Journal of The Korea Society of Computer and Information
Vol. 19, No. 12, December 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.12.123

부트로더와 물리적으로 독립된 저장장치를 이용한 모바일 운영체제 무결성 검증

박 재 경*, 이 상 훈**, 권 미 영**, 김 효 남***

A Mobile OS Integrity Verification Using Bootloader & Physically Independent Storage Device

Jae-Kyung Park*, Sang-Hun Lee**, Mi-Young Kwon**, Hyo-Nam Kim***

요 약

본 논문에서는 보안의 문제가 발생 할 경우 앱보다 치명적일 수 있는 운영체제 무결성에 대한 검증 기법에 대해 연구하였다.최근 스마트폰의 보급은 급속도로 발전하고 있으며, PC와 유사한 서비스를 제공하는 단계까지 왔다. 그리고 그에 따르는 보안 위협도 PC와 유사한 부분이 많다. 최근의 모바일 네트워크 환경에서 단말기의 루트 권한 탈취 및 운영체제 위변조 등에 의한 피해가 날로 늘어나고 있는 추세이며 이를 종합적으로 대응할 수 있는 시스템에 대한 연구가 필요하다. 모바일 위협에 대한 피해를 막기 위한 방법으로 운영체제의 무결성을 검사하는 방법이 사용되기도 한다. 무결성 검증 알고리즘은 운영체제 부팅 이전 레벨에서의 검증과 제어권이 운영체제로 넘어간 시점에서의 검증으로 구분될 수 있다. 무결성 검증은 앱 무결성과 운영체제 무결성으로 나뉘어 볼 수 있는데, 본 논문에서는 부트로더와 단말기 내부의 물리적으로 독립된 저장장치를 이용하여 운영체제의 무결성을 검증하는 기법을 제안하였다.

▸ Keywords : 안드로이드, 무결성, 커널, 무결성 검증, 부트로더

Abstract

In this paper, we study the verification techniques for OS integrity that can be more fatal than applications in case of security issues. The dissemination of smartphones is rapidly progressing and there are many similarities of smartphones and PCs in terms of security risks. Recently, in mobile network

• 제1저자 : 박재경, • 교신저자 : 권미영, 이상훈
• 투고일 : 2014. 12. 2, 심사일 : 2014. 12. 6, 게재확정일 : 2014. 12. 26.
* 한국과학기술원 사이버보안연구센터(KAIST Cyber Security Research Center)
** 국방과학연구소 (Agency for Defense Development)
*** 청강문화산업대학교 게임전공 (Dept. of Computer Game, ChungKang College of Culture Industries)
※ 본 연구는 국방과학연구소에서 수행 중인 "사이버 공격에 대비한 군용 다기능 모바일 단말기의 보안 SW 개발" 과제의 지원으로 수행되었습니다.

environment, there is a trend of increasing damages and now, there are active researches on a system that can comprehensively respond to this. As a way to prevent these risks, integrity checking method on operation system is being researched. As most integrity checking algorithms are classified by verification from the levels before booting the OS and at the time of passing on the control to the OS, in which, there are minor differences in the definitions of integrity checking or its methods. In this paper, we suggests the integrity verification technique of OS using a boot loader and a physically independent storing device in the mobile device.

▸ Keywords : Android, Integrity, Kernel, Integrity Verification, Boot Loader

I. 서 론

모바일 단말은 시대의 흐름에 따라 다양하게 변화되고 있으며, 그 방향성을 살펴보면 어플리케이션이나 콘텐츠의 경우 폐쇄적인 구조에서 개방형으로 바뀌고 있으며, 모바일 운영체제 역시 이와 비슷하게 전용 운영체제에서 범용운영체제로 바뀌게 되었다[8]. 선 하드웨어적 측면에서 살펴본다면, 단순 이동전화 서비스를 위한 모듈만을 장착하던 때에서 RF, Wi-Fi, 스크린, 대용량 저장장치, 고성능의 CPU 등과 같이 PC에서 사용되는 정도의 고사양 스펙을 탑재하는 시대로 변화되어왔다. 이러한 변화에 따라 모바일 단말의 보안위협이 급격히 높아지고 있다[5]. 개방형 플랫폼, 앱스토어를 통한 애플리케이션 유통, 다양한 네트워크 접속 환경의 지원, 이동편의성 및 모바일 오피스의 지원 등, 달리 생각해보면, PC의 보안위협을 뛰어 넘고 있다고 해도 과언이 아닐 정도이다. 각 계각층에서 이러한 모바일 보안위협을 대응하기 위한 연구와 노력이 이루어지고 있지만, 다방면에서 이루어지는 악의적인 위협으로부터 완전하게 보호하고 있지 못하는 실정이다. 최근 널리 보급되는 오픈 플랫폼과 같은 운영체제의 경우 어플리케이션의 해킹보다 그 위험성이 매우 높다[9,10]. 운영체제의 커널영역은 해커에 의해서 여러 가지 위협에 노출되게 된다. 프로세스 제어, 레지스트리 제어, 네트워크 제어, 키보드 제어 등 을 그 예로 들 수 있다.

본 논문에서는 이러한 오픈형 모바일 운영체제의 위협으로부터 보다 안전한 모바일 컴퓨팅을 보장하기 위해 단말기 내부의 독립된 저장장치를 이용하는 운영체제 무결성 검증 기법을 제안하며 이를 부트로더와 단말기 내부의 물리적으로 독립된 저장장치를 이용하여 운영체제의 무결성 검증 기법을 제안하였다.

II. 관련 연구

무결성은 대상 정보가 반드시 허가/인증된 대상에게만 개방되고, 또 그 대상에 의해서만 수정이 될 수 있음을 보장하는 것이다. 이러한 무결성을 보장하기 위해 취해지는 조치들로는 물리적 환경에 대한 통제, 데이터 액세스 제한, 그리고 엄격한 인증 절차의 유지 등이 해당한다. 다음은 최근 연구되고 있는 운영체제의 무결성에 대한 연구를 살펴보기로 한다.

2.1 삼성전자 KNOX

최근 삼성전자는 KNOX라는 명칭을 사용하는 모바일 보안 솔루션을 상용화하였다. 해당 솔루션은 Linux 커널과 Android 운영 체제를 통해 변조를 방지하는 디바이스 하드웨어 기반의 다각적 보안 솔루션이다. 해당 솔루션은 Android 운영 체제의 무결성 검증을 포함한 앱 및 정보 보호, 앱 별 VPN 연결을 통한 네트워크 인프라를 통한 위협 대응방안, BYOD 추세에 따른 개인과 기업 업무 데이터의 안정적인 관리 방안, 단말기 도난 관리 등을 포함하는 광범위한 모바일 보안 플랫폼이라고 할 수 있다[1].

FOSDEM 2014에서 발표된 Samsung KNOX의 무결성 검증 기법을 살펴보면 운영체제의 무결성 검증과, 블록레벨의 무결성 검증기법을 제시하고 있다. 특히 Secure Boot, Trusted Boot 등으로 이루어지는 부트 레벨에서의 운영체제 무결성 검증 기법을 살펴볼 필요가 있다. Trusted Boot의 경우 ARM TrustZone 기술을 사용하여, 모든 부트로더와

운영체제 커널의 암호화된 핑거프린트를 저장하고, 이 정보를 이용하여 무결성 검증을 실시하고 있다[7].

대부분의 Android 디바이스에서 Android 부트 로더는 디바이스 커널의 진위를 확인하지 않는다. 디바이스에 대한 제어 권한을 강화하려는 사용자라면 디바이스를 루팅하는 해킹된 커널을 설치할 수 있다. 해킹 커널은 모든 데이터 파일과 앱 및 리소스에 대해 슈퍼유저 액세스 권한을 제공한다[2,3]. 해킹된 커널이 손상되면 서비스 거부가 발생할 수 있고, 커널에 악성 소프트웨어가 포함되어 있으면 기업 데이터의 보안에 위해 요소가 발생한다. 이를 방지하기 위해 KNOX는 Secure Boot를 개발하였고 이는 부팅 과정에서 인증되지 않은 부트 로더와 커널이 로드되지 못하도록 하는 보안 메커니즘을 제공한다. Secure Boot는 KNOX Workspace를 사용하는 디바이스에서 악성 공격을 차단하는 일차 방어선의 역할을 한다.

다음 그림 X와 같이 KNOX는 체계적인 보안 검사를 통해 디바이스에서 올바른 커널만 사용한다. 하드웨어 수준에서 1차 부트 로더는 PKI 인증서를 검사하여 2차 부트 로더 1의 무결성을 확인한다. 마찬가지로 2차 부트 로더 1은 1차 부트 로더 2의 무결성을 확인하고, 2차 부트 로더 2는 Android 부트 로더의 무결성을 확인한다. Android 부트 로더는 Root of Trust로 삼성 인증서를 가진 삼성 인증 커널만 로드하는 구조로 개발되었다.

그림 1. KNOX의 부팅 과정
Fig. 1. Booting Process of KNOX

다만, 이러한 KONX의 단점으로는 플랫폼이 매우 무겁고 효율성이 떨어진다는 점이며 KONX에서 수행되는 앱이 매우 제한적이라는 점에서 매우 융통성이 떨어지는 솔루션이라고 할 수 있으며 이스라엘 네게브 벤 구리온 대학 사이버 보안 연구원에 의하면, 삼성의 보안 소프트에어에 심각한 취약성이 발견되었다고 한다[5].

2.2 KI_MON

KAIST에서 개발한 KI-Mon은 하드웨어 기반의 외장형 커널 무결성 모니터 시스템이다. 이벤트 발생 메커니즘 방식에 따라 커널의 객체를 모니터링하며 커널의 무결성을 위한 탐지 및 검증을 수행한다[4]. KI-Mon은 모니터링 하는 객체에서 발생되는 수정에 대해 메모리 주소나 값을 생성할 수 있다. 이벤트 생성은 불필요한 소프트웨어는 배제를 하는 화이트리스트 기반으로 만들어진다. 기존의 커널 모니터링 방식은 모니터와 커널이 같은 공간에 설치되어 매번 발생하는 커널의 수정에 대해서 모두 반영하기에는 성능적인 문제나 효율적인 문제가 매우 떨어졌다. 추가적으로 KI-Mon API를 통해 프로그래밍이 가능하며 모니터링 규칙도 수정이 가능하다.

다음 그림 2와 같이 KI-Mon의 아키텍처처럼 별도의 하드웨어에 FPGA 기반의 보드를 통해 독립적인 커널 무결성 모니터링이 가능하며 CPU의 부하를 줄일 수 있는 방식의 VTMU를 사용하여 메모리 수정을 감지할 수 있는 구조이다.

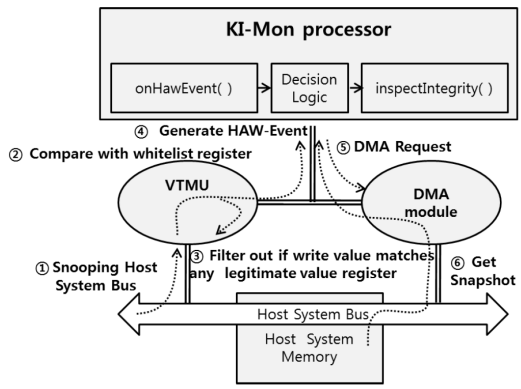

그림 2. KI-Mon 아키텍처
Fig. 2. Architecture of KI-Mon

다만, KI-Mon은 별도의 하드웨어를 장착하여야하며 이는 호스트의 특성에 따라 제약이 발생할 수 있다. 디바이스 드라이버 제작이나 인터페이스가 일치해야한다는 단점을 가질 수 있다. 따라서 본 논문에서는 이러한 단점을 보완하기 위해 별도의 하드웨어 장치를 사용하지 않고 별도의 하드웨어 메모리 공간을 통한 무결성 검증을 연구하였다.

III. 본 론

본 논문에서 제안하고자 하는 모바일 운영체제 무결성 검증 기법에 대해서 설명한다. 본 기법은 무결성 검증 데이터가 적재 되는 독립적인 보안 메모리 영역을 확보를 하고, 해당 보안 메모리 영역으로의 접근이 가능한 인터페이스를 커널 모듈로 제작한다. 해당 모듈은 부트로더에서 운영체제 부팅과 함께 커널로 적재하게 되며, 이 모듈을 이용하는 무결성 검증

어플리케이션을 통해 해당 데이터로 접근을 하게 된다.

3.1 독립적인 보안 메모리영역 확보

본 논문에서 제안한 보안 메모리영역은 기존 모바일 기기에 존재하지 않는 새로운 저장매체를 뜻한다. 해당 저장매체는 기기의 다른 부품들과 같이 디바이스 드라이버가 있어야만 운영체제상에서의 접근이 가능하다. 기본적으로 PC에 장착되는 모든 주변장치들은 각각의 장치를 제어하는 디바이스 드라이버를 가지고 있다. 이는 실제 해당 장치를 제어하는 인터페이스를 정형화하여, 사용자들이 공통적인 과정을 통해 해당 장치를 사용할 수 있게 하는 방법이다[11]. 운영체제는 이렇게 디바이스 드라이버로 정의된 I/O 인터페이스를 통해 사용자의 요청에 따라 실제 디바이스에게 I/O를 수행하게 된다. 이런 디바이스 드라이버들은 운영체제에 포함이 되어있는 상태로 배포되기도 하며, 일부는 모듈이라고 하는 하나의 오브젝트 형태로 별도 관리 되어 운영체제와 별도로 관리되기도 한다. 모바일 단말기 역시 PC와 같은 방식을 사용하고 있으며 디바이스 드라이버의 존재 역시 동일하다.

본 논문에서 제시하는 독립적인 보안 메모리 영역은 그림 3과 같이 디바이스 드라이버를 직접 제작하여 해당 영역으로의 접근을 제어하도록 설계된다. 해당 메모리 영역으로의 접근은 부트레벨과 운영체제레벨에서 모듈 Read 접근이 가능하도록 되어있다. 그러나 Write 기능의 경우 부트로더 영역에서만 가능하도록 설계가 된다. 이는 악의적인 사용자에 의한 운영체제변조 등이 일어날 경우, 운영체제의 제어권을 탈취 당하게 되며, 준비된 보안 메모리영역으로 접근이 가능하게 되므로 Read 기능은 제공하되, 혹시 모를 해당 영역의 데이터 위변조를 사전에 방어하기 위하여 Write 기능을 제거한 것이다.

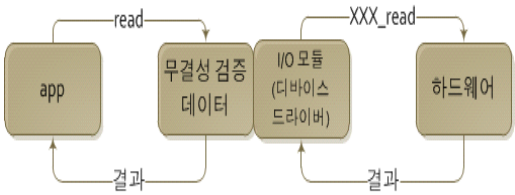

그림 3. 무결성 검증 데이터 접근
Fig. 2. Access of Integrity Verification Data

그리고 한 단계 더 나아가 디바이스 드라이버에는 입출력 시 저장된 데이터로의 접근을 2차적으로 관리하기 위하여 암복호화 기능을 추가 한다. 그리고 해당 드라이버는 커널 모듈 형태로 미리 제작을 하여 커널과 별도로 관리를 하며 부트 레벨에서 커널 이미지를 로드시킨 후 해당 드라이버를 커널에 적재 하도록 한다.

3.2 부트로더를 활용한 무결성 검증

일반적인 부트로더는 과거 LILO에서부터 현재 상용 Linux 운영체제에서 사용되고 있는 GRUB가 있다. 그리고 임베디드 장비에서 주로 사용이 되고 있는 u-boot 부트로더가 있다. 부트로더는 운영체제 부팅 이전에 H/W의 이상 유무 체크 및 운영할 운영체제를 RAM영역에 적재하는 역할을 하게 된다. 각각의 부트로더들은 정형화된 로직들이 존재하며 순차적으로 장치의 상태를 점검하고 실제 운영에 사용될 운영체제를 동작시키도록 하는 역할을 한다. 따라서 해당 영역을 악용하여 변형된 운영체제로 부팅이 되는 행위가 일어날 수 있다. 실제로 모바일 단말의 보안상 가장 큰 문제로 지적되고 있는 루팅을 하는 행위 역시 단순히 운영체제 취약점을 이용하는 방법이 아니라 이 부트영역의 수정으로부터 시작되는 경우도 많이 발생 되고 있다[6]. 그에 따라 모바일 단말기 제조사 및 기타 연구진들은 앞서 언급한 KNOX와 같은 솔루션을 적용하여 안전한 운영체제만을 선별하여 올릴 수 있도록 여러 가지 방안을 활용하여 부트레벨의 처리를 하고 있다.

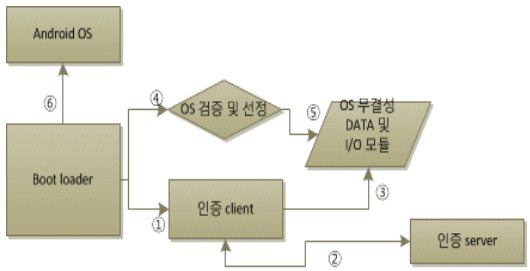

그림 4. 부트레벨에서의 무결성 검증
Fig. 4. Integrity Verification of Boot Level

본 논문에서 제안하는 방법은 그림 4와 같이 부트로더는 메인운영체제로 진입하기 전에 초기화 과정을 진행하고 CF 메모리나 HDD와 같은 보조기억장치에서 시스템에 사용 될 운영체제를 찾아서 부팅을 시켜주는 작업만을 하는 것이 아니라, 개발자가 직접 프로그래밍을 하여 부가적인 기능들을 추가 할 수 있게 되어있다. 그 예로 네트워크 통신을 통해 외부에서 운영체제를 다운로드 받아와서 메인메모리에 적재를 하거나 특정 디바이스 장치를 컨트롤하는 행위들을 할 수 있다. 이런 점을 착안을 하여 앞서 준비한 보안메모리 영역으로의 접근을 수행하여 운영체제 무결성 인증을 시도 할 수 있다. 다음 그림5는 임베디드 Linux와 안드로이드 시스템의 구조

적인 차이점을 나타내고 있다.

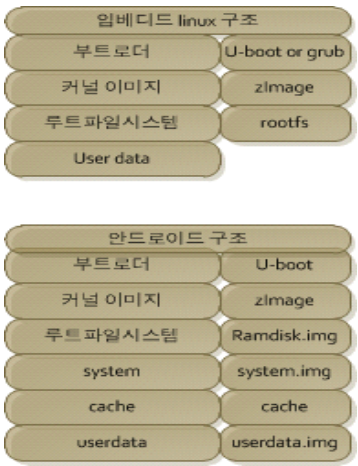

그림 5. 임베디드 linux와 안드로이드 시스템
Fig. 5. Embedded Linux vs Android System

본 논문에서 제안한 운영체제의 무결성 검증 과정을 제시한다. 최초 단말기는 제조 시점에 해당 메모리 영역에 인증된 운영체제 이미지의 무결성 검증 데이터를 기록한 상태로 배포가 되며 다음과 같은 순서에 의해 검증이 진행된다.

- ①②③ 단계 : 단말기 부팅 시 순차적으로 부팅과정을 진행 한 후 보안 메모리영역의 운영체제 무결성 검증 데이터는 운영체제 위변조 검사 단계에서 해당 단말 제조사와의 통신을 통해 데이터를 갱신 할 수 있도록 인터페이스 및 프로세싱을 진행
- ④ 단계 : 운영체제 위변조 여부를 확인하기 위한 검증 기능을 실행
- ⑤ 단계 : 위변조 검증을 위하여, 보안 메모리 영역의 데이터와 실제 운영체제의 값이 일치하는지 판단
- ⑥ 단계: 검증이 완료된 경우 해당 운영체제를 메모리에 적재함과 함께 보안 메모리 영역에 포함되어있는 I/O 모듈을 운영체제가 사용할 파일 시스템으로 복사를 진행하며, 제어권을 운영체제로 전달

안드로이드의 운영체제는 커널 이미지와 램디스크(ramdisk.img)로 분리할 수 있는데 다시 이 램디스크를 해지(unpackaging)하게 되면 실제 파일 시스템을 확인 할 수 있다.

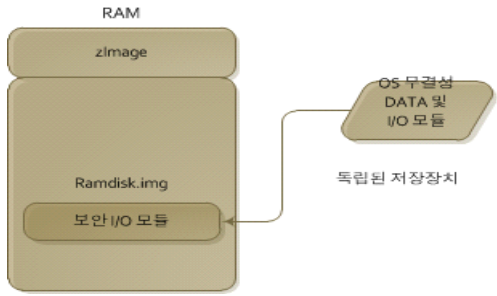

그림 6. 보안 I/O 모듈 운영체제와 병합 과정
Fig. 6. Secure I/O Module OS & Merge Process

이를 부트로더에서 메모리로 적재를 하게 되는 과정에 I/O 모듈을 파일시스템의 modules/ 위치로 복사를 하여, 운영체제 부팅 이후에 관련 어플리케이션에서 보안 메모리 영역으로의 접근을 가능하도록 한다.

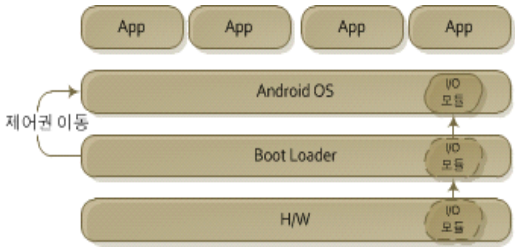

그림 7. 보안 I/O 모듈 포지션 변화
Fig. 7. Position Change of Secure I/O Module

위와 같은 과정을 통해서 부트로더가 관리하던 보안메모리 영역에 대한 커널 I/O 모듈을 안드로이드 운영체제로 적재를 시키고 제어권이 넘어가게 된다.

3.3 운영체제레벨에서의 무결성 검증

앞 절에서 언급한 I/O 모듈을 이용한 운영체제의 운영 상태에서 무결성 검증 단계이다. 해당 모듈은 보안 저장장치에 대한 디바이스 드라이버이며, 해당 드라이버의 I/O 기능상에는 write 기능이 배재되어 있다. 즉 open, read, close 기능만을 구현하여 운영체제 상에서는 해당 데이터의 접근만 가능하며 별도의 수정작업은 불가능하도록 설계하였다. 이는 악의적인 주변의 공격으로부터 운영체제 무결성 검증 데이터의 위변조를 방어하는 것이 목적이다.

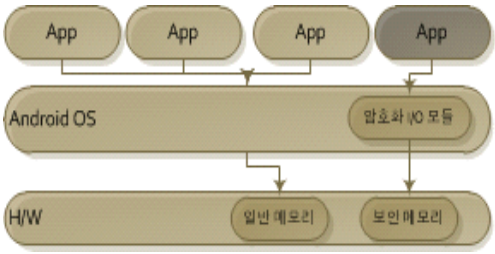

그림 8. 운영체제 운영중 무결성 검증
Fig. 8. Integrity Verify in OS Operating

운영체제에 모듈 형태로 insert된 디바이스 드라이버를 통해서 신뢰 할 수 있는 무결성 검사앱이 주기적으로 운영체제의 시스템 영역의 무결성 유지 여부를 체크한다. 이때 무결성 체크를 하는 앱의 신뢰도가 매우 중요하게 된다.

3.4 검증을 위한 프로토타입 모델 제작

3.3절에서 언급한 '부트로더와 물리적으로 독립된 저장장치를 이용한 모바일 운영체제 무결성 검증'을 구현하기 위해서는 하드웨어적인 부분에서 제약 조건이 발생한다. 따라서 본 논문에서는 이러한 제약조건을 탈피하기 위하여, OS 부팅 이전의 과정을 형식화하여 필요한 요소들을 리눅스 어플리케이션으로 별도 제작하여 검증 고찰을 실시하였다.

우선 물리적으로 독립된 저장장치 부분을 리눅스에서 제공하는 가상메모리를 활용한다. 이는 블록디바이스 드라이버를 제작하고 가상으로 하나의 저장장치를 제작하여 마운트하는 방식을 취하게 된다.

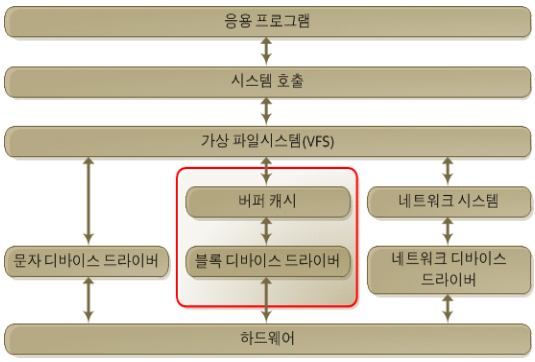

그림 9. Linux 시스템 구조
Fig. 9. Linux System Architecture

그림 9는 프로토타입을 제작하기 위한 리눅스 시스템의 구조상에서의 블록 디바이스의 위치를 보여주고 있다. 프로토타입은 무결성 값을 저장하기 위하여 독립 저장 장치인 메모리의 일부분을 보안 메모리 영역으로 파티션을 설정하는 가상 램 디스크를 사용한다. 가상 램 디스크란 메모리를 하드디스크 드라이브인 것처럼 설정하는 것으로 메모리에 파일을 저장할 수 있으며 블록 디바이스 드라이버를 통해서 생성된 가상 램 디스크에 무결성 값을 저장하여 커널 및 app의 위. 변조를 확인할 수 있다. 프로토타입은 최초 단말기 부팅 시 순차적으로 부팅 과정을 진행 한 후 무결성 검증 데이터를 해당 단말 제조사와의 통신을 통해 데이터를 갱신해야 하지만 제조사와의 협약을 맺어야 하는 문제점이 있어 프로토타입에서는 무결성 검증 데이터를 암호화 어플리케이션에서 갱신하는 것으로 대체하였다.

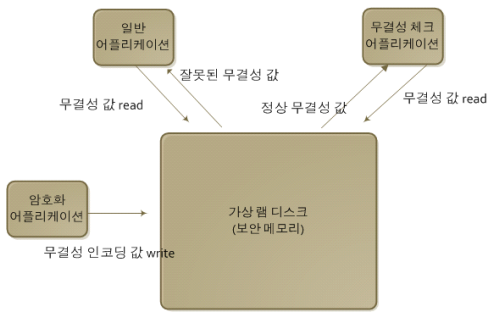

그림 10. 프로토타입 모델 구성도
Fig. 10. Prototype Model Diagram

그림 10은 프로토타입 모델 구성 도를 설명한 것을 보여주는 것으로 가상 램 디스크와 어플리케이션으로 구성되어 있다. 암호화 어플리케이션은 위. 변조를 탐지 할 일반 파일이나 바이너리 파일의 해시 값을 추출하여 해당 값을 이미 생성되어 있는 가상 램 디스크에 저장하여 추후 무결성 체크 어플리케이션에서 위. 변조를 탐지 할 수 있게 한다. 일반 어플리케이션은 무결성 체크 어플리케이션 외에 다른 어플리케이션을 지칭한 것으로 가상 램 디스크에 접근하는 인증되지 않은 어플리케이션을 말하며, 무결성 체크 어플리케이션은 실시간으로 가상 램 디스크 안에 암호화 어플리케이션에서 저장한 무결성 값을 읽어 현재 무결성 체크 할 파일의 해시 값과 비교하여 위. 변조를 탐지 한다.

3.4.1 블록 디바이스 드라이버를 통한 가상 램 디스크 생성 및 등록

가상 램 디스크를 생성하기 위해서 다음 그림 11의 함수를 사용하여 블록 디바이스를 등록 및 해제한다.

```
int register_blkdev(unsigned int major, const char *name)
int unregister_blkdev(unsigned int major, const char *name)
```

그림 11. 블록 디바이스 등록 / 해제 함수
Fig. 11. Block Device Register / Unregister

각 함수의 파라미터 major 번호는 블록장치 혹은 문자장치의 그룹에 할당되는 번호이며 Major 번호가 주어진 경우에는 이것을 사용하여 빈자리가 있는지를 확인하게 되지만, 그렇지 않은 경우는 가장 큰 번호에서부터 빈자리를 검사해서 블록 디바이스 드라이버를 설치할 자리를 찾으며 블록 디바이스의 이름과 드라이버에서 제공하는 블록 디바이스 연산에 대한 포인터를 적용한다 .name은 생성 할 가상 램 디스크 파일명을 말한다. 다음 그림 12는 응용 프로그램에서 가상 램 디스크를 제어할 수 있도록 정의하는 구조체이다.

```
struct file_operations xxx_fops = {
        .owner  = THIS_MODULE,
        .open   = secure_open,
        .release = secure_release,
        .read   = secure_read,
        .ioctl  = secure_read
                };
```

그림 12. file_operations 구조체
Fig. 12. file_operations structure

file_operations 구조체는 디바이스 드라이버와 응용 프로그램을 연결하는 구조체로 각각의 멤버 변수를 다음과 같다.

- owner : 파일 오퍼레이션의 소유자를 나타내며 디바이스 드라이버의 사용 횟수를 커널에서 관리해야 하기 때문에 필드 지정
- open : 응용 프로그램에서 디바이스를 처음 사용하는 경우를 처리하는 함수를 지정
- release : 응용 프로그램이 디바이스를 더 이상 사용하지 않아서 닫기를 구현하는 함수를 지정
- read : 디바이스 드라이버의 읽기를 구현하는 함수를 지정
- ioctl : read와 write로 구현하기 곤란한 디바이스 드라이버의 입/출력 처리를 구현하는 함수를 지정

그 외에 여러 가지의 멤버 변수를 지정할 수 있지만 필요한 변수만 지정하여 다른 기능의 제한 및 보안 등급을 높였다. 가상 램 디스크 생성 및 블록 디바이스 드라이버가 등록되면 응용 프로그램에서 가상 램 디스크에 저장되어 있는 값을 읽어 무결성을 체크할 수 있다. 다음 그림 13은 가상 램 디스크 및 블록 디바이스에 대한 코드를 나타내고 있다.

```
int secure_read()
{
    [해당 app에 무결성 값 전달]
}
int secure_ioctl(){
  [무결성 값 write]
  return 0;
}
int secure_open(){
  return 0;
}
int secure_release(){
  return 0;
}
/* file_operations 구조체 등록*/
struct file_operations xxx_fops={
  .owner  = THIS_MODULE,
  .open   = secure_open,
  .release=secure_release,
  .read   = secure_read,
  .ioctl  = secure_ioctl,
};
int device_init(void)
{
  [register_blkdev() 함수 호출하여 등록]
  [가상 메모리 할당]
}
void device_exit(void)
{
  [unregister_blkdev()함수 호출하여 해제]
  [가상 메모리 해제]
}
module_init(device_init)
moduel_exit(device_exit);
MODULE_LICENSE("GPL");
```

그림 13. 가상 램 디스크 및 블록 디바이스 코드
Fig. 13. Virtual RAM disk & Block Device Code

무결성 값을 저장하기 위하여 device_init함수를 통해 블록 디바이스를 생성 및 등록하게 되며 무결성 값이 저장될 수 있는 가상 메모리를 할당한다. 무결성 값을 저장할 수 있는 공간이 만들어 지게 되면 암호화 어플리케이션 및 무결성 체크 어플리케이션이 등록된 file_operation 구조체에 정의되어 있는 함수들을 통해 가상 램 디스크에 접근 및 연산을 할 수 있다. secure_read 함수는 인증 된 어플리케이션에서 값을 읽을 시 무결성 값을 전달하며 secure_ioctl 함수는 암호화 어플리케이션에서 무결성 값을 가상 램 디스크에 저장하는 함수이다.

3.4.2 암호화 어플리케이션, 무결성 체크 어플리케이션

```
int data_integrity_insert()
{
  if( 가상 램 디스크 open){
    [ 무결성 체크할 파일 해시 함수를 통한 값 도출 ]
    [ 해시 함수를 통하여 도출된 값 가상 램 디스크에 저장 ]
}
```

그림 14. 암호화 어플리케이션 코드 알고리즘
Fig 14. Algorithm of Encryption App code

암호화 어플리케이션은 최초 부팅 시 가상 램 디스크에 체크할 파일을 해시 함수를 통하여 값을 도출하고 도출된 해당 값을 생성된 가상 램 디스크에 저장하여 추후 무결성 체크 어플리케이션에서 사용할 수 있게 한다.

```
int data_integrity_check()
{
  if( 가상 램 디스크 open){
    [ read() 함수를 통한 무결성 값 읽어 옴 ]
    [ 무결성 값과 매칭하여 위. 변조 확인 ]
  }
```

그림 15. 무결성 체크 어플리케이션 코드 알고리즘
Fig. 15. Algorithm of Integrity Verification App code

무결성 체크 어플리케이션은 이벤트 발생 및 설정된 시간에 무결성을 체크한다. 무결성 체크를 하게 되면 가상 램 디스크를 open 함수를 통해서 접속한 후 read 함수를 통하여 무결성 값을 읽어 온다. 읽어 온 값과 체크 할 파일들의 해시 값을 비교하여 위. 변조가 이루어졌는지 검사한다.

IV. 검증 및 고찰

본 실험은 프로토타입 모델을 기반으로 1개의 커널 모듈과 3개의 어플리케이션으로 구성되어 실험을 진행하였다. 부트레벨에서의 실험은 한계가 있으므로 프로토타입으로 개발한 코드를 향후 부트레벨로 적용하면 동일한 결과를 얻을 수 있으므로 본 논문의 실험은 확인이 가능한 응용 레벨에서 진행하였다. 본 논문의 실험 단계는 다음과 같다.

- 가상 램 디스크 생성
- 암호화 어플리케이션을 통한 무결성 값 저장
- 무결성 체크 어플리케이션에서 블록 디바이스 인터페이스를 통한 가상 램 디스크 접근 및 무결성 값 도출
- 일반 어플리케이션에서 블록 디바이스 인터페이스를 통한 가상 램 디스크 접근 및 무결성 값 도출

다음 그림 16은 무결성 값을 저장하기 위한 가상의 램 디스크를 먼저 생성하여 무결성 값을 저장하기 위한 공간으로 활용하였다.

```
[root@csrc imsi]#
[root@csrc imsi]# mknod /dev/secure_memory b 250 0
[root@csrc imsi]# insmod virtual_ramdisk.ko
[root@csrc imsi]# mkfs -t ext2 /dev/secure_memory
[root@csrc imsi]# mkdir /mnt/secure_memory
[root@csrc imsi]# mount -t ext2 /dev/secure_memory /mnt/secure_memory/
```

그림 16. 가상 램 디스크 생성
Fig. 16. Creation of Virtual RAM Disk

블록 디바이스 노드를 생성하기 위해 mknod 명령어를 사용하여 /dev 디렉터리 밑에 major 번호 250, minor 번호 0 인 secure_memory 특수 파일을 만든다. major 번호는 다른 디바이스 드라이버와 중복되지 않도록 임의의 번호를 지정하였다. 가상 램 디스크를 생성하기 위해 블록 디바이스를 등록 및 해제하는 드라이버를 커널 모듈로 구현 하였으며 insmod 명령어를 통해 모듈을 실행중인 커널에 링크하였으며 생성된 가상 램 디스크는 mkfs 명령어를 통하여 ext2 파일시스템으로 포맷하였다. 마지막으로 mount 방식을 사용하여 무결성 값을 저장할 수 있도록 하였다.

```
[root@csrc imsi]# ./data_integrity_insert virtual_ramdisk.ko
프로세스 명 : data_integrity_insert
MD5 변환 된 값 : 99cd556342f0a1edc49df7cc30089764
[root@csrc imsi]#
[root@csrc imsi]#
```

그림 17. 암호화 어플리케이션 실행
Fig. 17. Execution of Encrypt App

data_integrity_insert는 암호화 어플리케이션으로 그림 17에서와 같이 무결성 값을 가상 램 디스크에 저장하는 프로세스이다. 해당 프로세스는 무결성을 체크 할 파일 리스트를 가지고 있으며 실행 시 해시 함수를 사용하여 리스트에 속해 있는 파일들의 해시 값을 도출 한 후 해시 값은 가상 램 디스크에 저장한다.

```
[root@csrc imsi]# ./data_integrity_check
프로세서 명 : data_integrity_check
MD5 변환 된 값 : 99cd556342f0a1edc49df7cc30089764
[root@csrc imsi]#
[root@csrc imsi]#
```

그림 18. 무결성 검사 어플리케이션 실행
Fig. 18. Execution of Integrity Verify App

data_integrity_check는 무결성 검사 어플리케이션으로 무결성을 검사하는 실행 파일이다. 해당 프로세스는 무결성을 검사하기 위하여 그림 18과 같이 가상 램 디스크에 접근하여 무결성 값을 읽어온다. 무결성 검사를 위한 값을 비교하기 위하여 무결성 검사 파일 리스트를 가지고 있으며 암호화 어플리케이션과 동일 한 해시 함수를 사용하여 해시 값 을 도출한다.

```
[babisss@csrc imsi]$
[babisss@csrc imsi]$ ./data_integrity_check
프로세서 명 : data_integrity_check
MD5 변환 된 값 : 99cd556342f0a1edc49df7cc30089764
결과 : 무결성 값이 일치합니다
[babisss@csrc imsi]$
```

```
[babisss@csrc imsi]$
[babisss@csrc imsi]$ ./data_integrity_check
프로세서 명 : data_integrity_check
MD5 변환 된 값 : 99cd556342f0a1edc49df7cc30089764
결과 : 무결성 값이 일치하지 않습니다
[babisss@csrc imsi]$
```

그림 19. 무결성 검사 결과
Fig. 19. Integrity Test Result

그림 19에서와 같이 도출된 해시 값을 가상 램 디스크에서 가지고 온 값과 비교하여 악의적인 행위에 의해 파일이 변조되었는지를 검사한다. 만약 악성코드나 루팅 등을 통해 파일이나 기타 저장 정보가 변경되었을 경우에는 무결성 검사를 통해서 확인이 가능하며 이때 모든 기능을 정지시키고 대기상태로 변경하는 것이 가능하다.

모바일 사용자는 자신의 기기가 대기 상태에 빠졌을 경우 이를 복구하기 위한 절차를 수동으로 진행하여 최악의 상황을 모면할 수 있다.

그림 20. 일반 어플리케이션 실행
Fig. 19. Execution of General App

general_application는 일반 어플리케이션으로 가상 램 디스크에 접근 하는 인증 되지 않은 프로세스이다. 해당 프로세스는 무결성 검사 어플리케이션과 동일한 방법으로 가상 램 디스크에 접근하였지만 그림 19의 실험 결과에서 알 수 있는 것처럼 인증되지 않은 프로세스가 접근하여 값을 읽어올 시 정상 무결성 값이 아닌 잘못된 무결성 값을 전달하는 것을 알 수 있었다. 이는 정해진 방식이 아닌 불법적인 접근을 차단할 수 있다는 것을 알 수 있다.

본 논문에서 제시하는 프로토타입을 통해 실험 한 결과 특정한 공간을 물리적으로 운영하여 파일이나 운영체제의 무결성을 보장할 수 있음을 확인할 수 있었다. 따라서 본 논문이 제시하는 방안에 따라 부트레벨에서의 무결성 제공이 가능하며 이를 통해 악성코드의 악의적인 운영체제의 변경이나 공격을 원천적으로 막을 수 있다.

IV. 결 론

본 논문에서 제안하는 모바일 운영체제의 무결성 검증은 별도의 저장장치를 두고 운영체제가 올라오기 이전의 부트레벨에서 1차적인 검증을 수행하고, 운영체제에게 제어권이 넘어간 후에도 같은 검증 데이터를 활용하여, 지속적인 무결성 검증을 실시하도록 설계하였다. 그리고 해당 저장장치로의 I/O를 담당하는 디바이스 드라이버를 커널 모듈로 미리 제작을 하여 커널과 분리하여 관리한다. 해당 모듈은 운영체제에 포함하여 배포가 되는 것이 아니라 부트레벨에서 부트로더가 운영체제를 메모리에 적재할 때 같이 적재를 함으로써 펌웨어의 유출 등에 대해 안전을 보장 할 수 있다. 다만, 본 논문에서는 부트레벨의 실험이 매우 제한적이며 실험 결과를 확인하기 어려워 가상의 램 디스크를 설계한 후 앱을 통해 결과를 확인하였다. 이러한 설계 방식에 따라 모바일 환경에 적용한다면 무결성을 안전하게 유지하는데 매우 유리할 것으로 판단한다.

참고문헌

[1] Samsung KNOX. http://www.phonearena.com/news/Samsung-Knox-found-to-have-a-serious-vulnerability_id50670

[2] Vulnerability report: Xen 3.x. http://secunia.com/advisories/product/15863.

[3] Xen : Security vulnerabilities. http://www.cvedetails.com vulnerability-list/vendor_id-6276/XEN.html. Last accessed April 4, 2012.

[4] Vulnerability report: Xen3.x. http://secunia.com/advisories/product/15 86 3. Last accessed April 4, 2012.

[5] Survey of Security Threats and Countermeasures on Android Environment. Joonhyouk Jang 2013.12

[6] RAISE. Enye lkm rookit modified for ubuntu 8.04. http://packetstormsecurity.com/files/75184/Enye-LKM-Rookit-Modified-For-Ubuntu-8.04.html. Last accessed Sep 4, 2012.

[7] Integrity Protection Solutions for Embedded Systems, FOSDEM 2014 Brussels, Belgium, February , 2014

[8] Sangorrin, Daniel, Shinya Honda, Hiroaki Takada. "Dual operating system architecture for real-time embedded systems." In Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), Brussels, Belgium, pp. 6-15. 2010.

[9] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky. Hypersentry:

enabling stealthy in-context measurement of hypervisor integrity. In Proceedings of the 17th ACM conference on Computer and communications security, CCS '10, pages 38{49, New York, NY, USA, 2010. ACM.

〔10〕 DINABURG, A., ROYAL, P., SHARIF, M., AND LEE, W. Ether: malware analysis via hardware virtualization extensions. In Proceedings of the 15th ACM conference on Computer and communications security (New York, NY, USA, 2008), CCS '08, ACM, pp. 51-62.

〔11〕 D. Clarke, G. E. Suh, B. Gassend, M. van Dijk, and S. Devadas. Checking the integrity of a memory in a snooping-based symmetric multiprocessor (smp) system. Technical report, MIT LCS memo-470, http://csg.csail.mit.edu/pubs/memos/Memo-470/smpMemoryMemo.pdf, 2004.

저 자 소 개

박 재 경

1994: 동국대학교 컴퓨터공학과 공학사.
1996: 홍익대학교 전자계산학과 이학석사.
2002: 홍익대학교 전자계산학과 이학박사
현 재: 학국과학기술원 사이버보안연구센타 책임연구원
관심분야: 네트워크 보안, 사이버 보안
Email : wildcur@kaist.ac.kr

이 상 훈

1994: 홍익대학교 컴퓨터공학과 공학사.
1996: 홍익대학교 전자계산학과 이학석사
현 재: 국방과학연구소 제2기술연구본부 선임연구원
관심분야: 시스템 보안, 모바일 보안
Email : shljhl@add.re.kr

권 미 영

1987: 이화여자대학교 전자계산학과 이학사.
1992: 고려대학교 경영정보대학원 경영학석사
현 재: 국방과학연구소 제2기술연구본부 책임연구원
관심분야: 시스템 보안, 모바일 보안
Email : kmyadd@add.re.kr

김 효 남

1988 : 홍익대학교 전자계산학과 이공학사.
1990 : 홍익대학교 전자계산학과 이공학석사.
2014 : 홍익대학교 전자계산학과 이공학박사.
현 재 : 청강문화산업대학교 교수
관심분야: 사이버 보안, 게임 보안
Email : hnkim@ck.ac.kr