기 위한 특정 악성링크로 이동시키는 것이다. 두 번째 과정은 가장 취약한 웹 어플리케이션의 취약점을 공격하여 쉘코드를 실행 시킨 후 최종 악성코드를 drive by download 하는 것이다. 2장에서 보았듯이 쉘코드가 실행되면 웹 브라우저 자체에서 URLDownload-ToFile이라는 API를 사용해 최종 악성코드를 다운로드 받고 최종 악성코드를 실행하는 경우와 웹 브라우저에서 CreateRemoteThread API를 사용해 사용자의 운영체제에서 이미 실행중인 explorer.exe의 프로세스 내부에 원격으로 쓰레드를 생성하여 감염시키고 explorer.exe 프로세스가 최종 악성코드를 다운로드 받아 실행하게 만드는 경우가 있다. 마지막 세 번째 과정은 두 번째 과정에서 실행 된 최종 악성코드의 활동이다. 위의 3가지 과정을 보면 첫 번째 과정인 웹 페이지 이동과 두 번째 과정인 쉘코드 실행은 웹 브라우저에서 발생하는 일이고 세 번째 과정인 최종 악성코드의 활동은 웹 브라우저나 웹 브라우저에게 감염된 프로세스로부터 발생되는 것이다. 그렇다면 웹 브라우저에서 최초 링크부터 쉘코드가 실행되기까지 방문하는 링크들을 참조하면 어떤 어플리케이션 취약점에 의해 일어나는 공격인지 알 수 있고 일반적인 웹 페이지 방문만으로는 다운로드 이벤트나 파일 실행은 일어나지 않으므로 웹 브라우저에서 다운로드 이벤트와 프로세스 및 쓰레드 생성 이벤트를 탐지하고 웹 브라우저로부터 파생되는 모든 프로세스와 쓰레드의 행동을 추적한다면 최초 악성링크 방문부터 쉘코드의 실행 그리고 최종 악성코드의 행위까지 전체적인 악성행위를 탐지 할 수 있게 된다.

WMDS에서는 이와 같은 이론을 바탕으로 Observer라는 악성행위 탐지 모듈을 설계 하였다. Observer의 내부는 〔그림 10〕과 같이 File Monitor, Registry Monitor, Service Monitor,

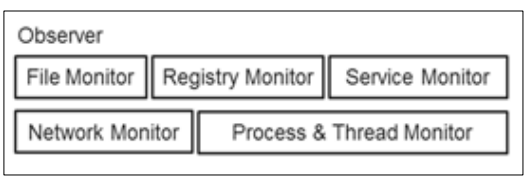
Network Monitor, Process & Thread Monitor로 이루어져 있고 〔그림 11〕과 같이 프로세스 단위로 붙어서 해당 프로세스에서 벌어지는 파일, 레지스트리, 서비스, 네트워크, 프로세스 및 쓰레드에 대한 이벤트를 감지한다.

File Monitor는 CreateFile, CopyFile, MoveFile, WriteFile, DeleteFile API를 후킹하고 Registry Monitor는 RegCreateKey, RegSetValue, RegOpenKey API를 후킹하며 Service Monitor는 CreateService, StartService API를 후킹하고 Network Monitor는 gethostbyname, inet_addr, URLDownloadToFile, InternetConnect, HttpOpenRequest API를 후킹하며 Process&Thread Monitor는 CreateProcessInternal과 ZwResumeThread API를 후킹한다. Observer가 사용하는 API 후킹 기법은 인라인 후킹이기에 Native API를 직접 호출하는 악성코드에 대한 분석도 가능하다. 또한 인라인 후킹이기에 웹 브라우저가 달라도 Native API는 같기에 크롬이나 파이어 폭스에서도 WMDS 사용이 가능하다.

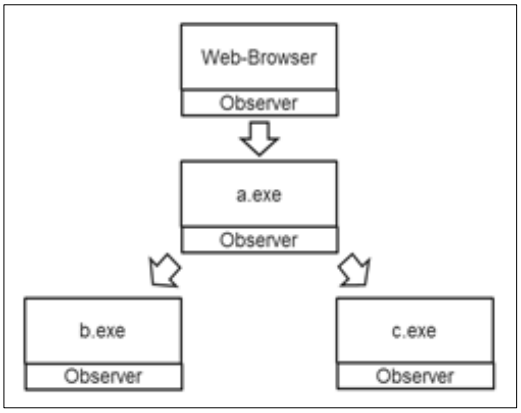
Observer는 〔그림 12〕에서와 같이 3 단계의 모듈 핵심 내용을 가지고 있다. 최초 프로세스에서 동작 시 해당 프로세스에서 사용되는 Kernel32.dll, Advapi32.dll, Ws2_32.dll, urlmon.dll, wininet.dll 등에 정의되어 있는 API 중 Observer 내부의 모니터들이 필요로 하는 API 함수를 분류하고 후킹하여 이벤트 발생 시 해당 모니터가 선점하여 처리하게 해준다. 하나의 예를 들자면 Observer가 최초 프로세스에서 동작 시 File Monitor를 위해 해당 프로세스에서 사용하는 Kernel32.dll의 CreateFile, CopyFile, MoveFile, DeleteFile등 파일 조작에 관한 API 후킹을 하고 파일 조작에 관한 이벤트 발생 시



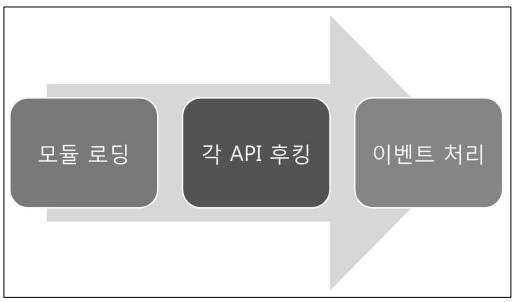
〔그림 9〕 웹 어플리케이션 취약점 공격의 3 단계



〔그림 10〕 Observer 동작 방법



〔그림 11〕 각 프로세스 별 첨부 된 Observer

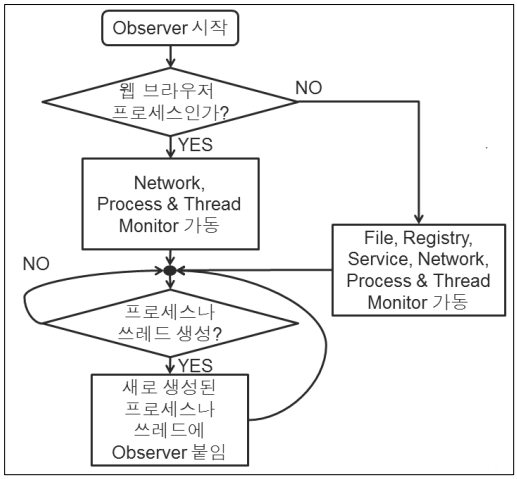


〔그림 12〕 Observer 모듈의 핵심 내용 3 단계

File Monitor가 선점하여 처리 할 수 있도록 해주는 것을 예로 들 수 있다.

WMDS의 전체적 과정 및 기존 연구와의 차별성

최초 웹 브라우저 실행 시 웹 브라우저 프로세스에 하나의 Observer가 붙게 되고 〔그림 13〕에서와 같이 웹 브라우저 프로세스에 붙여진 Observer는 웹 어플리케이션 취약점 공격을 위한 링크 이동과 다운로드 이벤트, 프로세스 및 쓰레드의 생성만 확인하면 되기 때문에 Network Monitor와 Process & Thread Monitor만 동작한다. 이후 웹 브라우저나 웹 브라우저에 의해 감염된 프로세스에서 프로세스 및 쓰레드 생성 시 해당되는 프로세스에 또 하나의 Observer를 붙이고 재귀적으로 생성되는 모든 프로세스 및 쓰레드에 Observer를 붙이게 된다. 웹 브라우저 이외에 파생된 모든 프로세스에 붙여진 Observer는 모든 모니터를 동작하게 함으로서 웹 어플리케이션 취약점 공격부터 최종 악성코드까지의 모든 악성 행위를 탐지 할 수 있다. 이로 인해서 Capture-HPC [4]의 단점인 예외리스트를 작성할 필요가 없고 BLADE [6]의 단점인 스크립트 코드 추적



〔그림 13〕 Observer 실행 시 순서도

불가도 결국 스크립트 인터프리터 프로세스가 실행되어 스크립트 코드를 실행하므로 인터프리터 프로세스 생성 시 Observer가 붙게 되어 스크립트 코드도 악성행위 탐지가 가능해진다. 또한 Strider Monkey [5]의 단점인 모든 메모리, 프로세스, 레지스트리, 프로세스를 저장하고 비교하는 것이 아니고 실시간으로 악성 행위를 탐지하는 것이기에 시간적 제약도 사라지게 된다.

4.3 WMDS의 로그 분석

Observer 모듈을 핵심으로 만들어진 WMDS로 인해 웹 어플리케이션 취약점을 노리는 악성링크 탐지가 가능하게 되었다. WMDS는 웹 어플리케이션 취약점을 노리는 악성링크의 최초 방문부터 가장 취약한 웹 어플리케이션 공격을 위한 링크 이동, 쉘 코드 실행 및 최종 악성코드의 동적 분석까지 가능하다. 악성링크 방문 후 〔그림 14〕에서 볼 수 있듯이 WMDS는 전체 과정에 대한 로그를 남긴다.

로그의 내용을 분석해 보면 Object는 이벤트의 주체가 되는 프로세스이고 Operation은 탐지된 이벤트 내용이다 그리고 Parameter는 이벤트 발생 시의 인자 값이다. 최초 웹 브라우저 실행 후 Observer 모듈이 웹 브라우저의 프로세스에 붙게 되고 네트워크와 프로세스 및 쓰레드의 생성을 감시한다. 최초 i.html을 방문하고 i.js를 통해 exp.mid를 호출하게 되는데 이것은 CVE-2012-0003인 Windows Media Player 플러그 인 취약점을 이용한 공격임을 알 수 있다. MIDI 취약점 공격 성공 후 쉘코드가 실행 됨으로서 URLDownloadToFile API를 호출하여 최종 악성코드인 ppx.exe를 로컬 시스템에 a.exe로 다운로드 받고 CreateProcess API를 이용해 최종 악성코드를 실행 시킨다. 웹 브라우저 프로세스에서 악성행위를 감시하고 있던 Observer는 a.exe의 프로세스 생성을 탐지하고 a.exe에 새로운 Observer를 붙이고 모든 모니터를 동작 시킴으로서 최종 악성코드인 a.exe의 모든 행위를 추적 할 수 있다. a.exe의 행위는 WMDS의 로그만으로 충분히 파악 할 수 있는데, 윈도우즈 폴더 내의 ws2help.dll 파일을 다른 이름으로 바꾸고 imm32.bmp라는 파일을 만들어 ws2help.dll로 탈바꿈 시킨다. 마지막으로 자기 자신의 삭제 명령을 내리고 결국 최종 악성코드는 사라지게 된다. ws2help.dll은 Windows Socket 2.0 Helper DLL로서 ws2_32.dll을 보조하는 역할을 한다. 최종 악성코드에 의해 바뀐 ws2help.dll의 내부를 살펴보면 각종 백신을 무력화 시키는 기능과 각종 게임 계정을 탈취하기 위한 기능을 갖고 있다.

```
WMDS Starts!!

Object:C:\Program Files\Internet Explorer\iexpore.exe
Operation:HttpOpenRequestA
Parameter:www.xxxxx2333.com/i.html

Object:C:\Program Files\Internet Explorer\iexpore.exe
Operation:HttpOpenRequestA
Parameter:www.xxxxx2333.com/i.js

Object:C:\Program Files\Internet Explorer\iexpore.exe
Operation:HttpOpenRequestA
Parameter:www.xxxxx2333.com/exp.mid

Object:C:\Program Files\Internet Explorer\iexpore.exe
Operation:URLDownloadToFile
Parameter1:www.xxxxx2333.com/ppx.exe
Parameter2:C:\Documents and Settings\Administrator\Application Data\a.exe

Sample was created at c:\a.exe

Object:C:\Program Files\Internet Explorer\iexpore.exe
Operation:CreateProcess
Parameter:C:\Documents and Settings\Administrator\Application Data\a.exe

Object:C:\Documents and Settings\Administrator\Application Data\a.exe
Operation:MoveFileExA
Parameter1:C:\WINDOWS\system32\ws2help.dll
Parameter2:C:\WINDOWS\system32\ws2help.dll.xJR.tmp

Object:C:\Documents and Settings\Administrator\Application Data\a.exe
Operation:CreateFileA
Parameter1:C:\WINDOWS\imm32.bmp
Parameter2:GENERIC_WRITE

Object:C:\Documents and Settings\Administrator\Application Data\a.exe
Operation:MoveFileExA
Parameter1:C:\WINDOWS\imm32.bmp
Parameter2:C:\WINDOWS\system32\ws2help.dll

Object:C:\Documents and Settings\Administrator\Application Data\a.exe
Operation:CreateProcessInternalA
Parameter:C:\WINDOWS\system32\cmd.exe /c del
          "C:\Documents and Settings\Administrator\Application Data\a.exe"

Object:C:\WINDOWS\system32\cmd.exe
Operation:DeleteFileW
Parameter:C:\Documents and Settings\Administrator\Application Data\a.exe
```

〔그림 14〕 WMDS 로그

위의 WMDS 로그 분석을 통해 웹 어플리케이션 취약점 공격부터 최종 악성코드 분석까지 모든 과정을 분석 할 수 있다는 것을 증명하였다.

4.4 평가

전 세계적으로 사람들이 많이 접속하는 사이트 일수록 보안에 관한 관리가 잘 되는 정상링크 일 것이라는 판단 하에 Alexa TOP Sites [17]에서 제공하는 사이트들 중 1위부터 100위까지의 사이트를 정상링크 샘플로 지정하고, Google Safe Browsing [16]을 통해 탐지되는 100개의 악성링크를 악성링크 샘플로 지정하였다. WMDS의 성능 평가를 위해 정상링크와 악성링크를 합친 200개의 URL을 가지고 악성링크 탐지율을 실험 하였다.

WMDS 평가 시 운영체제는 Windows XP SP2 상에서 운영되었고 웹 브라우저는 Internet Explorer 6, 7 버전에서 테스트 되었다.

200개의 URL을 사용하여 악성링크 탐지율에 관한 성능 평가를 한 결과는 〔표 1〕에서 볼 수 있듯이 정상링크 100개를 모두 정상링크로 진단 하였고, 악성링크 100개 중 93개의 링크를 악성이라고 판단하였는데 분석 해 본 결과 나머지 7개는 〔그림 15〕와 같이 공격자가 웹 어플리케이션 취약점 공격을 위해 정상링크에 악성링크로 유도하기 위한 악성 코드를 삽입 하였으나 악성링크에는 실질적인 공격 코드를 추가하지 않고 공격 전 사전 검사를 하기 위한 카운터 링크만 추가하여 악성행위가 발생하지 않아 탐지를 못 하는 것이었다. 〔그림 15〕을 보면 3 개의 링크가 일렬로 늘어선 것을 볼 수 있는데 최초 tj518.html을 시작으로 s20xx.php을 방문하게 되고 카운터 링크인 hzs20xx.com을 연달아 방문하게 되는 형식이다. 여기에 웹 어플리케이션 공격을 위한 링크가 추가되면 〔그림 2〕와 같은 형태를 띄면서 방문자의 웹 어플리케이션 환경에 따라 특정 링크를 방문하게 만든다.

좀 더 신빙성 있는 시험을 위해 악성링크로 판명된 93개 URL에서 drive by download 된 최종 파일 중 해시 값이 같은 것을 제외한 70개의 파일 샘플을 VirusTotal [15]에서 검사 해 본 결과 〔표 2〕에서 볼 수 있듯이, 6개의 의심이 가는 파일 이외에 64개의 파일은 악성코드로 결과가 나왔다. 이로 인해 악성링크 탐지를 위한 WMDS의 탐지 효율성이 증명 되었다.

V. 결론 및 향후과제

기관이나 회사의 서버를 향한 공격이 주로 이루어지던 이전과는 다르게 근래에는 비교적 보안에 대한 지식이 낮고 공격이 용이한 클라이언트를 향한 공격이 늘어나게 되었다. 웹 어플리케이션에 대한 의존도가 높아진 지금 공격자들에게 웹 어플리케이션 취약점은 좋은 공격 도구가 되었다. 웹 어플리케이션 취약점 공격을 위해 악성링크가 쓰이고 이러한 악성링크를 탐지하기 위해 많은 연구가 이루어 졌다.

이 논문에서 제안한 새로운 웹 브라우저 기반 악성행위 탐지 시스템인 WMDS는 비교적 간단한 이론을 바탕으로 기존에 있던 시스템의 단점을 극복할 수 있는 시스템이다. 그러나 초기 시스템 모델이기에 몇 가지 한계점이 존재하는데 취약점 공격이 성공되기 위한 환경이 아니라면 취약점 공격은 이루어지지 않게 되고 결국 탐지를 할 수 없게 된다.

이 논문을 작성 할 당시 주로 쓰이던 취약점 공격은 인터넷 익스플로러 자체 취약점, MIDI 취약점, Adobe Flash Player 취약점, Java Applet취약점 총 4가지 웹 어플리케이션을 노린 공격이기에 논문을 작성 할 당시의 모든 공격을 탐지 할 수 있는 환경을 구축해 놓아서 모든 탐지가 가능하였다.

현재까지는 4개의 웹 어플리케이션에 대한 환경만 구축 되어 있는데 모든 웹 어플리케이션에 관한 환경을 구축한다면 모든 취약점 공격을 위한 악성링크를

〔표 1〕 WMDS의 악성링크 탐지 결과

	Total	True Positive	False Positive	True Negative	False Negative
URLs	200	100	0	93	7

〔표 2〕 최종 악성코드 VirusTotal 결과

	Total	GameTheft	Backdoor	Suspicious	Trojan	Dropper	Exploit	KillAV	Rootkit
Files	70	42	7	6	7	5	1	1	1

찾아 낼 수 있을 것이다.

1장에서 설명하였듯이 기관이나 회사의 서버를 목표로 하는 공격에서 보안상 취약한 일반 사용자를 겨냥한 공격이 늘어남으로 인해 서버에서만 활용되던 허니팟이 클라이언트에서도 활용되어지고 있다. 이러한 공격자들의 경향으로 인해 호스트 기반 침입 탐지 시스템인 HIDS에 대한 여러 연구가 진행 중이다. 향후에 웹 브라우저로부터 프로세스나 쓰레드 생성 또는 다운로드 이벤트 발생 시 유저가 의도한 것인지 아닌지 판단 할 수 있는 모듈을 개발 한다면 WMDS는 HIDS로서 임무를 수행 할 수 있을 것이다. 또한 본 논문을 통해 소개한 WMDS는 고 상호작용 클라이언트 허니팟이기에 WMDS와 연동이 가능한 저 상호작용 클라이언트 허니팟을 구현하여 저 상호작용 클라이언트 허니팟으로 빠른 시간에 1차적인 필터링을 하고 의심이 가는 링크를 고 상호작용 클라이언트 허니팟이 처리 하도록 하여 효율적인 하이브리드 클라이언트 허니팟을 구현해 보고자 한다.

참고문헌

〔1〕 JSUnpack.http://jsunpack.blogspot.com/

〔2〕 JSbeautifier. http://jsbeautifier.org/

〔3〕 U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors," Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more, pp. 8-8, 2009.

〔4〕 Capture-HPC. https://projects.honeynet.org/capture-hpc

〔5〕 Y.M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen and S. King, "Automated web patrol with strider honeymonkeys," Proceedings of the 2006 Network and Distributed System Security Symposium, pp. 35-49, 2006.

〔6〕 L. Lu, V. Yegneswaran, P. Porras and W. Lee, "Blade: an attack-agnostic approach for preventing drive-by malware infections." Proceedings of the 17th ACM conference on Computer and communications security, pp. 440-450, 2010.

〔7〕 J. Nazario, "PhoneyC: a virtual client honeypot," Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more, pp. 6-6, 2009.

〔8〕 MITRE. Common Vulnerabilities and Exposures, 2011. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3544

〔9〕 MITRE. Common Vulnerabilities and Exposures, 2012. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0003

〔10〕 MITRE. Common Vulnerabilities and Exposures, 2011. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2140

〔11〕 MITRE. Common Vulnerabilities and Exposures, 2010. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0806

〔12〕 MITRE. Common Vulnerabilities and Exposures, 2011. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0754

〔13〕 MITRE. Common Vulnerabilities and Exposures, 2011. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1255

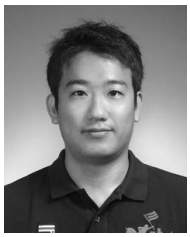
〔14〕 이스트소프트(Estsoft), 알약 월간 보안동향 보고서, 2011년 12월. http://alyac.altools.co.kr/SecurityCenter/Issue/TrendReport.aspx

〔15〕 VirusTotal. http://www.virustotal.com/

〔16〕 Google Safe Browsing. https://developers.google.com/safe-browsing/

〔17〕 Alexa Top Sites. http://www.alexa.com/topsites/

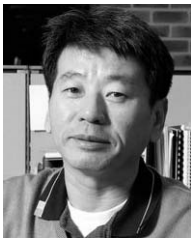
〈著者紹介〉



이 영 욱 (Youngwook Lee) 학생회원
2011년 7월 : Tsinghua University 소프트웨어공학 학사 졸업
2011년 9월~현재 : KAIST 정보보호대학원 석사과정
〈관심분야〉 정보보호, 컴퓨터공학, 소프트웨어 보안

정 동 재 (Dongjae Jung) 학생회원
2011년 8월: 아주대학교 정보 및 컴퓨터공학 학사 졸업
2011년 9월~현재: KAIST 정보보호대학원 석사과정
〈관심분야〉 정보보호, 컴퓨터공학, 소프트웨어 보안

전 상 훈 (Sang-hun Jeon) 정회원
2000년 2월: 울산대학교 산업공학과 졸업
2010년 8월~현재: KAIST 사이버보안연구센터 연구개발팀장
2011년 5월~현재: 빛스캔(주) 개발팀장
〈관심분야〉 보안동향, 침해사고 대응, 보안이슈 분석



임 채 호 (Chae-ho Lim) 종신회원
1986년: 홍익대학교 전산학과 학사
2001년: 홍익대학교 전자계산학과 박사
2006년~2009년: NHN(주) 보안실 실장, 연구센터 수석
2009년: 한국정보보호학회 부회장
2010년 8월~현재: KAIST 사이버보안연구센터 연구부소장
2011년 2월~현재: KAIST 정보보호대학원 연구교수
〈관심분야〉 인터넷 보안, 정보보호 위험 관리, 정보보호 관리 및 정책