방화벽 우회 테크닉

저자: Ka0ticSH(asm.coder@verizon.net)
번역 및 편집: vangelis (vangelis@wowwhacker.org)
Wowcode Team of Wowhacker

호스트 탐지:

방화벽 뒤에 어떤 호스트가 살아있다면 그 호스트를 탐지할 수 있는 방법은 여러 가지가 있다. 이 시리즈의 앞에서는 주로 traceroute와 ICMP 패킷을 사용하는 것에 대해 주로 다뤘다. 여기서는 호스트를 탐지하는 다른 방법들에 대해 다룰 것이다.

여기서 다룰 테크닉은 보내진 어떤 패킷에 설정된 플래그의 다른 조합들을 포함하고 있다. 플래그가 붙은 패킷은 보내지고, 우리는 어떤 패킷이 우리에게 돌아올지 관찰할 것이다. 이를 위해 TCP에 대한 기본적인 지식이 필요하다. 여기에 대해 대략 알아보자.

클라이언트는 목적지에 SYN 플래그비트가 설정된 패킷을 보낸다. 목적지 호스트는 RST 비트로 플래그가 붙은 패킷으로 응답하거나 또는 SYN 비트로 플래그가 붙은 보내진 패킷을 인정하는 SYN|ACK 패킷으로 응답한다. 호스트는 SYN 비트나 ACK 비트로 플래그가 붙은 보내진 패킷을 인정하기 위해 ACK 패킷을 다시 보낸다.

이전에 언급했던 것처럼 우리는 ICMP 패킷이나 보통 traceroute 패킷으로 확인하는 방법을 사용했왔다. 인터넷이 성장하고, 보안 문제가 중요해지자 방화벽/라우터의 설정은 보통 ping에 사용되는 것으로 알려진 ICMP의 Type 8 패킷을 막고 있다. 사실 ping은 사용가능한 가장 쉬운 호스트 탐지툴이다.

TCP 플래그가 붙은 패킷을 사용함으로써 우리는 네트워크의 상태를 확인하는데 약간 더 접근을 할 수 있다. 특히 전형적인 방화벽/라우터는 TCP 패킷을 막지는 않을 것이다. 그래서 이 패킷들에 대한 플래그를 조작함으로써 우리는 호스트가 시스템을 확인하기 위해 우리가 분석할 수 있는 방식으로 응답하도록 할 수 있다. 이 테크닉들은 호스트에 대해 포트 스캐닝을 할 수 있도록 해주며, 어떤 호스트가 온라인상인지 오프라인상인지 탐지하고, 그리고 광범위하게 그것의 운영체제를 탐지할 수 있도록 한다.

호스트가 살아있는지의 여부 탐지:

우리는 hping(http://www.kyuzz.org/antirez/hping2.html)을 사용할 것이다. 호스트 탐지는 단지 패킷을 보내고, 그것에 대한 응답을 기다리는 가장 기본적인 수준으로 시작할 것이다. 만약 아무런 응답이 없을 경우 호스트는 다운되어 있거나 패킷이 필터링되거나 드롭되었다고 생각한다.

TCP Ack 패킷:

ACK 비트 플래그가 붙은 패킷이 어떤 호스트에 보내지면 만약 그 호스트가 살아있다면 RST(reset) 비트 플래그가 붙은 다른 패킷으로 응답을 한다. 이 테크닉에 하나 더 추가한 것은 ACK 플래그가 붙은 패킷을 보낸 포트가 열려있는지 닫혀 있는지 차이가 없다는 것이다. 만약 그 호스트가 살아있다면 RST 플래그가 붙은 패킷으로 응답할 것이다.

```
[root@localhost]#hping 10.10.1.17 -A -c1 -p 2
hping 10.10.1.17 (eth0 10.10.1.17) : A set, 40 data bytes
60 bytes from 10.10.1.17: flags=R seq=0 ttl=59 id=0 win=0 time=0.3ms
```

RST 비트 플래그가 붙은 리턴된 패킷을 주목하자. 이것은 호스트가 살아있으며, 패킷이 TCP ACK 플래그가 붙은 패킷을 허용한다는 것을 나타낸다.

TCP Syn/Ack 패킷:

이 테크닉은 SYN 및 ACK 비트 플래그가 붙은 패킷을 보낸다. 이것은 나중에 살펴보겠지만 광범위한 운영체제 탐지 테크닉에도 도움을 준다. 하지만 지금은 호스트 탐지에 대해서만 이야기 하도록 하자. 호스트가 SYN 및 ACK 비트 플래그가 붙은 패킷을 받으면 RST 비트 플래그가 붙은 패킷을 리턴한다.

```
[root@localhost]#hping 10.10.1.16 -SA -c1 -p 2
hping 10.10.1.16 (eth0 10.10.1.16) : SA set, 40 data bytes
60 bytes from 10.10.1.16: flags=SA seq=0 ttl=59 id=0 win=0 time=0.5ms
```

포트 스캐닝:

열린 포트와 닫힌 포트를 탐지하기 위해 TCP 플래그 패킷 방법을 사용할 수 있다. 또한 이 테크닉을 포트 탐지를 위해 사용할 수도 있다. 이제 열린 포트 또는 닫힌 포트에 보내질 수 있는지 알아보기 위해 패킷이 분석될 수 있는 다양한 방법들에 대해 알아보자.

TCP Syn 패킷:

이 테크닉은 아주 효과적인 방법이다. 보통 SYN 비트 플래그가 붙은 패킷은 필터링되거나 드롭되지 않는다. 보통 열린 포트는 이 패킷을 받을 것이며, 예상된 TCP SYN/ACK 플래그가 붙은 패킷을 리턴한다. 닫힌 포트는 RST/ACK 비트 플래그가 붙은 패킷을 종종 리턴하기도 한다.

예]

```
[root@localhost]#hping 10.10.1.17 -S -c1 -p 21
hping 10.10.1.17 (10.10.1.17) : S set, 40 data bytes
60 bytes from 10.10.1.17: flags=SA seq=0 ttl=59 id=0 win=0 time=0.5ms
```

위의 예시는 ftp 포트 21번이 열려 있는지 확인하기 위해 호스트에 보내진 것이다. 21번이 열려 있는지 알 수 있다. 예에서도 볼 수 있듯이 SYN/ACK 비트 플래그가 붙은 패킷이 돌아왔다.

```
[root@localhost]#hping 10.10.1.17 -S -c1 -p 2
hping 10.10.1.17 (10.10.1.17) : S set, 40 data bytes
60 bytes from10.10.1.17: flags=S seq=0 ttl=59 id=0 win=0 time=0.3ms
```

위의 경우는 닫힌 포트 2번으로 SYN 비트 플래그가 붙은 패킷이 보내진 것으로 보인다.

TCP Fin 패킷:

이 테크닉은 포트의 열린 여부와 호스트의 온라인 및 오프라인 여부를 탐지하기 위해 사용된다. 이 테크닉은 SYN 비트 플래그가 붙은 패킷 만큼 효율적이지는 않다. 왜냐하면 열린 포트는 FIN 비트 플래그가 붙은 패킷을 드롭시킬 것이기 때문이다. 닫힌 포트는 RST/ACK 플래그 패킷으로 응답할 것이다. 다음은 그 예이다.

```
[root@localhost]#hping 10.10.1.17 -F -c1 -p 2
hping 10.10.1.17 (10.10.1.17) : F set, 40 data bytes
60 bytes from 10.10.1.17: flags=RA seq=0 ttl=59 id=0 win=0 time=0.5ms
```

위의 예에서 본것처럼 FIN 비트 플래그가 붙은 패킷은 닫힌 것으로 추정되는 2번 포트로 보내졌다. 응답은 RST/ACK 비트 플래그가 붙은 또 다른 패킷이었다. 이것이 포트 스캐닝의 또 다른 방법이다. 그런데 방화벽이 필터링을 하고 있을 경우, 그래서 닫힌 포트에 보내진 패킷이 방화벽에 의해 드롭되면 그 포트가 열려있다고 추정할 수도 있는 문제도 있다.

TCP Null 패킷:

TCP 플래그를 사용하는 또 다른 테크닉은 어떤 것도 사용하지 않는 것을 포함하고 있다. 이 패킷은 닫힌 포트로 보내져 RST/ACK 플래그 패킷이란 결과를 초래할 것이다. 전과 마찬가지로 열린 포트에 보내진 Null 패킷은 아무런 응답이 없거나 또는 그 패킷이 폐기될 것이다.

```
[root@localhost]#hping 10.10.1.17 -c1 -p 2
hping 10.10.1.17 (10.10.1.17) : NO FLAGS are set, 40 data bytes
60 bytes from 10.10.1.17: flags=RA seq=0 ttl=59 id=0 win=0 time=0.5ms
```

받은 응답은 RST/ACK 비트 플래그가 붙은 패킷이며, 이것은 21번 포트가 닫혀있다는 것을 보여준다. TCP FIN 패킷 테크닉과 아주 유사하지만 유연성은 없다. 어떤 방화벽은 null 패킷을 드롭하도록 설정될 수도 있다.

TCP Xmas 패킷:

또한 위의 두 가지 테크닉과 마찬가지로 포트의 열린 여부를 확인한다. 만약 포트가 닫혀있다면 RST/ACK 플래그 패킷으로 응답 받을 것이다. 그리고 만약 포트가 열려있다면 호스트는 그것에 반응하지 않을 것이다.

```
[root@localhost]#hping 10.10.1.17 -c1 -F -S -R -P -A -U -X -Y -p 2
hping 10.10.1.17 (10.10.1.17) : RSAFPUXY set, 40 data bytes
60 bytes from 10.10.1.17: flags=RA seq=0 ttl=59 id=0 win=0 time=0.5ms
```

위의 보기에서 호스트는 포트가 닫혀있는 것을 입증하면서 RST/ACK 플래그 패킷으로 응답하고 있다. Null 패킷과 마찬가지로 보통 Xmas 패킷이 필터링되기 때문에 더 이상 호스트를 확인하는 것이 허용되지 않는다.

이 테크닉들은 만약 허용된다면 호스트를 탐지하고, 그 호스트가 살아있는지의 여부를 알아보기 위해 사용될 수 있다. 닫힌 포트를 찾아낼 개연성은 열린 포트를 찾아내는 것보다 훨씬 더 쉬울 것이다. 이것들과 같은 테크닉은 패킷이 일상적인 트래픽처럼 보이도록 만들어 방화벽을 우회할 것이다.

ICMP 테크닉:

가끔 네트워크 문제를 해결하거나 기본적인 정보를 수집하기 위해 사용되는 ICMP 패킷은 네트워킹의 일반적이면서도 중요한 부분이다. 여기서는 ICMP에 대해 간단히 알아보기로 한다. Ping은 가장 일반적인 ICMP 타입이다. 만약 방화벽이 ping을 막지않는다면 대부분 ICMP Type 8, ECHO REQUEST, 및 ICMP Type 0, ECHO REPLY이다. 방화벽이 막지않는 다른 ICMP 패킷을 이용해 호스트가 살아있는지 탐지할 수 있다.

ICMP Timestamp Request, Type 13 :

```
[root@localhost]#icmpush -vv -tstamp 10.10.1.17
-> Outgoing interface = 10.10.1.1
-> ICMP total size = 20 bytes
-> Outgoing interface = 10.10.1.1
-> MTU = 1500 bytes
-> Total packet size (ICMP + IP) = 40 bytes

ICMP Timestamp Request packet sent to 10.10.1.17 (10.10.1.17)

Receiving ICMP replies...
10.10.1.17 -> Timestamp reply transmitted at 06:17:42
icmpush: Program finished OK
```

위의 결과를 보면 ICMP 패킷은 방화벽을 통해 허용되어 있다. 관리자는 탐지가 안되었을거라고 생각할지 모르지만 탐지가 되었다. Timestamp 테크닉은 보통 *nix 시스템에 대해 사용된다. 좀더 많은 정보는 다음 섹션을 참고해라.

이 정도만 하자. 보통 막힌 ICMP 패킷들은 ICMP type 8 ECHO REQUEST와 ICMP type 0 ECHO REPLY이며, 둘 다 ping을 위해 사용된다.

광범위한 OS 추측:

이 테크닉들의 일부는 어떤 플랫폼에서는 작동하지 않을 수도 있다. 이것은 운영체제를 추측하는데 최상의 기술은 아니다. 정확하지도 그리고 항상 성공하지도 않는다. 이 시도들 및 방법들은 시행착오에 기반을 둔 것이며, 어떤 패킷에 운영체계가 반응하는 것에 기반을 둔 것이다.

Windows:

아마 여태까지 언급된 호스트 탐지법 대부분이 윈도우상에서는 작동하지 않는다는 것을 알게 되었을지도 모르겠다. 단지 TCP 플래그가 붙은 패킷 테크닉들 중의 하나가 윈도우 시스템에 작용하지 않으며, 이것은 Xmas 테크닉이다. 또한 윈도우 시스템에 작동하지 않는 다른 하나는 브로드캐스트 테크닉이다.

여기서는 간단하게 다루겠는데, 만약 어떤 방화벽이 ICMP Type 8 패킷을 통과하도록 허용하고, 그 네트워크에 ping을 보내거나 주소를 브로드캐스트한다면 ICMP Type 8 패킷을 그 네트워크상에 있는 모든 호스트에 보내는 결과가 나오며, 그것들로 하여금 당신에게 ICMP Type 0(ECHO REPLY)을 보내도록 한다. 이것은 그 네트워크에 대한 정보를 알아낼 수 있다. 그러나 이것은 어떤 서비스 팩이 설치된 NT4가 아니라면 윈도우 시스템에는 작용하지 않는다.

앞에서 언급된 것처럼 timestamp 요청 테크닉은 윈도우 시스템에서는 작동하지 않으며, *nix 시스템에 대해서는 아주 효과적인 적으로 입증되었다. 여태 언급하지 않은 것 중 윈도우 시스템에서 작동하지 않는 다른 테크닉은 ICMP Type 15(Information Request packet)이다. 어떤 호스트로부터 요청하기 위해 사용되는 것은 네트워크 주소이다. 어떤 정보에 의하면 ICMP Type 15와 ICMP Type 16은 이미 구식이 되어버렸다.

Linux:

앞에서 언급된 테크닉들의 대부분과 반대로 ICMP Type17은 리눅스 시스템에 대해서는 작동하지 않았다. 하지만 윈도우 시스템에서는 작동했다. ICMP Type 17인 Address Mask Request는 목표 네트워크상의 subnet mask에 요청을 하고, 정보를 획득하는데 사용된다. 응답(ICMP Type 18, Address Mask Reply)는 subnet mask를 포함한다.

```
[root@localhost]#icmpush -vv -mask 10.10.1.17
-> Outgoing interface = 10.10.1.1
```

-> ICMP total size = 12 bytes

-> Outgoing interface = 10.10.1.1

-> MTU = 1500 bytes

-> Total packet size (ICMP + IP) = 32 bytes

ICMP Address Mask Request packet sent to 10.10.1.17 (10.10.1.17)

Receiving ICMP Replies...

icmpush: program finished OK

위는 리눅스 시스템에 보내졌지만 아무런 응답을 받지 못한 ICMP Type 17(Address Mask Request)의 예이다.

BSD:

BSD 네트워크 코드는 Syn/Ack 패킷에 응답하지 않는 것으로 최적화되어 있다. 아키텍처가 그 이유가 될 수 있지만 또한 운영체계를 탐지하고 추측하는데 도움이 될 수 있다. 리눅스를 포함해 다른 운영체계들과 Windows도 이 테크닉에 성공적으로 반응한다. 아래는 그 예이다.

[root@localhost]#hping 10.10.1.17 -c1 -p 23 -S -A

hping 10.10.1.17 (eth0 10.10.1.17) : SA set, 40 data bytes

위의 예에서는 Syn/Ack 비트로 플래그가 붙은 TCP 패킷을 보내기 위해 hping을 사용하고 있다. 이 패킷은 BSD 시스템에 보내졌으며, 그래서 어떤 응답도 받지 못했다.

Routers:

ICMP 테크닉을 사용하여 우리는 라우터도 역시 탐지할 수 있다. 라우터를 탐지하는 것은 아주 간단하며, 유용하다. 우리가 라우터라고 추정하는 시스템에 보내진 ICMP Type 10(Router Solicitation)을 이용하여 ICMP Type 9(Router Advertisement) 패킷이 응답을 받는지 알아보기 위해 점검할 수 있다.

[root@localhost]#icmpush -vv -rts 10.10.1.16

-> Outgoing interface = 10.10.1.1

-> ICMP total size = 20 bytes

-> Outgoing interface = 10.10.1.1

-> MTU = 1500 bytes

-> Total packet size (ICMP + IP) = 40 bytes

ICMP Router Solicitation packet sent to 10.10.1.16 (10.10.1.16)

Recieving ICMP replies...
10.10.1.16 -> Router Advertisement (10.10.1.16)
icmpush: Program finished OK

이제 우리는 네트워크상에 있는 라우터를 발견했으며, 그것은 나머지 네트워크에 하나의 필터링 패킷일 수 있다. 이 테크닉은 아주 유용하며, 어떤 시스템인지 알아낼 수 없을 때 간단한 해결책이 될 수 있다.

방화벽 침투:

자, 이제 우리는 방화벽을 찾아냈으며, ACL에 대해 파악했다. 그리고 커뮤니케이션을 나누고자 하는 시스템도 찾아냈다. 어떻게 할 것인가? 그 시스템과 커뮤니케이션을 하는 동안 계속 그 방화벽을 우회해라. 몇 가지 툴을 사용하면 그렇게 어려운 작업이 아닐 것이다.

프로토콜 터널링:

프로토콜 터널링은 하나의 프로토콜의 명령어들을 다른 프로토콜에 혼합시키는 행위이다. 우리는 이 테크닉을 구현한 몇몇 트로이를 보았다. 그리고 관리자들은 보안을 위해 그들이 사용하는 프로토콜(대부분 TCP와 UDP)을 필터링하기 시작했다. 여전히 대부분의 관리자들은 터널링을 위해 어떤 프로토콜들이 사용되고 있는지, 그리고 그들의 방화벽이 얼마나 형편없이 구멍이 뚫려 있는지 알아야 할 것이다.

ICMP/UDP Tunneling:

앞에서도 말했듯이, 대부분의 관리자들은 ICMP Type 8와 ICMP Type 0 패킷을 봉쇄할 것이다. 하지만 여전히 어떤 관리자들은 ICMP 헤더에 있는 ICMP Tunneling wrap 명령어나 데이터를 봉쇄하지 않고 있다. Loki/Lokid와 같은 프로그램은 가장 단순한 형태로 이 테크닉을 실행한다. 다음 인용문을 보자.

" Loki Project의 개념은 단순하다. 그것은 ICMP_ECHO 및 ICMP_ECHOREPLY 패킷의 데이터 부분에서 임의의 정보를 터널링하는 것이다. Loki는 ICMP_ECHO 트래픽 내에 존재하는 은밀한 채널을 공격한다. 이 채널은 네트워크 장치가 ICMP_ECHO 트래픽의 내용을 필터링하지 않기 때문에 존재한다. 네트워크 장치들은 ICMP_ECHO 트래픽의 내용을 단순히 통과하거나 드롭시키거나, 또는 리턴시킨다. 트로이 패킷 그 자체는 일반적인 ICMP_ECHO 트래픽으로서 머스커레이드된다. 우리는 우리가 원하는 어떤 정보를 캡슐화(터널링)시킬 수 있다. 여기서부터 계속 Loki는 정보를 터널링하는 ICMP_ECHO 트래픽을 가리킬 것이다. "

이것을 읽어보면 loki/lokid가 기본적인 트래픽으로 보여진다는 것을 입증할 것이다. 또한 loki는

'방화벽 침투 툴'로써 목적을 두기보다는 훨씬 더 은밀한 방법으로 데이터를 전송하기 위한 것이다. 방화벽 침투를 위한 것이라면 loki의 서버인 lokid는 장악한 시스템이나 방화벽 뒤에 존재하는 시스템에 설치되어야 한다.

```
[root@localhost]#lokid -p -i -v 1
```

그런 다음 전체 터널링 구조와 함께 클라이언트에 연결한다.

```
[root@localhost]#loki -d 10.10.1.17 -p -i -v 1 -t 3
```

ICMP 터널링은 아주 효과적인 것으로 입증되었는데, 그것은 일반적인 방화벽이 ICMP 패킷을 통과시키기 때문이다. 그래서 loki/lokid를 가장 강력한 백도어로 만들어준다.

TCP Ack 터널링:

TCP/IP 핸드쉐이크가 작동하는 방법을 살펴보면, Syn 패킷은 보내지고, 그런 다음 Syn/Ack 패킷은 받게 되고, 그런 다음 Ack 패킷이 다시 보내진다. 일반적인 관리자는 어떤 Ack 패킷은 핸드쉐이크나 연결 구도의 일부라고 추정할 것이다. ICMP/UDP 터널링처럼 트래픽이 정상적인 것으로 관리자들은 믿을 것이다.

이것을 위한 이상적인 프로그램은 AckCmd이다. Loki/lokid처럼 AckCmd는 클라이언트와 서버 부분으로 구성되어 있다. 서버는 호스트에서 실행되고, 클라이언트는 그 서버와 통신하기 위해 사용된다. 클라이언트는 서버와 통신하기 위해 Ack 패킷만 사용한다. 클라이언트가 서버의 IP를 알고 있다면 패킷은 일반적인 트래픽처럼 방화벽을 통과한다.

```
C:₩>ackcmds.exe
```

이 명령어는 AckCmd의 서버를 실행시킨다. 이것은 방화벽 뒤의 호스트에서 실행된다.

```
C:₩>ackcmdc.exe 10.10.1.17
```

이 명령어는 AckCmd의 클라이언트를 구동시킨다. 그 IP에 따라 서버도 실행된다. 그런 다음 "AckCmd>" 프롬프트가 나타날 것이다.

Outbound 트래픽:

"최근 개인 방화벽의 외부로 나가는 트래픽을 막는 것에서의 취약점이 개인 방화벽 제조업체들에

의해 제공되는 보안 차원에 대한 진지한 관심을 일으키고 있다. 보안 전문가들은 유명한 방화벽에서 외부로 나가는 트래픽을 막는 것을 구현하는데 있어서의 결함을 지적하고 있다. 필수적으로 이 취약점은 악의적인 프로그램이 “신뢰받은 어플리케이션”에 코드를 삽입시킴으로서 외부로 트래픽이 나가는 것을 막는 것을 우회하도록 하며, TCP/IP 패킷을 보낼 수 있는 프로토콜 드라이버를 만들어 Microsoft TCP/IP 스택을 우회하도록 해준다. 다음 세대에 아마 이용 가능한 트로이는 이 취약점을 공격하는 것이 될 것이다. 사실, 보안 연구자들은 모든 트래픽을 막도록 설정되어 있어도 개인 방화벽을 통과할 수 있는 도구들을 이미 만들었다.”

위 글은 http://www.sygate.com/alerts/Outbound-Blocking.htm에서 발췌했으며, 개인 방화벽이 외부로 나가는 트래픽을 막을 수 있는지 테스트 해본 결과 많은 것들이 실패했다. 이것에 대한 글은 다음을 참고하자.

http://keir.net/firehole.html
http://www.hackbusters.net/ob.html
http://online.securityfocus.com/archive/1/244026
http://grc.com/lt/leaktest.htm

결론:

많은 방화벽들은 악의적인 패킷이 통과하는 것을 허용하지 않는 아주 뛰어난 기반을 가지고 있지만 새로운 테크닉이 발표되면서 그 패킷을 통과하도록 해주는 것이 가능해졌다. 정상적인 패킷처럼 보이지만 조작된 패킷일 수 있다. 아주 철저하게 분류된 데이터만 전송되도록 하여 악의적인 패킷이 통과되는 것을 막아야 한다.

[참고문헌]

AckCmd - http://ntsecurity.nu/toolbox/ackcmd/

Project Loki/Lokid - http://phrack.org/show.php?p=49&a=6

Advanced Host Detection - http://packetstormsecurity.nl/papers/protocols/host-detection.pdf

Firewall/VPN/Intrusion Detection Systems - http://www.is-it-true.org/fw/

Firehole - http://keir.net/firehole.html

Leaktest - http://grc.com/lt/leaktest.htm

Sygate Security Alerts - http://www.sygate.com/alerts/Outbound-Blocking.htm

Outbound Traffic Flaws - http://www.securityfocus.com/archive/1/244026

HackBuster's OutBound Flaw Program - http://www.hackbusters.net/ob.html

Hping - http://packetstormsecurity.org/UNIX/scanners/hping2.0.0-rc1.tar.gz

Loki/Lokid - http://packetstormsecurity.org/crypt/LIBS/loki/loki-3.0.tar.gz