

네트워크 통신의 비밀 ARP

웹 브라우저를 띄운 후 아무 생각없이 접속하는 도메인(예:www.tt.co.kr) 이지만 실제로 이 도메인 네임은 내부적으로 DNS 에 의해 IP 주소로 바뀌어 IP 로 접속이 되는 것이다. 과연 그것이 다인가? 그렇지 않다. 실제로 IP 요청은 다시 ARP 라는 프로토콜에 의해 MAC 주소라는 하드웨어 주소로 변경된 후 접속이 되는 것이다. 이 과정은 그리 복잡하지는 않지만 이 과정을 제대로 이해하지 않고서는 “네트워크 보안”을 이야기 할 수 없을 정도로 많은 것들을 놓치게 된다. 실제로 ARP를 이용한 해킹은 그리 낯선 방법도 어려운 방법도 아니다. 이번 호에서는 많은 사람들이 그 중요성을 놓치고 있는 LAN 구간에서의 통신 프로토콜인 ARP 와 이와 관련하여 숨겨져 있는 재미있는 사실들에 대해 알아보도록 하자.

오늘과내일 넷센터 홍석범(antihong@tt.co.kr)

WAN 과 LAN

흔히들 일상적인 네트워크 구간을 크게 WAN(Wide Area Network) 구간과 LAN(Local Area Network) 구간으로 나눈다. 흔히 이러한 WAN과 LAN 구간을 나누는 장비로는 게이트웨이 역할을 하는 라우터가 담당하는데, 혹시 라우터라는 장비를 보신 분들도 있을지 모르겠지만 라우터라는 장비를 보면 적게는 두 개에서 많게는 몇 십개까지의 많은 인터페이스(랜카드)로 이루어져 있는 것을 볼 수 있을 것이다. 이 인터페이스들은 담당하는 역할에 따라 WAN 구간을 담당하는 시리얼(Serial)인터페이스와 LAN 구간을 담당하는 이더넷(Ethernet) 인터페이스로 나눌 수 있다. 따라서 흔히들 WAN 구간을 시리얼 구간, LAN 구간을 이더넷 구간이라 부르기도 한다. WAN 구간에서는 라우터와 라우터간에 서로 약속한 라우팅 프로토콜을 이용해 통신을 하게 되는데,(이를테면 전세계적으로 ISP간에 연동시에는 BGP 라는 프로토콜을 이용하여 연동한다. 이에 대해서는 CISCO 홈페이지등 라우팅 관련 서적을 참고하기 바란다.) 그렇다면 LAN 구간에서는 어떤 프로토콜을 이용하여 서로 통신하게 될까? 그렇다. 바로 “ARP” 이다.

ARP

ARP 는 Address Resolution Protocol 의 약자로서 마치 DNS 가 도메인 주소를 IP 주소로 바꾸어주는 것처럼 IP 주소를 MAC(Media Access Control) 주소로 변환해 주는 프로토콜이다. 그렇다면 MAC 주소는 또 무엇인가? MAC 주소는 각 이더넷 카드마다 유일한 하드웨어 주소로서 이 정보는 리눅스의 경우 ifconfig 로, 윈도우즈의 경우 ipconfig /all 로 확인할 수 있다.

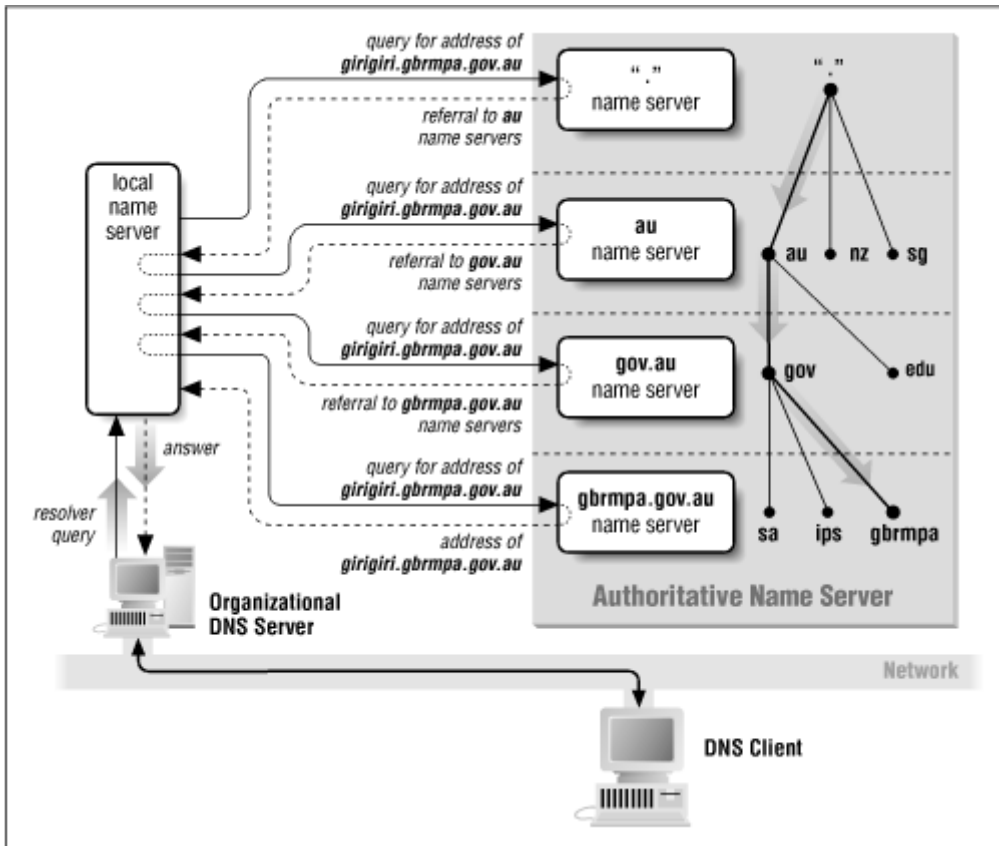
ARP는 일반 유저들에게는 그리 친숙하지 않은 단어이지만 이 프로토콜은 로컬 네트워크의 매우 중요한 규약이며 ARP를 이해하여야만이 네트워크의 보안을 설명할 수 있을 정도로 반드시 숙지하여야 할 중요한 프로토콜이다.

DNS와 ARP

ARP를 쉽게 이해하기 위해 아래 그림과 같이 자신의 PC에서 인터넷을 접속하는 경로를 생

각해 보자. 자신의 PC에서 브라우저를 실행하여 `www.tt.co.kr` 을 입력한 후 엔터를 치면 해당 도메인의 웹 페이지가 접속이 된다. 너무 당연한 것 같은 이 절차는 실제로 어떠한 경로를 통해 접속이 되는 것일까? 다소 지루할 지 모르겠지만 전체 네트워크를 이해하기 위해 처음 과정부터 거슬러 올라가 보도록 하자.

자신의 브라우저에서 도메인명을 입력하면 이 요구는 제일 먼저 자신의 PC에서 설정한 DNS 서버(이 DNS 서버를 편의상 Client DNS 라 하자.) 로 가게 된다. Client DNS 서버는 `www.tt.co.kr` 의 Resolving 요구(도메인을 IP주소로 변환) 를 받아 자신 스스로가 위임 권한이 있는지(즉, 자신의 `/etc/named.conf` 파일에 질의한 도메인이 정의되어 있는지) 여부를 확인하여 없을 경우에는, 질의한 도메인에 대한 캐싱이 있는지 여부를 확인하고 있을 경우 바로 해당 정보를 넘겨주지만 없을 경우에는 소위 root 서버라는 최상위 DNS 서버에 질의를 하게 된다. 일반적으로 웹 접속시 `http://www.tt.co.kr` 이라고 입력하지만 실제로 엄격히 이야기한다면 `http://www.tt.co.kr.` 와 같이 주소의 제일 끝에 `.` 을 붙이는 것이 맞는 것이다. 여기에서 주소 제일 끝에 있는 `.` 을 root 서버로 부른다. root 서버는 전 세계적으로 13 개가 있으며 만약 root 서버에 장애가 생기면 전 세계의 인터넷은 마비가 되고 말 것이다. 여하튼 질의를 받은 root 서버는 질의한 도메인의 주소가 `.kr` 인 것을 보고 `.kr` 을 관장하는 DNS 서버는 한국 전산원의 DNS 서버라는 답을 해 주고 이 답을 받은 Client DNS 서버는 한국 전산원의 DNS 서버에 다시 `www.tt.co.kr` 을 질의하게 된다. 이 질의를 받은 한국 전산원의 DNS 서버는 질의한 도메인이 `.co.kr` 인 것을 보고 이번에는 `co.kr` 을 관장하는 DNS 서버에 다시 질의하라고 응답하고, 이 응답을 받은 Client DNS 서버는 다시 `co.kr` 을 관장하는 DNS 서버에 `www.tt.co.kr` 을 질의하게 된다. 이 질의를 받은 DNS 서버는 `tt.co.kr` 을 보고 이 도메인을 관장하는 DNS 서버는 `ns1.tt.co.kr` 이라는 것을 알려주게 되고 이 응답을 받은 Client DNS 서버는 최종적으로 `ns1.tt.co.kr` 에게 `www.tt.co.kr` 을 질의하여 `211.47.65.1` 이라는 IP 주소를 알게 되어 이 주소를 요청한 PC 에게 알려주게 된다. 참고로 이와 같이 root 서버부터 시작해서 위임 권한이 있는 각 DNS 서버로 내려오면서 반복적인 질의를 하는 방법을 recursion 이라 한다.

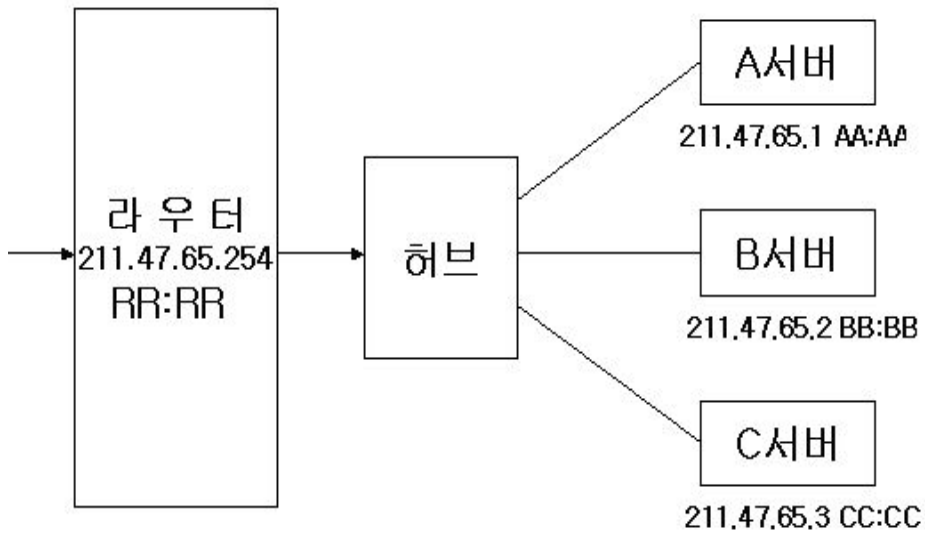


<그림1> DNS의 recursion 예

Client DNS 서버로부터 IP 주소를 받은 PC는 자신의 라우팅 정보에 따라 이 요구를 PC에 설정되어 있는 게이트웨이, 즉 라우터로 넘긴다. 이 요청을 받은 라우터는 라우터 자신의 라우팅 프로토콜에 따라 가장 최적화된(비용이 적은) 여러 라우터를 경유하여 접속하려는 서버가 위치한 최종 라우터까지 다다르게 된다. 이 경로는 자신의 윈도우즈 PC의 경우 도스 프롬프트에서 “tracert -d 목적지” 나 리눅스등의 유닉스 계열의 서버일 경우 “traceroute -n 목적지” 로 확인할 수 있다.

내부 네트워크에서의 통신원리

이제 해당 서버가 있는 라우터에서부터는 LAN 구간이므로 앞에서 이야기한대로 ARP 가 작동하게 된다. 이 부분을 아래와 같이 간단히 도식화하였고 라우터 하단에는 아래와 같이 각각의 IP 주소에 해당하는 MAC 주소가 각각 192.168.1.1(AA:AA), 192.168.1.2(BB:BB), 192.168.1.3(CC:CC) 인 시스템이 있다고 약식으로 가정하였다.



<그림2> 내부 네트워크 구성도

만약 라우터를 통해 접속하려는 요구가 A 서버인 211.47.65.1 이라면 여기에서부터 ARP가 작동하게 된다. 그런데, LAN 구간에서는 서버의 MAC 주소를 알아야 통신을 할 수 있는데, 라우터는 211.47.65.1의 MAC 주소를 아직 모르기 때문에 라우터에서는 밑단의 네트워크에 대해 211.47.65.1 인 서버는 MAC 주소를 알려 달라는 Broadcasting 패킷(이 패킷에는 라우터의 소스IP 주소와 소스 MAC 정보를 포함하고 있다.)을 보내게 된다. 이 패킷은 라우터 하단의 모든 서버들이 받게 되며 이 패킷의 목적지 IP 가 자신이 아닌 B나 C 서버는 이 요청에 대해 응답하지 않으며 이 요청을 받은 A서버(IP가 211.47.65.1)는 자신의 MAC 주소를 적은 패킷을 라우터에게 Unicast(즉, arp 요청을 한 라우터에게만)로 응답하게 된다. 이렇게 해서 라우터와 A 서버는 서로의 MAC 정보를 알게 되었으므로 드디어 통신을 할 수 있게 되었다. 아래는 실제 로컬 랜상의 arp 가 실시간으로 broadcasting 하고 있음을 tcpdump 로 캡처한 예이다. 아래에서 보는 것처럼 arp 요청 (arp who-has)은 Broadcasting 을 하지만 arp 응답(arp reply)은 Unicast(즉 arp를 요청한 IP 에게만 응답)하는 것을 알 수 있다. 참고로 아래 그림에서 arp reply를 하는 패킷이 하나가 아니라 몇 개 있는 것은 이 시스템의 환경이 dummy 이기 때문이다. 이에 대해서는 아래에서 다시 설명하기로 한다.

현재 리눅스 시스템의 arp 캐시정보는 아래와 같이 arp -a를 입력하면 된다.

```
[root@test1 ~]# arp -a
? (211.47.65.15) at 00:50:8B:9A:2E:A8 [ether] on eth0
? (211.47.65.254) at 00:02:FC:08:C4:A0 [ether] on eth0
as5.tt.co.kr (211.47.65.75) at 00:50:8B:9A:2C:AF [ether] on eth0
? (211.47.65.51) at 00:50:8B:9A:37:9A [ether] on eth0
[root@test1 ~]#
```

<그림4> arp 캐시 정보 확인

아래와 같이 라우터에서도 arp를 캐시하고 있는 것을 확인할 수 있다.

```
7513_ROUTER>
7513_ROUTER>sh arp
Protocol Address Age (min) Hardware Addr Type
Internet 211.47.64.184 0 00c0.26dc.02c4 ARPA
Internet 211.47.64.185 3 5254.05e0.abdd ARPA
Internet 211.47.64.186 0 004f.4e07.7588 ARPA
Internet 211.47.64.187 15 004f.4e07.7595 ARPA
Internet 211.47.64.180 1 5254.05f3.1f75 ARPA
Internet 211.47.64.181 1 004f.4e07.7589 ARPA
Internet 211.47.64.183 12 004f.4e07.7583 ARPA
Internet 211.47.64.178 10 004f.4e07.758f ARPA
Internet 211.47.64.172 42 004f.4e07.3851 ARPA
Internet 211.47.64.173 7 00c0.26dd.8972 ARPA
Internet 211.47.64.174 0 004f.4e07.758d ARPA
Internet 211.47.64.168 2 0050.ce30.0a3d ARPA
Internet 211.47.64.169 6 0050.bf75.9d2f ARPA
Internet 211.47.64.170 1 0002.2ac2.2e76 ARPA
Internet 211.47.64.171 1 0050.ce30.0a5f ARPA
Internet 211.47.64.164 0 0050.bf07.fc54 ARPA
Internet 211.47.64.165 6 0000.2104.24c6 ARPA
```

<그림5 > 라우터의 arp 캐시

여기에서 주의할 점은, arp 는 내부 LAN 에서만 의미가 있으므로 라우터를 벗어난 WAN 구간에서는 의미가 없으며 LAN 구간에서도 브로드 캐스팅 도메인 영역에서만 의미가 있다는 것이다. 따라서 arp 캐시는 해당 서버에 접근하는 모든 클라이언트나 서버에 대해서 캐시하는 것이 아니라 C Class 로 netmask 되어 있을 경우 해당 영역(위의 경우 211.47.65.0 대역)만 캐시하는 것을 볼 수 있다. 가끔 여러분의 서버나 PC의 이더넷 카드를 변경하였을 때 바로 인식이 되지 않고 빠르면 몇 분, 길게는 몇 시간까지 기다린 후에야 네트워크가 다시 작동하는 것을 경험해 보았을 것이다. 이를테면 이더넷 카드를 기존의 RealTek 8139에서 3com 계열로 바꾸었고, 커널에서도 잘 인식이 되었다. 모듈도도 해보고 커널에 정적으로 포함시켜 보기도 하고, 분명 시스템에서는 정상적으로 이더넷 카드를 인식하고 설정도 전혀 문제가 없는데, 네트워크가 작동하지 않는 경우가 있다. 왜 그럴까? 이제 여러분은 이 이유를 알 수 있을 것이다. 그렇다. 바로 arp 캐시 때문이다. 앞에서 라우터는 평균 4시간정도 arp를 캐시한다고 말했다. 따라서 이더넷 카드를 변경했다면 해당하는 IP 에 대한 MAC 주소가 변경되었는데, 라우터에서는 이전의 정보를 캐시하고 있는 것이다. 즉, 211.47.65.1의 MAC 주소를 3com 이 아닌 이전의 RealTek 으로 인식하고 있는 것이다. 따라서 외부에서 라우터를 통해 211.47.65.1로 접속 요구가 들어왔을 때 라우터는 이전의 정보인 AA:AA MAC 주소를 가지고 있는 서버를 찾는데, 이미 211.47.65.1은 3com 의 MAC 주소인 HH:HH 로 변경되었고 기존의 AA:AA MAC 주소는 존재하지 않기

때문에 접속할 수 없게 되는 것이다. 만약 이더넷 카드를 변경 후 바로 네트워크가 되었다면 다행히 arp 캐시가 작동하지 않았거나 캐시 시간이 짧았기 때문이었을 것이다. 그런데 arp 캐시 때문에 캐시가 업데이트 될 때까지 무작정 기다리기만 해야 한다면 너무 불합리하지 않은가? 당장 서비스도 중지없이 해야 하는데 말이다.

이를 위해서는 Gratuitous arp 라는 것을 사용하면 된다. gratuitous arp 는 로컬 랜상의 arp 캐시 정보를 업데이트하는 arp 로서 이를 실행하면 모든 캐시 정보를 바로 업데이트 할 수 있다. 리눅스에서는 send_arp 라는 실행파일을 이용하면 되며 이 실행 파일은 클러스터링 패키지인 piranha 라는 패키지에 포함되어 있다.

사용 방법은 send_arp src_ip_addr src_hw_addr targ_ip_addr tar_hw_addr 인데, 만약 /usr/sbin 디렉토리에 send_arp 실행 파일이 있고 211.47.65.1의 MAC정보를 기존의 AA:AA에서 HH:HH 로 변경하여 이 정보를 업데이트하려면 아래와 같이 실행하면 될 것이다.

```
# /usr/sbin/send_arp 211.47.65.1 HHHH 211.47.65.255 ffffffff
```

위와 같이 실행하면 211.47.65.255 즉 211.47.65.0 대역의 서버중 211.47.65.1 에 대한 arp 캐시를 가지고 있는 모든 서버에 대해 211.47.65.1의 MAC 주소는 HH:HH 로 업데이트하라고 브로드 캐스트하는 것이다. 이를 실행하면 라우터를 포함하여 모든 네트워크 장비가 arp 캐시를 업데이트하며 네트워크가 다시 작동하는 것을 확인할 수 있을 것이다. 참고로 자신의 MAC 정보는 아래와 같이 ifconfig 를 실행하면 확인할 수 있다.

```
[root@test1 ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:8B:9A:1B:1E
          inet addr:211.47.65.106 Bcast:211.47.65.255 Mask:255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2788822 errors:0 dropped:0 overruns:0 frame:1864
          TX packets:85554 errors:0 dropped:0 overruns:0 carrier:0
          collisions:26111 txqueuelen:100
          RX bytes:1794962086 (1711.8 Mb)  TX bytes:101435312 (96.7 Mb)
          Interrupt:11 Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:147 errors:0 dropped:0 overruns:0 frame:0
          TX packets:147 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:112998 (110.3 Kb)  TX bytes:112998 (110.3 Kb)
```

<그림6> MAC 주소의 확인

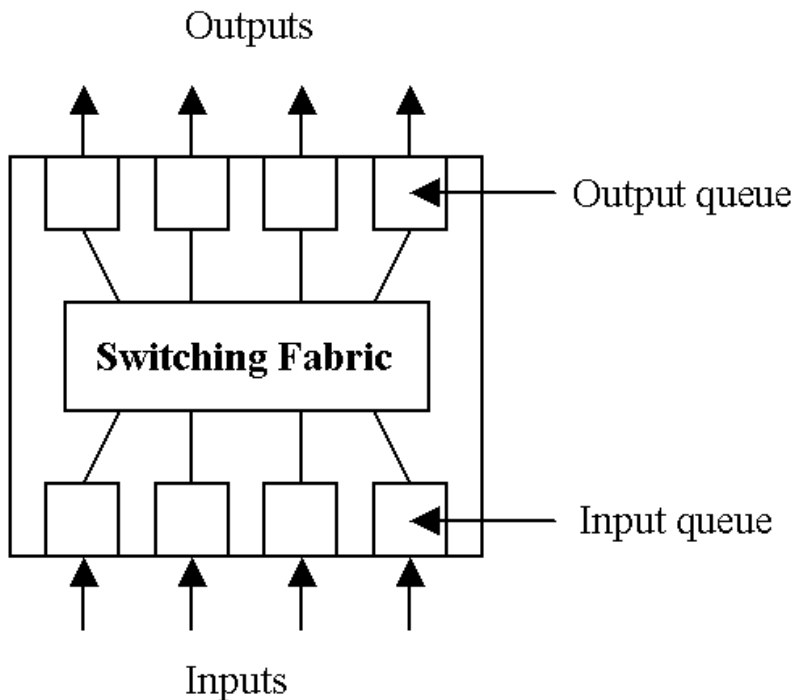
실제로 gratuitous arp를 이용하면 한 서버가 다운시 다른 서버가 대신 서비스하는 클러스터링의 용도로도 사용되며 매우 유용한 프로그램이라 할 수 있다. 이외 ARP와 관련하여 Unarp 라는 것도 있는데, 이는 Gratuitous arp 가 기존의 arp 캐시를 다른 정보로 업데이트하는 것이라면 Unarp 는 기존의 arp 캐시가 있는 시스템이 있다면 단지 arp 캐시를 삭제만 하는 것이다. 따라서 접속할 때마다 IP가 변하는 ADSL 등의 DHCP의 경우는 접속을 끊을 때 Unarp를 Broadcasting 하고, IP를 할당받아 네트워크를 시작할 때에는 Gratuitous arp를 실행하면 arp 캐시로 인하여 접속이 되지 않는 문제는 해결이 될 것이다.

이렇듯 gratuitous arp 는 매우 유용하지만 다른 한편으로 악용될 경우에는 쉽게 다른 시스템을 무력화시킬 수 있는 무서운 공격 프로그램이 될 수도 있다는 점을 양지하기 바란다.

Switch 와 Dummy 허브

이번에는 로컬구간에서의 통신에 대해 알아보도록 하자.

많은 사람들은 LAN 네트워크의 보안을 위해 더미 허브대신 스위치를 사용하라고 권장한다. 과연 그런가? 실제로 스위치는 애초에 디자인시 보안을 고려했기 보다는 Performance 를 염두해 두고 개발된 것인데, 어떻게 하다보니 우연히 더미(Dummy) 허브에 비해 보안에 좋은 것 뿐이지 반드시 그러한 것은 아니다. 그럼, 여기서 잠깐 더미허브와 스위치의 차이에 대해 알아보도록 하자. 더미 허브와 스위치는 일반적으로 차선의 개념과 연관지어 생각하면 쉽다.

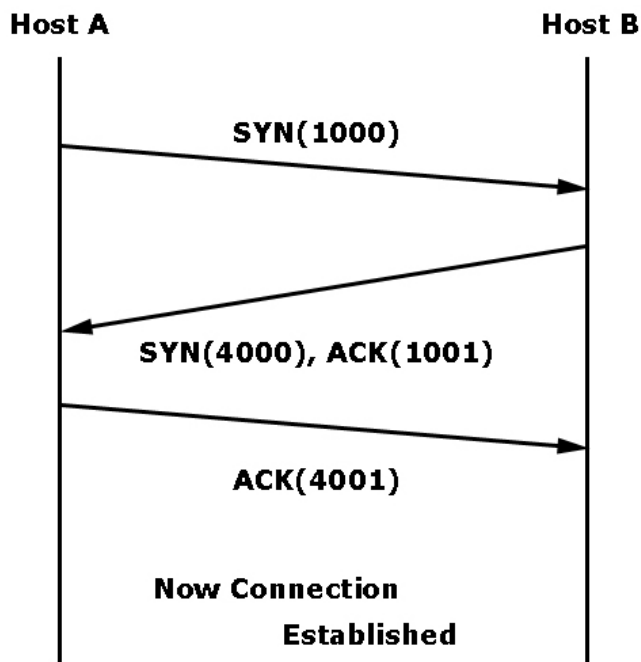


<그림7> Switch 의 개념

스위치의 경우 위와 같이 input이 4차선이면 output도 4차선인 개념이다. 반면에 더미 허브는 input 은 4차선이지만 output 은 1차선인 개념이므로 당연히 output에서 병목현상이 발생할 수밖에 없게 된다. 이때 병목 현상으로 인하여 동시에 output 에 도달하려면 필연적으로 collision(충돌) 이 생기게 되어 결국 속도 저하가 발생하게 된다.(리눅스에서 특정 인터페이스의 collision 여부는 ifconfig 로 확인할 수 있다.) 그런데, 이러한 속도 저하보다는 좀 더 중요한 차이점이 있다. 즉, 스위치는 지정된 IP로 패킷을 바로 포워딩해 주지만 더미 허브는 모든 패킷을 모든 포트에 broadcast 한다는 것이다. 그럼 여기에서 각각 더미허브 및 스위치 환경에서 데이터가 통신하는 경로에 대해 다시 생각해 보자.

Dummy

더미(dummy) 환경에서 A 서버와 C 서버가 telnet 과 같은 TCP 통신을 한다면, A 서버에서는 제일 먼저 ARP Broadcasting을 하게 된다. 이때 이 요청은 Broadcasting 이므로 B, C 모두에게 전달된다. 이중 B 서버는 ARP 목적지 IP가 자신의 IP가 아니므로 이 요청에 응답하지 않고 C 서버는 목적지가 자신의 IP 이므로 이 요청에 대하여 요청을 한 A 서버에게 ARP reply 를 한다. 그러나 dummy 환경에서는 모든 패킷이 broadcast 하므로 B 서버에서도 C 서버의 unicast 응답을 캡처할 수 있다.(이는 위의 tcpdump 그림에서 확인하였다.) A 와 C 간에 ARP broadcast와 reply를 한 후에는 TCP 통신을 하려고 하므로 본격적인 데이터 교환을 하기 전에 TCP 3 Way handshake 과정을 선행하게 된다.



<그림8> TCP 3 way handshake

tcp 3 way handshake 과정은 위 그림에서와 같이 서버와 클라이언트간에 위 3가지 절차를 거친 후에야 연결이 성립되어 데이터가 교환된다는 것을 익히 알고 있을 것이다. 그런데, dummy 에서는 위 3 way handshake 뿐만이 아니라 Connection 이후의 모든 데이터 교환 역시 broadcast 하게 된다. 아래에서는 211.47.65.106 서버에서 www35.tt.co.kr 서버로 telnet 접속시 반응하는 패킷을 www35.tt.co.kr 서버에서 tcpdump 로 잡은 예이다. tcpdump 에서 보는 것처럼 먼저 arp 통신이 되어 서버의 MAC 정보를 교환한 후에야 비로써 3 way handshake 가 이루어지는 것을 확인할 수 있을 것이다. 만약 한번 시도를 한 후에는 arp 캐시가 있으므로 캐시가 있는 상태에서 재시도를 하면 arp 요청과 reply 절차가 없이 바로 3 way handshake 가 이루어지는 것을 <그림10> 에서 확인해 볼 수 있다.

```
[root@www35 /root]# tcpdump host 211.47.65.106
tcpdump: listening on eth0
14:17:37.558524 arp who-has www35.tt.co.kr tell 211.47.65.106
14:17:37.558553 arp reply www35.tt.co.kr is-at 0:50:8b:9a:30:6
14:17:37.562238 211.47.65.106.1028 > www35.tt.co.kr.telnet: S 2152
7822[|tcp] > (DF) [tos 0x10]
14:17:37.562325 www35.tt.co.kr.telnet > 211.47.65.106.1028: S 2145
K.timestamp 150286178[|tcp] > (DF)
14:17:37.562482 211.47.65.106.1028 > www35.tt.co.kr.telnet: . ack
10]
14:17:37.562859 211.47.65.106.1028 > www35.tt.co.kr.telnet: P 1:25
) [tos 0x10]
14:17:37.562889 www35.tt.co.kr.telnet > 211.47.65.106.1028: . ack
14:17:37.634866 www35.tt.co.kr.telnet > 211.47.65.106.1028: P 1:13
F) [tos 0x10]
14:17:37.634998 211.47.65.106.1028 > www35.tt.co.kr.telnet: . ack
x10]
```

<그림9> arp cache가 없을때의 tcpdump

```
[root@www35 /root]# tcpdump host 211.47.65.106
tcpdump: listening on eth0
23:27:21.387971 211.47.65.106.1025 > www35.tt.co.kr.telnet: S
289(0) win 5840 <mss 1460,sackOK,timestamp 346718[|tcp] > (DF)
23:27:21.388017 www35.tt.co.kr.telnet > 211.47.65.106.1025: S
670(0) ack 2613508290 win 5792 <mss 1460,sackOK,timestamp 153
23:27:21.388146 211.47.65.106.1025 > www35.tt.co.kr.telnet: .
p,nop,timestamp 346718 153584560 > (DF) [tos 0x10]
23:27:21.388541 211.47.65.106.1025 > www35.tt.co.kr.telnet: P
5840 <nop,nop,timestamp 346718 153584560 > (DF) [tos 0x10]
23:27:21.388576 www35.tt.co.kr.telnet > 211.47.65.106.1025: .
op,nop,timestamp 153584560 346718 > (DF)
23:27:21.439056 www35.tt.co.kr.telnet > 211.47.65.106.1025: P
n 5792 <nop,nop,timestamp 153584565 346718 > (DF) [tos 0x10]
23:27:21.439178 211.47.65.106.1025 > www35.tt.co.kr.telnet: .
op,nop,timestamp 346723 153584565 > (DF) [tos 0x10]
23:27:21.439223 211.47.65.106.1025 > www35.tt.co.kr.telnet: P
5840 <nop,nop,timestamp 346723 153584565 > (DF) [tos 0x10]
```

<그림 10> arp cache가 있을때의 tcpdump

switch

switch 환경에서 A 서버와 C 서버가 telnet 과 같은 TCP 통신을 한다면 A 서버에서는 C 서버의 MAC 주소를 알기 위해 먼저 ARP Broadcasting 을 한다. 이때 이 요청은 Broadcasting 이므로 B, C 모두에게 전달된다. 이 중 B 서버는 ARP 목적지 IP가 자신이 아니므로 이 요청에 응답하지 않고, C 서버는 목적지가 자신의 IP 이므로 이 요청에 대하여 요청을 한 A 서버에 ARP reply를 한다. 여기까지는 스위치나 dummy 환경이 동일하다. 그러나 switch 환경에서는 arp broadcast 이외의 모든 패킷이 broadcast 하지 않으므로 B 서버에서는 C 서버의 unicast 응답을 캡처할 수 없다. A 와 C 간에 ARP broadcast와 reply를 한 후에는 TCP 통신을 하려고 하므로 본격적인 데이터 교환을 하기 전에 TCP 3 Way handshake 과정을 거치게 된다. A 와 C 간에 서로의 MAC 정보를 교환한 후에는 서로의 MAC 주소를 알기 때문에 3 way handshake 뿐만이 아니라 connection 이후의 모든 데이터 교환 역시 broadcast 하지 않고 A와 C 간에만 통신을 하게 되며 B 에서는 A와 C 간의 통신을 엿볼 수 없게 된다.

즉, dummy 환경에서는 (1) arp 요청과 응답

(2) tcp 3 way handshake

(3) 데이터 교환

이 모든 과정이 broadcasting 되어 B 에서도 A와 C 간의 통신을 엿볼 수 있으나 switch 환경에서는 (1) arp 요청만이 broadcasting 하며

(2) arp 응답

(3) tcp 3 way handshake

(4) 데이터 교환 모두 A 와 C 간에 unicast 로 작동하기 때문에

B 에서는 A 와 C 간의 통신을 엿볼 수 없게 된다.

여기에서 B처럼 자신을 출발지 또는 목적지로 하지 않은 A 와 C 간의 통신을 엿보는(?) 일련의 행위를 스니핑(Sniffing) 한다고 한다.

스니핑(Sniffing)

그렇다면 스니핑은 어떻게 가능한가?

크게 두 가지 이유로 가능하다. 첫 번째는 위에서 본 것처럼 dummy 환경에서는 모든 패킷을 broadcast 하여 직접적인 통신과는 관계없는 다른 PC나 서버에서도 다른 시스템의 패킷을 캡처할 수 있다는 특성을 이용한 것이고, 두 번째는 인터넷의 표준 프로토콜로 사용중인 TCP/IP 의 전송 방식이 암호화하지 않은 Plain text(평문)를 사용한다는 점을 이용한 것이다. 그렇다고 해서 Plain text 방식으로 broadcasting 한다고 바로 모든 패킷을 스니핑할 수 있는 것은 아니다. 왜냐하면 기본적으로 목적지 IP가 자신의 IP가 아닌 패킷은 이더넷에서 패킷을 드롭(drop)하는 특성이 있기 때문이다. 이처럼 스니핑을 하려면 자신의 목적지가 아닌 패킷도 받아들여야 하는데, 그렇게 하려면 인터페이스가 promiscuous (또는 promisc)모드로 작동하도록 설정하여야 한다. promisc 모드는 우리말로 “무차별 모드”라고도 하는데, promisc 모드로 설정되면 이더넷에서 패킷을 드롭하지 않고 모든 패킷을 다 받아들여지게 되어 스니핑이 가능하게 된다. 리눅스에서 인터페이스가 promisc 모드인지 여부를 알 수 있는 명령어는 앞에서 본 것처럼 ifconfig 라는 명령어로 확인 가능하다. 이때 아래와 같이 PROMISC 라는 문자열이 보이면 인터페이스는 promisc 모드로 되어 있는 것인데, promisc 모드를 수동으로 설정하려면

ifconfig eth0 promisc 와 같이 하면 되고, 설정을 해제하려면

ifconfig eth0 -promisc 로 실행하면 된다.

```
[root@test1 ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:8B:9A:1B:1B
          inet addr:211.47.65.106  Bcast:211.47.65.255  Mask:255.255.255
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:1718009  errors:0  dropped:0  overruns:0  frame:17384
          TX packets:1753  errors:0  dropped:0  overruns:0  carrier:0
          collisions:54  txqueuelen:100
          RX bytes:1185221290 (1130.3 Mb)  TX bytes:212471 (207.4 Kb)
          Interrupt:11  Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

<그림11> promisc 설정 여부 확인

특별한 이유가 없다면 정상시에는 promisc 모드로 설정되어 있을 필요가 거의 없으므로 만약 자신의 시스템이 promisc 모드로 설정되어 있다면 스니핑 여부를 주의깊게 검토해 보아야 할 것이다. 그러나 시스템에 따라 자동으로 promisc 로 설정하는 경우도 있으니 promisc로 설정되어 있다고 해서 반드시 스니핑이 작동하고 있는 것은 아니라는 것을 참고하기 바란다.

스니핑에 대한 대책

그렇다면 이에 대한 대책은 무엇인가? 3가지 정도의 방법을 대안으로 제시할 수 있겠다. 첫 번째는, Plain text로 전송되는 TCP/IP 대신 암호화 전송 프로토콜을 사용하는 것이다. 이의 대표적인 예가 telnet 대신 SSH, http 대신 https(SSL 웹서버)등을 사용하는 것이 그 예이다. 여기서 잠깐 SSL 에 대해 이야기하고자 한다. 적지 않은 사람들이 SSL 보안 웹서버를 설치하면 마치 방화벽처럼 모든 보안에 완벽해 지는 것으로 오해하는 경우가 있는데, SSL 은 80번 포트를 사용하는 http 프로토콜 역시 TCP/IP 에 속하므로 평문으로 전송되어 스니핑 될 수 있기 때문에 기존의 http 대신 암호화 전송 프로토콜인 SSL을 사용하는 것 뿐이지 다른 것은 아닌 것이다. 아마도 보안 웹서버라는 이름 때문에 다소 혼동하는 것이 아닌가 한다.

두 번째는 앞에서 본 것처럼 broadcasting을 하는 dummy 대신 스위치를 사용하는 것이다. 세 번째는 기업간 통신시 암호화된 통신을 하는 가상 사설망인 VPN(Virtual Private Network) 을 사용하는 것이다.

이 중 가장 근본적인 방법은 첫번째로 제시한 암호화 전송 프로토콜을 사용하는 것이다. 여기에서 잠깐 두 번째 방법으로 제시한 dummy 대신 스위치에 대해 알아보도록 하자. 앞에서 본 대로 스위치를 사용할 경우 초기 arp broadcast 요청시에만 broadcast 하고 이후에는 모두 unicast 로 통신하므로 상대적으로 스니핑에 안전하다는 것을 이해했을 것이다. 하지만 과연 스위치를 사용한다면 스니핑에 안전할까? 답은 “결코 그렇지 않다.”이다. 왜냐하면 쉽지는 않지만 스위칭 환경에서도 스니핑을 할 수 있는 방법이 있기 때문이다. 바로 arp의 특성을 이용하여 아래와 같은 몇 가지 방법이 가능하다.

스위치 환경에서도 가능한 스니핑 기법

(1) MAC Flooding

스위치에서는 각각의 포트에 해당하는 물리적인 MAC 주소를 기억하고 있는데 이 메모리에는 어느 정도의 한계치가 있다. 이러한 스위치의 특성을 이용하여 위조된 MAC을 지속적으로 발생시켜 스위치의 ARP 테이블을 Flood 시키는 방법으로 MAC 정보가 Flood 되면 스위치는 마치 dummy처럼 모든 포트에 broadcast 하게 된다. 보안적인 용어로 이러한 것을 “fail open“ 이라 한다.

(2) ARP Spoofing

ARP Spoofing은 스니핑하고자 하는 서버인 것처럼 MAC을 위조한 패킷을 Broadcast 하여 트래픽을 포워딩하는 것을 말하는데, 앞의 그림2와 같은 스위치 환경에서 B 서버가 A와 C 서버간의 통신을 스니핑 할 수 있는 방법으로 아래와 같이 ARP 의 원리만 알면 아주 쉽게 스니핑 할 수 있다. B 서버에서 A 서버에게 211.47.65.3 의 MAC 주소가 CC:CC가 아니라 B 서버의 MAC주소인 BB:BB 라는 패킷을 발송하고, C 서버에는 211.47.65.1 의 MAC 주소가 AA:AA 가 아니라 B 서버의 MAC주소인 BB:BB 라는 패킷을 발송하는 것이다. 그렇게 되면 A 서버와 C 서버는 이 정보를 의심없이 받아들여 정보를 갱신한 후 A와 C간 통신 시 모든 패킷은 B 서버로 보내지게 되며 B 서버는 모든 패킷을 받은 후 원래의 서버로 포워딩만 해 주면 되는 것이다. 이때 B 서버의 `net.ipv4.ip_forward=1` 로 설정되어 있어야 원래의 서버로 패킷을 포워딩하게 되며 A와 C 서버는 정상적인 데이터 전송이 되므로 패킷이 B를 거쳐간다는 사실을 알 수 없게 되는 것이다.

(3) ARP Redirect

ARP Redirect 역시 ARP Spoofing과 비슷한 방법인데, 자신이 마치 Gateway 인 것처럼 위조된 MAC 을 Broadcast하여 모든 트래픽이 자신을 통과하게 하는 방법이다. 즉, A와 C 서버에게 Gateway 주소인 211.47.65.254 의 MAC 주소가 라우터의 MAC 주소인 RR:RR 이 아니라 B 서버의 MAC 주소인 BB:BB 라는 패킷을 broadcast 하는 것이다. 그렇게 되면 로컬 랜상에 있는 서버들은 이 정보를 아무런 의심없이 받아들여 캐시 정보를 갱신한 후 게이트웨이로 보내는 모든 패킷을 211.47.65.254가 아니라 B서버인 211.47.65.2로 보내게 되며, B 서버는 모든 패킷을 받은 후 원래의 게이트웨이인 211.47.65.254 로 포워딩만 해주면 되는 것이다. 이때 B 서버의 `net.ipv4.ip_forward=1` 로 설정되어 있어야 원래의 게이트웨이로 패킷을 포워딩하게 되며 A와 C 서버는 정상적인 데이터 전송이 되므로 패킷이 B를 거쳐간다는 사실을 알 수 없게 되는 것이다.

(4) MAC Duplicating

만약 B 서버에서 A 서버의 트래픽을 스니핑하고 싶다면 B 서버에서 A 서버의 MAC주소와 같은 MAC 정보로 설정을 하는 방법이 바로 MAC Duplicating이다. `ifconfig`를 이용하면 이는 어렵지 않게 설정할 수 있는데,이렇게하면 `eth0` 의 MAC 주소가 00:50:8B:9A:1B:1B인 경우 이를 AA:AA:AA:AA:AA:AA 와 같이 설정하고자 한다면 먼저 `ifconfig eth0 down` 으로 `eth0` 인터페이스를 다운시킨 후 `ifconfig eth0 hw ether AA:AA:AA:AA:AA:AA up` 와 같이 실행하면 `eth0` 인터페이스의 MAC 정보를 변경할 수 있다. 이러한 경우 스위치를 혼란시켜 두 포트 모두 같은 MAC 주소를 가진 것처럼 인식하게 되며 스위치는 A 서버로 보낼 트래픽을 A와 B 모두에게 발송하게 되는 것이다. 데이터는 두 포트 모두에게 보내어지므로 B 서버에서는 IP 포워딩을 설정할 필요가 없다.

실제로 위와 같이 스위치 환경에서 스니핑하는 것은 그리 어렵지 않으며

`hunt`나 `dsniff`(<http://www.monkey.org/~dugsong/dsniff/>)등과 같은 관련 툴을 이용하면 어렵지 않게 구현할 수 있다.

스니핑 차단하기

이제 스위치 환경도 결코 스니핑에 안전하지 않다는 사실을 알았을 것이다. 그렇다면 스위치 또는 dummy 환경에서 어떻게 보안을 강화할 수 있을까? 아래의 몇 가지 방법을 대안으로 제시할 수 있다.

(1) IP Flitering

스위치에서 제공하는 IP Filtering 기능을 사용함으로써 각각의 포트에서 오가는 트래픽을 필터링 할 수 있다. 그러나 이 기능을 제공하는 스위치에서만 가능하며 매번 포트가 바뀔 때마다 수작업으로 설정해 주어야 하므로 많은 수고가 따르게 된다.

(2) Port security

만약 스위치에서 Port security 기능을 지원할 경우 MAC Flood 나 MAC Spoofing을 예방할 수 있는 방법으로 각각의 포트에 물리적인 MAC 주소를 정적(Static)으로 설정하는 것이다. 각종 ARP 기반의 스니핑에 가장 확실한 방법이라 할 수 있지만 과연 이렇게 하는 곳이 얼마나 될까는 의문이다.

원격지 서버의 스니핑 모니터링 프로그램

특정 서버에서 스니핑이 작동하고 있는지 리눅스의 경우 ifconfig를 실행시 PROMISC가 설정 되었는지 여부로 확인할 수 있다고 하였다. 그런데, 관리하는 서버가 여러 대라면 매번 로그인하여 ifconfig 로 PROMISC 설정 여부를 확인하겠는가? 거기다가 만약 ifconfig 실행 파일 자체가 변조되어 ifconfig 결과를 신뢰할 수 없다면? 이는 아무런 의미가 없게 되는 것이다. 이를 위해 원격에서도 특정 서버의 스니핑 작동여부를 체크할 수 있는 Sentinel 과 AntiSniff 라는 프로그램을 소개하고자 한다. Sentinel 을 설치하려면 미리 패킷 캡처 라이브러리인 Libnet 1.0과 libpcap 가 설치되어야 하는데, 각각

(<http://www.packetfactory.net/Projects/libnet>) 와 (<ftp://ftp.ee.lbl.gov/libpcap-0.4.tar.Z>)에서 다운로드 받아 압축 해제 후 압축이 풀린 디렉토리에서 ./configure; make ; make install 로 설치하면 된다. 그리고 Sentinel 은

<http://www.packetfactory.net/Projects/sentinel/> 에서 다운로드 할 수 있으며 다운로드후 압축 해제를 하고 압축이 풀린 디렉토리에서 make all 로 컴파일하여 설치하면 된다. 현재 Sentinel이 원격지 시스템에서 스니핑 여부를 감지하는 방법은 3가지가 있는데, 이 방법은 각각 DNS test, Etherping test, ARP test 등이다. 참고로 이 방법은 ARP를 이용한 방법이므로 같은 네트워크 세그먼트에 속해 있어야만 탐지의 의미가 있다는 점을 양지하기 바란다.

먼저 각각의 방법이 가능한 원리에 대해 간단히 살펴보면, 우선 DNS test 의 경우 목적지 서버에 위조된 연결 요청을 보내어, 일반적인 스니핑 프로그램이 요청한 시스템의 IP 주소를 역리졸브(Inverse DNS lookup) 한다는 특징을 이용하여 DNS 트래픽을 감시하여 스니핑 여부를 감지하는 방법이다. Etherping test는 목적지에 ping 패킷을 보낼 때 목적지의

IP 는 맞지만 목적지의 MAC 주소는 존재하지 않는 정보로 위조하여 Icmp Echo Packet 을 보내어 응답이 오는지 여부를 감시하는 방법으로 대부분의 정상적인 시스템에서는 존재하지 않는 MAC 정보이기 때문에 패킷에 대해 응답하지 않지만 promisc mode가 설정된 시스템에서는 이와 관계없이 응답을 한다는 특징을 이용하여 감시하는 방법이다. 마찬가지로 ARP test 역시 IP 는 목적지의 IP로 설정하지만 목적지의 MAC 주소를 다르게 하여 icmp 대신 ARP 요구를 보내는 방법으로 Promisc 모드가 아닌 경우에는 패킷이 목적지까지 갈 수 없으므로 목적지에서는 응답하지 않지만 Promisc 모드인 경우에는 모든 패킷을 받아들이므로 결국 응답한다는 특징을 이용하여 스니핑 여부를 감시하는 방법이다. 각각의 방식에 대한 실행 예는 아래와 같다.

```
./sentinel -a -t 211.47.65.4 # ARP 테스트
./sentinel -d -f 1.1.1.1 -t 211.47.65.4 # DNS 테스트
./sentinel -e -t 211.47.65.4 # Etherping 테스트
./sentinel -t 211.47.65.4 -f 1.1.1.1 -d -a -e # 3개의 테스트를 동시에 수행
```

위와 같이 실행시

Results: 211.47.65.4 tested positive to etherping test.

와 같이 탐지 결과가 positive 가 나오면 Promisc 모드로 설정되었다는 의미이므로 해당 인터페이스의 PROMISC 여부를 조사하여야 한다. 그런데, 한 시스템에 대해 각각의 테스트를 동시에 실행했을 때 결과가 각기 다르게 나오는 경우가 있는데, 이는 리눅스의 커널 버전에 따라 종종 발생하는 현상으로 세 가지 방법중에 어느 하나라도 positive 라는 결과가 나온다면 반드시 스니핑 작동 여부를 확인하기 바란다.

아울러 Antisniff 의 경우 <http://www.securitysoftwaretech.com/antisniff/> 에서 다운로드 가능하며 리눅스 버전과 윈도우 버전의 프로그램도 사용할 수 있는데, 테스트 하는 원리는 위의 Sentinel 과 비슷하며 추가적으로 Latency test 라는 재미있는 방법이 있는데, 이 방법은 스니핑이 작동시 promisc 모드로 설정되어 있을 경우에는 로컬 네트워크상의 모든 트래픽을 받아들인다라 시스템의 로드가 전반적으로 높아진다는 점을 이용해 불필요한 쓰레기 트래픽을 전송하여 시스템의 응답 시간이 길어지는지 여부를 조회하는 방법으로 아직까지는 100% 신뢰할 수는 없는 방법이며 계속적으로 개선중인 기능이다.

윈도우 버전의 Antisniff 의 경우 모니터링하고자 하는 IP 대역을 지정하여 한꺼번에 검사 가능하고, 검사 결과에 대해 각종 통계도 볼 수 있으며, 얼마의 주기로 테스트를 할 것 인지를 시간대별, 날짜별, 주별로 정할 수 있는 예약 기능 및 검사 결과가 변경시 메일로 발송하거나 음악이 나오게 하는 등의 알람 기능도 있어 편리하게 사용이 가능하다.

아울러 arp와 관련된 프로그램으로 작지만 강력한 프로그램인 ARPWatch 라는 프로그램을 소개하고자 한다. 앞에서 “tcpdump -e arp” 를 실행하였을때 실시간으로 IP 주소와 MAC 정보가 broadcast 되는 것을 확인하였을 것이다. 이 프로그램은 이러한 ARP 트래픽을 실시간으로 모니터링하여 MAC 주소와 IP 간의 매칭을 감시하는 프로그램으로서 만약 현재의 ARP 정보가 데이터베이스에 저장되어 있는 ARP 정보와 다르거나 새로운 MAC 주소가 추가/확인시에는 해당하는 내용을 지정된 관리자에게 메일로 통보하게 된다. arpwatch 프

로그인을 이용하면 MAC 주소나 ARP를 이용하는 공격에 대한 대응 및 네트워크 관리에 매우 유용하다.

암호화 전송 프로토콜 사용하기

앞에서 TCP/IP 가 얼마나 취약한지 그리고 이를 위한 대책이 어떤 것이 있는지 알아보았다. 물론 TCP/IP 의 취약점을 개선하기 위해 ipv6 등 새로운 개발이 가시화되고 있지만 현재의 상태에서 가장 확실한 방법은 암호화 전송 프로토콜을 사용하는 것에는 모두 동의할 것이다. 이는 telnet 대신 ssh를, http 대신 https를 사용한다고 말했었다. 그렇다면 ftp, smtp, pop3등 다른 프로토콜은 어떻게 할 것인가? 만약 telnet 만 사용한다면 그냥 ssh를 사용하기만 하면 되겠지만 한 서버에서 telnet 과 ftp, pop3등을 함께 사용한다면 어느 하나의 서비스만 암호화하는 것은 그리 큰 의미가 없다. 어차피 TCP/IP 자체가 취약한 것이므로 암호화 전송 프로토콜을 사용하려면 모든 서비스에 대해 암호화 전송 프로토콜을 사용해야 하기 때문이다. 그래서 이를 위해 기존 TCP/IP 기반의 서비스에 암호화 전송 프로토콜을 사용하려면 아래와 같은 두 가지 방법이 가능하다.

- (1) sslwrap를 이용한 암호화
- (2) SSH의 포트전송 기능을 이용한 암호화

(1) sslwrap를 이용한 암호화

sslwrap 는 pop3나 imap, smtp 등의 TCP 서비스를 감싸 TSL/SSL 을 이용하여 모든 TCP 데이터 전송을 암호화하는 간단한 유닉스 서비스로서 기존의 서비스를 다시 설치할 필요없이 어렵지 않게 사용이 가능하다. sslwrap를 이용하려면 openssl 이나 ssleay를 설치하여야 하는데, 여기에서는 openssl을 사용하는 방법을 알아보도록 하자.

먼저 <http://www.openssl.org/> 에 접속하여 최신의 openssl을 다운로드하여 설치할 서버에 업로드한다.

또는 설치할 서버에서 직접 “lynx <http://www.openssl.org/source/openssl-0.9.6b.tar.gz>” 나 “wget <http://www.openssl.org/source/openssl-0.9.6b.tar.gz>” 로 다운로드 받아도 된다. 다운로드 후 압축 해제하여 압축이 풀린 디렉토리에서 아래와 같이 실행하여 컴파일하여 설치를 한다.

```
[root@www ~/openssl-0.9.6b]# ./config ; make; make test; make install
```

이번에는 sslwrap을 설치할 차례이다.

역시 "lynx <http://www.rickk.com/sslwrap/sslwrap.tar.gz>" 나

"wget <http://www.rickk.com/sslwrap/sslwrap.tar.gz>" 으로 다운로드한 후 압축 해제하여 make 만 실행하여 컴파일하면 된다. 컴파일후 생성된 sslwrap 파일을 /usr/sbin/ 디렉토리에 복사한다. 혹 컴파일이 잘 안되는 경우에는 <http://rpmfind.net/> 에서 자신의 버전에 맞는 적당한 rpm 파일을 다운로드하여 설치해도 된다.

이번에는 인증서를 생성할 차례이다. 물론 베리사인이나 Thawte등 공인된 CA(Certificate

Authority) 기관에서 일정정도 비용을 지불하고, CSR 를 생성후 인증서를 구입하는 방법도 있지만 여기에서는 자기 자신이 사인한 인증서를 발급하는 방법에 대해 알아보도록 하자. 인증서를 발급하는 명령어는 아래와 같다.

```
# openssl req -new -x509 -nodes -out /usr/local/include/sslwrap.pem -keyout /usr/local/include/sslwrap.pem -days 365
```

이 명령어를 한줄에 이어서 입력하기 바란다. 아래는 위와 같이 실행시 보이는 화면이다.

```
Using configuration from /usr/share/ssl/openssl.cnf
Generating a 1024 bit RSA private key
.....++++++
..++++++
writing new private key to '/usr/local/include/sslwrap.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:KR
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:Seoul
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Today & Tomorrow
Organizational Unit Name (eg, section) []:Net-Center
Common Name (eg, your name or your server's hostname) []:test1.tt.co.kr
Email Address []:antihong@tt.co.kr
```

위에서 굵게 표시된 부분은 각자의 상황에 맞게 적절히 입력하여야 하는 내용이다. 각각에 대해 잠깐 알아보도록 하자.

```
Country Name (2 letter code) [AU]:KR
--> 국가 이름을 코드로 입력한다. 한국은 KR 이라 입력한다.
State or Province Name (full name) [Some-State]:
--> 주는 존재하지 않으므로 적당히 입력하거나 그냥 엔터를 입력한다.
Locality Name (eg, city) []:Seoul
--> 도시이름을 입력한다.
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Today & Tomorrow
--> 회사나 조직, 기관의 이름을 입력한다.
Organizational Unit Name (eg, section) []:Net-Center
```

--> 부서이름을 적절히 입력한다.

Common Name (eg, your name or your server's hostname) []:test1.tt.co.kr

--> 인증서가 설치되는 서버의 받는 메일서버 주소를 입력한다.

만약 여기에서처럼 pop3, smtp 에 대해 암호화 프로토콜을 사용하려면 이 주소는 아웃룩 익스프레스등의 메일 프로그램에서 보내는 메일서버 또는 받는 메일서버로 입력할 주소이다. Kildong, Hong등과 같이 자신의 이름을 입력하지 않도록 주의하기 바란다.

Email Address []:antihong@tt.co.kr

--> 자신의 e-mail 주소를 입력한다.

이제 인증서 생성을 끝났으므로 sslwrap를 inetd 나 xinetd 에 설정할 차례이다.

먼저 /etc/services 파일을 열어

```
pop3s          995/tcp
```

```
smtps          465/tcp
```

와 같이 설정되어 있는지 확인하고 없으면 위와 같이 추가한다.

그리고 xinetd 의 경우 xinetd.conf 설정 파일에 아래와 같이 추가한다.

```
service smpts
{
    disable          = no
    flags             = REUSE
    socket_type      = stream
    wait             = no
    user              = nobody
    server            = /usr/sbin/sslwrap
    server_args      = -cert /usr/local/include/sslwrap.pem -quiet -port 25
    log_on_failure   += USERID
}
```

```
service pop3s
{
    disable          = no
    flags             = REUSE
    socket_type      = stream
    wait             = no
    user              = nobody
    server            = /usr/sbin/sslwrap
    server_args      = -cert /usr/local/include/sslwrap.pem -quiet -port 110
    log_on_failure   += USERID
}
```

inetd 의 경우라면 pop3s 서비스에 대해 inetd.conf 파일에 아래와 같이 설정할 수 있다.

```
pop3s stream tcp nowait nobody /usr/sbin/tcpd /usr/sbin/sslwrap  
-cert /usr/local/include/sslwrap.pem -port 110
```

이와 같은 방식으로 다른 서비스도 아래와 같이 설정할 수 있다.

```
https stream tcp nowait nobody /usr/sbin/tcpd /usr/sbin/sslwrap  
-cert /usr/local/include/sslwrap.pem -port 80
```

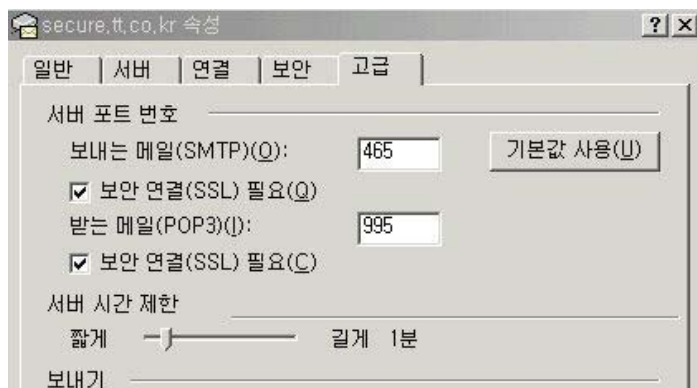
```
imaps stream tcp nowait nobody /usr/sbin/tcpd /usr/sbin/sslwrap  
-cert /usr/local/include/sslwrap.pem -port 143
```

```
telnets stream tcp nowait nobody /usr/sbin/tcpd /usr/sbin/sslwrap  
-cert /usr/local/include/sslwrap.pem -port 23
```

위와 같이 설정 후 inetd 또는 xinetd를 재시작하면 변경한 내용이 적용될 것이다.
설정된 포트가 열렸는지 아래와 같이 확인한다.

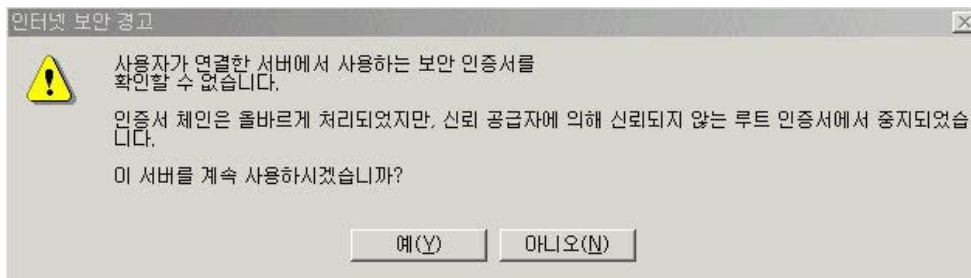
```
# telnet localhost pop3s  
# telnet localhost smtps
```

이제 서버에서의 설정은 끝났으므로 메일 클라이언트 프로그램에서 설정할 차례이다.
메일 프로그램에서 받는메일서버(pop3), 보내는메일서버(smtp)를 위에서 설정한 서버이름으
로 입력하고, 고급 탭에서 보내는 메일서버와 받는 메일서버에 아래 그림과 같이 “보안연결
(SSL)필요”를 선택하면 465와 995번 포트가 설정되는 것을 확인할 수 있다.



<그림> 아웃룩 익스프레스 설정

이제 드디어 모든 설정이 끝났다. 메일 송수신이 되는지 테스트해 보기 바란다. 그런데, 메
일 프로그램에서 보내고 받기를 클릭하면 아래와 같이 처음 프로그램을 시작할 때마다 경고
메시지가 뜨게 되어 약간 불편하지 않을 수 없다.



<그림> 공인된 인증서가 아닌 경우 경고 화면

이는 공인된 인증서가 아니라 자기 자신이 사인한 인증서를 발급하였기 때문이며 이러한 경우에는 다음과 같이 추가적인 작업을 해주면 이 메시지가 나타나지 않도록 할 수 있다.

먼저 인증서 파일인 /usr/local/include/sslwrap.pem 파일을 열면

```
-----BEGIN RSA PRIVATE KEY-----로 시작해서 -----END RSA PRIVATE KEY----- 로 끝나는 부분과, -----BEGIN CERTIFICATE----- 로 시작해서 -----END CERTIFICATE----- 로 끝나는 부분이 있는데, 이 중에서 "-----BEGIN CERTIFICATE ..... END CERTIFICATE-----" 부분만 복사해서 Windows 등의 클라이언트 PC에 server.crt 라는 파일로 복사를 한다. 그리고 이 파일에서 오른쪽 마우스를 클릭하면 “인증서 설치” 라는 메뉴가 보이는데, 여기에서 “인증서 설치”를 실행한다. 이후 나오는 화면에서 모두 “다음-->”다음“을 선택 후 마침을 선택하면 PC에서의 설치가 완료된다. 다음부터는 자체적으로 사인한 인증서를 사용하더라도 경고 화면이 나오지 않는 것을 확인할 수 있을 것이다. 그럼, 실제로 pop3s를 사용할 때 실제로 암호화가 되어 스니핑할 수 없는지 실제로 테스트해 보도록 하자. 각종 리눅스 스니퍼 프로그램을 이용해도 되지만 여기에서는 tcpdump를 이용해 테스트 해 보았다.
```

먼저 아래와 같이 서버에서 설정하여 연결을 캡처할 수 있도록 준비하고 pop3 로 메일을 받아보도록 하자.

```
# tcpdump -x host 211.1.1.1 > pop3.txt
```

이번에는 암호화 전송 프로토콜인 pop3s로 패킷을 캡처할 수 있도록 준비하고 pop3s 로 받아보도록 하자.

```
# tcpdump -x host 211.1.1.1 > pop3s.txt
```

이 명령어는 메일 클라이언트 프로그램을 실행하는 211.1.1.1 로부터의 패킷을 16진수로 캡처하여 각각 pop3.txt와 pop3s.txt 라는 파일로 저장을 한다는 의미이다. 이후 캡처한 데이터를 16진수로 표기하는 아스키 코드표로 해석을 하면 된다. 16진수로 표기하는 아스키 표는 <http://www.asciitable.com/>를 참고하기 바란다.

또는 아래처럼 16진수를 아스키로 변환해 주는 trans.pl 프로그램을 이용해서 변환할 수도 있다.

```
#!/usr/bin/perl
```

```

while(<>){
    s/ //g;
    s/([a-zA-Z0-9][a-zA-Z0-9])/pack("c",hex($1))/eg;
    print $_;
}

```

위와 같이 작성 후 perl trans.pl pop3.txt 또는 pop3s.txt를 하면 아래와 같이 덤프받은 데이터를 아스키로 변환할 수 있다. 물론 아래처럼 깨끗하게 변환은 되지 않지만 pop3.txt 파일의 내용을 읽을 수 있다는 것을 확인할 수 있을 것이다. 반면에 pop3s.txt 파일을 변환해보면 암호화가 되어 거의 어떤 문자인지 알아볼 수 없다는 것을 알 수 있을 것이다.

pop3를 스니핑한 예

```

+OK POP3 test1.tt.co.kr v2001.77 server ready
user user1
+OK User name accepted, password please
pass dkssud
+OK Mailbox open, 2 messages

```

#pop3s를 스니핑한 예

```

E[u@@??Aj
?@.?霏D57?
P a?u?8??뵤?
?ㄱ4?Ti夫
栗 EFv@@??Aj
?@.?霏w57?

```

sslwrap를 이용한 방법에 대해서는 <http://www.quiltaholic.com/rickk/sslwrap/>를 참고하기 바란다.

(2) SSH의 포트 포워딩 기능을 이용한 암호화

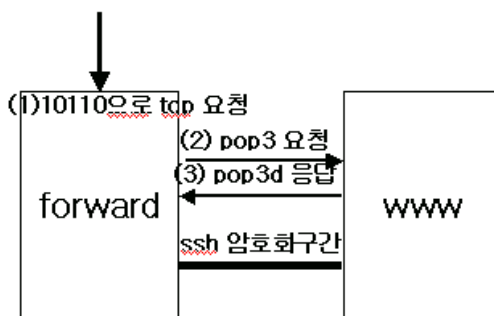
ssh 클라이언트와 서버 연결 구간에는 암호화 통신을 한다는 것을 알고 있다. 이 특성을 활용하여 ssh 에서 제공하는 포트 포워딩 기능을 이용하면 암호화가 제공되지 않는 프로토콜도 쉽게 암호화 전송할 수 있다. 쉽게 이야기하면 ssh 에서 기존의 서비스를 캡슐로 싸서 서비스한다고 생각해도 된다.

포트 포워딩 기능을 이용해 ssh 클라이언트와 ssh 서버 사이에 암호화를 위한 터널을 구성할 수 있는데, 이를 구성하기 위한 방법은 “로컬포트 포워딩“과 ”원격포트 포워딩“ 두 가지 방법이 있다. 두 방법 모두 거의 유사한 방법이므로 여기에서는 로컬포트 포워딩 방법에 대해 알아보도록 하자.

ssh 클라이언트와 ssh 서버간에는 암호화 전송이 되기 때문에 일단 tcp 연결을 할 기존의 서버외에 연결을 할 ssh 클라이언트로 사용할 추가적인 한 대의 서버가 더 필요하다.

아래와 같이 두 대의 서버가 있다고 하자.

forward 서버는 ssh 클라이언트 연결을 하여 클라이언트의 연결을 받아 ssh를 이용하여 TCP 연결을 포워딩 할 서버이고 www 서버가 포워딩을 받아 실제로 서비스를 제공할 서버이다.



<그림> ssh 포트 포워딩서버 구성도

위와 같은 환경에서 하여야 할 일은 먼저 암호화를 할 포트에 대해 ssh 터널을 구성하는 것이다. 이는 클라이언트(즉 forward) 서버에서 다음과 같은 형식으로 ssh 연결을 하면 된다.

```
ssh -L 로컬포트:원격호스트:원격포트:SSH서버호스트
```

위와 같은 환경에서는 원격 호스트와 SSH 서버 호스트가 같으므로 forward 서버에서 단지 `ssh -L 10110:www:110:www` 와 같이 접속을 하면 된다. 아래는 실제 접속 예이다.

```
[root@forward /root]# ssh -L 10110:www:110 www
root@www's password:xxxxxxx <--- 암호 입력
[root@www /root]#
```

이 상태에서 forward 와 www 서버에서 각각 netstat을 해 보면

forward 서버의 경우

```
0 *:10110          *:*              LISTEN
foirward:47883   www:22           ESTABLISHED
```

와 같이 10110번 포트가 리스닝하고 있으며 www 서버에 22번 ssh 포트에 접속해 있는 상황을 발견할 수 있을 것이다.

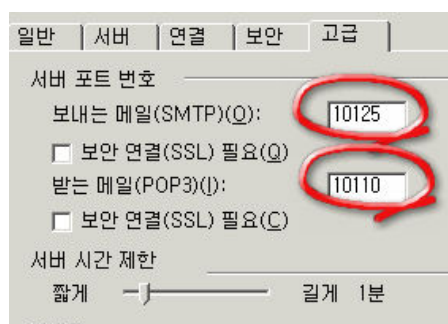
그리고 www 서버의 경우 아래와 같이 forward 서버의 ssh 연결이 성립되어 있는 것을 확인할 수 있다.

```
www:22          forward:47883    ESTABLISHED
```

이 상태에서 외부에서 www 서버에 pop3d 연결을 하려면 바로 www서버에 110번으로 연결하는 것이 아니라 forward 서버에 10110번으로 접속을 하면 된다. 아래와 같이 forward 서버에 10110번 포트에 접속을 하면 forward서버가 아니라 www 서버의 110번 포트가 반응하는 것을 확인할 수 있다.

```
# telnet forward 10110
Trying 211.1.1.2...
Connected to forward.
Escape character is '^ ]'.
+OK POP3 www v2001.75 server ready
```

실제로 아웃룩 익스프레스등 메일 프로그램에서도 받는 메일 서버(pop3)에 기존의 www 대신 forward 로 입력하고 “고급” 탭에서도 아래와 같이 기존의 110번 포트 대신에 10110을 입력한다. 그리고 이와 같은 ssh의 포트 포워딩 방식으로 smtp 나 telnet, ftp 서비스도 암호화 전송할 수 있는데, 만약 smtp 서비스에 대해 기존의 25번 대신 10125와 같이 포워딩 하였을 경우에는 아래와 같이 포트를 변경하여 설정하면 된다.



<그림> 포트 포워딩시 설정예.

이 상태에서 앞에서 했던 방법으로 tcpdump를 이용하여 스니핑을 해서 포트 포워딩된 패킷을 캡처해 보면 암호화 전송이 되고 있는 것을 확인할 수 있을 것이다.

마치면서

TCP/IP가 디자인 될 당시에는 네트워크는 그리 복잡하지 않았고 상호간에 신뢰할 수 있었기 때문에 그다지 보안을 고려하지 않아 TCP/IP는 그 자체적으로 매우 많은 보안적인 결함을 가지고 있다. 앞에서 밝힌 것 외에도 다른 많은 보안적 결함이 있으며 이를 이용한 많은 공격들이 인터넷상에 존재한다. 이 글을 계기로 늘 사용하고 있는 TCP/IP 프로토콜이 그 자체로 얼마나 보안적인 취약점을 안고 있는지 알 수 있는 계기가 되었기를 바라며 아울러 실제적인 방법이나 기법보다는 그 작동 원리에 대해 조금이나마 알게 되었기를 바란다. 이외 더 많은 정보에 대해서는 아래의 사이트가 도움이 될 것이다.

참고사이트 :

<http://packetstorm.decepticons.org/sniffers/>

http://www.linuxsecurity.com/resource_files/network_security/sniffing-faq.html

<http://www.robertgraham.com/pubs/network-intrusion-detection.html>

<http://www.certcc.or.kr/paper/tr2000/2000-07/tr2000-07.htm>

<http://www.ietf.org/rfc/rfc0826.txt?number=826>