네트워크 포렌직을 위한 트래픽의 실시간 비손실 압축에 관한 연구

유상현[0], 김기창
인하대학교 정보통신공학부
babobaram@super.inha.ac.kr [0], kchang@inha.ac.kr

Real-time Lossless Compression of Traffic for Network Forensics

Sang-hyun Yoo[0], Ki-chang Kim
School of Information & Communication Engineering, Inha Univ.

요 약

최근 가파른 증가세를 보이며 그 기법 또한 다양해지고 있는 공격에 대한 사후처리와 동일 침입 방지를 위해, 네트워크 포렌직에 대한 관심이 커지고 있다. 포렌직을 위해서는 정보의 손실없는 네트워크 트래픽을 수집하여 보존하는 것이 필요하지만, 그 양이 막대하며, 이는 더 큰 용량의 디스크를 필요로 한다. 이를 해결하기 위해서는 압축을 수행하는 것이 필요하며, 또한 IP와 같이 필요로 하는 몇몇 정보를 압축해제 없이 접근할 수 있게 한다면, 간단한 작업을 위해서도 압축을 풀기 위해 컴퓨팅 파워와 시간을 줄일 수 있을 것이다. 따라서, 이 논문에서는 수집한 패킷마다 압축을 수행할 부분과 수행하지 않을 부분으로 나누고, 압축을 수행한 뒤, 해당 정보를 4바이트의 헤더로 만들어 덧붙임으로써, 기존 트래픽을 압축함과 동시에 패킷들에 대한 간단한 정보들을 압축해제 없이 접근할 수 있는 모델을 제안하였다.

1. 서 론

전 세계적으로 인터넷을 중심으로 한 네트워크 인프라가 발전함에 따라 네트워크 침해 사례가 증가할 뿐 아니라 다양해지고 있으며, 또한 최근 이러한 공격이 국가 혹은 국가를 초월한 해커 집단에 의해 조직적인 행태를 띰으로써 국가 혹은 기업 기밀 유출, 서비스 중단 등의 결과로 이어지고 있다. 이러한 공격들을 탐지하고 차단하기 위해, 침입차단, 침입탐지, 침입방지 등의 방법이 도입되어 사용되고 있으나, 이들도 공격기법의 변화속도를 따라잡을 수 있을 정도로 완벽하지는 못하다.

이러한 방법들을 보완하면서, 공격에 따른 이후의 사후조치를 위해 최근 네트워크 포렌직이 많은 관심을 받고 있다. 네트워크 포렌직[1]을 위해서는 해당 서브넷에 오가는 모든 트래픽을 잡아볼 수 있어야 하지만, 네트워크 속도를 감당할 수 없음으로 인한 패킷의 드롭은 차제하고라도, 막대한 크기의 트래픽량은 곧 하드웨어 비용의 증가로 이어질 것이다. 트래픽량의 크기를 줄이기 위해 주기적인 압축을 실시한다고 해도, 그 내용 확인을 위해서는 압축을 풀어야만 하며, 이 또한 많은 컴퓨팅 파워와 시간을 필요로 할 것이다.

따라서, 이 논문에서는 네트워크 포렌직을 위해, 기존의 압축 라이브러리들을 활용하여 트래픽을 실시간 압축하고, 프로토콜 헤더의 특정 필드들에 압축해제 없이 접근하기 위한 한 접근방법을 제안하고 이에 따른 결과를 실험을 통해 보이고자 한다.

이 논문의 2절에서는 관련연구로써 네트워크 포렌직에 대한 소개와, 침입탐지 시스템에서 감사자료 축약을 위해 제안된 방법들, 논문을 위해 검토하였던 압축 라이브러리들과 패킷수집을 위한 알고리즘에 대하여 간략히 설명하고, 3절에서는 논문에서 제안하고 있는 바를, 4절에서는 그를 이용한 실험과 결과를, 끝으로 5절에서는 결론과 향후 연구과제들에 대하여 설명하며 결론을 맺는다.

2. 관련연구

2.1 네트워크 포렌직

일반적인 네트워크 모니터링은 개별적인 트래픽 흐름과 그 내용들을 살펴보게 되지만, 때로는 모든 트래픽을 수집하여, 그 부분을 떼어 분석하는 것이 필요한데, 이러한 작업을 네트워크 포렌직이라고 한다[1].

보안의 측면에서, 네트워크 포렌직 시스템은 수집된 네트워크 트래픽을 가지고, 사후에 종합적인 분석을 통해 침입탐지 혹은 침입방지 시스템이 미처 발견해내지 못한 비정상행위 등을 찾아내고, 트래픽을 이용해 이를 재현하거나, 법적 대응을 위한 증거수집, 이후 침입방지를 위한 해결책 등을 제공해 줄 수 있다. 이를 위해서는 모든 네트워크 트래픽의 수집을 필요로 한다.

2.2 데이터 축약

최근 침입탐지 시스템들에서는 비정상행위 탐지를 위해 발생하는 다량의 감사데이터를 줄이고 탐지 속도를 향상시키기 위하여, 데이터 마이닝[2], 신경망[3] 등을 이용하여 감사데이터를 축약하고 있지만, 포렌직을 위해서는 손실없는 원본데이터를 필요로 하기 때문에, 이들을 적용할 수는 없다.

2.3 압축 라이브러리

2.3.1 lzo

lzo는 oberhumer에 의해 개발된 LZ계열의 알고리즘을 사용사용하는 실시간 비손실 데이터 압축 라이브러리로 블록단위(sliding dictionary)의 비교를 통한 압축을 수행한다. 압축률은 zlib에 비해 떨어지지만, 압축/해제 속도가 대단히 빠르기 때문에, NASA의 화성탐사 로버(rover)인 Spirit와 Opportunity에 사용되고 있다[4].

2.3.2 zlib

zlib은 GNU tool인 gzip에 사용되는 라이브러리로 압축 알고리즘(deflate)은 LZ77[5]과 허프만 코딩[6]의 조합을 이용하고 있으며, lzo와 유사하게 입력 데이터로부터 중복된 스트링을 찾아서, 이를 이전에 찾았던 스트링에 대한 포인터, 길이쌍으로 대체함으로써 압축을 수행한다. 이 알고리즘은 IETF의 네트워크 워킹그룹에 의해 표준화 작업이 진행되어 RFC1950, 1951, 1952[7, 8, 9]에 반영되어 있다. zlib은 또한 입력 데이터를 데이터 스트림으로 다루면서 압축을 수행한 데이터의 일부분을 플러쉬(flush)할 수 있다.

3. 실시간 트래픽 압축

이 연구에서는 실시간 비손실 트래픽 압축과 동시에 압축해제 없는 IP와 그 상위계층 헤더로의 접근을 위해 그림 3.1과 같은 모델을 제안한다.

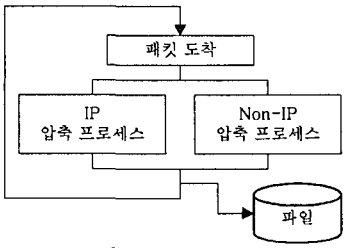

그림 3.1 제안된 모델

표 3.1 트래픽 분석 결과

분류	패킷수	비율
IP	31,407,915 개	92.36 %
non-IP	2,598,661 개	7.64 %
계	34,006,576 개	

모델은 크게 IP 압축 프로세스와 non-IP 압축 프로세스로 나뉜다. 대상 네트워크의 트래픽을 수집한 결과, 표 3.1과 같은 분포를 보임을 알 수 있었지만, ARP 스푸핑 등 non-IP 패킷을 이용한 공격에 대한 고려가 필요했기 때문에, 그림 3.1과 같이 IP와 non-IP의 두 부분으로 나뉘어졌다. 소스 데이터는 2계층 헤더를 포함한다.

3.1 패킷 수집

패킷의 수집은 tcpdump와 Snort 등 많은 오픈소스 소프트웨어들이 사용하고 있는 pcap 라이브러리를 통해서 이루어지며, 특별한 필터링 룰은 적용되지 않았다. 매 패킷이 도착할 때마다, 이더넷 헤더의 type 필드를 체크하여 IP 패킷과 non-IP 패킷을 구별한다.

3.2 압축 프로세스

3.2.1 압축 라이브러리의 선택

매 패킷마다 수행하는 압축 프로세스를 위해서는 zlib 라이브러리가 사용되었다. lzo의 경우, 매 패킷이 도착할 때마다 중복을 체크하기 위한 사전이 초기화되기 때문에, 압축률이 매우 떨어지는 문제점을 지니고 있다. 반면, zlib은 이를 스트림으로 관리하기 때문에, 매번 패킷이 도착할 때마다, 스트림에 데이터를 넣어주면, 32K 이내의 옵셋내에서 이전의 중복 데이터에 대한 정보를 유지하기 때문에, 압축률을 높일 수 있다.

간단한 실험을 통해 확인해본 결과, 표 3.2에서 보는 바와 같이 펜티엄 II 400에서 10MB 크기의 파일을 압축하였을 경우, lzo가 zlib에 비해 약 1.5배정도 빠르기는 하지만, 단일 패킷의 크기가 MTU에 의해 일반적으로 최대 1,500 바이트 정도의 크기에 한정되기 때문에, 초당 1,300 KB 이상을 압축할 수 있는 zlib을 이용하여도 무방하였다.

표 3.2 파일 압축 시간(파일 크기: 10MB)

라이브러리	소요 시간	KB/sec(환산)
lzo	2.742 sec	약 3,562 KB/sec
zlib	7.093 sec	약 1,377 KB/sec

3.2.2 압축 정보 헤더

압축후에 원하는 데이터를 압축 해제 없이 얻기 위해, 패킷마다 그림 3.2와 같이 4 바이트 크기의 압축 정보 헤더를 덧붙였다.

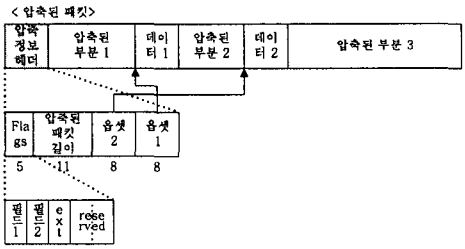

그림 3.2 압축 정보 헤더

이 헤더의 옵셋은 IP, TCP, UDP 등 정형화된 헤더의 필드들을 가리키게끔 만들어진 것이며, 그림 3.2에서와 같이, flag을 이용해 패킷의 헤더중 원하는 헤더내의 필드를 선택하고, 그 필드까지의 옵셋을 제공하여, 곧바로 찾을 수 있게 하였다. 또한, 압축된 패킷 길이를 이용하여, 다음 패킷으로 접근할 수 있게 하였다. 하지만, 4 바이트 헤더의 경우 옵셋을 최대 2개까지만 제공할 수 있기 때문에, flag에 ext 필드를 두어, 더 많은 필드를 원할 경우, 헤더를 확장할 수 있게 하였다.

3.2.3 압축 프로세스

압축 프로세스는 그림 3.3와 같다.

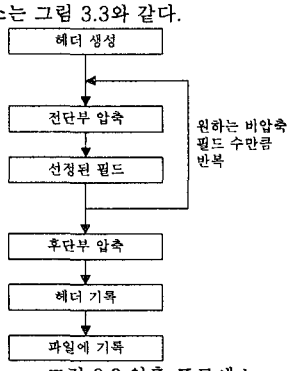

그림 3.3 압축 프로세스

기본적인 아이디어는 매우 간단하다. 패킷을 원하는 필드를 기준으로 전·후단부로 나누고, 원하는 필드는 압축을 수행하지 않는다. 압축 이후의 필드까지의 정보가 헤더의 옵셋에 저장되는데, 이때, 헤더를 위한 4 바이트 외에 선택된 필드를 압축하지 못함으로 인한 오버헤드가 약간 발생한다. 이 부분을 허프만 코딩을 이용하여 분리 압축하는 방법에 대해 고려해 보았지만, 이는 곧 프로세싱 시간에 대한 오버헤드로 이어질 것이므로, 압축률을 약간 포기하는 쪽으로 선택이 이루어졌다. 또한, 압축 정

보 헤더와 중간 압축 부분들은 임시버퍼에 저장된다.

4. 실험 및 결과

실험은 압축률과 압축후 특정 필드에 대한 접근에 걸리는 시간을 측정하는 두 가지로 진행되었다. 실험은 그림 4.1과 같은 네트워크상에 펜티엄II-400 프로세서를 갖는 스니퍼를 두고, 그로부터 수집된 패킷을 대상으로 이루어졌으며, 이 연구에서 제안한 모델과 앞서 소개한 두 압축 라이브러리를 이용한 압축 프로그램인 gzip, lzop에 의한 결과가 비교되었다.

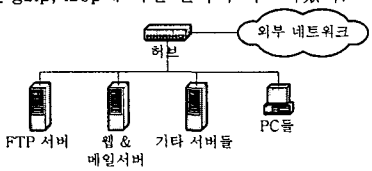

그림 4.1 대상 네트워크

실험을 통한 비교를 위해 트래픽을 제안 모델을 이용하여 실시간 압축하는 동시에 동일 트래픽을 파일로도 저장하였으며, 그 크기는 제안 모델에 의해 압축된 파일의 크기를 기준으로 1G, 100M, 10M로 나뉘어 수집되었다.
제안된 모델을 통해 압축시 선택된 필드는 IP 주소였다.

4.1 압축률

일반적으로 중복의 제거를 통한 압축 알고리즘의 압축률은, 압축대상에서의 중복의 분포 및 횟수에 크게 의존하므로, 수집된 트래픽의 특성에 따라 압축률이 큰 차이를 보일 수 있다. 때문에, 압축률을 그대로 보이는 대신, 가장 좋은 압축률을 보여주는 gzip에 대한 상대적인 압축률을 계산하였으며, 그 결과는 표 4.1과 같다.

표 4.1 압축률 비교(gzip 기준)

파일크기	제안모델	lzop	gzip
1G	113.77 %	103.50 %	100.00 %
100M	120.55 %	109.44 %	100.00 %
10M	110.64 %	102.25 %	100.00 %
평균	114.99 %	105.06 %	100.00 %

제안된 모델의 경우, 패킷당 4 바이트의 오버헤드가 존재하는데다 선택된 필드는 압축을 수행하지 않기 때문에, 압축률은 기존 압축프로그램에 비해 많이 떨어지는 모습을 보여주었다. 하지만, 압축되지 않은 raw 트래픽과 비교해서는 표 4.2에서와 같이 좋은 결과를 보여주고 있다.

표 4.2 제안된 모델을 이용한 raw 트래픽 압축률

파일크기		압축/raw	비율
1G	최적	950298976/2147483647	44.25 %
	최악	1000230283/1153836480	86.69 %
	평균	995842902/1648628193	60.40 %
100M	최적	100000582/325238325	30.75 %
	최악	100000059/106269763	94.10 %
	평균	100000219/168029488	59.51 %
10M	최적	10000010/30631505	32.64 %
	최악	10000674/13472614	74.23 %
	평균	10000323/20915288	47.81 %

4.2 필드 검색시간

압축률과는 달리 필드의 검색시간은 해당 트래픽 내에서의 IP 패킷 수(IP 주소를 검색하기 때문에)에 따라 큰 차이를 보였다. 때문에, 압축률에서와 마찬가지로 gzip을 기준으로 하여 비교를 수행하였으며, 그 결과는 표 4.3와 같다.

표 4.3 IP 주소 검색시간 비교(gzip 기준)

파일크기	제안모델	lzop	gzip
1G	15.89 %	98.98 %	100.00 %
100M	19.83 %	94.11 %	100.00 %
10M	24.67 %	68.84 %	100.00 %
평균	20.13 %	87.31 %	100.00 %

위의 결과와 같이 검색시간에서 큰 차이를 보이는 이유는, 기존 툴을 이용하여 압축을 해제하는데 소요되는 시간이 컸기 때문이다. 오로지, raw 트래픽을 검색하는 시간과 비교하여서는 거의 동일한 시간이 소요되었다.

5. 결론 및 향후 연구

네트워크 포렌식이나 네트워크 침입탐지와 같이, 감사데이터로써 네트워크 트래픽을 필요로 하면서, 각 패킷의 헤더내에 있는 특정 필드로의 접근이 빈번한 경우를 위해, 트래픽을 실시간 압축하면서, 압축해제 없이 원하는 데이터를 빠른 시간에 얻을 수 있는 모델을 제안하였다. 또한, 실험을 통해, 일정정도의 압축률을 얻으면서도, 상대적으로 대단히 빠르게 데이터에 접근할 수 있음을 볼 수 있었다.
하지만, 실험에서는 패킷을 수집하고 압축하는 동안 어느 정도의 패킷이 폐기되었는지는 고려되지 않았다. 따라서, 현재의 초고속 통신망에서는 CPU의 업그레이드, 이 모델을 하나의 하드웨어 모듈로 만드는 것, 트레픽을 분산시키는 방법 등을 통한 폐기 패킷의 최소화가 고려될 수 있을 것이다.

참고문헌

[1] V. Corey, C. Peterman, S. Shearin, M. S. Greenberg, J. V. Bokkelen, Network Forensics Analysis, Internet Computing, IEEE, vol. 6, issue 6, pp. 60-66, Nov.-Dec. 2002.
[2] W. Lee, S. J. Stolfo, K.W. Mok, Mining Audit Data to Build Intrusion Detection Models, Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, pp. 62-72, Aug. 1998.
[3] S. Mukkamala, A. H. Sung, Audit Data Reduction Using Neural Networks and Support Vector Machines, Proceedings of Digital Forensics Research Workshop, Aug. 2002.
[4] M. Oberhumer, LZO - A Real-time Data Compression Library, www.oberhumer.com/opensource/lzo/lzodoc.php, Jul. 2002.
[5] Lempel, A., and J. Ziv, A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, vol. IT-23, No. 3, May 1977.
[6] D. A. Huffman, A Method for the Construction of Minimum Redundancy Codes, Proceedings of IRE, 40, pp. 1098-1101, 1952.
[7] P. Deutsch, J. L. Gailly, ZLIB Compressed Data Format Specification ver. 3.3, RFC-1950, May 1996.
[8] P. Deutsch, DEFLATE Compressed Data Format Specification ver. 1.3, RFC-1951, May 1996.
[9] P. Deutsch, GZIP File Format Specification ver. 4.3, RFC-1952, May 1996.