임베디드 리눅스에서 서명 검증 방식을 이용한 악성 프로그램 차단 시스템

[Preventing ELF(Executable and Linking Format)-File-Infecting Malware using Signature Verification for Embedded Linux]

이 종 석 † 정 기 영 ††
(JongSeok Lee) (Ki Young Jung)
정다니엘 †† 김 태 형 ††
(Daniel Jung) (Tae-Hyung Kim)
김 유 나 † 김 종 †††
(Yuna Kim) (Jong Kim)

요 약 오늘날 모바일 기기들의 발전과 통신망의 고속화, 광역화와 함께 사용자의 중요한 정보를 유출하거나 특정 기기의 사용을 방해하는 보안 위협도 점점 증가하고 있다. 모바일 기기에서 널리 사용될 것이라 예상되는 임베디드 리눅스 또한 이러한 보안 위협으로부터 안전하지 못하다. 본 논문에서는 임베디드 리눅스를 위협하는 악성 프로그램의 특징에 대해 알아보고 그에 대한 대응책으로 임베디드 시스템의 특성을 고려한 서명 검증 방식을 이용한 악성 프로그램 차단 시스템을 제안한다. 제안하는 시스템은 악성 프로그램 검사 엔진 서버와 LSM 기반의 커널 모듈로 구현된 시스템으로 구성되며, 메모리에 상주하여 악성 프로그램을 감시하는 일반적인 실시간 감시 프로그램과는 달리, 커널 레벨에서 프로그램이 실행되는 순간 파일의 변조 여부를 검사하여 악성 프로그램의 실행을 사전 차단한다. 실험을 통해 제안한 시스템이 적은 오버헤드로 악성 프로그램의 실행을 효과적으로 사전 차단하는 것을 확인하였다.

키워드 : 임베디드 리눅스, 임베디드 보안, 악성프로그램, 바이러스, 시스템 보안

Abstract These days, as a side effect of the growth of the mobile devices, malwares for the mobile devices also tend to increase and become more dangerous. Because embedded Linux is one of the advanced OSes on mobile devices, a solution to preventing malwares from infecting and destroying embedded Linux will be needed. We present a scheme using signature verification for embedded Linux that prevents executable-Infecting malwares. The proposed scheme works under collaboration between mobile devices and a server. Malware detection is delegated to the server. In a mobile device, only integrity of all executables and dynamic libraries is checked at kernel level every time by kernel modules using LSM hooks just prior to loading of executables and dynamic libraries. All procedures in the mobile devices are performed only at kernel level. In experiments with a mobile embedded device, we confirmed that the scheme is able to prevent all executable-Infecting malwares while minimizing damage caused by execution of malwares or infected files, power consumption and performance overheads caused by malware check routines.

Key words : Embedded Linux, Embedded System Security, Malware, Virus, System Security

1. 서 론

최근에 모바일 임베디드 시스템에 적용하고자 하는 연구가 활발히 진행되고 있다. 하지만 이렇게 발달하는 환경에 따른 부작용으로 기존의 PC 및 서버 등에 존재하던 보안 위협들이 플랫폼을 넘어서 모바일 임베디드 시스템으로 쉽게 전이될 수 있다는 우려가 나오고 있다. 이렇게 전이된 보안 위협들은 사용자의 중요한 정보를 유출하거나 특정 기기의 사용을 방해하는 행위를 할 수가 있다. 또한 통신-금융 시스템의 컨버전스를 통해 모바일 기기에 전자상거래 기능이 추가됨에 따라 악성프로그램에 적절한 대응을 하지 못할 경우 단순한 피해를 넘어선 막대한 경제적 손실이 발생할 수 있다.

따라서 임베디드 리눅스 환경을 보호할 수 있는 보안 프로그램의 개발이 반드시 필요한 상황이지만, 현재까지의 진행되어 온 연구는 상대적으로 미흡하다. 그 이유는 현재 리눅스는 윈도우에 비해 상대적으로 일반 사용자

· 이 논문은 제34회 추계학술대회에서 '임베디드 리눅스에서 서명 검증 방식을 이용한 악성 프로그램 차단 시스템'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 포항공과대학교 컴퓨터공학과
spicajk@postech.ac.kr
(Corresponding author임)
existion@postech.ac.kr
†† 비 회 원 : 포항공과대학교 컴퓨터공학과
zzangkv@postech.ac.kr
dennis@postech.ac.kr
kth@postech.ac.kr
††† 정 회 원 : 포항공과대학교 컴퓨터공학과 교수
jkim@postech.ac.kr

논문접수 : 2007년 12월 14일
심사완료 : 2008년 5월 28일

정보과학회논문지 : 컴퓨팅의 실제 및 레터 제14권 제6호(2008.8)

층이 적고, 루트 권한과 사용자 권한으로 나뉘어져 있어 악성 프로그램이 침투하더라도 루트 권한을 얻지 않는 한 시스템에 큰 피해를 입힐 수 없다는 점에서 찾을 수 있다. 하지만 임베디드용으로 제작된 리눅스 시스템은 제한된 성능과 자원을 가지고 있고 유지보수가 신속하게 이루어지기 힘들어 약한 공격에 노출되더라도 지속적으로 안정적인 사용 환경을 보장할 수 없으며, 리눅스 상에서 동작하는 악성 프로그램의 수가 매년 증가하고 있다는 점을 볼 때, 임베디드 리눅스를 보호해줄 수 있는 보안 프로그램이 반드시 필요하다.

그래서 본 논문에서는 임베디드 리눅스 시스템의 특성을 고려하여 보안 프로그램으로 인해 발생하는 오버헤드를 최소한으로 줄이면서 악성 프로그램에 효과적으로 대응하기 위해, 서명 검증 방식을 이용한 악성 프로그램 차단 시스템을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 연구 동기를 살펴보고 연구 목표를 설정한다. 3장에서 제안하는 시스템에 대해 소개한다. 4장에서는 제안 시스템을 적용하여 실험한 결과를 분석한다. 5장에서는 본 논문과 관련된 연구들을 살펴보고 6장에서 결론 및 향후 연구 방향을 제시한다.

2. 연구 동기 및 목표

리눅스 환경에서 존재하던 기존의 악성 프로그램들은 주로 ELF(Executable and Linking Format) 형식의 바이너리 파일을 변형시켜 악성 코드를 삽입, 실행하려 한다. 이러한 방법은 다른 파일을 감염시켜 자가 복제를 계속 발생시키는 고전적인 형태의 바이러스 외에도 웜, 트로이 목마, 루트킷 등 다양한 유형의 공격에서 활용된다.

이러한 악성프로그램들은 임베디드 리눅스가 PC 및 서버에서 주로 사용되어 온 범용 리눅스와 프로그램 실행을 위해 거의 동일한 환경을 제공한다는 특징을 이용해 모바일 임베디드 시스템에 그대로 전이될 수가 있다. 따라서 임베디드 리눅스를 공격하는 악성 프로그램들이 증가하는 상황을 대비해 기존에 존재하던 악성 프로그램들을 참고하여 임베디드 리눅스에 대한 공격을 막기 위한 방법을 강구해야 한다.

그런데 기존에 존재하는 다른 모바일 운영체제를 위한 보안 소프트웨어들[2-7]과 같이 시스템 내부에 악성 프로그램 탐지를 위한 엔진을 내장할 경우, 수많은 악성 프로그램의 정보를 저장하고 검사하기 위해 필요한 시스템 자원이 막대하여 한정된 자원을 가지고 있는 모바일 임베디드 시스템의 가용성을 저하시킬 수 있다. 그리고 시그니처 기반의 탐지 방법의 한계로 인해 제로데이(zero-day) 공격과 다양한 변형 공격에 약점을 보여 임베디드 시스템의 안정성과 가용성을 완벽히 보장하기가 어렵다. 약한 공격에도 큰 피해를 입을 수 있는 임베디드 시스템의 환경을 고려할 때, 이런 문제점은 반드시 해결이 되어야 한다. 또한 안정성과 가용성을 보장하기 위해서는 시스템이 안전하게 유지가 되는지 실시간으로 확인해야 하나, 기존 어플리케이션 기반의 방법들은 하나의 독립적인 프로세스로서 메모리에 상주해야 하기 때문에 임베디드 시스템의 자원을 일정부분 지속적으로 소비하는 문제점이 생긴다. 따라서 기존의 PC 및 서버 등의 환경에서 사용되었던 보안 시스템보다 임베디드 시스템의 자원을 효율적으로 사용하는 방법이 필요하다.

한편 각 시스템에서 사용되는 임베디드 리눅스가 범용 리눅스와는 다르게 특정한 목적을 위한 하드웨어에 최적화된다는 점을 감안하여 기존 시스템의 변경을 필요로 하지 않으면서 쉽고 빠르게 시스템에 적용/해제가 가능한 방법을 개발하여야 한다.

따라서 본 논문에서는 임베디드 시스템에 쉽게 적용이 가능하며, 기존의 방법들보다 오버헤드를 줄일 수 있는, ELF 바이너리 파일 대상 악성 프로그램 실시간 감시 방법으로 커널 레벨에서 서명 검증 기법을 이용한 악성 프로그램 차단 시스템을 제안한다.

3. 제안하는 시스템

3.1 개요

그림 1은 제안하는 시스템의 구조를 표현하고 있다. 메모리에 상주하여 악성 프로그램을 검색하는 일반적인 실시간 감시 프로그램과는 달리, 제안하는 시스템은 ELF 바이너리 파일이 실행 명령 후 커널 레벨에서 파일이 실행되는 순간 그 파일에 대해 검사를 한다.

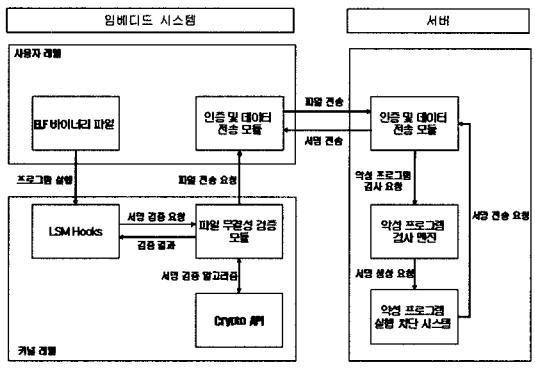

그림 1 제안 시스템 구조

이 때 프로그램을 실행할 때마다 어떤 악성 프로그램에 감염되었는지 자세히 검사를 하는 것은 임베디드 시스템에 부담을 주게 되므로, 3장에서 설명한 서명 검증

기법을 사용하여 실행하고자 하는 바이너리 파일의 변조 여부를 먼저 확인한 뒤, 파일이 변경되었거나 아직 서명을 생성하지 않았을 경우에만 악성 프로그램 검사를 하도록 한다. 악성 프로그램 검사는 임베디드 단말에서 하지 않고, 별도의 서버로 파일을 전송하여 수행한다. 이와 같은 방법을 사용하는 이유는 임베디드 시스템 내부에 악성 프로그램 검사 엔진을 둘 경우 전체 악성 프로그램 목록을 검사하는 과정을 수행하는 데 필요한 자원을 임베디드 시스템이 감당하기 힘들고, 악성 프로그램 엔진에서 사용하는 데이타베이스를 유지하고 갱신하는 것 또한 시스템 자원을 필요로 하기 때문이다.

서버는 임베디드 시스템으로부터 전송받은 악성 프로그램을 검사하여 안전한 파일로 판정될 경우, 검사한 파일에 대한 서명을 생성하여 임베디드 시스템으로 전송하며, 임베디드 시스템은 이것을 받아 바이너리 파일의 끝에 첨가한다. 이후 해당 파일의 실행 시에는 시스템에서 파일의 서명 검증 과정을 통해 감염이 안 된 안전한 파일로 판단하고 호출을 계속 수행한다. 그러나 서명 검증 과정이 실패한 경우에는 서명 생성 후 파일에 대한 변형이 발생한 것이므로 다시 서버로 보내어 파일이 안전하다는 검증 과정을 거쳐야 한다.

3.2 프로그램 실행 후킹

악성 프로그램의 실행을 사전에 차단하기 위해서는 언제, 어떻게 프로그램이 실행이 되는지 알아야만 한다. 그러나 리눅스에서 프로그램들이 실행되어 시스템에서 활동하게 되는 경로는 매우 다양하므로 사용자 단계에서 프로그램의 실행을 사전 차단하는 방법은, 실행되는 모든 프로그램을 검사하는 것이 불가능하기 때문에 적절하지 않다. 이와 같은 이유로 인해 우리는 커널 내부에서 프로그램이 실행되는 경로를 후킹하는 방법을 채택하였다. 리눅스 환경에서 모든 프로그램은 결국 시스템 콜을 통하여 사용자 레벨에서 커널 레벨로 내려와 메모리 영역을 할당받게 되므로, 메모리에 적재되기 직전에 우리가 원하는 작업을 수행하도록 커널 내부에 후킹 함수를 만들면 언제, 어떠한 경로로 실행되더라도 검사 과정을 우회할 수 없게 된다.

이러한 방법을 통해 동적 링크가 되는 공유 라이브러리를 포함한 모든 바이너리 파일의 실행을 검사할 수 있다. 이 때 후킹을 하기 위해서는 커널 내부 구조를 변경해야 하는데, 본 논문에서는 3장에서 소개한 LSM을 이용하는 방법을 제안하므로 기존에 사용하던 임베디드 리눅스의 커널을 재컴파일할 필요 없이 바로 적용할 수 있다.

그림 2은 LSM을 통해 교체할 수 있는 후킹 함수들 중 프로그램 실행과 관련된 단계별 함수들을 보여준다. 본 논문에서는 이 중에서 do_mmap() 함수 다음에 수행되는 file_mmap() 함수를 변경하여 프로그램이 사용하는 파일이 메모리에 적재되어 활성화되기 직전에 그 파일의 무결성을 검사하는 방법을 사용한다. 이를 통해 바이너리 실행 파일 형태로 존재하는 바이러스나 웜 등의 악성 프로그램 외에도 기존 프로그램의 취약점을 이용하여 바이너리 실행 파일이나 동적 라이브러리 파일의 내용을 변경하는 코드 삽입 공격(Code Injection Attack), 루트킷 실행 등을 막을 수 있어 시스템을 안전하게 보호할 수 있다.

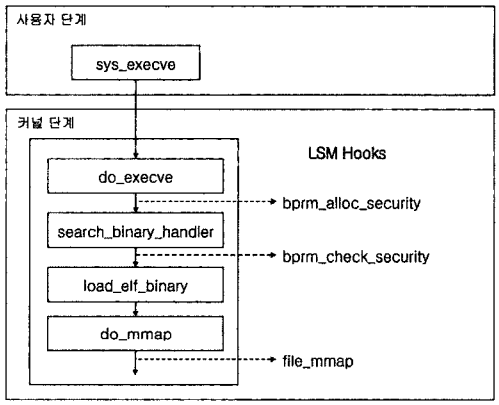

그림 2 프로그램 실행 단계별 LSM 후킹 함수

3.3 파일 무결성 검증

LSM 후킹 함수를 사용하여 프로그램의 실행 과정을 후킹하게 되면 파일의 무결성 검증을 위해 서명 검증을 수행한다. 본 논문에서는 서명 검증에 사용되는 알고리즘으로 HMAC-MD5를 선택하였다[13]. 단순히 MD5 알고리즘만으로 서명을 생성한다면 공격자가 감염된 파일에 MD5로 생성한 서명을 붙여서 정상적으로 서명 인증이 된 파일인 것처럼 배포할 수가 있다. 반면 HMAC-MD5는 각 임베디드 시스템별로 서버와 미리 공유한 비밀키를 이용하여 서명 검증을 하기 때문에 공유키를 모르는 경우에는 위조된 서명을 만들어 낼 수 없으므로 서버와 임베디드 시스템의 공유키가 노출되지 않는 한 서명 위조를 방지한다.

3.4 인증 및 키분배

서버와 임베디드 시스템은 파일 무결성 검증을 위해 사용되는 비밀키를 생성하기 위해 상호 인증 및 키분배 과정을 거친다. 본 논문에서는 이를 위해 ISAKMP (Internet Security Association & Key Management Protocol)[14] 프로토콜을 사용한다. 이것을 통해 서버와 임베디드 시스템이 서로의 정보를 확인하고 인증을 한 뒤 SA(Security Association)를 설정하고 키를 교환한다. ISAKMP는 정해진 시간마다 자동으로 SA와 키를 갱신하므로 암호화 프로토콜의 특성상 발생할 수 있는

암호분석학(Crypto Analysis)적 공격을 막을 수 있다. 또한 메시지 전송 시 난수값을 포함하기 때문에 재전송 공격(Reply Attack)을 방지할 수 있다. ISAKMP 프로토콜은 인증을 하는 방법에 다양한 기술들을 적용할 수 있으므로 일반적으로 많이 사용되는 공인인증서를 사용하여 인증을 수행하는 방법 외에 OTP(One Time Passward)와 같은 강한 인증(Strong Authentication) 기술을 사용할 수가 있다.

4. 실험 및 결과

4.1 실험 환경

실험에 사용된 임베디드 시스템은 ARM 코어 기반의 Intel xScale PXA270 프로세서를 사용하였으며 128MB SDRAM, 32MB NOR Flash Memory을 포함하였다. 운영체제는 2.6.18 버전 리눅스 커널을 포팅하여 시스템에 적용하였다. 서버로 사용한 시스템은 Intel Pentium 4 1.4MHz에 512MB SDRAM의 성능을 가졌으며 악성 프로그램 검사를 위한 엔진으로는 오픈 소스로 공개되어 있는 ClamAV 백신 프로그램[15]의 엔진을 추출하여 사용하였다. 실험을 위해 사용된 ELF 바이너리 파일은 busybox로 크기는 1.5MB이다.

4.2 실험 결과

먼저 제안하는 방법이 효과적으로 ELF 바이너리 파일의 변조를 방지하는지 확인하기 위해 ELF 바이너리 파일을 변조하는 공격들을 수행하였다. 실험 결과, ELF 바이너리 실행 파일을 감염시키는 바이러스들은 물론, 공유 라이브러리를 변조하여 백도어를 설치하는 루트킷 또한 차단하는 것을 확인할 수 있었다.

또한 제안하는 방법을 임베디드 시스템에 적용하였을 때 발생하는 오버헤드를 분석하기 위해 두가지 경우를 설정하여 각각 상황에 따른 오버헤드의 차이를 비교하였다.

표 1 프로그램 수행 시간 비교

	/bin/busybox ls-al	/bin/busybox pwd	tar jxvfp linux-2.6.18.tar.bz2
일반 커널	real 0.03s user 0.01s sys 0.02s	real 0.00s user 0.00s sys 0.01s	Real 2m59.34s User 2m48.15s Sys 0m6.88s
제안하는 방법을 적용한 커널	real 0.59s user 0.02s sys 0.51s	real 0.23s user 0.00s sys 0.18s	Real 3m04.57s User 2m50.19s Sys 0m7.12s

첫 번째 실험은 실행하고자 하는 파일이 정상적인 서명을 가진 것으로 검증이 되는 경우로, 프로그램이 실행을 시작하여 종료를 할 때까지 걸리는 수행 시간을 제안하는 방법을 시스템에 적용하기 전과 적용한 뒤로 나눠 비교하였다. 수행 시간을 비교할 때 사용한 ELF 파일 실행 명령은 /bin/busybox ls와 /bin/busybox pwd, tar jxvfp linux-2.6.18.tar.bz2이다. 표 1은 일반 커널과 제안하는 방법을 적용한 커널에서 측정한 수행 시간 결과이다. 표 1에서 real은 프로그램의 전체 수행 시간을 나타내고 user와 sys는 각각 사용자 레벨에서의 수행 시간과 커널 레벨에서의 수행시간을 나타낸다.

실험 결과 일반 커널에 비해 제안하는 방법을 적용한 커널에서의 수행 시간이 증가하는 것을 확인할 수 있다. 그러나 위의 예제 중 /bin/busybox ls -al와 /bin/busybox pwd가 사용자 레벨에서의 수행 시간은 거의 없고 커널 레벨에서의 수행 시간도 매우 짧은 프로그램들이라 성능 평가에 있어서 가장 나쁜 조건(Worst Case)이라는 점과 실험에 참여한 사용자들이 느끼기엔 수행 속도의 차이가 체감적으로 거의 없을 정도라는 사실에 주목하자. 일반적으로 사용자가 오래 사용하는, 즉 수행 시간이 긴 프로그램들은 tar jxvfp linux-2.6.18.tar.bz2에서 볼 수 있듯이 사용자 단계에서 대부분의 작업을 수행하기 때문에, 제안하는 방법은 처음에 프로그램을 실행시키는 과정 중에 커널 레벨에서 단 한 번만 수행되므로 전체적인 시스템 성능에 큰 지장을 주지 않는다. 또한 기존의 리눅스 시스템에서 사용되는 ELF 바이너리 파일의 크기도 대부분 512KB 이하라는 점을 감안할 때 임베디드 시스템에서 사용되는 바이너리 파일의 크기 증가에 따른 수행 시간 증가도 크게 우려하지 않아도 될 것이다[7].

표 2 악성프로그램 검사 수행 시간 비교

	전체 수행 시간 (second)	
악성프로그램 샘플 수	제안하는 방법	기존 방법
100	3.071	3.248
500	3.055	3.354
1000	2.985	3.521
3000	3.148	4.112
5000	3.179	4.771
10000	3.381	6.294
30000	3.977	12.366
50000	4.610	17.887

두 번째는 실행하고자 하는 파일이 서버로부터 서명을 받지 못했거나 파일의 내용이 변경되어 서명과 일치하지 않아 악성프로그램에 의해 감염이 되었는지를 판단하기 위한 작업을 수행하게 되는 경우로, 제안하는 방법과 기존의 보안프로그램들처럼 시스템 내부에서 검사하는 방법의 검사 수행 시간을 비교하였다. 검사의 대상이 되는 ELF 파일은 busybox를 선정하였고 데이타베이스에 저장된 악성프로그램의 샘플 수는 최소 100개부터 시작하여 최대 50000개까지 변경하면서 측정하였다.

표 2는 제안하는 방법과 기존의 백신 엔진이 내장된 방법의 수행 시간을 측정한 결과이다.

표 2에서 확인할 수 있듯이, 기존의 방법보다 제안하는 방법이 데이타베이스의 악성프로그램 샘플 수가 증가하면 증가할수록 더 좋은 성능을 낸다.

따라서 제안하는 방법이 적은 오버헤드로 시스템을 효과적으로 보호할 수 있음을 확인할 수 있다.

5. 관련 연구

Milenković 등은 프로세서에 서명 검증 알고리즘을 삽입한 칩을 추가하여 프로세서가 프로그램 코드를 실행할 때 실시간으로 프로그램의 무결성을 검사하는 방법을 제안하였고[16], Apvrille 등은 리눅스 커널 단계에서 RSA 서명 검증 기법을 사용한 실시간 바이너리 파일 인증 방법을 제안하였다[7]. 그러나 이러한 방법들은 프로세서에 하드웨어를 추가해야 하거나 모든 바이너리 파일에 대한 서명을 미리 만들어야 하는 문제가 있어서 기존에 존재하는 임베디드 시스템에 적용할 수 없다. 또한 현재 시스템이 안전한 상태에 있다는 가정을 바탕으로 하고 있기 때문에, 제안된 방법들을 적용하기 전에 시스템이 공격당했다면 이에 적절히 대응할 수 없다.

6. 결론 및 향후 연구

본 논문에서는 임베디드 시스템의 특성을 고려한 임베디드 리눅스 기반의 서명 검증 방식을 이용한 악성 프로그램 차단 시스템을 제안하였다. 제안하는 시스템은 악성 프로그램 검사 엔진 서버와 LSM 기반의 커널 모듈로 구현된 시스템으로 구성되며, 메모리에 상주하여 악성 프로그램을 감시하는 일반적인 실시간 감시 프로그램과는 달리, 커널 레벨에서 프로그램이 실행되는 순간 파일의 변조 여부를 검사하여 악성 프로그램의 실행을 사전 차단한다. 실험을 통해 제안한 시스템이 적은 오버헤드로 악성 프로그램의 실행을 효과적으로 사전 차단하는 것을 확인하였다.

제안한 방법은 ELF 바이너리 파일의 변조 여부로 공격 여부를 판단하기 때문에, 인증을 받은 정상적인 프로그램 내부에 존재하는 취약점을 통해 직접 쉘 코드를 실행시키는 형태의 공격을 방어하지 못한다. 또한 제안하는 방법을 실제 모바일 환경에서 적용할 경우, 공격자가 임의의 악성프로그램 파일들을 이동 단말 기기에 대량으로 배포하여, 임베디드 시스템들로 하여금 동 시간대에 패킷 전송량을 비정상적으로 증가시켜 서버와 임베디드 시스템 사이의 네트워크를 공격하는 일종의 DoS(Denial of Service) 공격을 시도할 수 있다.

향후 연구는 이러한 문제점들을 해결하기 위해 다양한 정상 프로그램들의 취약점을 통해 공격을 받더라도 시스템을 안전하게 보호할 수 있는 메모리 보호 기법과, 기존에 많이 연구되었던 모바일 환경에서의 DoS 공격 차단 기법들을 본 논문에서 제안한 방법에 적용시키는 방향으로 진행할 것이다.

참 고 문 헌

[1] Axelle Apvrille, David Gordon, Serge Hallyn, Makan Pourzandi, Vincent Roy, "DigSig: Run-time Authentication of Binaries at Kernel Level," Proceedings of the 18th USENIX conference on System administration (LISA 2004), 2004.

[2] Symantec AntiVirus for Handhelds, Available at http://www.symantec.com/

[3] Kaspersky AntiVirus Mobile, Available at http://www.kaspersky.com/trials?chapter=171229147

[4] Trend Micro Miblie Security, Available at trend-micro.com/mobilesecurity

[5] F-Secure Mobile Anti-Virus, Available at http://mobile.f-secure.com/

[6] McAfee Mobile Security, Available at http://www.mcafee.com/us/enterprise/products/mobile_security/index.html

[7] V3 Mobile, Available at http://www.ahnlab.com/

[8] 금융결제원, 공인인증서비스, Available at http://www.yessign.or.kr/service/certi_about.php

[9] Peter Loscocco, Stephen Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," In Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01), June 2001.

[10] Winfried Trumper, "Summary about POSIX.1e," http://wt.xpilot.org/publications/posix.1e, July 1999.

[11] Serge Hallyn, Phil Kearns, "Domain and Type Enforcement for Linux," In Proceedings of the 4th Annual Linux Showcase and Conference, October 2000.

[12] Linux Security-Module (LSM) Framework, Available at http://lsm.immunix.org.

[13] H. Krawczyk, M. Bellare, R. Canetti, "RFC 2104 - HMAC: Keyed-Hashing for Message Authentication," 1997.

[14] D. Maughan, M. Schertler, M. Schneider, J. Turner, "RFC 2408 - Internet Security Association and Key Management Protocol (ISAKMP)," 1998.

[15] ClamAV, Available at http://www.clamav.net/

[16] Milena Milenković, Aleksandar Milenković, Emil Jovanov, "Hardware Support for Code Integrity in Embedded Processors," Proceedings of the 2005 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2005), 2005.