Bloom Filter를 이용한 악성 코드 탐지 방안

이상훈^O 허환조 김효곤 최 린
고려대학교
smileland4u@hanmail.net^O, {urmajest, hyogon, lchoi}@korea.ac.kr

Filtering of Malicious Codes using Bloom Filter

Sanghoon Lee^0 Hwanjo Heo Hyogon Kim Lynn Choi
Korea University

요 약

바이러스로 시작된 악성 코드는 웜이라는 형태로 발전하였다. 인터넷 망의 고속화와 확장에 의해 웜의 전파 속도와 감염 범위는 증가하였지만, 아직까지 웜을 차단할 수 있는 획기적인 방법은 개발되지 않았고, 웜에 의한 피해는 갈수록 치명적인 결과를 낳고 있다. 본고에서는 Bloom Filter[1]를 이용한 content filtering 방법을 제안한다. 실험을 통해, 이미 알려진 웜에 대한 Bloom Filter의 성능을 검증하였으며, 알려지지 않은 웜에 대한 Bloom Filter의 적용 방법도 제안한다.

1. 서 론

2003년 1월 25일 발생한 SQL-Slammer 웜은 MS-SQL 서버의 취약점을 이용하여 전파되었고, 이로 인해 국내외 인터넷 망이 마비되었다. CAIDA(Cooperative Association for Internet Data Analysis) 의 분석 보고서에 따르면, 8.5초 마다 감염된 서버의 수가 두 배로 증가하였으며, 10분 후에는 전 세계 MS-SQL 서버의 약 90%가 감염되었다.[2] 이와 같은 새로이 출현하는 웜에 의한 피해 뿐만 아니라, 이미 알려진 웜에 의한 공격도 꾸준히 계속되고 있다.[3]

본고에서는 이러한 악성 코드의 공격에 대응하기 위해, Bloom Filter를 이용한 새로운 content filtering 방법을 제안한다. 이 방법은 *pattern matching*에 기반을 둔 기존의 content filtering 방법과는 달리 두 가지 장점을 가진다. 첫째로, Bloom Filter는 hash 함수를 사용하므로 기존의 pattern matching 방법에 비해 속도가 빠르다. 두 번째는, Bloom Filter는 알려진 웜뿐만 아니라 알려지지 않은 웜도 차단할 수 있다는 점이다.

2. 알려진 웜에 대한 filtering

본고에서는 웜에 대해 두 가지 가정을 하였다.

① ***No polymorphism*** : 웜은 하나의 형태를 가지고, 변형을 생산하지 않는다.

② ***Worms in the fore part of payload*** : 웜은 payload의 앞부분에 위치하며, payload의 첫 c bytes 내에 웜의 정보가 포함되어 있다.

Content filtering이 가능하기 위해서는 첫 번째 가정이 필요하며,[4] 두 번째 가정은 웜의 크기가 최소한 c bytes 이상이어야 함을 의미한다. 또한, 첫 c bytes만 검사하므로 c는 패킷의 최소 길이보다 길어야 한다. 실험에서는 c는 256으로 설정하였다.

Bloom Filter는 hash 함수를 통해 x개의 주어진 signature에서 y-bits의 결과를 얻고, bit vector의 index로 사용한다.[1] Filter 크기가 m-bits이고, k개의 hash 함수를 사용해서 총 n개의 signature로 filter를 생성한 경우 false-positive는

$P_e = (1-(1-1/m)^{kn})^k$ 이다.

본고에서는 세 개의 hash 함수를 사용하였다.

① Bit extraction[5] : 이 함수는 payload의 첫 256bytes에서 선택적으로 16bits와 24bits를 추출한다. 이 때 filter 크기는 2^{16}과 2^{24}이 된다.

② Byte selection with XOR[5] : Payload의 첫 256bytes에서 h + 4i의 위치에 있는 byte를 선택하며, h와 i는 각각 $0 \le h \le 3$, $0 \le i \le 63$이다. Byte selection에서는 h값이 서로 다른 네 개의 함수를 사용한다. Byte selection을 통해 얻어진 키 값은 XOR를 통해 index 값으로 변환된다.

③ Class H3[6] : i×j 크기의 행렬 Q가 있고, q(k)는 행렬의 k번째 열을 나타낸다. x(k)는 bit extraction을 통해 구한 키 값의 k번째 bit이다. 이 때 Hash 함수 $h_q(x)$는 아래와 같이 정의된다.

$$h_q(x) = x(1) \cdot q(1) \oplus x(2) \cdot q(2) \oplus \cdots \cdots \oplus x(i) \cdot q(i)$$

각 함수의 성능을 비교하기 위해, 950MHz AMD Duron CPU와 384MB 메모리 상에서 7000개의 signature와 백만 개의 단일 패킷을 사용했다. [표1]은 연산시간에 대한 실험 결과를 나타내는데, 표에서 연산시간은 각각의 hash 함수를 사용한 키 값 생성지연과 주어진 키 값을 이용한 filtering지연이 포함된다. 결과에 따르면 Bit Extraction 함수를 사용했을 때 가장 성능이 탁월했다.

Hash 함수	연산시간(ms)
Bit Extraction	5750
XOR	20775
Class H3	419350

[표 1] 백만 개의 단일 패킷에 대한 연산시간

결과에 따르면 Bit Extraction은 약 177Kpps의 연산속도를

나타내는데, 이는 대략 900Mbps 급의 인터넷 트래픽을 처리할 수 있는 능력이다.[7]

[표2]에 따르면 Class H3를 사용한 Bloom Filter가 가장 낮은 false-positive를 나타낸다. 그러나 false-positive는 함수의 종류보다 오히려 bit vector의 크기와 hash 함수의 수에 더 큰 영향을 받는다. 특히 $m=2^{24}$일 때 false-positive는 매우 낮아진다. 단 하나의 hash 함수만으로도 10^{-4} 수준의 에러 확률을 얻을 수 있으며, 현재 네트워크의 일반적인 수준인 10^{-6}에서 10^{-9}의 에러 확률을 얻기 위해서는 두개의 hash 함수만으로도 충분하다. 그러나 여전히 정상 패킷에 대한 false-positive의 가능성이 남아있으므로 Bloom Filter에서 웜으로 인식된 패킷에 대한 2차적인 검사가 필요하다.

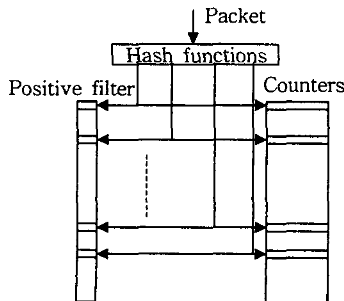
마지막으로, Bloom Filter가 요구하는 메모리 크기와 관련해서, $m=2^{24}$일 때 오직 2MB의 메모리가 필요하다. 이는 on-chip 메모리로 구현 가능한 크기이다.

			Bit Extraction	Byte selection with XOR	Class H3
$m=2^{16}$	k=1	이론값		0.1013	
		실험값	0.1051	0.1040	0.1011
	k=2	이론값		0.0370	
		실험값	0.0404	0.0430	0.0373
	k=4	이론값		0.0146	
		실험값	0.0165	0.0158	0.0148
$m=2^{24}$	k=1	이론값		4.17e-04	
		실험값	1.79e-04	1.54e-04	1.37e-04
	k=2	이론값		6.96e-07	
		실험값	0	0	0
	k=4	이론값		7.73e-12	
		실험값	0	0	0

[표 2] 백만 개의 단일 패킷에 대한 false-positive

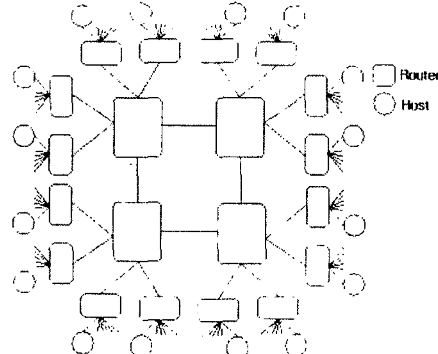
3. 알려지지 않은 웜에 대한 filtering

알려지지 않은 웜의 가장 큰 위협은 그것이 확산되기 전에는 웜의 signature를 정의하기 힘들다는 점이다. 그래서 본고에서 제안하는 방법은 웜의 일반적인 전파 특성을 -감염된 패킷의 전송 수가 기하급수적으로 증가- 이용한다. 이를 위해 Bloom Filter에 카운터를 추가했고, 그 구조는 [그림 1]과 같다.



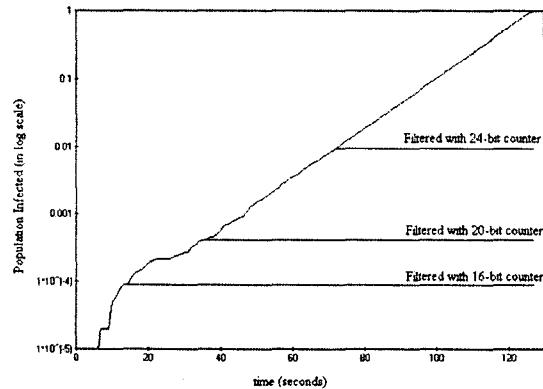
[그림 1] 알려지지 않은 웜에 대한 filter 구조

Positive filter는 알려진 웜에 대한 filter로서, 2장에서 소개한 Bloom Filter이다. Counter는 positive filter와 같은 index를 사용하며, hash 함수의 결과에 따라 각 counter 값이 1씩 증가한다. Counter 값이 기준값 이상 증가하면 filter는 해당 패킷을 알려지지 않은 웜으로 간주한다. 그런데, 증가만 반복할 경우 모든 counter값은 기준값을 넘게 되므로 일정시간마다 counter 값은 재설정되어야 한다. 이 때 재설정 주기는 네트워크 내의 filter의 위치에 따라 유연하게 설정되어야 한다. 실험에서는 재설정 시간 이내에 counter 값이 기준값을 초과하는 것으로 설정하였다.



[그림 2] 실험에서 사용한 네트워크 구조

실험에서는 [그림 2]와 같은 네트워크 구조를 사용했고, 2^{32}개의 호스트는 네트워크 가장자리에 동일하게 분포되어있다. 최초로 웜을 감지한 라우터는 주위의 모든 라우터에 웜에 대한 정보를 전송하며, 각 라우터는 자신의 positive filter를 갱신한다. 웜에 감염된 호스트는 PRNG[1] 함수를 사용해서 초당 4000개의 목적 주소를 생성하고, 생성된 주소를 향해 웜 패킷을 전송한다. 총 호스트 중 웜에 취약한 호스트는 총 10만개이며 웜 패킷을 전송받은 호스트는 1초 후부터 웜 패킷을 전송한다. 또한 hashing 지연은 패킷의 흐름에 영향을 주지 않을 만큼 작다고 가정하였다.



[그림 3] 시간에 따른 호스트의 감염 추이

[1] Pseudo Random Number Generation : SQL Slammer 웜에서 사용된 함수. $x'=(x*214013+2531011) \bmod 2^{32}$

[그림 3]은 웜의 전파 속도를 나타내는데, 실험에서는 hash 함수 값을 XOR함으로써 하나의 counter index만 사용하였다. 즉 모든 패킷은 하나의 카운터 index를 가진다. 웜은 지수적으로 감염자가 증가하는 전형적인 전염현상을 보인다. 실험에서 기준값은 counter가 저장 가능한 가장 큰 값으로 설정했고, 특정 counter값이 기준값에 도달하면 라우터는 해당 패킷을 웜으로 간주한다. 웜이 라우터에 의해 차단되기 전까지 Counter 크기에 따른 감염된 호스트의 수는 [표 3]과 같다. 실험에 따르면 라우터는 웜이 전파되기 시작한 후 1분 이내에 충분히 웜을 차단할 수 있었으며, 웜의 전파를 억제함으로써 감염 수를 아주 낮게 줄일 수 있었다. 즉, 알려지지 않은 웜이 전파되기 시작한 경우, 새로운 웜에 대한 적절한 대응책이 나오기 이전의 피해를 크게 줄일 수 있음을 뜻하다.

Counter bit	감염된 호스트 수
16 bits	3
20 bits	42
24 bits	781
28 bits	13,493

[표 3. Counter 크기에 따른 감염된 호스트 수]

4. 결 론

속도와 false-positive 두 가지 관점에서 Bloom Filter의 가능성에 대해 논했다. Bloom Filter는 900MHz의 CPU를 이용해서도 기가비트급에 가까운 처리 속도를 보장할 수 있었다. 그리고 2MB의 메모리만으로도 매우 낮은 false-positive를 가진다. On-chip 메모리를 사용한다면 빠른 처리 속도와 낮은 false-positive를 제공할 수 있다. 또한 Bloom Filter가 signature를 바탕으로 검색하므로, 알려지지 않은 웜의 차단에도 적용이 가능하다. 이는 Bloom Filter가 알려지지 않은 웜에 의한 피해를 효율적으로 막을 수 있는 하나의 방안임을 뜻한다.

실험에서는 실제 웜이 아닌 임의로 생성한 signature를 사용했고, 몇 가지 제약 조건이 있었다. 앞으로, 이러한 문제에 관해 추가적인 실험이 필요하고, 실제 네트워크 상에서의 성능과 개선점에 대한 연구가 필요하다.

참고

1. B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Communications of ACM*, vol. 13, no. 7, pp. 422-426, July 1970.

2. D. Moore, V. Paxson, S. Savage, C. Shannon, S.Staniford, N. Weaver, "The spread of the Sapphire/Slammer Worm," http://www.caida.org/outreach/papers/2003/sapphire/

3. http://isc.incidents.org/

4. D. Moore, C. Shannon, G. M.Voelker, S. Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code," *Proceedings of IEEE INFOCOM*, 2003.

5. E. Fu, E. Bahcekapili, M.V. Ramakrishna,"Efficient hardware hashing functions for high performance computers," *IEEE TransactionsComputers*, vol. 46, No. 12, December 1997.

6. L. Carter and M. Wegman, "Universal Classes of Hashing Functions," *J. Computer and System Sciences*, vol. 18, no. 2, pp. 143-154, 1979.

7. H. Kim, "Fast classification, calibration, and visualization of network attacks on backbone links," submitted for publication. http://net.korea.ac.kr/papers/ccv.pdf