情報保護學會誌
第18卷 第3號, 2008. 6

클라이언트 애플리케이션에서의 해킹 위협

김민성*, 정덕영**

요약

인터넷 사용의 증가와 함께, 일반인들의 악성 코드 혹은 악의적인 해킹에 대한 피해가 날로 증가해 가고 있다. 특히 과거 인터넷이 연결되어 있던 특정 서버만을 대상으로 삼았던 공격들이, 지금은 인터넷에 연결 되어 있는 일반 개인 사용자들의 애플리케이션 취약점을 이용한 공격들로 확대되고 있다. 특히 우리나라의 경우, 매우 급속한 인터넷 사용자들이 증가하고 온라인 상에서의 쇼핑과 게임 커뮤니티 활동이 매우 활발히 이뤄지고 있어, 인터넷을 이용한 해킹에 매우 많이 노출되어 있다. 본 글에서는 개인 사용자들의 대다수 PC에서 사용되는 애플리케이션인 온라인 게임 프로그램 및 웹 브라우저에서의 취약점들과 해킹 유형에 대하여 살펴본다.

Ⅰ. 서론

인터넷 사용의 증가와 함께, 일반인들의 악성 코드 혹은 악의적인 해킹에 대한 피해가 날로 증가해 가고 있다.

특히 과거 인터넷이 연결되어 있던 특정 서버만을 대상으로 삼았던 공격들이, 지금은 인터넷에 연결 되어 있는 일반 개인 사용자들의 애플리케이션 취약점을 이용한 공격들로 확대되고 있다.

우리나라의 경우, 매우 급속한 인터넷사용자들의 증가와 함께 온라인에서의 쇼핑과 게임 커뮤니티 활동이 매우 활발히 이뤄지고 있어, 인터넷을 이용한 해킹에 매우 많이 노출 되어 있다

본 글에서는 개인 사용자들의 대다수 PC에서 사용되는 애플리케이션인 온라인 게임 프로그램 및 웹 브라우저에서의 취약점들과 해킹 유형에 대하여 살펴보도록 하겠다.

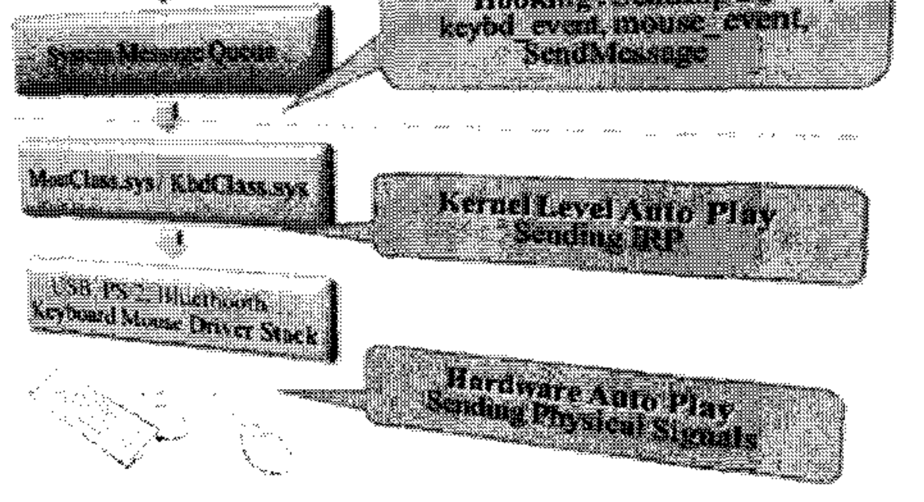
Ⅱ. 온라인 게임 애플리케이션 공격

온라인 게임에서의 해킹 유형으로써는 크게, 오토 마우스, 파일 및 메모리 변조, 스피드 핵 그리고 논 클라이언트 봇이 있다.

2.1 오토마우스

첫번째 유형인 오토 마우스는 크게 소프트웨어 방식의 오토 마우스와 하드웨어 형태의 오토 마우스로 분류할 수 있다[그림 1].

소프트웨어 방식은 주로 마이크로소프트 윈도(Windows)에서 사용자의 입력을 시뮬레이션 해줄 수 있는 API인 SendInput, keybd_event, mouse_event등의 함수를 사용하여 게임프로그램에 사용자의 입력이 이루어 진 것처럼 만들어 주는 방식을 사용하고 있으며, 하드웨어 방식 오토마우스는 USB 스틱에 마우스 키보드 관련 디바이스 드라이버를 포함하여, 커널 레벨에서 마우스와 키보드가 발생한 것처럼 흉내내도록 하는 방식을 사용하고 있다.



〔그림 1〕 오토마우스 분류

---

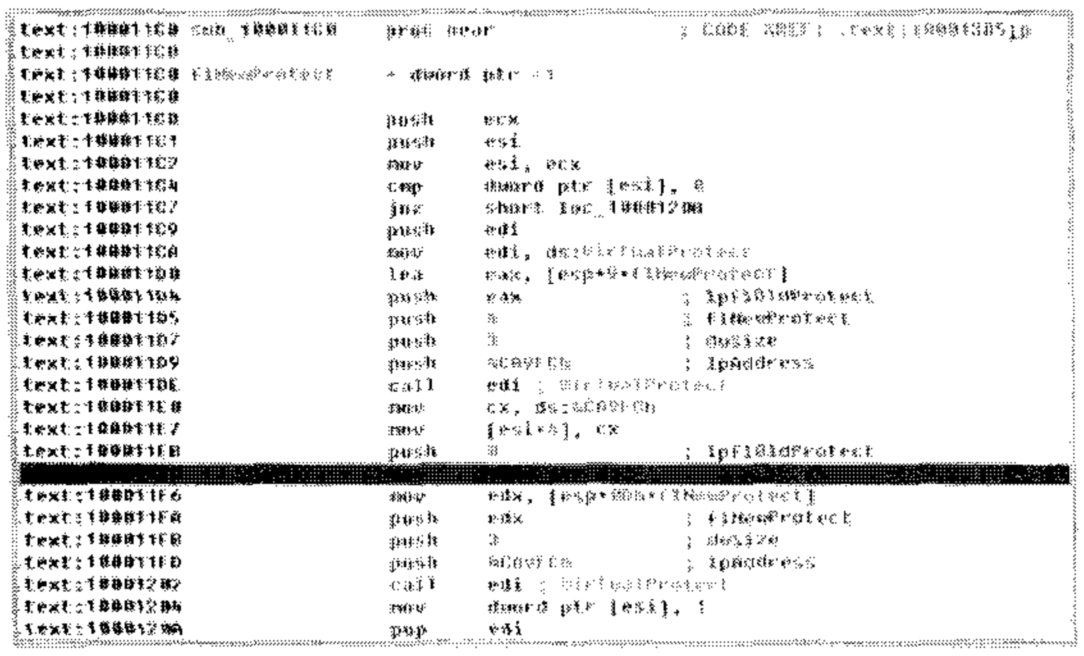
* 안철수연구소 ASEC팀 주임연구원 (dolka@ahnlab.com)

** 안철수연구소 선행기술팀 선임연구원 (human@ahnlab.com)

2.2 메모리 및 파일 변조

두번째 유형인 메모리 및 파일 변조는 게임의 주요 행동에 영향을 미치는 부분을 수정하는 해킹기법이다. 예를 들어, 농구 게임의 슛 성공판단 코드 부분을 수정하여 언제나 슛이 들어 간 것처럼 한다든지, 중요한 게임의 리소스량을 저장하고 있는 글로벌 변수를 수정하여, 게임의 리소스량을 가짜로 만들어 주는 등의 해킹이 많이 이루어 진다.

[그림 2]의 디스어셈블 코드는 게임해킹에서 많이 사용되었던 코드의 일부분으로서 특정 게임 프로그램의 메모리 조작을 위하여 게임 코드의 메모리 속성을 읽기 쓰기가 가능하도록 수정한 후 코드 변경을 수행 하는 코드이다.

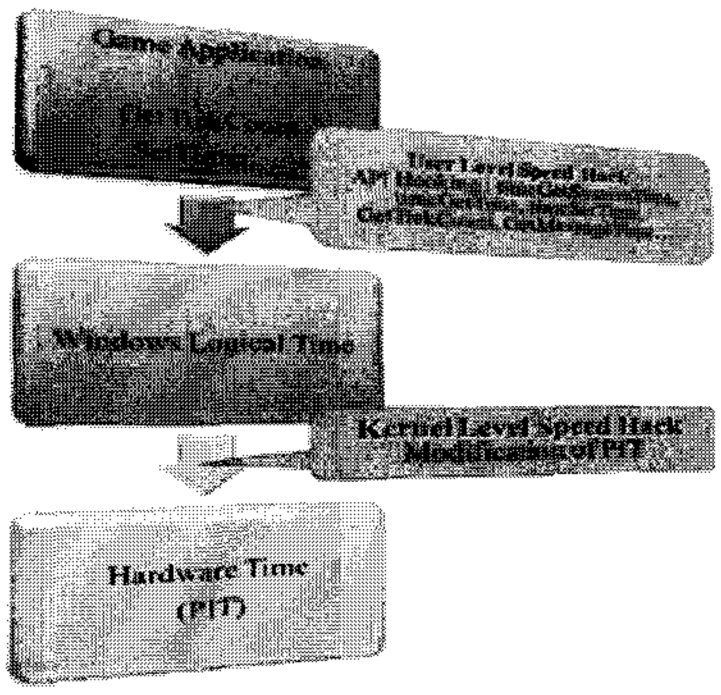


```
text:100011C0 sub_100011C0    proc near              ; CODE XREF: .text:100013B5↓p
text:100011C0
text:100011C0 flNewProtect    = dword ptr -4
text:100011C0
text:100011C0                 push    ecx
text:100011C1                 push    esi
text:100011C2                 mov     esi, ecx
text:100011C4                 cmp     dword ptr [esi], 0
text:100011C7                 jnz     short loc_1000120A
text:100011C9                 push    edi
text:100011CA                 mov     edi, ds:VirtualProtect
text:100011D0                 lea     eax, [esp+0Ch+flNewProtect]
text:100011D4                 push    eax              ; lpflOldProtect
text:100011D5                 push    4                ; flNewProtect
text:100011D7                 push    3                ; dwSize
text:100011D9                 push    4C09FCh          ; lpAddress
text:100011DE                 call    edi ; VirtualProtect
text:100011E0                 mov     cx, ds:4C09FCh
text:100011E7                 mov     [esi+4], cx
text:100011EB                 push    0                ; lpflOldProtect
[redacted]
text:100011F6                 mov     edx, [esp+0Ch+flNewProtect]
text:100011FA                 push    edx              ; flNewProtect
text:100011FB                 push    3                ; dwSize
text:100011FD                 push    4C09FCh          ; lpAddress
text:10001202                 call    edi ; VirtualProtect
text:10001204                 mov     dword ptr [esi], 1
text:1000120A                 pop     esi
```

〔그림 2〕 메모리 및 파일변조에 사용되는 코드

2.3 스피드핵

세번째 유형인 스피드핵[그림 3]은 윈도의 시간 관련 API들을 후킹(Hooking)하여 게임에서 해당 함수를 사용하여 원하는 타이밍을 맞추려 할 때, 그 타이밍을 늦게 혹은 빠르게 동작하도록 만드는 해킹 기법이다. 이러한 해킹 기법들은 현재 대부분의 게임 해킹 방지 솔루션들이 차단하고 있으며, 이로 인해 시스템의 8254 PIT(Programmable Interval Timer)칩을 조작하는 방식이 많이 사용된다, 이 방식은 윈도 시스템에서 동작하는 시간이 이 칩에 의해 발생하는 특정 시간 동안의 인터럽트 발생 횟수를 이용하여 카운트 하는 논리적 개념의 시간이기에, 이 인터럽트 주기를 변경함으로써, 전체적으로 Windows가 늦기도 빠르게도 보이도록 하는 해킹 방법이다.

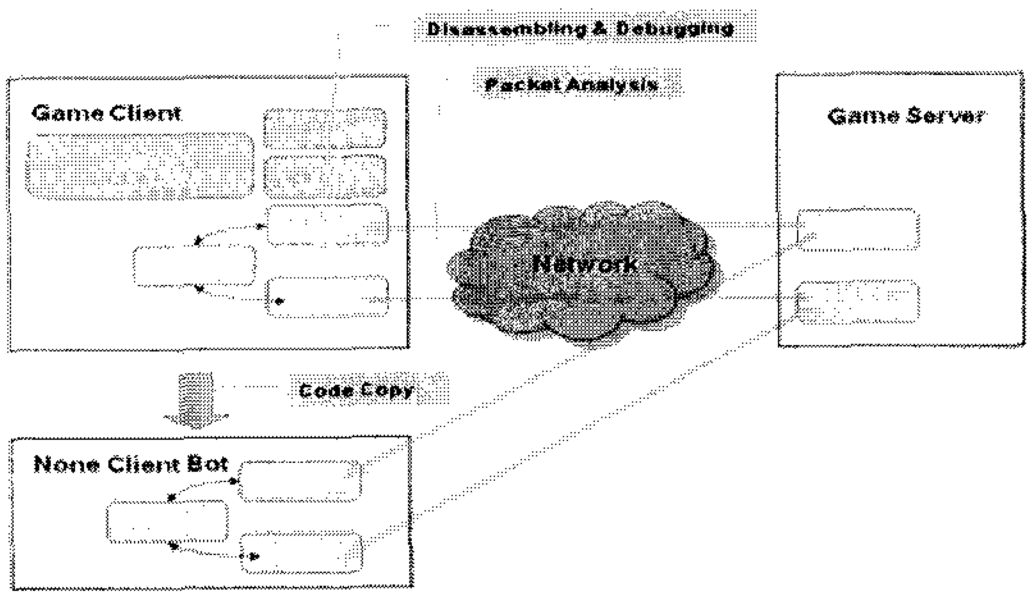


〔그림 3〕 스피드핵

2.4 논 클라이언트 봇(Non-Client Bot)

네 번째 유형으로는 논 클라이언트 방식의 해킹 기법이다. 이는 클라이언트에서 실제 게임이 동작하고 있지 않고 있음에도, 게임 서버에서는 마치 실제 게임이 동작하고 있는 것처럼 인식하도록 하는 방식이다. 이는 실제 게임의 주요 코드를 가짜로 만들어진 논 클라이언트 봇에 삽입한 후, 원하는 동작만을 수행하는 별도의 프로그램을 만들어 그 프로그램이 실제 게임서버에 접속하여 자동적으로 게임이 수행되도록 한다[그림 4].

이러한 논 클라이언트 봇은 게임의 주요 코드를 가짜 프로그램에 삽입하는 방식이므로 이러한 봇이 나온 경우 게임에 대한 많은 부분이 해커에 의해 자유 자재로 사용할 만큼의 분석이 이루어 졌다 할 수 있다



〔그림 4〕 논 클라이언트 봇 공격 방법

Ⅲ. 웹 브라우저 공격

3.1 웹 브라우저 공격 개요

취약점 공격 코드는 특정 애플리케이션의 취약점을

이용하여 해당 애플리케이션이 위치하는 시스템이 비정상적으로 동작하게 하거나 시스템의 권한을 획득하는 코드를 말한다. 일단 공격자가 해당 시스템의 권한을 획득하게 되면, 공격자는 웜이나 바이러스 같은 형태의 악성 코드를 사용자의 시스템에서 실행하거나 개인 정보를 획득할 수 있다.

현재 취약점 공격 코드의 대부분은 클라이언트 애플리케이션을 대상으로 하고 있으며, 이중 특히 웹 브라우저가 주요 공격 대상이 되고 있다. 그 이유는 다음과 같다.

첫째, 가장 많은 다수의 사용자를 공격대상으로 할 수 있기 때문이다. 인터넷의 사용이 대중화되면서, 웹브라우저는 일반 클라이언트 사용자가 가장 많이 사용하는 애플리케이션이 되었다. 따라서 웹브라우저를 공격대상으로 할 경우, 다수의 불특정 사용자들을 공격할 수 있으며 그 영향도 매우 광범위하다.

둘째, 다양한 애플리케이션을 공격할 수 있다. 웹 브라우저는 운영체제에서 사용하는 대부분의 파일 및 애플리케이션을 로드할 수 있다. 따라서 웹 브라우저를 공격대상으로 하는 공격코드는 브라우저 내에 존재하는 취약점뿐만 아니라 취약점이 존재하는 다른 애플리케이션을 공격대상으로 할 수 있다. 예를 들어, 취약점이 존재하는 ActiveX 애플리케이션의 공격을 위해 인터넷 익스플로러를 사용할 수 있다.

셋째, 메모리 조작이 쉽다. 웹 브라우저를 포함한 일부 애플리케이션들은 자바스크립트와 같은 스크립트 코드를 실행할 수 있다. 스크립트 코드가 사용하는 변수는 모두 동적으로 할당되므로 스크립트 코드가 실행되면 해당 애플리케이션은 메모리 할당을 동적으로 운영체제에 요청한다. 이러한 성질을 이용하면, 스크립트 코드를 사용하여 해당 애플리케이션의 메모리를 쉽게 조작할 수 있다. 일반적으로 애플리케이션의 취약점을 공격하는 취약점 공격 코드는 대부분 공격 대상 시스템의 권한을 획득하기 위해 해당 애플리케이션의 메모리를 조작하여 일련의 CPU 명령어들로 이루어진 쉘코드를 공격 대상 시스템의 메모리에 적재하고, 쉘코드가 적재된 메모리의 주소로 프로그램을 분기하여 쉘코드가 실행되도록 구성되어 있다. 쉘코드는 시스템의 권한 획득을 목표로 제작된 코드로, 쉘코드가 일반 사용자의 시스템에서 실행되면 공격자는 해당 시스템의 권한을 획득할 수 있다.

[그림 5]의 스크립트 코드는 “string” 문자열을 memory 배열의 첫 번째 원소에 저장하는 코드이다. [그림 1]의 코드가 웹 브라우저에서 실행되면, 웹 브라우저는 해당 문자열을 저장하기 위해 운영체제에게 동적인 메모리 할당을 요청한다. 결국 해당 문자열은 브라우저에 힙(Heap) 영역에 저장된다. 같은 방법으로 쉘코드에 해당하는 문자열을 특정 변수에 저장하는 스크립트 코드가 실행되면 쉘코드가 브라우저의 힙 영역에 저장된다. 즉, 웹 브라우저는 스크립트를 사용한 메모리 조작이 쉽게 가능하기 때문에 취약점 공격이 다른 애플리케이션에 비해 쉽다.

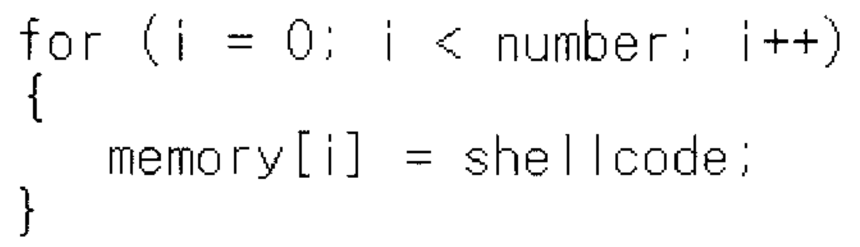
```
memory[0] = "string";
```

〔그림 5〕 문자열 변수의 동적인 할당

3.2 웹 브라우저 공격의 실제

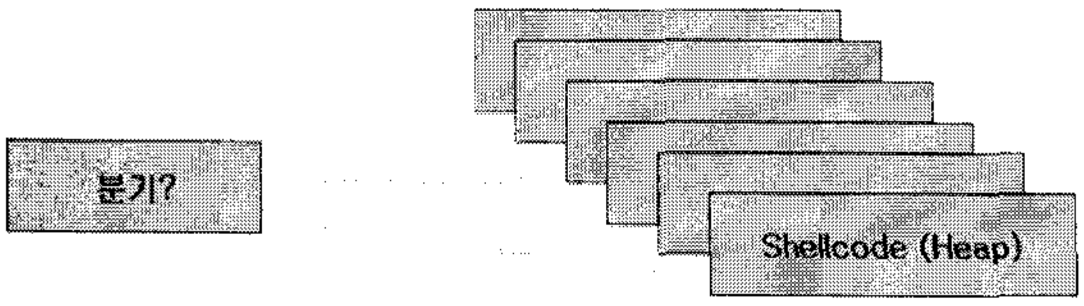
웹 브라우저 취약점 공격방법 중 가장 대중화된 방법은 힙 스프레잉(Heap Spraying) 공격이다. 이 방법은 스크립트 코드를 이용한 메모리 조작 공격기법으로, 힙 스프레잉 공격코드가 실행되면, 쉘코드를 브라우저의 메모리에 적재하고, 쉘코드가 존재하는 곳으로 프로그램을 분기하는 분기한다. 보통 전통적인 공격기법은 애플리케이션의 스택 메모리 영역에 쉘코드를 적재하지만, 스택의 주소를 추적하기 어렵거나, 스택에서 코드 실행이 불가능하도록 조치한 애플리케이션에서는 쉘코드가 스택에서 실행될 수 없다는 단점이 있다. 따라서 쉘코드의 실행이 가능하고 쉘코드가 저장된 메모리의 주소를 쉽게 알 수 있도록 쉘코드를 애플리케이션의 힙 영역에 저장한다. 이후 쉘코드가 위치한 곳으로 프로그램을 분기하여 쉘코드가 실행되도록 한다.

보통, 동적으로 할당되는 힙 영역의 메모리는 그 주소가 항상 동일하지 않기 때문에 쉘코드가 저장된 메모리의 주소를 추측하는 것은 쉽지 않은 작업이다. 하지만 인터넷 익스플로러와 같은 브라우저에서는 스크립트 코드 실행으로 인한 메모리의 레이아웃이 비교적 일정하다. 또한 힙 스프레잉 기법을 사용하는 공격자는 공격자가 원하는 위치에 쉘코드를 저장하기 위해 비교적 많은 수의 쉘코드가 담긴 메모리 블록을 힙 영역에 배치한다. [그림 6]의 스크립트 코드는 number만큼 애플리케이션의 힙 영역에 쉘코드를 저장하는 코드이다. number의 값이 커질수록 쉘코드가 존재하는 메모리 블록의 수가 많아지므로 공격자가 추측한 영역에 쉘코드가 배치될 확률이 높아지기 때문에 공격이 성공할 확률도 높아진다[그림 7].



```
for (i = 0; i < number; i++)
{
   memory[i] = shellcode;
}
```

(그림 6) 쉘코드 복사



(그림 7) 다수의 쉘코드 복사 후 분기

3.3 웹 브라우저 공격의 방어

힙 스프레잉 기법을 사용하여 웹 브라우저의 메모리를 조작하고, 취약점을 공격하는 것을 방어하기 위해서는 방어하는 방법은 코드의 시그너처를 사용하여 텍스트를 탐지하는 정적인 방법과 힙 스프레이 코드의 실행 특성을 탐지하는 동적인 방법이 있다.

3.3.1 시그너처 기반 탐지 방법

이 방법은 안티바이러스 제품이 악성코드를 탐지하는 전통적인 방법으로 힙 스프레잉 기법을 사용하는 공격 코드의 시그너처를 사용하여 스크립트 코드를 탐지하는 방법이다. 하지만 스크립트 코드가 암호화와 같은 방법으로 변형될 경우, 기존 시그너처로는 변형된 동일한 코드를 탐지할 수 없다는 단점이 있다. 현재 시그너처 기반의 탐지 방법을 우회하기 위해 고안된 암호화 방법은 계속해서 나타나고 있으며, 이 모든 경우를 탐지하기 위한 시그너처의 수는 암호화된 스크립트의 수를 따라가지 못하고 있다.

(표 1) 쉘코드 복사

<table>
<tr><th>원스크립트</th><th>암호화된 스크립트</th></tr>
<tr><td>alert ("javascript")</td><td>eval(function(p,a,c,k,e,r){e=String;if(!''.replace(/^/,String)){while(c--)r[c]=k[c]||c;k=[function(e){return r[e]}];e=function(){return'\\w+'};c=1};while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p}('0("1")',2,2,'alert|javascript'.split('|'),0,{}))</td></tr>
</table>

[표 1]은 원 스크립트 코드와 암호화 기법을 사용하여 암호화된 스크립트를 나타낸다. 두 개의 코드 모두 같은 역할을 하는 코드이지만, 원 스크립트와 암호화된 스크립트의 모습은 매우 상이하다.

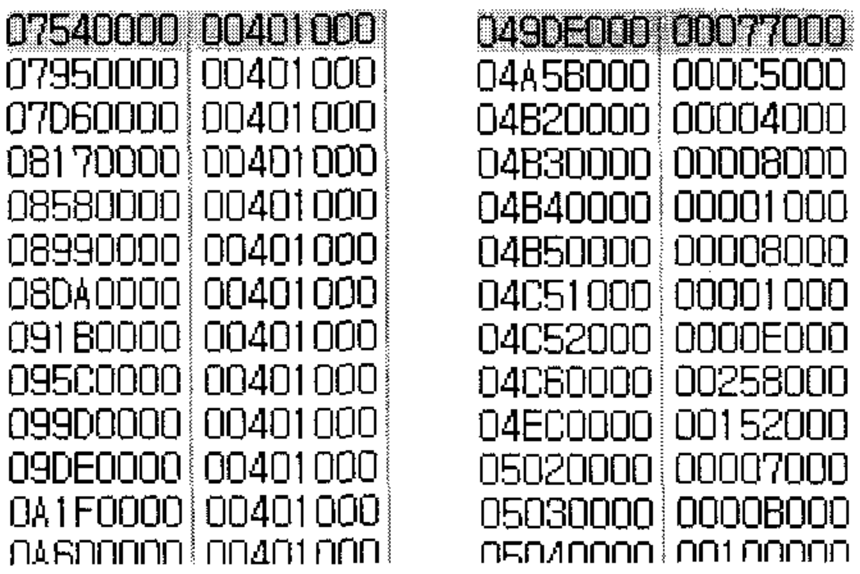
3.3.2 행위 기반 탐지 방법

이 방법은 스크립트가 실행될 때 발생하는 행동 특성에 의거하여 힙 스프레잉 공격 코드의 실행을 탐지하고 차단하기 위한 방법이다. 힙 스프레잉 코드가 실행될 때의 행동 특성은 스크립트의 암호화 여부에 관계없이 모두 같기 때문에 행위 기반 탐지 방법을 사용하면 거의 모든 힙 스프레잉 기법을 사용한 공격코드의 실행을 탐지하고 차단할 수 있다. 힙 스프레잉 코드가 실행될 때 발생하는 행위 특성은 다음과 같다.

3.3.2.1 메모리 레이아웃

힙 스프레잉 기법을 사용하는 코드는 공격자가 원하는 주소에 쉘코드를 저장할 수 있도록 많은 수의 메모리 영역의 할당을 동적으로 요청하고, 할당된 블록에 쉘코드를 저장한다. 따라서 힙 스프레잉 기법을 사용하는 코드가 실행된 이후 애플리케이션의 메모리 영역에는 동일한 크기를 가진 일련의 메모리 블록이 위치한다.

[그림 8]의 왼쪽과 오른쪽 그림은 각각 정상적인 경우와 힙 스프레잉 코드가 실행된 이후 웹 브라우저의 메모리 레이아웃을 나타낸다. 크기가 각기 다른 메모리 영역이 배치되어 있는 왼쪽 그림과 달리, 오른쪽 그림에서는 크기가 동일한 일련의 메모리 영역이 배치되어 있다.



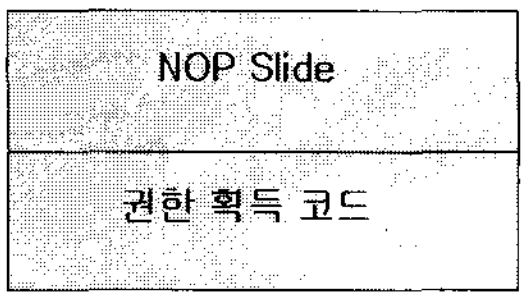
```
07540000 00401000      049DE000 00077000
07950000 00401000      04A5B000 000C5000
07D60000 00401000      04B20000 00004000
08170000 00401000      04B30000 00008000
08580000 00401000      04B40000 00001000
08990000 00401000      04B50000 00008000
08DA0000 00401000      04C51000 00001000
091B0000 00401000      04C52000 0000E000
095C0000 00401000      04C60000 00258000
099D0000 00401000      04EC0000 00152000
09DE0000 00401000      05020000 00007000
0A1F0000 00401000      05030000 0000B000
```

(그림 8) 메모리 레이아웃 비교

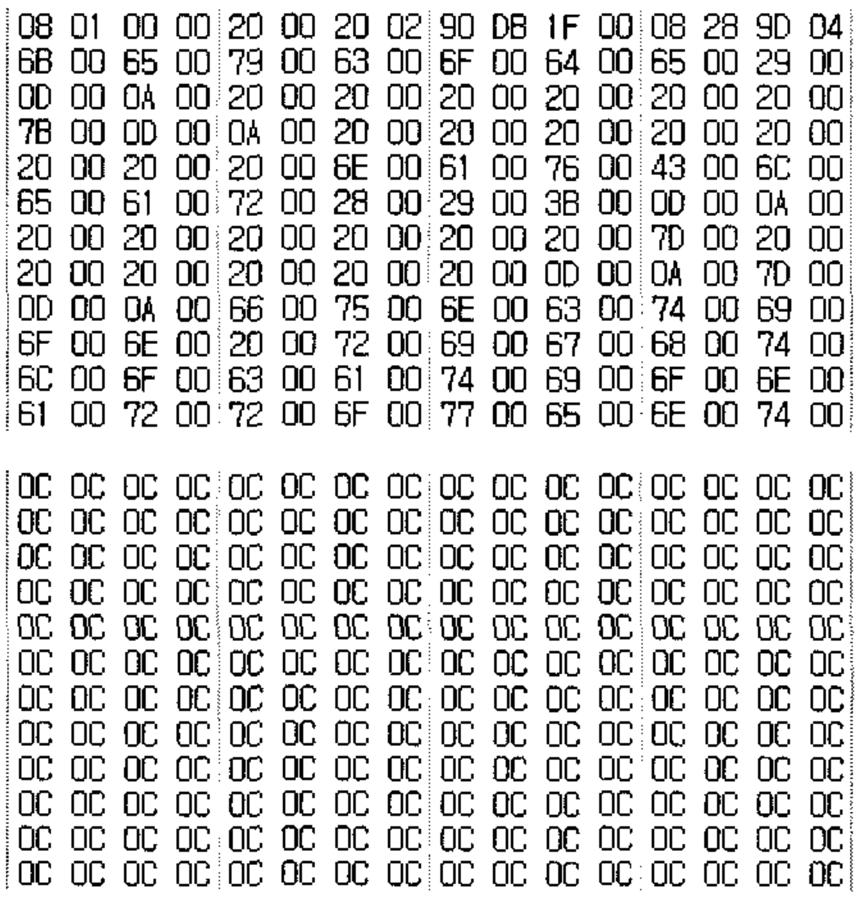
3.3.2.1. 엔트로피

일반적으로 쉘코드는 권한을 획득하기 위한 실제 코드앞에 많은 수의 NOP(No Operation) 명령어들을 가

지고 있는 것이 일반적이다[그림 9]. 쉘코드가 저장된 위치로 프로그램이 분기하도록 하려면, 공격자는 쉘코드가 저장된 메모리의 주소를 정확히 알고 있어야 한다. 하지만 많은 수의 NOP 코드가 쉘코드 앞에 존재하면 공격자는 쉘코드가 시작하는 정확한 위치를 알지 못하더라도 NOP 명령어 중 어느 한 곳으로 프로그램이 분기하도록 하면 되기 때문에 공격이 성공할 확률이 높아진다. 따라서 쉘코드가 배치하는 메모리 블록에는 NOP 명령어의 수가 쉘코드에 해당하는 명령어의 수보다 많다[그림 10]. 이러한 특성은 메모리 레이아웃의 엔트로피에도 영향을 준다. 엔트로피는 특정한 데이터 집합의 분포의 균질성을 측정하는 데 사용되는 수치로[식 1]을 사용하여 계산할 수 있다. 일반적으로, 특정한 데이터 집합 내에 같은 값을 나타내는 데이터가 많을수록 엔트로피의 값은 낮아지고 그렇지 않은 경우 높아진다. 쉘코드가 존재하는 메모리영역은 NOP 코드가 많기 때문에 블록 전체의 엔트로피는 낮아지게 된다. [표 1]은 정상 메모리 블록 2개와 쉘코드가 위치한 블록 1개의 엔트로피 값을 비교한 것이다.



(그림 9) 쉘코드 구성



```
08 01 00 00 20 00 20 02 90 DB 1F 00 08 28 9D 04
6B 00 65 00 79 00 63 00 6F 00 64 00 65 00 29 00
0D 00 0A 00 20 00 20 00 20 00 20 00 20 00 20 00
7B 00 0D 00 0A 00 20 00 20 00 20 00 20 00 20 00
20 00 20 00 20 00 6E 00 61 00 76 00 43 00 6C 00
65 00 61 00 72 00 28 00 29 00 3B 00 0D 00 0A 00
20 00 20 00 20 00 20 00 20 00 20 00 7D 00 20 00
20 00 20 00 20 00 20 00 20 00 0D 00 0A 00 7D 00
0D 00 0A 00 66 00 75 00 6E 00 63 00 74 00 69 00
6F 00 6E 00 20 00 72 00 69 00 67 00 68 00 74 00
6C 00 6F 00 63 00 61 00 74 00 69 00 6F 00 6E 00
61 00 72 00 72 00 6F 00 77 00 65 00 6E 00 74 00

0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
```

(그림 10) 정상블록(위)과 쉘코드(아래)블록

(표 2) 메모리 엔트로피 비교

메모리 블록	엔트로피
정상 블록 1	721.659
정상 블록 2	720.782
쉘코드	52.204

Ⅵ. 결 론

지금까지 클라이언트에서 주로 발생하는 온라인 게임과 웹 브라우저에서의 해킹 유형과 방어기법들에 대하여 살펴보았다.

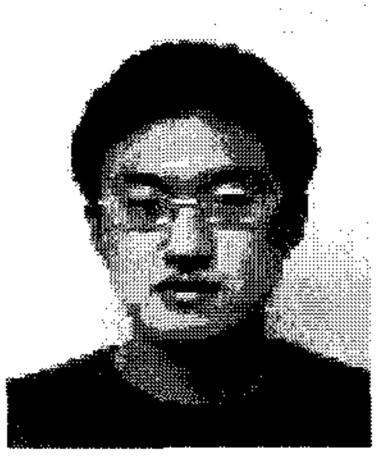
현재 많은 애플리케이션들이 인터넷과의 연동을 통한 온라인 서비스를 제공하고 있으며, 이는 이전보다 많은 보안 위협을 가지고 있음을 의미하기도 한다.

과거 애플리케이션 개발은 해당 목적에 필요한 기능과 성능에 초점을 맞추어 설계와 구현을 하고, 에러들을 처리함으로써 안정성을 확보하였다면, 지금은 이러한 설계와 구현 과정에서 보안 취약점들에 대해 분석함으로써 보안성을 강화하는 방법에 대하여 본격적으로 고민해 보아야 할 것이다.

참고문헌

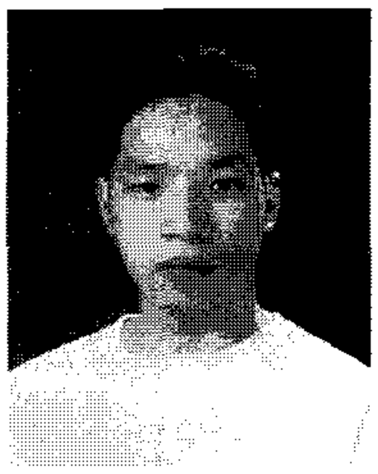
[1] 정덕영, 이호웅, "New Attack Trend in Online Game Security Area", Proc. of AVAR 2007

〈著者紹介〉



김 민 성 (Kim, Minseong)

2002년~2006년 : 서울대학교 전기 컴퓨터 공학부 박사과정 수료
2006년 ~ 현재 : 안철수연구소 ASEC팀 주임연구원
<관심분야> 소프트웨어 역공학, 취약점 분석, 네트워크 보안



정 덕 영 (Jung, Deokyoung)

1995년~2004년 : 광운대학교 컴퓨터 소프트웨어 공학과 졸업
1998년~현재 : 안철수연구소 선행기술팀 선임연구원
2003년 12월 : Windows 구조와 원리 그리고 Codes 저술 (가남사)
2006년 3월 : Windows 구조와 원리 저술 (한빛 미디어)