루트킷 탐지를 위한 리눅스 커널 감시 코드 삽입의 시스템 성능 부하에 관한 연구

문현곤, 허인구, 이진용, 이용제, 백윤흥
서울대학교 전기정보공학부
e-mail : {hgmoon, igheo, jylee, yjlee, ypaek}@sor.snu.ac.kr

A Study on Performance Degradation due to Code Instrumentation of Linux Kernel for Rootkit Detection

Hyungon Moon, Ingoo Heo, Jinyong Lee, Yongje Lee, and Yunheung Paek
Dept. of Electrical and Computer Engineering, Seoul National University

요 약

시스템을 공격하는 악성코드 기술과 그 방어 기술이 발전하면서, 최근의 많은 악성코드들이 운영체제를 직접 변조하는 커널 루트킷을 포함하고 있다. 이에 따라 커널 루트킷에 대한 여러 대응책들이 나오고 있으며, 최근의 많은 연구들이 루트킷 탐지능력 향상을 위해 운영체제 커널에 코드를 삽입하고 있다. 이 논문에서는 앞으로 루트킷 탐지를 위해 커널에 대한 코드 삽입 기술이 지속적으로 사용될 것으로 보고, 이와 같은 코드 삽입이 운영체제 커널이나 전체 시스템의 성능에 어떠한 영향을 주는지를 알아보았다.

1. 서론

커널 루트킷이란 특정 목적을 달성하기 위해 운영체제 커널을 수정하는 소프트웨어 또는 그 방법을 가리키며, 흔히 루트킷이라고 줄여 부르기도 한다. 하지만 악성코드들이 그 목적달성을 위한 수단의 하나로 포함하는 경우가 많기 때문에 일반적으로 루트킷이라고 하면 여러 악성행위를 직접 수행하거나 돕기 위해 커널을 변조하는 악성코드 또는 공격 방법을 뜻하는 경우가 많다.

루트킷을 포함하는 악성코드는 그렇지 않은 악성코드에 비해 탐지하기가 까다롭다. 현재 널리 쓰이고 있는 대부분의 악성코드 대응 솔루션들은 운영체제 위에서 구동되는 소프트웨어의 형태로 구현되어 악성코드 탐지를 수행하는 과정에서 운영체제가 제공하는 시스템 콜 등의 서비스를 사용하고 있는데, 루트킷이 이들을 변호라 수 있기 때문이다. 커널 루트킷이 성공적으로 운영체제를 변조하면, 그 서비스들이 악성코드 제작자의 의도대로 동작하면서 운영체제 위에서 구동되는 솔루션의 수행을 방해하게 된다.

이와 같은 루트킷을 보다 잘 탐지하기 위해서는 루트킷이 시스템을 성공적으로 감염시키더라도 정상적으로 동작할 수 있는 탐지장치의 개발이 필수적이다. 이를 위해 여러 연구팀에서 가상화 기술이나 [1,7], 하드웨어 설계 기술[2,3,5]을 이용한 루트킷 탐지장치를 제안해 왔다.

운영체제 커널에 독립적으로 구현되어 루트킷의 공격으로부터 안전한 외부 모니터들이 처음 소개되었을 때에는 대부분의 연구가 감시대상 시스템의 메모리를 읽어온 후 그 내용을 분석하는 형태를 띠었다. 하지만 이 방식은 과도한 메모리 인터페이스 사용으로 인한 성능저하나, 감시대상 커널의 서비스를 전혀 이용하지 못하고 순수하게 메모리 내용만으로 현재 상태를 복구해내야 하는 시맨틱 갭(semantic gap)이라는 문제를 갖고 있었다.

그러한 문제들을 해결하기 해결하는 한 방법으로 과거와 같이 운영체제 위에서 구동되는 보안 솔루션에서와 같이 커널을 후킹하는 방법이 최근 연구되어 왔다. [4,6] 이렇게 커널의 특정 지점에 코드를 삽입하여 이를 감시에 활용하면, 감시대상 시스템의 현재 상태에 관한 최소한의 정보를 알 수 있기 때문에, 이전에 비해 보다 효율적으로 감시를 수행할 수 있는 것이다.

이 논문에서는 이와 같은 연구의 흐름으로부터 앞으로 가상화 기술이나 하드웨어 설계기술을 이용한 루트킷 탐지장치가 점점 이와 같은 커널 코드 삽입 기술을 많이 활용할 것으로 보고, 그와 같은 탐지장치가 유발할 성능 부하를 예측해보고자 하였다. 이를 위해 우리는 리눅스 커널이 디버깅이나 프로파일링, 응용 최적화 등을 위해 제공하는 커널 트레이싱 시스템을[8] 이용하여 커널에 대한 코드삽입 여부, 빈도 및 개별 코드의 길이 등이 성능에 주는 영향을 알아보고자 하였다. 이에 더하여, 다양한 벤치마크들을 이용해 이러한 성능부하의 구체적 형태를 확인하고, 실제 사용자가 느끼게 될 성능부하의 정도도 예측해보고자 하였다.

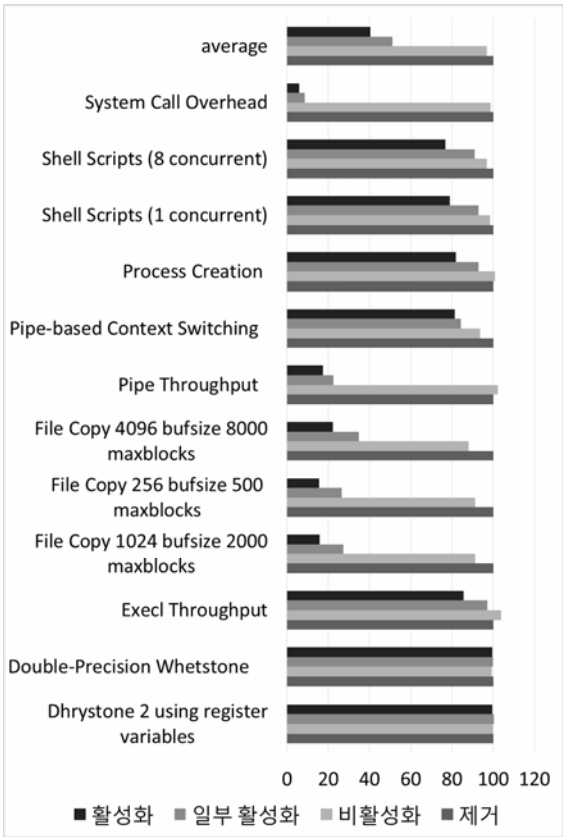

그림 1 유닉스벤치 실험결과

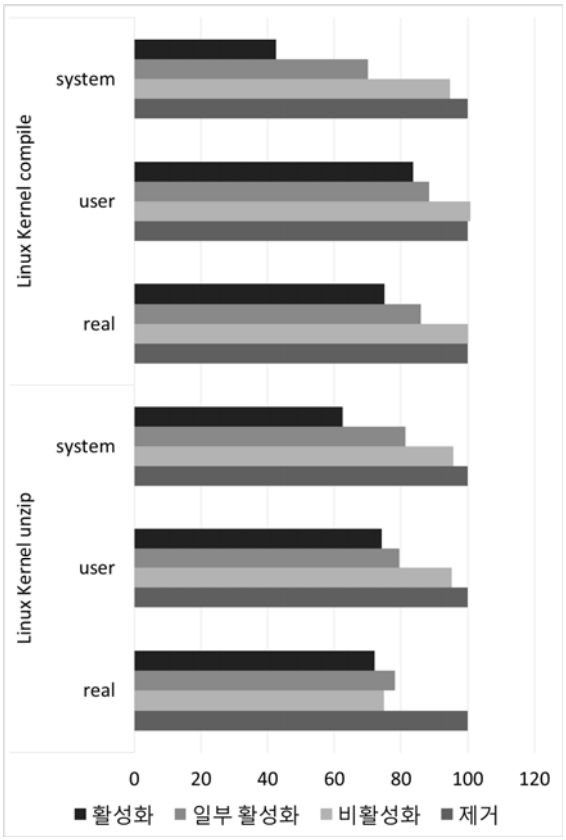

그림 2 커널 압축해제 및 컴파일 실험결과

2. 연구 방법

우리는 감시를 위해 삽입되는 코드가 시스템에 주는 성능부하를 예측하기 위해 서론에서 언급한 커널 트레이싱 시스템 중 트레이스 포인트(tracepoint)의[8] 설정 상태에 따른 시스템의 성능을 측정해 보았다. 트레이스 포인트란 커널의 개발단계에서 디버깅이나 프로파일링을 위한 로그생성이 필요한 부분에 코드를 삽입하는 방법 중 하나로, 커널을 컴파일하는 과정에서 포함 여부를 선택할 수 있으며, 컴파일을 통해 포함시키더라도 활성화시키지 않으면 삽입된 코드의 부하가 극단적으로 줄어드는 특성을 가지고 있다.

코드삽입의 빈도와 삽입된 코드들의 길이 등에 의한 성능영향을 알아보기 위해 서로 트레이스 포인트 설정 상태가 다른 네 가지 상태의 커널을 준비하여 실험을 진행하였다. 첫 번째로, 기준이 되는 성능측정을 위해 컴파일단계에서 트레이스 포인트를 포함하지 않는 커널을 준비하여 성능측정을 진행하였다. 트레이스 포인트를 포함하도록 컴파일한 커널은 트레이스 포인트를 이용한 트레이싱을 모두 끈 상태와 시스템 콜에 관해서만 활성화 시킨 상태, 모두 활성화 상태의 세 상태에서 각각 성능 측정을 진행하여, 삽입된 코드들의 길이나, 삽입되는 빈도가 성능에 주는 영향을 알아보고자 하였다. 트레이스 포인트 없이 컴파일된 커널과 트레이스 포인트를 포함하고 있지만 활성화시키지 않은 상태인 커널의 성능을 비교함으로써 극단적으로 간단한 코드삽입에 의한 성능영향을 예측할 수 있을 것이며, 트레이스 포인트를 전혀 활성화시키지 않은 경우와 모두 활성화시킨 경우를 비교함으로써 삽입된 코드의 길이가 성능부하에 주는 영향을 확인할 수 있었다. 또한 이들과 시스템 콜에 대해서만 트레이스 포인트가 활성화된 커널의 성능을 비교함으로써 코드삽입의 빈도에 따른 성능영향을 확인할 수 있었다.

앞에서 언급한 바와 같이 성능부하의 형태를 확인하고 실제 사용자가 느끼게 될 성능부하의 정도를 확인하기 위해 크게 세가지 벤치마크를 이용해 성능 측정을 진행하였다. 먼저, 유닉스계열 시스템의 성능 측정에 널리 쓰이는 유닉스벤치(UnixBench)를[9] 이용하여 성능부하의 형태를 알아보고자 하였다. 유닉스벤치는 시스템을 구성하는 각 요소의 성능을 측정하기 위한 벤치마크를 모아둔 것으로, 이 연구에서 커널에 삽입된 코드가 시스템의 어느 부분에, 어느 정도 영향을 주는지를 확인하는 데에 도움을 주었다. 하지만 유닉스벤치는 실제로 사용하는 응용프로그램의 구조 등을 잘 고려하고 있지는 않기 때문에, 실 사용시의 성능영향을 알아보기 위해 위와는 별도로 리눅스 커널의 압축해제 및 컴파일 시간을 측정해 보았고, 이에 더하여 파섹(PARSEC) 벤치마크[10,11] 중 일반 사용자용 응용프로그램에 해당하는 x264 인코더, 이미지 처리를 수행하는 빕스(vips), 레이 트레이스(raytrace), 유체물리엔진(fluidanimate)의 네 가지를 이용한 성능 측정 또한 진행하였다.

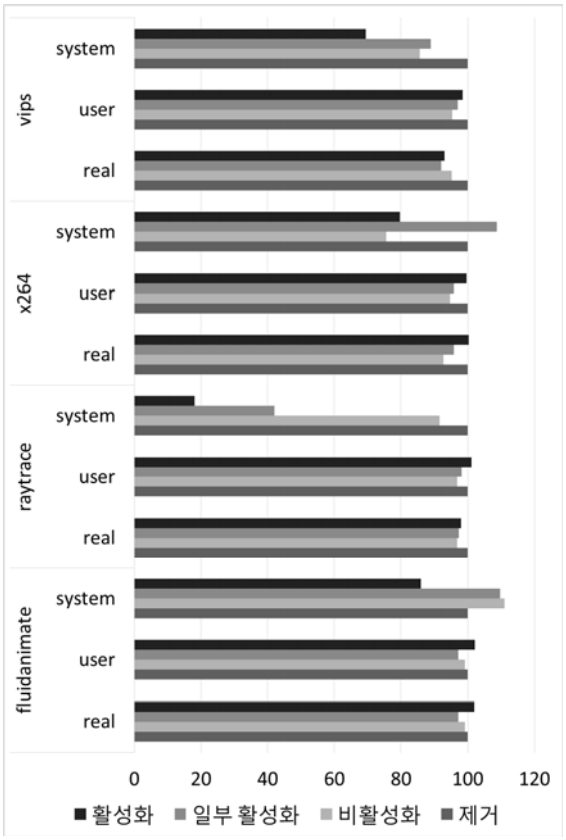

그림 3 파섹 벤치마크 실험 결과

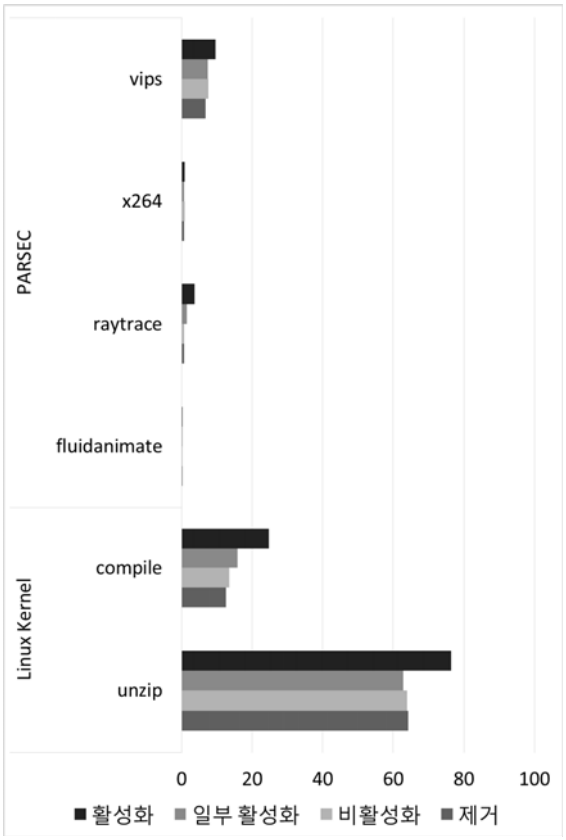

그림 4 벤치마크별 유저모드 실행시간 대비 커널모드 실행시간의 비율

3. 실험 결과

성능 측정은 버추얼 박스(Virtual Box)를 이용해 구동되는 가상 머신을 이용하였으며, 실험 과정에서 성능측정이 진행중인 가상 머신을 제외한 다른 워크로드(workload)가 해당 가상 머신을 구동중인 물리 머신상에서 통제되도록 하여 실험의 정확성을 높이고자 하였다. 실험에 사용된 물리 머신은 2.3GHz 로 동작하는 인텔 제온 E5-2630 프로세서 2 개를 가지고 있으며, 총 12 개의 물리 코어, 24 개의 논리코어를 갖는다. 그 중 4 개의 논리코어를 실험에 쓰인 가상 머신에 할당하였다. 또한 물리 머신이 가진 메모리 192GiB 중 4GiB 을 가상 머신에 할당하였다.

그림 1 은 유닉스벤치를 이용해 트레이스 포인트 상태에 따른 시스템의 성능을 측정한 결과를 나타내고 있다. 가장 아래쪽의 Dhrystone 이나 Whetstone 의 경우, 프로세서의 성능을 측정하기 위한 벤치마크로, 커널 서비스의 영향을 크게 받지 않기 때문에 성능차이가 거의 발생하지 않는 것을 알 수 있다. 나머지 벤치마크들은 운영체제가 제공하는 각 기능의 성능을 측정하기 위한 것으로, 트레이스 포인트 중 시스템콜 관련 부분만을 활성화시켜도 비교적 큰 성능저하가 나타남을 볼 수 있었다. 특히 파일시스템 접근이나 시스템콜 자체의 성능이 크게 나빠짐을 확인할 수 있었다. 이와 같이 유닉스벤치 중 운영체제가 제공하는 서비스의 성능을 평가하는 부분에서는 트레이스 포인트가 모두 활성화 된 경우 최소 15%, 최대 94%가량의 성능 저하가 발생하였다.

그림 2 는 리눅스 커널의 압축해제 및 컴파일에 걸리는 시간을 트레이스 포인트의 상태에 따라 측정하여 비교한 것으로, 유닉스벤치의 출력형식에 맞추기 위해 시간당 처리량의 단위로 변경한 후 그 비율을 그래프로 나타내었다. 두 경우 공통적으로 유저모드의 성능에(user) 비해 커널모드의 성능이(system) 더 감소한 것을 볼 수 있으며, 사용자가 느끼게 되는 실제 수행시간(real)의 경우 20%전후의 성능저하가 발생하였다.

그림 3 은 실제 사용자가 느끼는 시스템의 성능을 측정하기 위해 개발된 파섹 벤치마크 중 네 가지 벤치마크를 이용한 성능측정결과를 나타내고 있으며, 역시 시간당 처리량의 단위로 변경한 후 그 비율을 그래프로 나타내었다. 전반적으로 커널모드의 성능은 20%에서 최대 80%까지 감소하였으나, 유저모드의 성능이나 실제 수행시간은 거의 차이가 없는 것을 볼 수 있다.

4. 논의

첫 번째로 제시된 유닉스벤치 기반의 성능 측정 결과는 커널에 대한 코드 삽입이 커널이 제공하는 개별 서비스의 성능을 크게 감소시킬 수 있음을 보여주고 있다. 비록 트레이스 포인트를 비활성화한 경우 최소 0%, 최대 13%만의 성능저하가 발생하지만, 이 또한 무시할 수 있는 수준은 아니며, 루트킷 탐지를 위한 코드 삽입시 발생할 최소의 성능저하일 것이라는 점 또한 고려해야 할 것이다.

하지만 이들은 개별 기능의 성능영향만을 측정한 것으로, 실제 사용자가 이와 같은 성능저하를 느낄 것이라는 결과는 아니다. 실제로 두 번째와 세 번째 실험 결과는 이를 뒷받침하고 있다. 일반적으로 소프트웨어 개발과정에서 높은 시스템의 성능을 필요로 하는 압축해제 및 컴파일 과정이나, 실제 사용자가 느끼는 시스템의 성능 측정을 목적으로 만들어진 벤치마크에 대해서는 트레이스 포인트를 이용한 코드삽입의 성능저하가 비교적 적었다. 이는 실제 사용되는 응용프로그램의 경우 실행시간 동안 커널 서비스만을 이용하는 것이 아니라 자기자신의 기능 수행을 위한 코드도 수행하기 때문이며, 많은 경우 이 과정에서 실제 커널 서비스 수행에 소모되는 시간의 비율이 매우 적기 때문인 것으로 보인다.

그림 4 는 리눅스 커널 처리 및 파섹 벤치마크 수행과정에서 유저모드 수행시간과 커널모드 수행시간의 비율을 %로 나타낸 것으로, 수치가 높을수록 커널모드 수행시간이 긴 것을 의미한다. 앞에서 예측한 바와 같이 비교적 전체적인 성능 저하가 컸던 리눅스 커널 처리과정이 유저모드 수행 시간에 비해 긴 커널모드 수행 시간을 갖고 있었다. 이는 주로 잦은 파일 시스템 접근에 기인한 것으로 보인다. 반면 성능저하가 비교적 적었던 파섹 벤치마크의 경우 유저모드 수행시간에 비해 커널모드 수행시간이 무시할 수 있을 정도로 적어, 커널모드 수행시간의 증가가 전체적인 수행시간에 큰 영향을 주지 않았다. 비교적 큰 성능 저하를 관찰할 수 있었던 빕스의 경우 비교적 긴 커널모드 수행시간을 갖고 있었다.

5. 결론

우리는 이번 연구를 통해 디버깅 및 성능분석을 목적으로 준비되어 있는 커널 코드 삽입 시스템인 트레이스 포인트를 이용하여 감시목적으로 커널에 코드를 삽입할 경우 발생할 성능 부하를 예측해볼 수 있었다. 유닉스벤치, 리눅스 커널 압축해제 및 컴파일, 파섹 벤치마크 등 여러 종류의 벤치마크를 이용해 성능을 측정한 결과, 커널에 대한 코드삽입을 최소화하지 않을 경우, 커널이 제공하는 개별 서비스들의 성능은 크게 저하될 수 있는 것으로 나타났다. 하지만, 많은 응용프로그램들에서 커널의 성능이 응용프로그램 전체의 성능에 주는 영향이 크지 않기 때문에, 커널에 대한 코드 삽입을 효율적으로 진행할 경우, 응용프로그램의 성능에 큰 영향을 주지 않으면서도 원하는 감시기능을 수행하는 것이 가능할 것으로 생각된다.

참고문헌

[1] O.S. Hofmann, A.M. Dunn, S. Kim, I. Roy, E. Witchel, Ensuring operating system kernel integrity with OSck, in: ACM SIGPLAN Notices, ACM, 2011, pp. 279-290.

[2] H. Lee, H. Moon, D. Jang, K. Kim, J. Lee, Y. Paek, B.B. Kang, KI-Mon: A Hardware-assisted Event-triggered Monitoring Platform for Mutable Kernel Object}, in: Presented as part of the 22nd USENIX Security Symposium}, USENIX}, 2013, pp. 511-526.

[3] H. Moon, H. Lee, J. Lee, K. Kim, Y. Paek, B.B. Kang, Vigilare: toward snoop-based kernel integrity monitor, in: Proceedings of the 2012 ACM conference on Computer and communications security, ACM, 2012, pp. 28-37.

[4] B.D. Payne, M. Carbone, M. Sharif, W. Lee, Lares: An architecture for secure active monitoring using virtualization, in: Security and Privacy, 2008. SP 2008. IEEE Symposium on, IEEE, 2008, pp. 233-247.

[5] N.L. Petroni Jr, T. Fraser, J. Molina, W.A. Arbaugh, Copilot-a Coprocessor-based Kernel Runtime Integrity Monitor, in: USENIX Security Symposium, San Diego, USA, 2004, pp. 179-194.

[6] M.I. Sharif, W. Lee, W. Cui, A. Lanzi, Secure in-vm monitoring using hardware virtualization, in: Proceedings of the 16th ACM conference on Computer and communications security, ACM, 2009, pp. 477-487.

[7] Z. Wang, X. Jiang, W. Cui, P. Ning, Countering kernel rootkits with lightweight hook protection, in: Proceedings of the 16th ACM conference on Computer and communications security, ACM, 2009, pp. 545-554.

[8] M. Desnoyers, Using the Linux Kernel Tracepoints, Linux Kernel Documentation, https://www.kernel.org/doc/Documentation/trace/tracepoints.txt

[9] UnixBench, https://code.google.com/p/byte-unixbench/

[10] C. Bienia, K. Li, Benchmarking modern multiprocessors, Princeton University USA, 2011.

[11] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, A. Gupta, The SPLASH-2 programs: Characterization and methodological considerations, in: ACM SIGARCH Computer Architecture News, ACM, 1995, pp. 24-36.

Acknowledgement

본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(과제번호 2012-0000470), 중소기업청에서 지원하는 2012 년도 산학연공동기술개발사업(No. C0019562), 미래창조과학부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신) [No. 10047212, 1kB 이하 암호문 간의 연산을 지원하는 동형 암호 원천 기술 개발 및 응용 연구] 및 IDEC 의 지원을 받아 수행하였습니다..