DDoS 공격패킷 탐지를 위한 Pi 분석방법

이현주° 이형준 홍만표

아주대학교 정보통신전문대학원

{cutehj, prime, mphong }@ajou.ac.kr

Pi analysis mechanism for detecting DDoS attack packet

Hyunju Lee° Hyungjoon Lee Manpyo Hong

Graduated School of Information and Communication AJOU University

요 약

분산 서비스 거부 공격(Distributed Denial of Service, DDoS attack)의 목적은 정상적인 사용자가 서비스를 이용하는 것을 거부하는 것이다. 특히 범람 서비스 거부 공격(flooding-based DDoS attack)은 아직까지 방어하기 어려운 공격 형태이다. 그 이유는 공격 시 정상적인 트래픽이 왕복하는 것처럼 보이므로 공격 트래픽과 정상 트래픽을 구별하기 어렵기 때문이다. 하지만, 범람 서비스 거부 공격의 특징을 잘 이용하면 공격을 방어할 수 있다. 범람 서비스 거부 공격의 특징은 단독의 공격 호스트로부터의 트래픽 양이 여러 개의 정상 호스트들로부터의 트래픽 양보다 훨씬 많다는 것이다. 이러한 특징을 이용하면 특정 호스트로부터의 트래픽 양이 많은지 아닌지에 따라 공격이 발생했는지 여부를 결정할 수 있다. 이 논문에서는 Pi(Path Identification) 라는 메커니즘을 이용한 분산 서비스 거부 공격의 방어 기법을 제안한다.

1. 서 론

최근 인프라 공격(infrastructure attack)이 빠르게 증가하고 있다. 인프라 공격은 인터넷의 인프라를 소모시켜 정상 사용자들이 이용할 수 없도록 만드는 공격 형태이다. 그러한 공격의 예로 분산 서비스 거부 공격, 슬래머 웜(slammer worm), DNS cache poisoning 등이 있다 [1]. 특히 분산 서비스 거부 공격은 웹서버, 라우터, DNS 서버 같은 중요한 서비스를 모두 소모해서 정상 사용자가 이용할 수 없도록 만드는 공격 형태이다. 대부분의 분산 서비스 거부 공격 전략은 다수의 분산 공격 에이전트들이 공격 대상 네트워크의 제한된 리소스를 모두 소모하는 것이다.

분산 서비스 거부 공격을 막는데 소스 아이피 어드레스는 중요한 요인이 된다. 소스 아이피 어드레스를 관찰하면 공격 패킷의 근원지를 추적할 수 있고, 트래픽 분석이 가능하기 때문이다. 하지만 소스 아이피 어드레스는 스푸핑되기 쉬우므로 소스 어드레스 스푸핑 여부와 상관 없는 대응책이 필수적이다. 그러한 대응책으로 가장 일반적인 기법은 IP traceback 이며, Pi 도 그러한 대응책 중 하나이다.

이 논문에서는 Pi를 이용한 분산 서비스 거부 공격의 방어 기법을 제안한다. 이 기법은 트래픽의 양에 따라 네트워크의 모든 패스를 공격 패스와 정상 패스로 구분함으로써 공격 패스로부터 흘러오는 모든 패킷을 필터링하는 것이다.

2. 관련연구

IP traceback은 소스 아이피 어드레스 스푸핑을 이용한 공격에 대한 일반적인 대응책이다 [2]. 라우터들은 공격 호스트로부터 피해 호스트까지의 경로를 흐르는 패킷의 아이피 헤더 필드 중 자주 쓰이지 않는 특정 필드에 자신의 아이피 어드레스를 마킹한다. 피해 호스트는 라우터들의 아이피 어드레스 정보가 담긴 패킷을 충분히 받은 후, 공격 호스트까지의 패스를 재구성할 수 있다. 이러한 정보를 이용한 하나의 방어 기법으로 상위 라우터에게 공격 패스로부터 흘러나오는 패킷을 필터링 해달라고 요청하는 방법이 있다. IP traceback의 단점은 충분히 많은 수의 패킷을 받은 후에야 공격 호스트까지의 정확한 패스를 재구성할 수 있다는 것이다. 이를 위해 라우터와 피해 호스트 둘 다 막대한 작업을 수행해야 한다.

Pi는 각 패킷에 패스 정보가 삽입된 새로운 패킷 마킹 기법이다 [3]. 피해 호스트는 이 정보를 이용해 패스를 구별할 수 있다. Pi는 결정론적인 메커니즘이다. 같은 패스를 경유하는 모든 패킷은 같은 마킹 값을 가진다. 그러므로 피해 호스트가 정상 패스와 공격 패스를 구별할 수 있다면, 공격 패스에서 오는 모든 패킷을 필터링 할 수 있다. Pi는 또한 패킷 단위의 마킹 기법이다. 피해 호스트가 받는 각 패킷에 경로상의 모든 라우터의 정보가 들어있으므로 패킷 단위로 필터링을 할 수 있다. 그러므로 라우터는 마킹만 하면 되고, 피해 호스트는 마킹 정보를 이용해 필터링만 하면 된다.

Pi의 기본적인 마킹 방법을 설명하자면, 라우터들은 그들이 전송하는 패킷의 IP Identification field에 자신의 아이피 어드레스의 마지막 n 비트를 마킹한다. 마킹할 위치를 결정하기위해 16비트를 n 개의 섹션으로 나누고($\lfloor 16/n \rfloor$) 인덱스로 패킷의 TTL 값을 이용한다(TTL mod $\lfloor 16/n \rfloor$). 그림 1은 Pi의 마킹 방법의 한 예를 보여준다. 그림 1에서 공격 호스트 A 로부터 피해 호스트 V 까지의 경로에 5개의 라우터가 존재하고, 이 라우터들은 자신을 지나가는 패킷의 IP Identification field에 자신의 아이피 어드레스 중 마지막 2비트를 마킹한다. 마킹할 위치를 결정하기 위해 16비트를 2비트씩 나누어 총 8개의 섹션을 만든다. 첫 번째 라우터는 자신의 TTL 값 255 mod 8 = 7, 즉 첫 번째 섹션에 마킹을 하고 두 번째 라우터는 마찬가지로 자신의 TTL 값 254 mod 8 = 6, 즉 두 번째 섹션에 마킹을 한다. 이러한 절차를 거친 패킷은 라우터 경로를 대표하는 16비트 Pi를 갖게 된다.

공격자는 같은 경로의 Pi 마킹 값을 여러 개의 마킹 값으로 바꾸기 위해 초기 TTL 값을 변경할 수 있다. 이에 대응하기 위해 피해 호스트는 TTL 값을 검사해서 패킷 내의 가장 오래

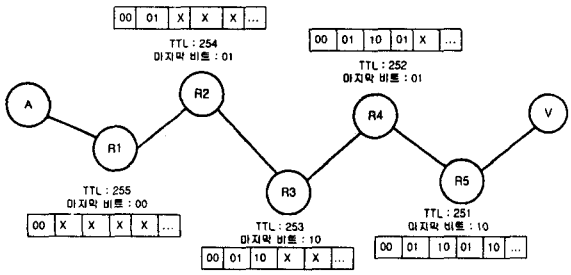

그림 1. Pi 마킹 방법의 예

된 마킹 위치를 찾을 수 있고, 이 위치를 기준으로 나머지 값들을 회전시켜서 다른 TTL 값에 대해 고유한 Pi 마킹 값을 얻을 수 있다. 이러한 방법을 TTL Unwrapping 이라고 부른다.

Pi의 단점은 적은 수의 라우터를 거치는 패킷에 대하여 무효한 마킹 값이 존재할 수 있다는 것이다. 라우터들이 자신의 어드레스를 1비트씩 마킹한다면 16비트의 IP Identification field를 모두 유효한 마킹 값들로 채우려면 16개의 라우터가 마킹에 참여해야 한다. 마찬가지로 2비트씩 마킹한다면 8개의 라우터가 마킹에 참여해야 한다. 만약 공격자가 피해 호스트 가까이에 위치한다면 마킹되지 않은 값들이 Pi에 많이 남아 있을 것이다. 공격자가 초기에 임의의 값들로 채워진 패킷을 보내고 u 개의 마킹되지 않은 값들이 남아있다면 공격자는 u 개의 마킹 값을 2^u 개 사이의 마킹 값들로 변경할 수 있게 되는 것이다. 이러한 경우 정확하게 패스를 결정하는 것은 불가능하다. 다음 장에서는 피해 호스트 가까이 위치한 공격 호스트를 막을 수 있는 방법을 제안한다.

3. Pi 분할 분석

이 논문에서 제안하는 기법은 모든 네트워크 패스를 공격 패스와 정상 패스로 구분하고 공격 패스로부터 오는 패킷을 필터링하는 것이다. 이 때, 공격 패스와 정상 패스를 구별할 수 있는 기준은 트래픽 양이다. 즉, 자주 발생하는 마킹 값을 포함하는 패킷을 탐지하는 것이다. 이를 위해 3.2에서는 패킷이 들어오면 마킹 값의 빈도수를 세서 자주 발생하는 마킹 값을 찾아내는 단순 빈도수 검사 기법을 생각해 볼 수 있다. 그러나, 만약 공격 호스트가 피해 호스트 가까이 위치하면 마킹되지 않은 값들이 많이 남아서 탐지 및 필터링이 정확하게 이루어지지 않는다는 문제가 발생한다. 이 문제점을 해결하기 위해, 3.3에서는 TTL Unwrapping을 적용하고 16비트의 IP Identification field를 여러 섹션으로 분할해서 필터링하는 Pi 분할 분석을 제안한다.

3.1 실험 환경

가상적인 DDoS 공격 트래픽을 발생하기 위해 루트 노드가 피해 호스트, 중간 노드가 라우터, 단말 노드가 공격 에이전트로 구성된 랜덤 이진트리를 만들었다. 이 때 단말노드의 차수는 15인데 그 이유는 인터넷 맵[4]과 스키터 맵[5]에서 실제 패킷을 수집한 결과 송신 호스트로부터 수신 호스트까지의 평균 라우터 수가 대략 15였기 때문이다. 이러한 실험 환경 아래, 공격 에이전트는 피해 호스트에게 패킷을 보내고, 라우터들은 자신이 전송하는 패킷에 그들의 어드레스의 해쉬값 중 마지막 한 비트를 마킹한다. 마지막으로 피해 호스트는 들어오는 모든 패킷을 수집해서 모든 마킹 값의 빈도수를 측정한다.

3.2 단순 빈도수 검사 기법

Pi의 기본 개념은 같은 패스로 들어오는 패킷은 모두 같은 마킹 값을 갖는다는 것이다. 즉, 같은 근원지에서 출발한 패킷은 같은 마킹 값을 가진다. 이러한 사실을 이용하면 들어오는 패킷이 같은 근원지로부터 출발한 것인지 아닌지를 결정할 수 있다. 또한, 공격이 발생하면 단독의 공격 에이전트가 방대한 트래픽을 소모한다. 그러므로 공격발생 시 들어오는 패킷을 관찰하면 특정 마킹 값이 자주 나타나는 것을 볼 수 있다. 이러한 사실을 근거로 자주 발견되는 마킹 값을 포함하는 패킷을 탐지하는 단순 빈도수 검사 기법을 생각할 수 있다. 이 기법은 들어오는 모든 패킷의 마킹 값의 빈도수를 세는 것이다. 이 때, 가능한 모든 마킹 값의 경우의 수는 $2^{16} = 65,536$ 이다.

단순 빈도수 검사 기법의 경우, 공격 호스트가 피해 호스트 가까이에 위치하면 마킹되지 않은 비트들이 Pi에 남아 특정 값에 대해 다양한 마킹 값들이 존재하게 된다. 즉, 마킹 값의 수는 더 증가하고 그들의 빈도수는 더 감소한다. 결국 필터링 리스트의 사이즈가 증가하는 것이다. 이처럼 증가한 필터링 리스트는 높은 오용탐지율(false-positive rate)을 초래한다. 3.3에서 제안하는 Pi 분할 분석의 목적은 오용탐지율을 줄이는 것, 다시 말해 필터링 리스트의 사이즈를 줄이는 것이다.

이 때, TTL Unwrapping을 적용하면 16비트의 IP Identification field 중 오른쪽 비트에 유효한 마킹 값들이 몰리고 이러한 유효 비트를 이용하면 피해 호스트 가까이 위치한 공격 호스트도 필터링 할 수 있을 것이다. 이를 증명하기 위해 16비트를 두 개로 나누어 왼쪽 비트와 오른쪽 비트에서

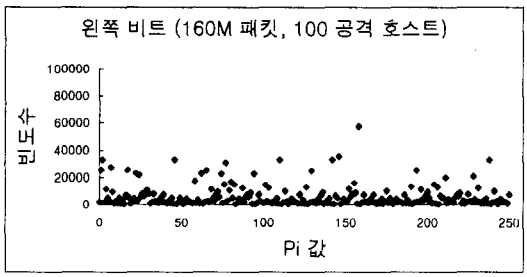

그림 2. 16비트를 두 개로 나눈 왼쪽 비트

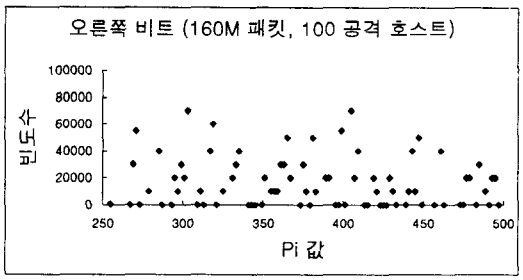

그림 3. 16비트를 두 개로 나눈 오른쪽 비트

빈도수의 분포를 비교해 보았다. 그림 2는 16비트를 두 개로 나눈 왼쪽 비트이고 그림 3은 오른쪽 비트이다. 그림을 보면 알 수 있듯이 그림 2의 값들의 분포가 그림 3의 값들의 분포보다 대체로 낮다. 그 이유는 그림 2의 값들이 마킹되지 않은 값들을 더 많이 포함하기 때문이다. 그래서 다양한 마킹 값(2^8 사이의 거의 모든 값들이 추출됨)들이 존재하고 그들의 빈도수는 낮다. 이에 반해 그림 3의 값들은 대부분 유효한 마킹 값들로 구성된다. 즉, 왼쪽 비트에 비해 적은 수의 마킹 값이 존재하고 그들의 빈도수는 더 높다.

만약 필터링 값을 40,000으로 설정한다면 그림 2에서는 극

소수의 값들만 필터링 되는데 반해 그림 3에서는 상당히 많은 값들이 필터링 된다. 이처럼 그림 2에서는 적절한 필터링 값을 설정하는 것이 어렵다. 하지만 16비트를 두 개로 나눈 오른 쪽 비트, 즉 유효 비트만 가지고 필터링을 한다면 피해 호스트 가까이 위치한 공격 호스트도 필터링 할 수 있게 된다.

3.3 Pi 분할 분석 (Pi dividing analysis)

우리가 제안하는 기법은 Pi 분할 분석으로 16비트의 IP *Identification field*를 여러 개의 섹션으로 나누어 분석하는 기법이다. Pi 분할 분석을 각 단계별로 설명하면, 처음 단계는 16비트의 IP *Identification field*를 여러 개의 섹션으로 나누는 것이다. 예를 들어 16비트를 4개의 섹션으로 나누면 각 섹션은 4비트로 구성되고 $2^4 = 16$ 개의 다양한 마킹 값들(0-15, 0: 4비트 모두 0으로 채워진 경우, 15: 4비트 모두 1로 채워진 경우)이 존재할 수 있다. 다음 단계는 패킷이 들어오면 모든 마킹 값의 빈도수를 세는 것이다. 그림 4와 5는 16비트를 4개의 섹션으로 나누어 분석한 결과이다. 그림 4와 5에서 공격 호스트의 수는 각각 10과 100으로, 정상 호스트의 수는 둘 다 1,000으로 설정했다. 그림 4와 5 모두 발생되는 패킷의 총 수

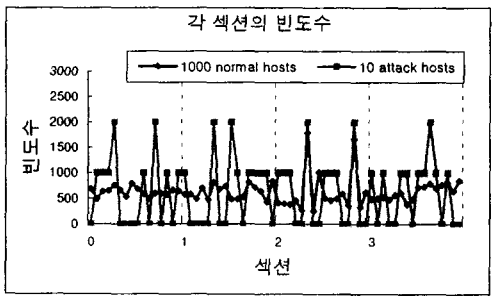

그림 4. 10개의 공격 호스트

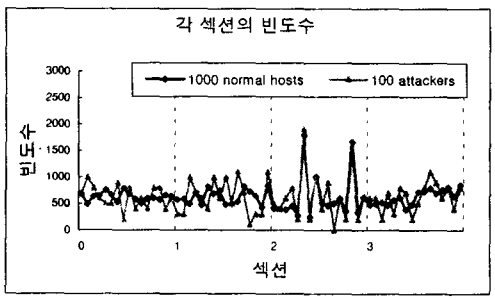

그림 5. 100개의 공격 호스트

는 20,0000이다. 그 중 반은 공격 호스트에 의해 발생되고, 나머지 반은 정상 호스트에 의해 발생된다. 그러므로, 그림 4와 5에서 공격 호스트는 각각 1,000개와 100개의 패킷을 발생한다. 빈도수를 살펴보면 1,000개의 정상 호스트가 발생하는 패킷의 빈도수보다 10개 혹은 100개의 공격 호스트가 발생하는 패킷의 빈도수가 더 높은 것을 볼 수 있다. 그러므로, 각 섹션별로 높은 빈도수를 가지는 값들을 조합하여 의심스러운 공격 리스트를 생성할 수 있다. 예를 들어, 그림 4에서 빈도수가 2,000(충분히 높은 빈도수)인 값들을 추출하면 첫번째 섹션에서 마킹 값 4와 11, 두번째 섹션에서 6과 9, 세번째 섹션에서 6과 14, 마지막 섹션에서 11에서 나타난다. 이 경우 의심스러운 공격 리스트에 추가되는 마킹 값의 총 수는 8개(2x2x2x1)이다. 마지막으로 의심스러운 공격 리스트에 있는 마킹 값들을 필터링 리스트에 추가한다. 그리고, 그 리스트에 있는 마킹 값와 일치하는 마킹 값을 갖는 패킷을 모두 필터링 한다.

4. 실험

나누는 섹션 수에 따른 공격 패킷의 필터링 비율을 측정하기 위해 다음과 같은 실험을 하였다. 그림 6은 단순 빈도수 검사 기법과 Pi 분할 분석에서 필터링 사이즈가 증가함에 따라 비례해서 증가하는 필터링 비율을 보여주는 실험의 결과이다. 우리는 섹션 수를 1,2,4로 선택했다. 이 때, 섹션 수가 1이라는 것은 단순 빈도수 검사 기법을 뜻한다. 실험 결과, 같은 필터링 사이즈일 때 섹션 수가 2일 때가 1일 때보다 필터링 비율이 더 높았다. 섹션 수가 4일 때는 1일 때보다 오히려 필터링 비율이 더 낮았다. 그 이유는 섹션 수가 증가할수록 의심스러운 공격 리스트의 사이즈는 증가하고 그에 따라 증가한 필터링 사이즈는 높은 오용탐지율을 초래하기 때문이다. 그림 6에서 필터링 사이즈가 1,500일 때를 기준으로 살펴보면 섹션 수가 1일 때는 필터링 비율이 0.7 이상이고 섹션 수가 2일 때는 0.5, 4일 때는 0.3이다. 즉, 섹션 수가 2일 때 가장 높은 필터링 비율을 보이고 있고 같은 필터링 비율을 기준으로 봐도 섹션 수가 2일 때 필터링 사이즈가 가장 작으므로 가장 낮은 오용탐지율을 보여준다.

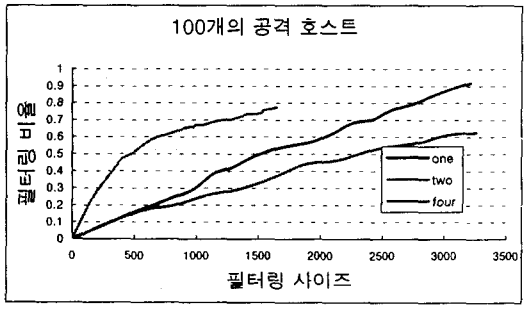

그림 6. 섹션 수의 영향

5. 결론

이 논문에서 Pi를 이용해 범람 서비스 거부 공격을 막을 수 있는 방법을 제안했다. 제안하는 기법은 모든 네트워크 패스를 공격 패스과 정상 패스로 분류해서 공격 패스로부터 흘러나오는 패킷을 필터링 하는 것인데, 패스를 구별하는 기준은 트래픽 양이다. 이를 위해 패킷이 들어오면 단순히 모든 마킹 값의 빈도수를 세는 방법을 소개했는데, 이 방법은 공격 호스트와 피해 호스트 사이의 거리가 짧은 경우 마킹되지 않은 값들이 많이 남는 문제가 생긴다. 이러한 문제를 해결하기 위해 TTL Unwrapping을 적용하고 16비트를 여러 섹션으로 분할해서 분석하는 기법을 제안했다. 실험 결과, 16비트를 두 개의 섹션으로 분할했을 때 가장 높은 필터링 효과를 보여주었다.

참고문헌

[1] David Moore. Slammer Worm. http://www.cs.berkeley.edu/~nweaver/sapphire/.
[2] M. Adler. "Tradeoffs in probabilistic packet marking for IP traceback", ACM, pp. 407 - 418, 2002.
[3] Abraham Yaar and Adrian Perrig. "Pi: A Path Identification Mechanism to Defend against DDoS Attacks", IEEE, pp. 93 - 107, 2003.
[4] Internet mapping. http://research.lumeta.com/ches/map/,2002.
[5] Caida. Skitter. http://www.caida.org/tools/measurement/skitter/,2000.