Diggin em Walls - 방화벽 탐지, 그리고 방화벽 뒤의 네트워크 확인

by Ka0ticSH(asm.coder@verizon.net)
번역: vangelis (vangelis@wowhacker.org)
Wowcode Team of wowhacker

서문

이 글은 방화벽의 확인, 탐지, 그리고 침입 테크닉에 대해서 다룰 것이며, 더 특별히 방화벽 이면에 존재하는 네트워크를 확인하는 능력을 다룰 것이다. 앞으로 계속될 버전에서는 방화벽을 무력화시키거나 변조하지 않고 방화벽 이면에 있는 목표 네트워크에 접근하는 가능성에 대해 다룰 것이다. 그리고 포트 리다이렉션 테크닉에 대해서도 다룰 것이다. 이 글을 읽기 위해 네트워킹, 유닉스 운영 시스템에 대한 지식, 그리고 nmap(http://www.insecure.org/nmap) 유틸리티가 가까이에 있으면 좋겠다.

자살 임무:

방화벽 없이 인터넷에 서버가 연결하도록 하는 것이다. 방화벽을 설치하고 배치하는 것에 대한 책을 Cheswick와 Bellovin가 쓴 이후 방화벽은 ISP, 호스팅 업체, 그리고 심지어는 개인 사용자들의 지속적인 요구사항이 되었다. 그리고 비경험자가 방화벽을 설정하는 것은 파괴적이거나 부주의하거나 자살행위나 마찬가지다. 보통 네트워크 또는 시스템 관리자들이 그렇다. 관리자들은 네트워크에 대한 경험을 가지고 있을지 모르지만 보안 기술은 제한적일 수 있다. 이것은 잘못 설정된 방화벽으로 이어지고, 과도하게 자신감을 가진 관리자는 그의 네트워크가 안전하다고 생각해서 자신의 네트워크 패킷 트래픽 필터링에 중요한 헛점을 만들어낸다.

도입:

현재 보안 시장에서 두 가지 방화벽이 인기를 끌고 있다. 그것은 Application Proxies와 Packet Filtering Gateways이다. 대부분의 사람들의 의견은 Application Proxies가 Packet Filtering Gateways 보다 더 안전하다는 것에 있다. 물론 각각은 장단점을 가지고 있다. Application Proxies는 패킷이 나가는 것에 대해 엄격한 만큼 안으로 들어오는 패킷에 대해 더 엄격하다. Packet Filtering Gateways는 대규모 회사 네트워크에서 찾아볼 수 있는데, 고도의 수행능력을 가지고 있으며, 내부로 들어오는 트래픽이라는 필수사항을 가지고 있다.

방화벽들은 지난 몇 년 동안 많은 시스템이나 네트워크를 안전하게 해왔다. 즉, 악의적인 패킷이나 감시의 눈길 등으로부터 안전하게 해왔다. 하지만 단순한 방화벽은 안전과는 거리가 멀다. 이 방화벽들에서 취약점들은 자주 발생한다. 각 타입의 방화벽을 공격할 새로운 exploit나 그 방화벽을 우회할 테크닉이 발견되고 있다. 가장 큰 취약점은 방화벽의 잘못된 설정이다.

제대로 설정되거나 고안되었거나, 업데이트되고 유지된다면 방화벽은 좀처럼 침입하기가 힘들다. 대

부분의 해커들은 이 방화벽에 달려들어 스스로 피곤한 시간을 보내지는 않을 것이다. 차라리 방화벽 이면에 있는 안전한 시스템들의 신뢰관계를 공격하거나 방화벽에 의해 필터링 되지 않는 서비스를 통해 공격할 것이다.

존재하는 방화벽 및 이면 탐지:

각 방화벽은 그것 자체의 독특한 확인 스타일을 가지고 있다. 공격자는 단순한 스캐닝이나 배너 확인 등을 통해 이 확인 시퀀시를 이용할 수 있다. 이렇게 해서 버전, 타입, 그리고 심지어는 어떤 규칙까지도 확인할 수 있다. 이 확인방법은 중요하지 않게 보이지만 일단 방화벽의 전략 포인트를 확인하면 취약점이 수면에 떠오르게 된다.

가장 단순한 탐지/확인 테크닉은 열린 포트에 대한 간단한 스캐닝이다. 어떤 방화벽들은 디폴트로 열린 포트에 의해 확인될 수 있다. CheckPoint의 Firewall-1은 디폴트로 TCP 포트 256, 257, 258이 열려 있다. 반면 Microsoft의 Proxy Server는 디폴트로 TCP 포트 1745와 1080이 열려 있다. 다른 방화벽들도 그들의 디폴트 포트가 열려 있다. nmap에 몇 가지 인자를 사용해 실행시키면 어떤 방화벽을 확인하거나 방화벽의 존재여부를 확인할 수 있다.

[root@localhost]# nmap -n -vv -P0 -p256, 257, 258, 1080, 1745 10.10.1.8

어떤 숙달되어 있거나 또는 심지어 새로 소개된 공격자가 어떤 방화벽이나 또는 패킷 필터링 장애물에 대한 정보를 제공하기 위해 어떤 네트워크에 대한 이 스캐닝을 수행할 것이 확실하다. 어쨌던 침입 탐지 시스템(Intrusion Detection System, IDS)은 이 스캐닝을 포착하지는 않을 것이다. 아마도 당신은 당신의 IDS 설정을 스캐닝을 포착하도록 설정할 수는 있지만 IDS는 보통 철저하게 위협이 되는 (“hardcore threatening scan”) 스캐닝만을 기록한다. 또한 주의 배너(note banner)를 통한 방화벽 확인 기술도 디폴트 포트 만큼이나 많이 사용된다.

C:₩> nc -v -n 10.10.1.8 23
(UNKNOWN) [10.10.1.8] 23 (?) open
Eagle Secure Gateway.
Hostname:

이것이 방화벽임을 확인할 더 많은 증거를 수집하기 위해 TCP 25(Send Mail Transfer Protocol, SMTP)로 netcat을 사용하기도 하기도 한다.

C:₩> nc -v -n 10.10.1.8 25
(UNKNOWN) [10.10.1.8] 25 (?) open
421 10.10.1.8 Sorry, the firewall does not provide mail service to you.

노트 배너를 통해 방화벽이 존재하고 있고, 25번 포트가 필터링 되고 있음을 알 수 있다.

방화벽의 존재 여부를 확인하는 제 삼의 방법은 포트 확인 테크닉이다. 여러 방화벽들은 일련의 숫자를 돌려주는데, 이것을 통해 방화벽의 타입과 버전을 확인할 수 있다. 예를 들어, TCP 259(SNMP)에 netcat을 이용해 다음과 같이 연결하면 Checkpoint Firewall-1이 구현되어 있음을 알 수 있다.

```
[root@localhost]# nc -v -n 10.10.1.72 259
(UNKNOWN) [192.168.21.1] 259 (?) open
30000003
```

```
[root@localhost]# nc -v -n 10.10.1.95 259
(UNKNOWN) [192.168.29.212] 259 (?) open
31300003
```

또 다른 하나의 테크닉은 디버깅 툴을 사용하는 것이다. Traceroute는 아주 잘 작동한다. traceroute 또는 윈도우 시스템에서는 tracert.exe는 어떤 호스트쪽으로 활성화된 hop과 양을 탐지하는데 사용되는 네트워크 디버깅 유틸리티이다. 디폴트로 UDP 데이터그램 패킷이나 스위치 디시즌으로 ICMP ECHO 패킷을 보낸다. 이 패킷들은 TTL (Time to Live) 필더로 설정되어 있다. TTL은 1로 설정되어 있다. 패킷 TTL 필더는 호스트가 탐지될 때마다 1씩 증가하고, 0의 TTL 필더를 가진 가장 최근의 탐지되지 않은 호스트에 도달한다. 0의 TTL 을 가진 이 패킷이 라우터에 도달할 때 라우터는 디폴트로 ICMP error 메시지로 원래의 트레이스라우팅하는 호스트에 응답할 것이다. traceroute는 실행되고 있는 어떤 서비스나 어플리케이션에 의해 사용되지 않을 것 같은 높은 UDP 포트를 선택하고, 그래서 어떤 에러도 발생하지 않을 것이다. 그래서 traceroute는 방화벽 탐지를 위해 사용될 수 있다. 하지만 어느 정도의 삭감이나 리딩이 사용자의 마음 속에 가능해야 하지만 그것은 가능하다. 다음 보기는 방화벽을 찾아내기 위한 기본적인 traceroute 시도를 보여준다.

Traceroute:
 이 시나리오에서 네트워크는 패킷 필터링 장치에 의해 보호되고 있으며, 이 장치는 ping과 ping 응답(각각 ICMP 타입 8과 0임)을 제외한 트래픽을 찾아내고, 모든 초과 패킷을 막는다. 우리는 어떤 호스트들이 이 필터링(방화벽일 수도 있고, 라우터가 될 수 있다. 라우터의 경우 보안 정책에 반하는 것이다.) 뒤에 있는지 알아보기 위해 traceroute 프로그램을 사용할 수 있고, 사용하도록 시도할 것이다.

```
[root@localhost]# traceroute 10.10.1.10
traceroute to 10.10.1.10 [10.10.1.10], 15 hops max, 20 byte packets
1 10.10.1.2 [10.10.1.2] 0.022 ms 0.024 ms 0.025 ms
2 10.10.1.3 [10.10.1.3] 1.327 ms 2.360 ms 2.380 ms
3 10.10.1.4 [10.10.1.4] 4.217 ms 4.612 ms 4.656 ms
4 10.10.1.5 [10.10.1.5] 4.927 ms 5.090 ms 5.238 ms
5 10.10.1.6 [10.10.1.6] 5.529 ms 5.812 ms 6.003 ms
6 10.10.1.7 [10.10.1.7] 56.921 ms 59.162 ms 61.762 ms
7 10.10.1.8 [10.10.1.8] 63.832 ms 63.682 ms 61.235 ms
8 * * *
9 * * *
10 * * *
```

10.10.1.8에서 Hop 7은 방화벽으로 추정된다. 물론 맞지 않을 수도 있다. 앞에서 배웠듯이 방화벽은 라우터일 수도 있고, 또는 어떤 다른 패킷 필터링 또는 리다이렉션 어플리케이션일 수도 있다. 우리의 패킷은 방화벽을 탐지하기 위해 지나간다. traceroute의 리눅스 버전에서 -I 인자를 사용해보자.

```
[root@localhost]# traceroute -I 10.10.1.10
traceroute to 10.10.1.10 [10.10.1.10], 15 Hops Max, 20 byte packets
1 10.10.1.2 [10.10.1.2] 0.022 ms 0.024 ms 0.025 ms
2 10.10.1.3 [10.10.1.3] 1.327 ms 2.360 ms 2.380 ms
3 10.10.1.4 [10.10.1.4] 4.217 ms 4.612 ms 4.656 ms
4 10.10.1.5 [10.10.1.5] 4.927 ms 5.090 ms 5.238 ms
5 10.10.1.6 [10.10.1.6] 5.529 ms 5.812 ms 6.003 ms
6 10.10.1.7 [10.10.1.7] 56.921 ms 59.162 ms 61.762 ms
7 10.10.1.8 [10.10.1.8] 63.832 ms 63.682 ms 61.235 ms
8 10.10.1.9 [10.10.1.9] 62.926 ms 66.125 ms 67.032 ms
9 10.10.1.10 [10.10.1.10] 70.482 ms 71.273 ms 71.762 ms
10 10.10.1.11 [10.10.1.11] 73.192 ms 76.921 ms 82.325 ms
```

우리는 제대로 작동하지 않는 것처럼 보이는 디폴트 UDP 데이터그램 TTL 패킷을 사용하는 것 대신 ICMP 패킷을 사용하기 위해 -I 인자를 붙여 traceroute를 사용하기로 결정했다. traceroute 결과를 유심히 보면 우리가 호스트들과 방화벽 이면에 있는 시스템들을 탐지할 수 있다는 것을 알 수 있다.

하나의 일반적인 시나리오 설정은 DNS를 제외하고 모든 연결과 트래픽을 막는 방화벽이다. 이것은 UDP 포트 53을 열어둔다.

```
[root@localhost]# traceroute 10.10.1.10
traceroute to 10.10.1.10 [10.10.1.10], 15 hops max, 20 byte packets
1 10.10.1.2 [10.10.1.2] 0.022 ms 0.024 ms 0.025 ms
2 10.10.1.3 [10.10.1.3] 1.327 ms 2.360 ms 2.380 ms
3 10.10.1.4 [10.10.1.4] 4.217 ms 4.612 ms 4.656 ms
4 10.10.1.5 [10.10.1.5] 4.927 ms 5.090 ms 5.238 ms
5 10.10.1.6 [10.10.1.6] 5.529 ms 5.812 ms 6.003 ms
6 10.10.1.7 [10.10.1.7] 56.921 ms 59.162 ms 61.762 ms
7 10.10.1.8 [10.10.1.8] 63.832 ms 63.682 ms 61.235 ms
8 * * *
9 * * *
10 * * *
```

위의 결과를 보면 마지막으로 탐지된 hop은 10.10.1.1에 있다. 우리는 DNS(UDP 53)을 제외하고 모든 것이 막혀 있다고 추정할 수 있다. 우리는 이 결과를 이용할 수 있다. traceroute를 사용하는 것은 방화벽 이면에 있는 목표를 확인할 수 있도록 해준다. 우리는 몇 가지를 통제할 수 있는데, 사용되는 시작 traceroute 포트(매 probe에 의해 증가함), 보내진 probe의 숫자(디폴트로 3으로 지정되어 있음) 등을 통제할 수 있다. 우리는 또한 우리가 공격하는 호스트와 방화벽 사이에 얼마나 많은 hop이 있는지 다음 방법을 사용할 수 있다.

출발 포트의 번호로 스캐닝을 시작한다.

(target_port - (number_of_hops * num_of_probes)) - 1

우리는 방화벽으로 하여금 우리가 DNS 쿼리 패킷을 보낸다고 받아들이게 함으로써 ACL(Access Control List)를 우회할 수 있다. 위에 열거된 공식(실제 공식은 아니지만)을 이용해 traceroute 패킷을 설정하는데 도움이 될 수 있도록 할 수 있다. 또한 방화벽들은 패킷의 내용을 분석하지 않는다는 것을 주목하자. 그래서 패킷을 조작하여 방화벽을 무력화시킬 수 있다.

(53 - (8 * 3)) - 1 = 28

새로 조작된 패킷은 접근 가능한 포트 번호를 가지게 되고, 그래서 ACL 제한을 우회할 수 있다. 다음 예를 보자.

```
[root@localhost]# traceroute -p28 10.10.1.10
traceroute to 10.10.1.10 [10.10.1.10], 15 hops max, 20 byte packets
1 10.10.1.2 [10.10.1.2] 0.522 ms 0.624 ms 0.625 ms
2 10.10.1.3 [10.10.1.3] 5.327 ms 6.360 ms 6.380 ms
3 10.10.1.4 [10.10.1.4] 7.217 ms 7.612 ms 7.656 ms
4 10.10.1.5 [10.10.1.5] 8.927 ms 9.090 ms 9.238 ms
5 10.10.1.6 [10.10.1.6] 9.529 ms 10.812 ms 12.003 ms
6 10.10.1.7 [10.10.1.7] 56.921 ms 59.162 ms 61.762 ms
7 10.10.1.8 [10.10.1.8] 63.832 ms 63.682 ms 61.235 ms
8 10.10.1.9 [10.10.1.9] 65.158 ms * *
9 * * *
10 * * *
```

traceroute가 보내진 각 probe에 대한 포트 번호를 증가시킨다는 것과 스캐닝이 목표의 방화벽을 지난 후 종결된다는 것을 생각하자. 왜냐하면 첫번째 probe는 UDP 53(DNS)의 포트를 부여했기 대문에 보내진 다음 probe는 UDP 54를 가진다. 이 패킷 필터링 어플리케이션을 위해 수집된 ACL에 바탕을 두고 UDP 54는 봉쇄된다. 더 많은 정보를 구하고, 네트워크에 대해 좀더 확인하기 위해 우리는 ACL을 우회할 상태로 패킷을 유지해야만 한다. 그리고 우리가 UDP 53에서 포트를 유지하고자 하기 때문에 우리는 각 probe의 패킷 포트번호의 증가를 비활성화하기 위해 또 다른 인자를 붙일 필요가 있다. 보내진 모든 패킷이 ACL 요구에 맞도록 하고, 받아들여지기 위해, 그래서 우리가 목표 네트워크를 추가로 확인하기 위해 여기서는 traceroute1.4a5(http://www.packetfactory.com에서 구할 수 있음)가 사용되고 있다.

```
[root@localhost]# traceroute -S -p28 10.10.1.12
traceroute to 10.10.1.12 [10.10.1.12], 15 hops max, 20 byte packets
1 10.10.1.2 [10.10.1.2] 0.522 ms 0.624 ms 0.625 ms
2 10.10.1.3 [10.10.1.3] 5.327 ms 6.360 ms 6.380 ms
3 10.10.1.4 [10.10.1.4] 7.217 ms 7.612 ms 7.656 ms
4 10.10.1.5 [10.10.1.5] 8.927 ms 9.090 ms 9.238 ms
5 10.10.1.6 [10.10.1.6] 9.529 ms 10.812 ms 12.003 ms
```

6 10.10.1.7 [10.10.1.7] 56.921 ms 59.162 ms 61.762 ms
7 10.10.1.8 [10.10.1.8] 63.832 ms 63.682 ms 61.235 ms
8 10.10.1.9 [10.10.1.9] 62.926 ms 66.125 ms 67.032 ms
9 10.10.1.10 [10.10.1.10] 92.332 ms 93.214 ms 96.016 ms
10 10.10.1.11 [10.10.1.11] 101.523 ms 103.273 ms 103.923 ms
11 10.10.1.12 [10.10.1.12] 104.516 ms 105.523 ms 105.682 ms
12 10.10.1.13 [10.10.1.13] 109.231 ms 111.122 ms 117.923 ms

Firewalking:
우리가 여전히 traceroute와 traceroute 패킷에 대해 이야기 하는 동안에도, 나는 간단히 Firewalking을 다룰 것이다. Firewalk는 traceroute처럼 방화벽을 지나간 하나의 hop을 정확하게 소멸시키기 위해 계산된 IP TTL로 패킷을 만들므로서 이루어진다. 예상된 결과는 패킷이 방화벽에 의해 허용되고, 통과되고, 지시된 대로 소멸하는 것이 허용될 것이다. 그래서 "ICMP TTL expired in transit" 메시지가 나온다. 물론 만약 그 패킷이 방화벽의 규칙에 의해 봉쇄된다면 그것은 떨어지고, 그리고 우리는 아무런 응답을 받지 않거나 또는 ICMP 타입 13 Admin Prohibited 필터 패킷을 받게 될 것이다. 연속적으로 probe를 보내고, 어떤 것이 응답하고 어떤 것이 응답하지 않는지를 기록함으로써 게이트웨이상에 있는 접근 목록이 결정될 수 있다. 우리가 시작하기 전에 우리는 두 가지 정보를 가지고 있어야 한다.

1) 방화벽이 kick in하기 전에 탐지된 마지막 게이트웨이의 IP 주소
2) 방화벽 이면에 있는, 즉 방화벽으로 보호된 영역에 위치한 호스트의 IP 주소

IP 주소 1(게이트웨이)은 우리에게 웨이 포인트(way point)로 제공한다. 만약 우리가 이 게이트웨이 넘어로부터 응답을 받지 못한다면 우리의 패킷이 우리가 통과하려고 했던 프로토콜이 무엇이든 그것에 의해 봉쇄되고 있다고 추정된다. 이미 이해했을지도 모르지만 IP 주소 2(호스트)는 패킷의 목적지로서 사용될 것이다. 다음은 작동중인 firewalk의 예이다.

[root@localhost]# firewalk -n -P1-8 -pTCP 10.10.1.7 10.10.1.12
Firewalking through 10.0.0.5 (towards 10.0.0.20) with a maximum of 25 hops.
Ramping up hopcounts to binding host...
probe: 1 TTL: 1 port 33434: [10.10.1.1]
probe: 2 TTL: 2 port 33434: [10.10.1.2]

probe: 3 TTL: 3 port 33434: [10.10.1.3]
probe: 4 TTL: 4 port 33434: [10.10.1.4]
probe: 5 TTL: 5 port 33434: [10.10.1.5]
probe: 6 TTL: 6 port 33434: [10.10.1.6]
probe: 7 TTL: 7 port 33434: Bound scan: 7 hops [10.10.1.7]
port 135: open
port 136: open
port 137: open
port 138: open
port 139: *
port 140: open

우리가 볼 수 있듯이, 방화벽의 ACL 규칙은 firewalk의 사용에 의해 우회된다. 소스 코드를 포함해 더 많은 정보는 www.packetfactory.net/projects/에서 찾을 수 있다.

다시 nmap으로 돌아가자. 그 이유는 nmap이 일반적인 스캐너처럼 호스트를 스캐닝하고, 어떤 포트가 열려있는지 아니면 닫혀있는지를 말해주고, 추가로 어떤 포트가 봉쇄되고 있는지 말해주기 때문이다. 포트가 필터링되고 있다고 응답이 오는 것의 3가지 이유나 상태가 있다.

1) 어떤 SYN/ACK 패킷도 받아들여지지 않았기 때문에
2) 어떤 RST/ACK 패킷도 받아들여지지 않았기 때문에.
3) 시스템이 code 13(Communication Administratively Prohibited)으로 ICMP 타입 3(Destination Unreachable)으로 응답

Nmap은 이 모든 조건을 사용할 것이고, 그것은 필터링되고 있는 포트로 간주한다, 예를 들어,

[root@localhost]# nmap -p21,23,53,80 -P0 -vv 10.10.1.10
Starting nmap V. 2.07 by Fyodor (fyodor@dhp.com, www.insecure.com/nmap/)
Initiating TCP connect() scan against (10.10.1.10)
Adding TCP port 21 (state Open).
Adding TCP port 53 (state Firewalled).
Adding TCP port 23 (state Firewalled).
Adding TCP port 80 (state Open).
Interesting ports on (10.10.1.10):
Port State Protocol Service

21 open tcp ftp
23 filtered tcp telnet
53 filtered tcp domain
80 open tcp http

위에서 출력된 결과를 보면 몇 개의 포트가 '방화벽으로 막혀' 있다. 우리는 그것이 필터링되고 있는 것의 이유를 알아보기 위해 tcpdump 출력을 사용할 수 있다.

Raw Packet 전송 및 HPING:

Raw packet 전송 또한 방화벽 너머의 네트워크를 확인하는데 사용된다. Transmission 또한 방화벽 이면에 있는 네트워크를 확인하는데 사용된다. Hping은 목적지 포트에 TCP 패킷을 보내므로서 작동하고, 그런 다음 받은 패킷을 분석하고 보고한다. Hping은 받아들여지고, 봉쇄되고, 떨어지고, 또는 거부된 패킷을 찾아내도록 해준다. 그래서 ACL 규칙들을 확인하게 해주어 좀더 깊은 탐색을 하게 한다.

[root@localhost]# hping 10.10.1.7 -c2 -S -p21 -n
HPING 10.10.1.7 (eth0 10.10.1.1) : S set, 40 data bytes
60 bytes from 10.10.1.1: flags=SA seq=0 ttl=242 id=65121 win=64240
time=144.4 ms

위의 보기를 보면 기본적으로 SYN/ACK 패킷인 SA 플래그 셋이 붙은 패킷을 받은 사실을 통해 우리는 10.10.1.7의 TCP 포트 21이 열려 있는 것을 볼 수 있다. 이것은 어떤 한 포트가 열려 있다는 것을 알 수 있지만 방화벽이 존재하는지의 여부는 알 수 없다. 그래서 좀더 확인이 필요하다.

[root@localhost]# hping 10.10.1.10 -c2 -S -p80 -n
HPING 10.10.1.10 (eth0 10.10.1.1) : S set, 40 data bytes
ICMP Unreachable type 13 from 10.10.1.8

이 hping으로 우리는 ICMP 타입의 13개의 패킷을 받았으며, 그것은 ICMP 관리자가 금지한 필터링 패킷이다. 이 몇 가지의 명령을 통해 10.10.1.8이 방화벽이라는 것과, 10.10.1.10의 80번 포트가 봉쇄되어 있다는 것을 확인했다. 방화벽이 설치된 호스트로부터 우리가 받을 수 있는 또 다른 하나의 응답은 다음과 같다.

[root@localhost]# hping 10.10.1.16 -c2 -S -p23 -n
HPING 10.10.1.16 (10.10.1.1) : S set, 40 data bytes
60 bytes from 10.10.1.1: flags=RA seq=0 ttl=59 id=0 win=0 time=0.5 ms

이것은 가능한 2개의 항목 중에서 하나를 보여준다. 1번은 방화벽에 의해 수락되고, ACL 규칙에 맞는 패킷이다. 하지만 호스트는 그 포트에 듣지 않고 있다. 또는 2번은 방화벽이 거부한 패킷이다.

이전에 받은 ICMP 타입 13 패킷을 사용하여 우리는 방화벽이 우리가 보낸 패킷을 허용하고 있지만 호스트는 듣고 있지 않다는 것을 추정할 수 있다. 물론 뛰어난 관리자는 방화벽이 모든 패킷을 봉쇄하도록 설정할 때 다음과 같은 결과를 받게 된다.

[root@localhost]# hping 10.10.1.16 -c2 -S -p23 -n
HPING 10.10.1.16 (10.10.1.1) : S set, 40 data bytes

Source Port Scanning:

이 기술은 패킷 필터링 어플리케이션과 상태를 유지하지 않는 장치에 적용된다. Cisco의 IOS가 그 예이다. 만약 방화벽이 상태를 유지하지 못한다면, 그래서 연결이 내부쪽인지 외부쪽인지 확인할 수 없을 경우를 생각해보자. 이 경우에 전송은 필터링되지 않고 통과할 수도 있을 것이다. 방화벽을 통해 허용된 일반 포트(예를 들면 UDP 포트 53)에 출발지 포트를 설정해보자. 이것은 nmap에 -g 옵션을 주어 사용한다.

[root@localhost]# nmap -sS -P0 -g 53 -p 139 10.10.1.15

만약 열린 포트의 긍정적인 출력을 받게 되면 취약한 방화벽이라고 생각할 수 있다.

잘못 설정된 ICMP 패킷:
여기서는 간단히 언급하기로 하겠다. 자세한 것은 http://www.bliackhat.com을 찾아보아라. 그 문서에 따르면 확인된 호스트로부터 되돌아온 ICMP error 메시지를 도출하고, 그것의 존재를 찾아내는 다양한 방법들을 사용할 수 있다. 그 방법들은 다음과 같다.

- IP 헤더 분해

- 다른 헤더 길이 필더

- IP 옵션 필더

- IP 헤더에 유효하지 않은 필더 값 사용

- IP 헤더에 유효한 않은 필더 값 사용

- Fragmentation 오용

만약 우리가 모든 프로토콜과 포트의 조합으로 목표 네트워크의 전체 IP 범위를 확인했다면 그것은 우리에게 목표 네트워크의 토폴로지 맵을 가져다 줄 것이고, 접근 목록을 확인할 수 있도록 해줄 것이다.

결론:

어떤 네트워크를 탐지하고 확인하는 방법들은 많이 있다. 이 글에서는 단지 다른 테크닉을 발견하는 것으로 이어질 몇 가지 주요 사항만 언급했다. 이 문서는 자신의 네트워크가 안전하다고 믿고 있는 관리자들을 목표로 했다. 앞에서 언급한 테크닉들은 방화벽의 ACL을 포함해 네트워크의 토폴로지를 확인하는데 지속적으로 사용될 수 있을 것이다.
사실 제대로 설정된 것은 우회하기가 힘들다. traceroute, nmap, hping, 그리고 사용될 수 있는 다른 툴들은 공격자가 라우터, 방화벽을 통해 어떤 경로로 접근하는 방법을 찾도록 도와준다. 많은 취약점들이 방화벽의 ACL에서의 설정오류 때문과 로그, 트래픽, 그리고 유지 및 관리의 부족으로 인해서 발견된다.

--Ka0ticSH
asm.coder@verizon.net