시스템 콜 시퀀스를 이용한 침입탐지 오토마타의 자동생성

문병은[0], 위규범
아주대학교 정보통신전문대학원
{lovtea,kbwee}@ajou.ac.kr

The Automatic Generation of Intrusion Detection Automata using System Call Sequences

Byung-Eun Moon[0], Kyu-Bum Wee
Graduate School of Information and Communication, Ajou University

요 약

침입 탐지 시스템 연구에서 정상 행위와 비정상 행위를 구별하기 위한 방법으로 시스템 콜 시퀀스를 이용하는 방법들이 많이 소개되었다. 그 중에서도 정상적인 시스템 콜 시퀀스를 프로파일링 하는데 있어서 오토마타를 이용하는 방법들이 제안되었다. 그러나 정상적인 시스템 콜 시퀀스의 오토마타를 생성하는데 있어서 수동적으로 생성하는 방법이 대부분이었고, 자동적으로 생성하는 방법도 제안되었다. 본 논문에서는 시스템 콜 시퀀스에 대한 오토마타를 자동으로 생성하는 방법을 제안한다.

1. 서 론

침입 탐지 시스템 기법은 크게 두 가지로 나뉜다. 하나는 알려진 침입 패턴을 놓고 이 패턴과 일치 또는 유사한 패턴을 침입으로 간주하는 오용탐지 기법이고, 다른 하나는 정상 행위를 모델링 하여 모델링한 행위로부터 벗어나는 것을 침입으로 간주하는 비정상탐지기법이다. 이 중 비정상탐지기법은 알려지지 않은 침입 패턴에 대해서도 탐지할 수 있다는 장점을 가지고 있다.

현재까지 정상 행위를 모델링하기 위해서 통계적 방법, neural network, Hidden Markov Model(HMM), Automata 등을 사용한 방법 등이 있다.. 이중 Forrest 가 제안한 방법은 정상 시스템 콜 시퀀스를 슬라이딩 윈도우 기법을 사용하여 일정한 길이로 자른 후 데이터베이스에 저장한다. 이후 새로운 시스템 콜 시퀀스가 발생하면, 그 시스템 콜 시퀀스에 대하여 동일한 방법을 적용하여 자른 후 데이터베이스에 있는 정상 부분 시퀀스와 비교하여 매치 되지 않는 횟수가 threshold를 넘어서면 침입으로 판별한다[3].

본 논문에서는 정상적인 시스템 콜 시퀀스를 모델링 하는데 있어서 기존의 오토마타를 사용하는 방법[1,4,5]의 단점을 극복할 수 있는 오토마타를 생성하는 자동화된 방법을 제안하고, 방법의 유용성을 판단하기 위하여 Forrest 가 제안한 방법과 결과를 비교하였다.

제 2장에서는 오토마타를 사용한 관련연구에 대해서 기술한다. 제 3장에서는 제안된 방법에 대한 전체적인 절차를 설명한다. 4장에서는 실험 결과 및 분석을 서술한다. 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련연구

정상행위를 모델링 하는데 오토마타를 사용하는 기존의 기법으로 Kosoresow가 제안한 방법이 있다. 이 방법은 다음의 단계를 거친다[4].

- ▶ 1단계 : 시스템 콜을 프로세스 별로 분리한다.
- ▶ 2단계 : 공통된 prefix와 suffix를 찾아, 각 시퀀스를 prefix, main part, suffix 로 분할한다.
- ▶ 3단계 : 시퀀스에 공통적으로 자주 나타나는 패턴을 매크로로 대체한다.
- ▶ 4단계 : 모델링에 사용된 모든 프로세스를 승인하는 오토마타를 생성한다.

Kosoresow가 제안한 이 방법은 매크로를 사용함으로써 오토마타의 크기가 작아지고, 전체 시퀀스를 대상으로 모델링 하였으므로 정상적인 시스템 콜 시퀀스의 부분 시퀀스들로 이루어진 침입 패턴에 대해서도 탐지할 수 있다는 이점을 가진다. 그러나 이 방법은 매크로를 정의하는 부분이나 오토마타를 생성하는 부분이 자동화되어 있지 않고 사용자가 개입해야 하므로 정상 행위를 모델링하기 어려운 단점이 있다.

3. 시스템 구조 및 알고리즘

본 논문에서 제안하는 방법은 그림1과 같은 단계를 거친다.

3.1. 중복 시스템 콜 제거 및 분류

원시 데이터의 형태는 프로세스별로 분리된 시스템 콜 시퀀스이다. 이러한 시스템 콜 시퀀스 중에는 동일한 시퀀스가 존재하므로 우선 동일한 시스템 콜 시퀀스는 하나만 남기고 제거한다. 이 때 trie 자료구조를 사용한다. Trie는 하나의 노드에 하나의 문자가 저장되는 자료구조로서 만일 시스템 콜 시퀀스의 prefix 가 동일한 경우에 같은 노드로 합쳐지게 된다. 따라

서 동일한 시스템 콜 시퀀스는 하나만 남게 된다. 그 후 prefix 에 따라 유사한 프로세스들의 그룹으로 분류한다.

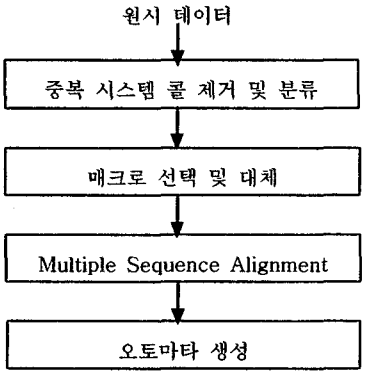

그림 1. 오토마타 생성 과정

3.2. 매크로 선택 및 대체[6]

매크로를 설정하기 위하여 suffix trie 라는 구조체를 사용하였다. suffix trie는 입력 문자열에 존재하는 모든 부분 문자열을 나타내도록 되어 있으며, 루트 노드로부터 각 노드까지의 경로가 각 부분 문자열을 나타낸다. 또한 루트 노드로부터 각 단말 노드까지의 경로는 각 suffix를 나타낸다. 그림 2는 입력 문자열이 ATACATA$인 경우의 suffix trie를 보여준다.

Suffix trie를 통하여 각 프로세스의 시스템 콜 시퀀스에 속하는 모든 부분 시퀀스를 찾을 수 있고, 이러한 부분 시퀀스 각각이 매크로로 선택될 수 있는 가능성을 가지게 된다.

부분 시퀀스 중 가능한 한 적은 매크로로 문자열을 짧게 만들 수 있도록 매크로를 선택한다. 전체 부분 시퀀스 중에서 전체 시스템 콜 시퀀스의 길이를 가장 많이 줄일 수 있는 부분 시퀀스가 첫 번째 매크로가 된다. 그 후에 줄여진 시퀀스에 대하여 위와 같은 매크로 선택 방법을 적용하여 매크로를 선택하게 된다. 이 작업은 더 이상 시퀀스에 길이가 줄여지지 않을 때까지 반복하게 된다.

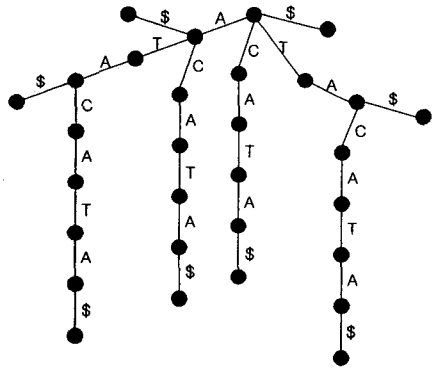

그림 2. 스트링 ATACATA 의 suffix trie.

3.3. Multiple sequence alignment[7]

Multiple sequence alignment 알고리즘은 정렬하고자 하는 스트링을 모두 동일한 것으로 간주하고, 중간에 공백을 추가하거나 동일한 위치의 다른 두 알파벳을 동일한 것으로 간주함으로써 스트링들을 정렬하는 알고리즘이다. Multiple sequence alignment를 수행하기 위해서는 우선 pairwise sequence alignment 과정을 수행해야 한다. 이 알고리즘은 두 스트링의 alignment를 하는 알고리즘이다.

두 개의 sequence의 alignment는 dynamic programming 기법을 이용한 효율적인 알고리즘이 있다[7].

여러 개의 sequence 에 대하여 위의 동일한 알고리즘을 적용하려면 복잡도가 높아지므로 star alignment 라는 heuristic algorithm을 적용한다.

Star alignment algorithm은 여러 개의 문자열에 대하여 하나의 문자열을 기준으로 잡은 후 각각을 pairwise alignment 하였을 경우에 score의 합이 가장 작은 문자열을 center로 잡아서 alignment를 진행한다.

3.4. 오토마타 생성

오토마타는 multiple alignment를 한 결과에서 동일한 위치에서 동일한 문자는 한 상태로 합쳐지고 다른 문자면 다른 상태로 분기되는 방법으로 만들어진다. 여기에 대한 예는 그림 3과 같다.

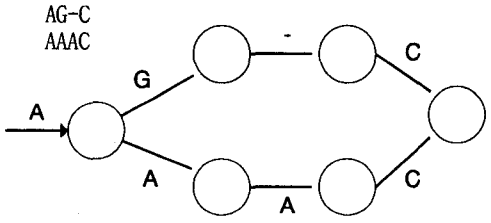

그림 3 오토마타 생성의 예

그림 3에서 문자열 AG-C와 AAAC 의 처음 문자열과 네 번째 문자열이 같으므로 같은 상태로 합쳐지게 된다.

오토마타를 생성한 후 임의의 한 문자열이 오토마타에서 승인되는지의 여부를 검사하는 방법은 다음과 같다. 먼저 검사하려는 문자열의 처음 문자와 오토마타의 시작 상태에서 그 문자열에 해당하는 전이가 있는지 살펴보고 있다면 해당 상태로 전이하면서 +3의 score를 준다. 만일 없다면 '-' 의 문자를 가지는 전이가 있는지 살펴보고 그 상태로 전이한 후 -1의 스코어를 준다. 이 두 경우 모두가 아닌 경우에는 다음 상태에서의 모든 전이를 살펴서 일치하는 전이가 있는지 살펴본다. 그리고 -3의 스코어를 준다. 그래서 승인 상태에 도달할 때까지 검사하려는 문자열의 문자가 남아 있으면 이것은 승인되지 않았다는 것을 의미한다. 그리고 승인상태에 도달하였더라도 threshold를 주어 만일 score가 threshold를 넘어섰을 경우만 승인상태라고 가정한다. 본 논문의 모든 실험에서는 threshold를 0으로 하였다.

4. 실험 및 결과

실험에 사용한 데이터는 UNIX system의 sendmail 과 lpr 에 관한 시스템 콜 시퀀스를 사용하였다. sendmail 데이터를 가지고 한 실험에서는 오토마타를 사용한 정상 행위 모델링이 침입 탐지 시스템에 적용될 수 있다는 가능성을 보였으며, lpr 데이터를 가지고 한 실험에서는 Forrest 가 한 실험과 비교하였다. 표 1은 sendmail 데이터에 대한 실험결과를 보여준다. 실험에서 사용한 모든 데이터는 Forrest의 홈페이지에서 얻었

다[1].

	training에 사용된 프로세스의 수	test	
		악성	정상
실험1	100	0/34	47/47
실험2	120	0/34	27/27
실험3	130	0/34	17/17

표 1. sendmail data 실험결과

표에서 training에 사용된 프로세스의 수는 오토마타를 생성하는데 사용한 정상 시스템 콜 시퀀스의 개수이고, test란의 악성은 악성 행위를 하는 프로세스들이 training에서 만들어진 오토마타에서 승인되는지의 여부를 실험한 결과를 나타내고, 정상은 정상 행위를 하는 프로세스들에 대한 실험 결과를 나타낸다. 이 때, 분모는 실험에 사용한 프로세스의 개수를 나타내고, 분자는 오토마타에서 인식한 숫자를 나타낸다. 악성에 대한 실험에서 승인되지 않았다는 것은 악성을 악성으로 옳게 판단한 경우이다. 정상에 대한 실험에서 승인되지 않았다는 것은 정상 행위인데 악성으로 판단한 경우이므로 false positive를 나타낸다. 위 표에서의 결과는 정상 행의 프로세스를 모두 정상으로 옳게 판단하였다.

표 2,3은 lpr data 에 대하여 동일한 실험을 한 결과이다. (5번 반복한 실험에 대한 평균이다)

	training에 사용된 프로세스의 수	test	
		악성	정상
mit lpr	700	0/1001	1992/1997
unm lpr	700	0/1001	530/532

표 2. lpr data 실험결과 (automata 이용) - threshold 0

	training에 사용된 프로세스의 수	test	
		악성	정상
mit lpr	700	0/1001	1968/1997
unm lpr	700	0/1001	528/532

표 3. lpr data 실험결과 (Forrest의 방법) - threshold 0.2

Forrest가 제안한 방법은 정상 시스템 콜 시퀀스를 슬라이딩 윈도우 기법을 사용하여 특정 길이로 잘라 DB를 작성하게 된다. 이번 실험에서는 그 길이를 10으로 하였다. 그리고 테스트하려는 모든 시퀀스에 대하여 다음 절차에 의한 S_를 구한다.

1. 테스트하려는 시퀀스를 길이 10으로 자른다.
2. 과정 1에서 잘라진 각 시퀀스와 DB에 있는 모든 시퀀스간의 hamming distance를 구한다. 이 때 hamming distance는 두 시퀀스간에 동일한 위치에서 동일한 시스템 콜 시퀀스가 아닌 경우 1씩 더해진다.
3. 각 시퀀스별로 과정 2에서 구한 hamming distance중 가장 작은 값을 찾는다. 이것을 minimum hamming distance라고 칭한다.
4. Minimum hamming distance중 가장 큰 값을 찾는다. 이 값을 슬라이딩 윈도우 사이즈 10으로 나눈 값이 S_ 이다.

만일 S_ 값이 threshold를 넘는다면 비정상이라고 판단하게 된다.

표 2,3을 보면 본 논문에서 제시한 오토마타를 이용한 실험과 Forrest가 제안한 방법 모두 악성 행위를 탐지하는 측면에서는 동일한 결과를 보여준다. 반면에 본 논문에서 제시한 방법이 Forrest가 제시한 방법에 비하여 False positive 가 낮음을 볼 수 있다.

5. 결론 및 향후 연구 방향

현재까지의 실험 결과에서 Forrest가 제안한 방법과 비교할 때 False positive 측면에서 어느 정도 향상을 가져왔다. 그리고 오토마타를 생성하기까지의 전 과정을 자동화할 수 있었다. 그러나 아직까지 다양한 데이터에 대하여 실험을 해야 할 필요성이 있으며, 오토마타에서 승인 여부를 검사할 때 기준이 되는 threshold를 설정하는 부분에 있어서도 더 많은 실험이 필요하다.

6. 참고 문헌

[1] http://www.cs.unm.edu/~immsec/systemcalls.htm
[2] 임영환, "프로세스 행위 파일링을 위한 DFA 자동 생성 기법", 아주대학교, 2001.
[3] S. Hofmeyr, S. Forrest, "Intrusion Detection using Sequence of System Calls", Journal of Computer Security, Vol. 6, pp.151-180, 1998.
[4] A. Kosoresow, "Intrusion Detection via System Call Traces", IEEE Software, Vol. 14, No.5 pp 35-42, 1997.
[5] R. Sekar and M. Bendre, "A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors", Proceeding of the 2001 IEEE Symposium on Security and Privacy, pp.144-155, 2001.
[6] J. Vilo, "Discovering Frequent Patterns from Strings", Department of Computer Science, University of Helsinki, Technical Report C-1998-9, May 1998.
[7] S. Carlos, "Introduction to Computational Molecular Biology", pp.49-80, PWS Publishing Company, 1997.