Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.5, Oct. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.5.1105

안드로이드 시스템 하에서의 SEAndroid 정책 보호 기법 설계 및 구현*

유 석 만,[1,2†] 박 진 형,[1] 이 동 훈[1‡]
[1]고려대학교 정보보호대학원, [2]삼성전자 무선사업부

SEAndroid Policy Protection Architecture Design and Implementation in Android*

Seok-man Yoo,[1,2†] Jin-Hyung Park,[1] Dong-hoon Lee[1‡]
[1]Graduate School for Information Security, Korea University,
[2]Mobile Devision, Samsung Electronics

요 약

SEAndroid(Security-Enhanced Android)는 안드로이드(Android) 운영체제의 핵심 보안 요소로 적용되어 있다. 리눅스와 안드로이드의 구조적 차이점이 존재함에 따라 SELinux(Security-Enhanced Linux)를 SEAndroid로서 안드로이드에 적용하고 있는데, SEAndroid의 보안은 신뢰성 있는 정책에 기반하므로 이러한 정책이 변조되는 경우에는 심각한 보안 문제가 발생할 수 있다. 따라서 SEAndroid의 가장 중요한 요소인 정책을 안전하기 보호하기 위한 방법에 관한 연구는 반드시 필요하다. 본 논문에서는 안드로이드에 현재 적용되어 있는 SEAndroid 정책 업데이트 프로세스를 분석하여 보안 취약점을 파악하고, 정책 변조 공격이 가능한 다양한 경로에 대해 분석한다. 그리고 이러한 변조 공격으로부터 정책을 보호할 수 있는 기법인 SPPA(SEAndroid Policy Protection Architecture)를 제안한다. SPPA는 개별 정책에 대한 검증 과정을 통해 정책의 신뢰성과 무결성을 보장하며, PWRM(Policy Writing Rule Monitoring)을 이용하여 버전 다운그레이드 공격을 방지한다. 마지막으로 모바일 디바이스에서 제안한 기법을 구현하여 기기에 미치는 영향 및 성능을 검증한다.

ABSTRACT

Android includes SEAndroid as a core security feature. SELinux is applied to Android OS as a SEAndroid, because there exists structural differences between Linux and Android. Since the security of SEAndroid depends on the reliable policy if the policy is tampered by the attacker, the serious security problems can be occurred. So we must protect policies which are the most important thing in SEAndoroid. In this paper, we analyze the process of SEAndroid policy updating to find out vulnerabilities and study the attack points on policy tampering. And we propose the SPPA to detect whether the policy is modified by an attacker. Moreover, we prove the performance and the effect of our proposed method on mobile device.

Keywords: SEAndroid, SE for Android, policy, update, integrity, SEAndroid Policy Protection Architecture

Received(08. 17. 2016), Modified(09. 05. 2016), Accepted(09. 09. 2016)
* 본 연구는 '15년도 삼성전자 학술과견 과정의 일환으로 수행하였습니다.
† 주저자, keisers.smyoo@gmail.com
‡ 교신저자, donghlee@korea.ac.kr(Corresponding author)

I. 서 론

스마트폰의 등장 이후 각축을 벌이던 스마트폰 운영체제 시장은 현재 안드로이드와 iOS로 양분되어 있으며, 특히 안드로이드는 '16년 1분기를 기준으로 스마트폰 운영체제 가운데 84.1%의 시장 점유율을 보이며 대세를 이루고 있다[1]. 안드로이드는 웨어러블 기기 등의 다양한 분야에도 적용되고 있으며, 이에 따라 보안 또한 필수적으로 요구되고 있다.

안드로이드는 리눅스 커널을 기반으로 동작하기 때문에, 리눅스에 존재하는 보안 취약점이 안드로이드에서도 동일하게 존재하는 경우가 있다.

리눅스의 기본 접근 통제 방법인 DAC(Discrete Access Control)에서는 파일 및 자원의 접근 제어가 오직 권한과 소유권에 의해서 결정된다. 초기 안드로이드에서는 리눅스와 마찬가지로 DAC 접근 제어를 사용하여 관리하였기 때문에, 루트(최상위 관리자 : root) 권한이 탈취되었을 경우에는 ProfileDroid[2]의 결과와 같이 공격자가 사용자 정보를 획득할 수 있으며, 더 나아가 센서 데이터 등을 조작하여 보안 사고를 발생시킬 수 있다[3].

DAC와는 달리 MAC(Mandatory Access Control) 방식에서는 신뢰성 있는 관리자에 의해 설정된 정책에 의해서만 접근 통제가 이루어지기 때문에, 공격자가 루트 권한을 획득한다 하더라도 시스템에서 정책으로 제한한 동작 이상은 허용되지 않으므로 중요 자원을 더욱 안전하게 보호할 수 있다. MAC 방식을 리눅스에 적용한 것이 SELinux[4]이며, 안드로이드에서 보안 강화 요소로 채택하고 있는 SEAndroid[5]는 SELinux를 안드로이드 구조에 확장 적용한 것이다.

그러나 SEAndroid의 보안 요소들 또한 공격자에 의해 무력화될 수 있는 가능성이 존재한다. SEAndroid의 접근 제어는 신뢰성 있는 정책을 기준으로 통제하는 것이므로, 만약 공격자가 정책을 변조한다면 정당하지 않은 방법으로 자원 및 정보의 접근이 가능해진다. 특히 SEAndroid는 여러 요인으로 인해 정책 업데이트 과정이 반드시 수행되어야 하는데, 이 과정에서 정책 변조 공격이 발생할 수 있는 문제점이 있다. 그러나 현재까지 안드로이드 보안, 특히 SEAndroid에 관한 연구는 구조 개선 및 활용 범위 확대에 집중되고 있으며[6], SEAndroid 정책 자체를 안전하게 보호할 수 있는 방법에 관한 연구는 진행되지 않고 있다.

본 논문에서는 SEAndroid 정책을 안전하게 보호할 수 있는 방법인 SPPA(SEAndroid Policy Protection Architecture)를 제안한다. SPPA는 본 논문의 SEAndroid의 정책 업데이트 과정 분석을 통해 발견된 정책 변조 공격으로부터 최신 버전의 정책 유지와 정책의 무결성을 보호할 수 있는 방법을 제공한다.

제안하는 방법은 다음과 같은 장점을 갖는다. 첫째, 향후 발생할 수 있는 신규 보안 취약점을 이용한 정책 변조 공격에 대해서도 대응 가능하다. 둘째, 현재 안드로이드 구조와는 별도의 추가 요소로 설계하여 기존 SEAndroid의 수정을 최소화할 수 있다. 셋째, 원격 검사와 같은 방법과는 달리 추가적인 하드웨어 요소가 필요하지 않기 때문에 비용 효율적이며, 추가적인 시스템 구축 시간이 없어 개발 비용과 빠른 시장 적시성을 최우선 과제로 여기는 실제 환경에 적용 시 진입 장벽을 최소화할 수 있다.

본 논문이 기여하는 바는 다음과 같다.

- 정책 업데이트 프로세스 분석을 통해 정책 변조가 가능한 보안 취약점을 발견하였다.
- SEAndroid 정책 보호 기법을 최초로 제안하였다.
- 제안하는 방법은 신규 취약점에도 대응 가능하도록 유연하고 효율적으로 동작하도록 설계되었으며, 기존 시스템에 대한 최소한의 수정만으로 쉽게 적용할 수 있다.
- 구현 및 실험을 통해 실제 환경에 적용이 가능함을 보였다.

본 논문의 구성은 다음과 같다. 2장에서는 SELinux와 SEAndroid의 기본 구조를 설명하고, 정책 변조 공격이 가능함을 보인 연구 사례를 알아본다. 3장에서는 정책 업데이트 프로세스와 관련된 보안 취약점에 대한 분석 결과를 보인다. 4장에서는 해당 취약점에 의해 가능한 공격을 보이고, 이에 대한 보호 기법을 제안한다. 5장에서는 제안한 기법이 시스템 성능에 미치는 영향에 대해 실제 구현 및 실험을 통해 측정한 결과를 제시한다. 마지막으로 6장에서 결론을 맺는다.

II. 배경 지식 및 관련 연구

2.1 SELinux

SELinux는 플라스크(FLASK, Flexible MAC Architecture) 아키텍처 기반으로 개발되었으며, 그 구조는 [Fig. 1]과 같다. SELinux를 구성하는 각 모듈의 역할은 다음과 같다.

- Object Manager : 어떤 Subject가 Object에 접근하려 할 때, 허용 여부를 AVC에게 질의하고, 그 결과에 따라 접근을 허용 및 거부한다.
- AVC(Access Vector Cache) : 그 동안의 결정을 유지하고 있으며, Object Manager가 질의한 내용에 대한 결정 내용을 가지고 있지 않을 경우 Security Server에 질의한다. Security Server에서 받은 응답을 Object Manager에 전달하는 동시에 스스로도 내용을 저장하여 다음 질의에 대비한다. 이는 성능 향상을 위한 구조로서 AVC는 메모리에 결정 내용을 유지한다.
- Security Server : SELinux 정책을 기반으로 접근 허용 여부를 결정한다. SELinux 정책은 파일 형태로 저장되어 있으나, 시스템 시작 시 Security Server가 그 내용을 읽어 메모리에 유지한다.

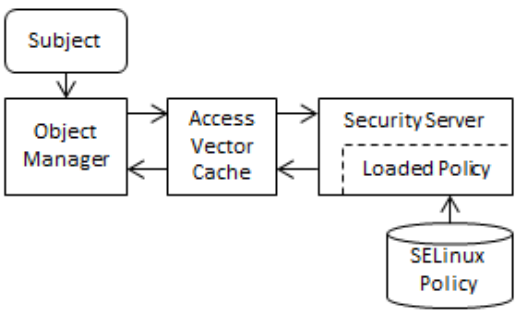

Fig. 1. SELinux architecture[7]

2.2 SEAndroid

안드로이드 또한 리눅스 커널을 기반으로 동작하고 있기에 SELinux를 적용할 수 있는 여지가 있었으나, 초기 안드로이드 개발 시는 여러 사유로 인해 SELinux 적용에 대해서는 고려되지 않았다. 스마트폰의 대두 당시는 안정적이며 풍부한 기능을 갖춘 스마트폰 운영체제의 개발이 우선이었다. 이는 곧 운영체제의 점유율 확산과 관련하여 향후 운영체제의 존속과 직결되어 있는 문제였기 때문이다.

구글 및 안드로이드 진영이 강한 보안의 필요성을 가지게 된 것은 안드로이드 버전 4.X 무렵이다. 안드로이드 운영체제가 시장에 안착했고 급속도로 점유율을 높여감에 따라, 그 활용도 또한 다양한 분야로 퍼져나갔다. 모바일 금융 분야가 대표적이며, 이는 절대적인 안전성과 보안을 요구한다. 게다가 급속한 발전은 스마트폰 시장이 곧 포화 상태가 될 것이라는 예측을 이끌어냈다. B2C(Business To Customer) 시장의 포화를 예측한 구글 및 안드로이드 진영은 B2B(Business To Business) 시장으로 시선을 돌리기 시작했다.

활용분야가 크게 확대되면서 안드로이드 악성 코드 및 보안 침해 사례는 급격히 증가하였으며 금융, 정부 및 기업 업무에서 보안 침해 사고가 발생한다면 이는 개인 차원에서의 피해 규모와는 차원을 달리하므로, 안드로이드에 대한 강한 보안 요소는 반드시 필요한 사항이 되었다.

이런 흐름 속에서 구글은 SEAndroid 개발 및 적용을 결정하였다. [Fig. 2]는 안드로이드 6.0을 기반으로 분석한 SEAndroid 구조를 나타낸다.

커널 스페이스에 존재하는 AVC와 Security Server의 동작은 SELinux와 동일하다. SELinux와 SEAndroid를 구분짓는 결정적인 요인은 Framework에 추가된 요소와 SEAndroid Policy 및 Context이다. SELinux MAC 방식을 앱 기반

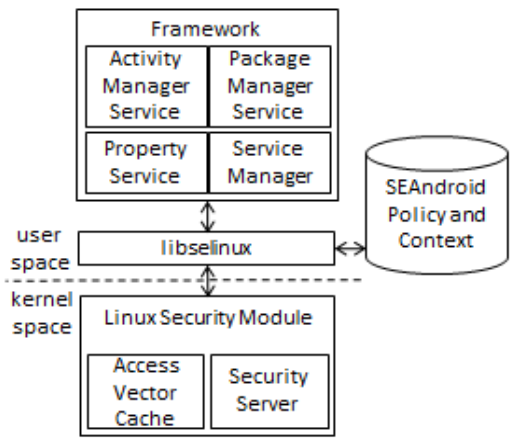

Fig. 2. SEAndroid architecture

자원 및 퍼미션과 같은 안드로이드 고유 접근 제어 방식에 적용하기 위하여 PMS(PackageManager-Service), AMS(ActivityManagerService) 등의 프레임워크 모듈에 관련 코드가 추가되었다. 또한, 이러한 안드로이드 고유 요소에 대한 접근을 제어하기 위해 SEAndroid 고유의 정책을 수립하여 적용하였다. 시스템 빌드 및 런타임 시, 프레임워크는 SEAndroid 컨텍스트를 참조하여 자원을 레이블링한다. Subject가 자원 접근 시도 시, 프레임워크는 libselinux 라이브러리를 통하여 LSM(Linux Security Module)에 질의하며, LSM은 메모리에 미리 읽어 놓은 정책을 기반으로 허용 여부에 대한 결정을 수행한다.

SEAndroid가 제어하는 안드로이드 자원의 종류를 알아보기 위한 가장 좋은 방법은 정책과 컨텍스트 내용을 분석하는 것이다. 정책 및 컨텍스트는 역할별로 여러 파일에 분산 저장되어 있으며, 각 파일의 내용은 다음과 같다.

- file_contexts : 파일 시스템에 존재하는 파일들에 대한 보안 컨텍스트 정의. 파일은 일반 파일 및 디바이스 노드, 소켓 등과 같은 파일 형태의 자원도 포함한다. 빌드 시점에 존재하는 파일은 빌드 시 레이블링되며, 런타임 시에 새로 생성되는 파일 또한 컨텍스트를 참조하여 레이블링된다.
- service_contexts : 안드로이드에서 동작하는 서비스에 대한 보안 컨텍스트를 정의한다.
- seapp_contexts : APP 프로세스와 디렉토리에 대한 보안 컨텍스트를 정의한다.
- property_contexts : 안드로이드에서 정의한 프로퍼티에 대한 보안 컨텍스트를 정의한다.
- mac_permissions.xml : APP 서명에 따른 보안 컨텍스트를 정의한다. 이는 곧 APP이 사용할 수 있는 권한, 즉 퍼미션과 직결되어 있다.
- sepolicy : Security Server에서 사용하는 MAC 정책을 가지고 있다. 위에서 정의된 보안 컨텍스트에 기반하여 정책을 검사하고 접근 허용 여부를 결정한다.

안드로이드 앱, 서비스, 프로퍼티, 그리고 퍼미션은 리눅스와 구분되는 안드로이드의 고유 요소로서, 이러한 요소들을 보호하는 것이 SEAndroid와 SELinux의 가장 큰 차이점이다.

그 외 eops.xml(Enterprise operations), ifw.xml(Intent Firewall)과 같은 정책 파일도 있으나 부가적인 정책이며, SEAndroid 정책 및 컨텍스트는 위에 명시한 6개 파일을 기본으로 동작한다. 따라서 본 논문에서 제안하는 정책 보호 기법 또한 6개 파일만을 주 보호 대상으로 하며, 편의상 정책 및 컨텍스트를 통틀어 SEAndroid 정책이라 통칭하고자 한다.

2.3 관련 연구

SEAndroid 정책을 보호하는 방법에 관한 연구는 현재까지 전무하다. 반면 정책 변조 공격과 관련된 연구는 발표된 것이 있는데, 대표적으로 O. Peles 등은 기존에 알려진 CVE-2014-7911[8] 취약점을 기반으로 동일 취약점을 가진 코드를 조사, 분석 및 실증한 바 있으며, 이를 기반으로 SEAndroid 정책을 변조할 수 있음을 보였다[9]. 이 취약점은 안드로이드 IPC(Inter-Process Communication)를 통하여 비정상적인 직렬화 객체를 송신했을 때, 수신 프로세스 측에서 수신된 개체의 정상 여부를 검사하지 않고 비직렬화하는 문제를 이용한다. 수신 프로세스의 GC(garbage collector)가 비직렬화된 객체를 소멸시킬 때 해당 객체의 소멸자를 호출하며, 몇몇 클래스의 소멸자는 클래스 멤버 변수에 저장되어 있는 메모리 주소를 참조하여 해당 위치에 존재하는 코드를 실행하도록 되어 있다. 공격자는 공격 코드의 위치를 클래스 멤버 변수에 저장하고, 이를 비정상적으로 직렬화하여 수신 프로세스에 전송한다. 수신 프로세스는 해당 객체를 비직렬화하나 정상적인 객체가 아니므로 사용하지 않으며, 수신 프로세스의 GC 가 해당 객체의 소멸자를 호출, 결과적으로 수신 프로세스의 권한 하에 공격 코드가 실행되게 된다. O. Peles 등은 이 공격이 유효함을 입증하기 위해 안드로이드의 system_server 프로세스에 비정상적인 직렬화 객체를 송신하여 SEAndroid 정책을 변조할 수 있음을 증명하였다. SEAndroid 정책 파일은 system_server와 init 프로세스만 변경 가능하도록 정책으로 설정되어 있으나[10], 취약점을 이용하여 system_server 하에서 공격 코드가 실행된다면, 이는 정책에 부합하는 동작이므로 방어할 수 없다. 이 연구의 공격 방법 자체는 본 논문에서 주목하

는 SEAndroid 정책 보호와 직접적인 관련은 없으나, 실제로 SEAndroid 정책 변조 공격을 수행했다는 것에 의의가 있다.

III. SEAndroid 정책 업데이트 프로세스

SEAndroid는 정책을 기반으로 한 강력한 접근 제어 방식인 MAC을 사용하며, 설계 상에서 현재까지 발견된 문제는 없다. 그러나 이를 실제 환경에 적용하기 위해서는 여러 가지 부가 요소가 추가적으로 고려되어야 하는데, 그 중 한 가지가 정책을 업데이트하기 위한 프로세스이다.

본 장에서는 안드로이드에서 SEAndroid 정책 업데이트 프로세스의 필요성과 그로 인한 구조적 한계에 대해 설명한다. 또한 실제 업데이트 코드 분석을 통해 발견한 보안 취약점에 관하여 설명한다.

3.1 정책 업데이트 프로세스 필요성 및 구조적 한계

SEAndroid 정책 변조 공격을 차단하기 위한 가장 간단한 방법은 system_server 프로세스를 포함한 어떤 프로세스도 SEAndroid 정책에 대해 쓰기 권한을 가지지 않도록 하는 것이다. 이는 SEAndroid 정책 설정을 통하여 가능하지만 현실적으로는 적용 불가하며 그 이유는 다음과 같다.

SEAndroid 정책은 강력한 보안을 제공해 주지만 그와 비례하여 사용성에 영향을 미칠 수 있으므로 정확히 수립되어야 하며, 이것이 바로 안드로이드 4.3에는 퍼미시브 모드, 4.4에는 부분적 인포싱 모드, 5.0에 이르러서야 비로소 전체 인포싱 모드로 적용된 이유이다. SEAndroid는 접근 허용/거부를 포함한 SEAndroid 관련 정보를 AVC log로 기록하며, 퍼미시브 모드에서는 정책에 어긋난 동작이라 하더라도 이를 허용하였고, AVC log에 해당 기록을 저장하였다. 이 로그를 바탕으로 더욱 정확한 정책을 수립하여 안드로이드 4.4에서는 부분적 인포싱 모드로 적용하였으나, 그럼에도 불구하고 실제 시장에서는 SEAndroid로 인하여 정상 동작이 되지 않는다는 보고가 종종 발생했다.

안드로이드 신규 버전 발표 시마다 새로운 기능 요소가 추가되고 있으며, 웨어러블[11], 자동차[12], 더 나아가 IoT 기기[13] 등 각 분야를 위한 안드로이드 운영체제도 개발되고 있다. 이에 따라 신규 기능 및 각 안드로이드 운영체제를 위한 SEAndroid 정책도 계속하여 추가되어야 하지만, 정확하고 안정적인 수준의 정책을 추가하기 위해서는 일정 기간 이상의 자료 수집이 필수적이다.

안드로이드 앱 또한 SEAndroid 정책과 밀접한 관련이 있다. 구글 및 안드로이드 탑재 기기 업체는 정책 수립 후 여러 앱을 설치하여 정상 동작 여부를 확인하는 과정을 거치나, 전 세계 차원에서 개발되는 안드로이드 앱의 수는 기업 차원에서 테스트할 수 있는 범위를 넘어선다. 이에 잘못 수립된 정책이 시장 배포 후 발견되는 경우가 있으며, 간혹 정상적인 거부 정책이지만 사용자 요구에 의해 허용해야 하는 경우도 존재한다.

위와 같이 다양한 상황에서 정책을 적시에 업데이트해야 하는 경우가 발생한다. 정책을 업데이트하는 가장 기본적인 방법은 펌웨어 업데이트 시 함께 수행하는 것이나, 이는 비용과 보안 측면에서 큰 문제를 내포하고 있다. 펌웨어 업데이트는 산업 구조에 따라 회당 고비용을 요구한다. 정책은 보안과 직결된 사항이므로 즉시 업데이트해야 할 경우가 빈번한데, 펌웨어 업데이트와 함께 수행될 경우 기업이 부담해야 할 비용이 커지게 된다. 또한 정책 결점으로 인한 중대하고 긴급한 보안 사고가 발생할 경우, 해당되는 정책 결점은 즉시 업데이트되어야 한다. 그러나 펌웨어 업데이트 프로세스는 대부분 오랜 시간을 필요로 하며, 사용자는 업데이트가 배포될 때까지 보안 위협에 노출된다.

구글(Google) 또한 이러한 점을 인식하여 SEAndroid 정책 업데이트 방식에 대해 펌웨어 업데이트와 별도의 프로세스를 구축하도록 강제하고 있으며, 구글이 운영하는 안드로이드 호환성 프로그램(Android Compatibility Program)[14]에 이를 언급하고 있다. 이 프로그램은 안드로이드 기기가 갖추어야 할 여러 요건을 제시하고, 그 요건에 맞도록 개발된 기기만 GMS(Google Mobile Services) 인증을 부여한다. GMS 인증을 획득한 기기만 구글 앱 스토어를 이용할 수 있기 때문에 GMS 인증은 매우 중요한 사안이다. 따라서 주요 안드로이드 탑재 기기 제조업체 입장에서는 시장 경쟁력 확보를 위해 GMS 인증을 반드시 획득해야 한다. GMS 인증 요건 명시 문서인 CDD(Android Compatibility Definition Document) 4.4의 9.7 항목에 SEAndroid 정책 업데이트 관련 요건을 “it MUST support dynamic updates of the SELinux policy file without requiring a

system image update"와 같이 명시하고 있다 [15].

즉, GMS 인증을 받기 위해서는 반드시 펌웨어 업데이트와 별개로 SEAndroid 정책 업데이트 프로세스를 갖추어야 하며, 이는 곧 외부로부터 정책 파일을 수신하여 기존 정책 파일을 업데이트할 수 있는 프로세스가 존재해야 한다는 것과 동일한 의미가 된다. 이러한 구조적 한계에 의해 정책을 변경할 수 있는 프로세스는 하나 이상 존재해야 하며, 바로 이 프로세스가 SEAndroid 정책 변조 공격의 보안 취약점이 될 수 있다. 공격자가 업데이트 프로세스의 컨텍스트 하에서 정책 변조 공격 코드를 수행시킬 수 있다면 SEAndroid는 무력화되며 2.3절에서 설명한 O. Peles 등이 수행한 공격이 이에 해당한다.

3.2 정책 업데이트 프로세스 분석

정책은 SEAndroid의 핵심이며 정책을 업데이트하는 과정에서 변조된 정책이 입력되면 SEAndroid의 목적을 위한 기능이 무력화되므로, 정책의 무결성은 반드시 검사해야 할 필요가 있다. 본 절에서는 안드로이드 6.0 AOSP(Android Open Source Project) 코드를 기반으로 정책 업데이트 프로세스를 분석하고, 보안 취약점이 발생할 수 있는 부분에 대해 파악한 결과를 제시한다.

[Fig. 3]은 SEAndroid 정책 업데이트 프로세스 구조를 나타낸 것이다. 점선으로 표시된 부분은 실제 AOSP 코드에는 존재하지 않는 부분이다. 구글은 실선으로 표시된 부분만 구현하고, 점선 부분은 각 제조사에서 자체적으로 제작하도록 하고 있다. 서버 부분은 단말 측이 아니므로 당연히 구글에서 제작할 부분이 아니다. AGENT는 각 제조사의 정책 배포 서버와 통신하는 부분이며, 제조사 별로 서버와의 통신 프로토콜이 다를 수 있기에 구글에서 일괄적으로 코드를 제작할 수 없다. 그러나 서버 및 AGENT는 일반적인 OTA 구조에서 반드시 존재해야 하므로, 정책 업데이트 프로세스에 포함되어야 한다.

업데이트 대상이 되는 파일은 [Fig.3]의 a)에 나타낸 8개 파일이다. selinux_version은 안드로이드의 BUILD_FINGERPRINT 값을 포함하고 있으며, 이는 정책이 적용되어야 할 펌웨어 버전을 나타낸다. /data/security/bundle의 version은 정책 번들의 버전을 의미하며, 이를 제외한 6개 파일은 2장에서 설명한 SEAndroid 정책 구성 파일이다.

정책 파일들은 개별 파일로서 기기에 전달되는 것이 아니다. 정책 번들이라는 하나의 파일 형태로 구성된 후[16] 배포 서버를 통해 각 기기에 전달되며, 부가 정보로 번들 버전을 함께 전송한다. AGENT는 정책 번들과 버전을 서버로부터 수신하고 인텐트(Intent)에 포함하여 system_server 프로세스로 전달한다. system_server에서 정책을 전달받는 리시버는 SELinuxPolicyInstallReceiver이며,

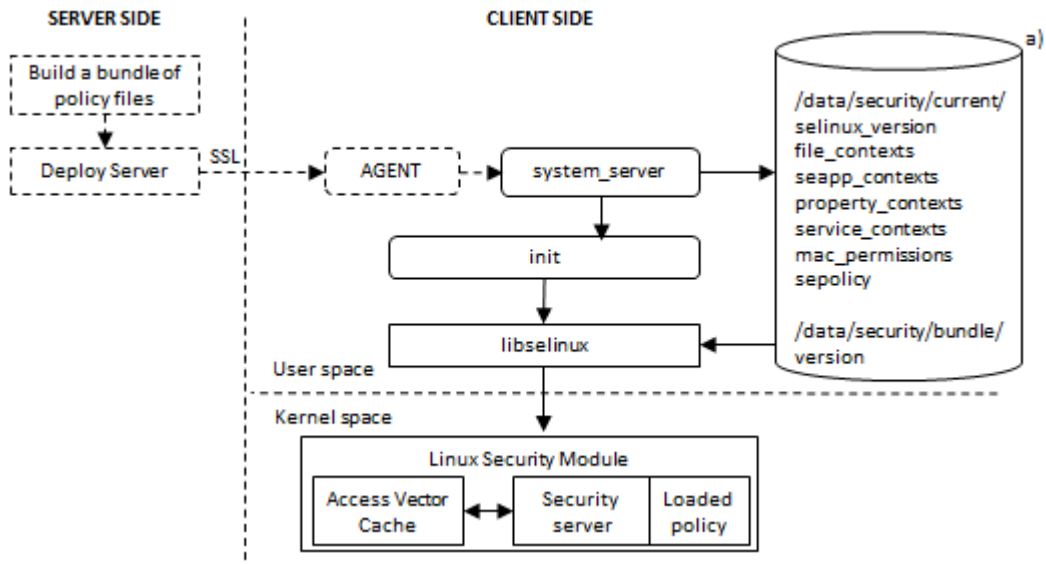

Fig. 3. SEAndroid policy update process. a)Policy-related files

〔Fig. 4〕와 같이 android.permission.UPDATE_CONFIG이라는 퍼미션으로 보호되어 있다. 이 퍼미션은 〔Fig. 5〕와 같이 system|privileged 권한을 가진 앱만 소유할 수 있으며, 이는 제조사 서명키로 서명되어 있고 출시 시점에서 선탑재되어 있는 프로세스만 사용 가능한 권한이라는 의미를 가지고 있다. AGENT와 같이 신뢰할 수 있는 앱 및 서비스로부터만 업데이트 정책 번들을 수신할 것이며, 이를 바탕으로 정책 번들의 신뢰성을 확보하려는 것을 알 수 있다.

system_server 프로세스는 다음과 같은 동작을 수행한다.

- 현재 기기에 적용되어 있는 정책 번들 버전(data/security/bundle/version)과 새로운 번들 버전의 비교. 새로운 버전이 기존 버전보다 높은 경우만 업데이트 허용
- 번들을 sepolicy 등의 각 파일로 분리하여 /data/security/current 디렉토리에 저장
- selinux.reload_policy 프로퍼티를 1로 셋팅

업데이트된 정책이 즉시 적용되어야 할 필요가 있다면 이는 selinux.reload_policy 프로퍼티를 셋팅하는 것으로 가능하며, 이러한 셋팅 동작은 system_server, init 프로세스만 할 수 있도록 SEAndroid 정책으로 제한되어 있다. system_server 프로세스에 의해 selinux.reload_policy 프로퍼티가 1로 셋팅되면 init 프로세스가 동작하여 정책들을 실제로 적용시킨다. libselinux가 sepolicy를 읽어들여 커널로 전달하면, 전달된

```
<receiver android:name=
 "com.android.server.updates.
 SELinuxPolicyInstallReceiver"
 Android:permission=
 "android.permission.UPDATE_CONFIG"/>
```

Fig. 4. The permission of SELinuxPolicyInstallReceiver

```
<permission android:name=
 android.permission.UPDATE_CONFIG
 "Android:protectionLevel=
 "signature|privileged"/>
```

Fig. 5. The priviledge of UPDATE_CONFIG permission

sepolicy는 Security Server에 의해 메모리에 적재되고, 그 순간부터 SEAndroid 정책으로서 역할을 수행하며, Linux Kernel Module 안에서의 AVC와 Security Server의 동작은 2장에서 설명한 바와 같다. 그 외 정책들은 sepolicy와 같이 selinux.reload_policy 프로퍼티가 1로 셋팅될 때 즉시 적용되나 mac_permission의 경우는 앱이 설치될 때에서야 비로소 적용된다.

그러나 이러한 정책 적용 단계에서 정책의 무결성과 신뢰성을 검증하는 프로세스는 존재하지 않으며, 모든 정책은 기기 재부팅 시에도 일괄적으로 적용된다.

3.3 SEAndroid 정책 변조 공격

정책의 무결성과 신뢰성은 SEAndroid 동작에 있어서 가장 중요한 요소이다. 그러나 3.2절에서 설명한 것과 같이 무결성 검증 프로세스는 존재하지 않으며, 신뢰성은 정책 수신 리시버의 퍼미션에만 의존한다는 것을 알 수 있다. 즉, AGENT와 리시버 사이의 구간을 제외한 모든 부분은 보안 취약점에 해당될 수 있다.

다음은 공격자의 능력에 따라 가능한 정책 변조 공격을 나타낸 것이다.

- 서버와 AGENT 구간 침투 : 만약 공격자가 서버와 AGENT 구간 사이에 개입할 수 있는 능력을 가지고 있다면 변조된 정책 번들을 주입하여 SEAndroid를 무력화시킬 수 있다. 이 구간은 제조사 자체적으로 구현하는 구간이며, 일반적으로 SSL 프로토콜을 사용한다. SSL은 안전한 프로토콜로서 대중적으로 사용되고 있으나 계속해서 취약점이 발견되고 있고, 개발자의 능력에 따라 구현 결과물이 보안 취약점을 가지고 있을 수도 있다. 제조사 자체적인 보안 프로토콜을 사용한다 해도 취약점은 존재할 수 있기 때문에 만약 공격자가 변조된 정책 번들을 AGENT에게 전달할 수 있다면, 변조된 정책이 AGENT를 통하여 system_server에 전달된 후의 과정에서는 신뢰성 및 무결성 검증 과정이 존재하지 않으므로 SEAndroid가 무력화 될 수 있다.
- system_server 권한 탈취 : system_server 프로세스는 init 프로세스와 더불어 SEAndroid 정책 파일을 쓸 수 있도록 허용되어 있다. 만약 공격자가 system_server의

권한을 탈취할 수 있는 능력을 소지하고 있다면, system_server의 권한으로 정책을 변조할 수 있다.

SEAndroid가 무력화되는 것은 여러 보안 침해 사고가 발생할 수 있는 시발점이 되며, 이에 대해서는 안드로이드 사이트에서 설명하고 있다[17]. 이 부분은 SEAndroid가 필요한 경우에 대해서 나열하고 있으나, 이는 SEAndroid가 적용되어 있지 않거나 무력화되었을 경우 발생할 수 있는 예시 또한 될 수 있다. 일례로 루트 권한으로 동작하고 있는 netd 프로세스가 공격당했을 경우, system server 프로세스만이 변경할 수 있는 파일들을 공격자가 변조할 수 있게 되며, 이 것이 SEAndroid 정책 변조 공격에 성공한 공격자가 이후 수행할 공격 시나리오가 될 것이다.

정책 변조의 대상으로 두 가지를 가정해 볼 수 있다. 첫 번째로는 /data/security/current에 쓰여진 정책 중 원하는 일부만을 변조하는 것이며, 두 번째는 기기에 쓰여진 정책보다 더 낮은 버전의 정책을 모두 덮어쓰는 것이다. 정책이 버전 업데이트되는 이유는 여러 가지가 있으나, 대표적으로 (1)잘못 수립된 정책으로 인한 정상 동작 방해 (2)정책의 빈틈을 이용한 공격이 발생하여 정책 보완 필요 (3)시스템 변경에 의해 불필요해진 정책 삭제 등을 생각할 수 있다. 이 중 주목해야 하는 것은 (2)의 경우이다. 버전 1.0 정책에 보안 약점이 존재하고, 그를 보완한 버전 2.0 정책이 이미 기기에 적용되어 있다 가정한다. 버전 1.0 정책은 이미 공중(公衆)에 배포되어 있어 공격자가 쉽게 획득 가능하며, 정당한 제작자에 의해 제작된 것이기 때문에 신뢰성 및 무결성 판별 과정을 통과할 수 있다. 공격자는 버전 1.0에 존재하는 보안 약점을 이용하여 원하는 공격을 수행할 것이다. 본 논문에서는 이러한 공격을 버전 다운그레이드 공격(version downgrade attack)이라 명명하여 이후 설명에 사용한다.

IV. SEAndroid Policy Protection Architecture

본 장에서는 SEAndroid 업데이트 프로세스를 분석한 내용을 바탕으로 SEAndroid 정책 변조 공격을 방어할 수 있는 기법인 SPPA(SEAndroid Policy Protection Architecture)를 제안한다.

4.1 시스템 설계 요구 사항

SEAndroid 정책 변조 방지 시스템은 다음과 같은 설계 요구 사항을 만족해야 한다.

i. 신뢰할 수 있는 제작자가 수립한 정책인지에 대한 여부를 판별할 수 있어야 한다.
ii. 정책 업데이트 프로세스 중 어떠한 단계에서 정책 변조 공격이 성공했는지에 관계없이, 변조된 정책은 기기에 적용되지 않아야 한다.
iii. 현존 혹은 향후 발생할 수 있는 어떠한 취약점에 의해 정책이 변조되었는지에 관계없이, 변조된 정책은 기기에 적용되지 않아야 한다.
iv. 현재 기기에 적용되어 있는 정책 번들의 버전보다 더 낮은 버전의 정책은 기기에 적용되지 않아야 한다.
v. 실제 적용 시 진입 장벽을 낮추기 위해 다음과 같은 사항을 만족해야 한다.
   a. 기존 코드의 변경을 최소화해야 하며, 추가되는 요소는 가능한 별도의 분리된 모듈로 구성되어야 한다.
   b. 추가적인 하드웨어 요소 배제를 통해 비용 효율성 만족 및 개발 시간을 단축할 수 있어야 한다.

4.2 SEAndroid Policy Protection Architecture

본 절에서는 SPPA를 위한 신규 번들 구조 및 SPPA에 해당하는 Integrity Checker, PWRM (Policy Writing Rule Monitor)과 그 세부 동작에 대해 설명한다.

4.2.1 신규 정책 번들 구조 제안

[Fig. 6]은 AOSP 6.0 기반에서 동작하는 기존의 정책 번들 구조를 나타내며, 정책 번들과 번들 버전이 인텐트를 통해 system_server 프로세스로 전달되어 정책 업데이트 프로세스가 수행된다. 번들 안에 포함되어 있는 내용은 selinux_version, mac_permissions, seapp_contexts, property_contexts, file_contexts, sepolicy, service_contexts로서, 번들의 앞 부분은 각 정책 내용의 길이를 포함하고 있으며, 각 길이는 각각 4 bytes 공간 안에 저장되어 있다. 즉, 정책 길이를 저장하는 앞 부분의 총 저장 공간은 28 bytes이다. 뒷 부분

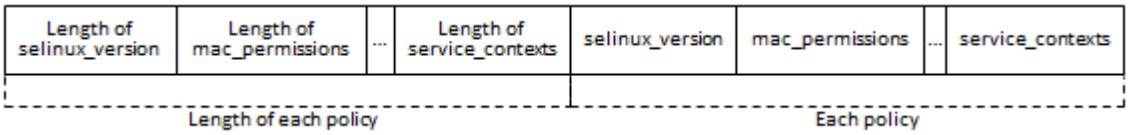

Fig. 6. Original policy bundle structure

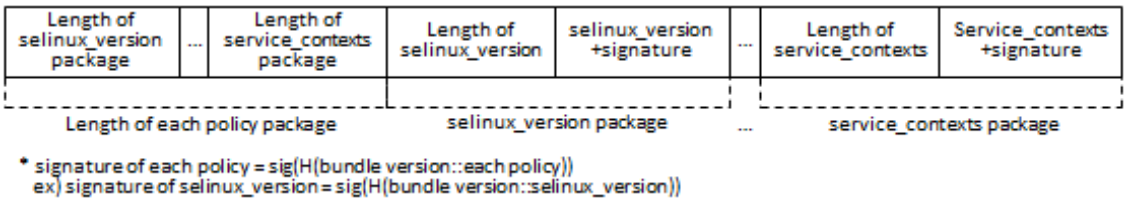

Fig. 7. A policy bundle structure for the SPPA

은 동일한 순서로 각 정책 내용이 포함되어 있다.

system_server는 정책 번들 앞 부분에 기록되어 있는 각 정책의 길이를 참조하고, 그를 이용하여 뒷 부분의 각 정책을 분리하여 /data/security/current 디렉토리에 저장한다. system_server는 인텐트 퍼미션의 보호 레벨로만 정책의 신뢰성을 의존하며, 정책 번들 자체에 대한 어떤 검증도 수행하지 않는다. 따라서 정책 번들 자체에도 서명 등의 인증을 위한 정보는 포함되어 있지 않다. 정책 파일들이 실제 정책으로 적용될 때에도 무결성 검증 등의 프로세스는 존재하지 않는다. 이로 인해 공격자가 변조된 정책 번들을 OTA 상에서 주입하거나 혹은 이미 /data/security/current 디렉토리에 쓰여진 정책 파일의 내용을 변조하는 공격에 대해서 무방비 상태가 된다.

이러한 문제를 보완하기 위해 〔Fig. 7〕과 같이 신규 정책 번들 구조를 제안한다.

먼저 기존 번들 구조에서 각 정책의 길이와 정책의 내용, 정책에 대한 서명값을 하나의 패키지로 구성한다. 신규 정책 번들의 앞 부분은 이러한 각 정책 패키지의 길이를 저장한다. 각 패키지의 길이는 기존과 동일하게 각각 4bytes의 공간 안에 저장하며, 그 순서 또한 기존과 동일하다.

뒷 부분은 각 정책의 패키지로 구성된다. 정책 패키지는 정책 자체의 길이, 그리고 정책 내용과 개별 정책의 서명을 포함하고 있으며, 그 순서는 앞 부분에서 패키지의 길이를 저장한 순서와 동일하다. 서명은 번들 버전과 정책을 붙여 해시한 결과를 정당한 제작자의 RSA 개인키로 서명한다. 해시 알고리즘으로는 SHA512를 사용하며, 2048비트의 RSA 개인키를 이용하여 서명하여야 한다.

제안하는 정책 번들 구조는 다음과 같은 특징을 갖는다.

- 정당한 제작자의 개인키로 서명함으로서, 출처의 신뢰성을 검증할 수 있다.
- 개별 정책 서명 구조 도입으로 검증 위치의 유연성을 확보함으로서 불특정 취약점에 대해 대응할 수 있다.
- 번들 버전을 포함한 서명 생성으로 버전 다운그레이드 공격을 방어할 수 있다.
-

〔Fig. 8〕은 전체적인 SPPA의 구조를 나타낸다. 정책 번들과 번들 버전을 전달받은 system_server는 우선 번들 버전을 /data/security/bundle 디렉토리에 저장한다. 이후 정책 번들의 앞 부분을 읽어 각 정책 패키지의 길이를 기록한 후, 그 길이에 따라 각 패키지를 분리하여 /data/security/current 디렉토리에 개별 파일로 저장한다. 번들 버전을 먼저 저장하는 것은 PWRM과 밀접한 관련이 있으며, 이는 4.2.3절에서 설명한다.

모든 파일이 정상적으로 저장되면, system_server는 selinux.reload_policy 프로퍼티를 1로 셋팅하여 정책들이 실제로 적용되도록 명령한다.

4.2.2 Integrity Checker

안전한 정책 보호를 위해서는 공격자가 정책 변조를 위해 사용한 취약점이나, 변조를 성공한 시점에

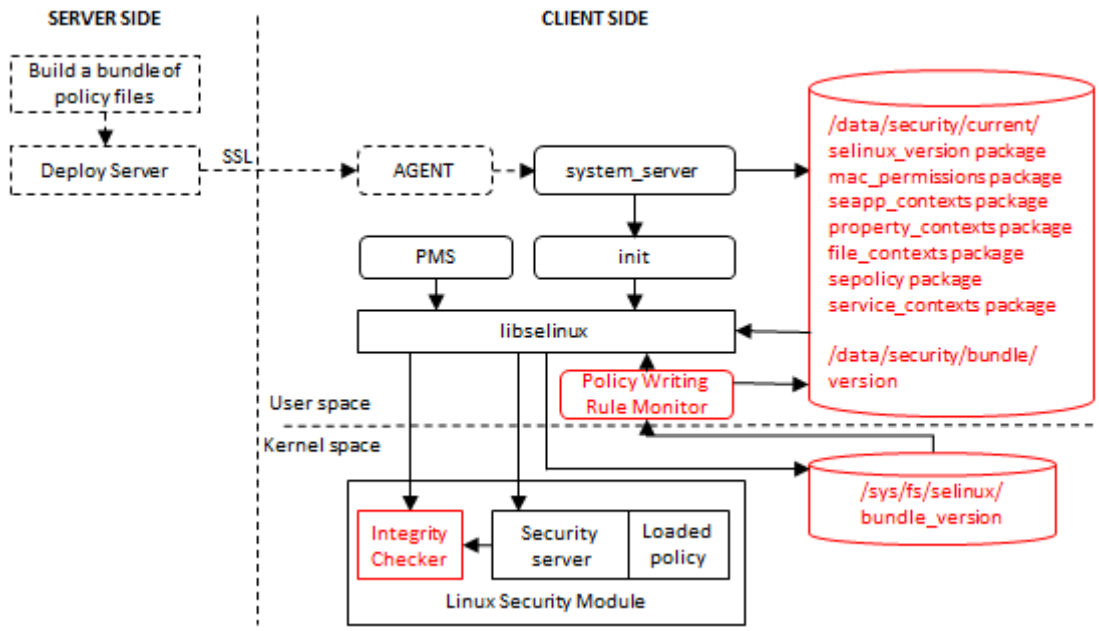

Fig. 8. SEAndroid Policy Protection Architecture

관계없이 변조된 정책을 기기에 적용할 수 없도록 보장해야 한다. 이를 위해 SPPA는 정책 적용 직전 단계에서 각 정책의 서명을 검증한다. 기존 AOSP의 문제점은 정책 번들의 신뢰성을 system_server에 존재하는 리시버의 퍼미션으로만 확인한다는 것이며, 그 이후 프로세스에서는 검증 단계가 존재하지 않는다는 것이다. 정책 적용 직전 단계에서 각 정책의 서명을 검증하면, 이전의 어떤 단계에서 어떠한 공격으로 인해 정책 변조가 발생하더라도 이를 검출할 수 있다.

정책 번들에 포함되어 있는 정책들이 모두 동일한 시점과 위치에서 적용된다면, 번들 자체에 서명하는 것이 효율적이며 일반적이다. 적용 직전 단계에서 번들 전체에 대한 서명을 검증한 후 적용하면 되기 때문이다. 그러나 현재 AOSP 6.0에 존재하는 SEAndroid의 구조는 그렇지 않다. 각 정책 별로 적용되는 위치가 다르며 이를 위한 코드 흐름도 다르다. 이는 각 정책이 적용되어 활용되는 부분이 다르기 때문이다. 예를 들어, sepolicy의 경우는 커널 메모리에 적용되어 동작의 허용 여부를 판단하는 역할을 담당한다. property_contexts는 유저 메모리에 적용되어 안드로이드 프레임워크에 존재하는 프로퍼티의 레이블링을 담당한다. 이를 위해 본 논문에서 제안하는 정책 번들은 각 정책별로 서명을 생성함으로써, 안드로이드의 정책 적용 구조를 변경하지 않도록 지원한다.

각 정책의 서명 검증은 Integrity Checker에서 담당한다. Integrity Checker는 〔Fig. 8〕과 같이 커널의 LSM 부분에 위치한다. 커널은 유저 영역에 비해 안전성이 높으며, 이를 보장하기 위한 여러 기술 또한 개발되어 있다. 이러한 기술의 예시로 삼성의 안드로이드 보안 강화 아키텍처인 Knox Real-time Kernel Protection〔18〕을 들 수 있다.

각 정책의 업데이트를 담당하는 Security Server, libselinux, PMS는 정책 업데이트 직전에 Integrity Checker로 정책 패키지를 전달하여 서명 검증을 요청한다. Integrity Checker는 서명 패키지의 앞 부분에 저장되어 있는 정책 자체의 길이를 읽고, 이를 이용하여 정책과 서명을 분리한다. 서명의 길이를 저장하는 공간은 4bytes로 일정하다. 서명을 검증하기 위해서는 두 가지 정보가 추가적으로 필요하며, 번들 버전과 서명 검증을 위한 공개키가 그에 해당한다. 번들 버전은 /sys/fs/selinux/bundle_version에 저장되어 있는 값을 읽어온다.

Integrity Checker를 이용하여 서명 검증을 수행하는 과정은 다음과 같다.

- sepolicy : selinux.reload_policy 프로퍼티가 1로 셋팅되면, init 프로세스의 프로퍼

티 서비스 코드가 동작하고 libselinux 함수를 이용하여 /data/security/current 에 저장된 sepolicy 패키지를 읽는다. 이 내용은 커널의 Security Server로 전달되며, Security Server는 정책을 integrity checker로 전달하여 sepolicy의 서명 검증을 수행한다. 검증이 성공적으로 완료될 경우 Security Server는 sepolicy 내용을 Loaded Policy, 즉 정책 데이터베이스 형태로 저장하여 이후 동작 허용 여부에 참조한다.

- file_contexts, property_contexts, seapp_contexts, service_contexts : selinux.reload_policy 프로퍼티가 1로 셋팅되면 init 프로세스의 프로퍼티 서비스 코드가 동작하고 libselinux에서 /data/seurity/current에 저장되어 있는 각 해당 패키지 파일을 읽는다. 이 내용은 안드로이드 프레임워크에서 레이블링에 활용하는 정보이므로 커널의 Security Server로 전달되지 않고 유저 메모리에 바로 저장된다. 따라서, libselinux는 각 정책 패키지를 Integrity Checker로 바로 전달하여 서명 검증을 요청한다.
- mac_permissions.xml : 이 내용은 신규 앱 설치 시 앱의 권한을 판별하는 것과 관련되어 있다. 따라서 PMS(PackageManager Service)에서 직접 읽고 저장한다. 앱이 설치될 경우 PMS가 /data/security/current에 저장되어 있는 mac_permission 패키지 파일을 읽고, libselinux를 통하여 Integrity Checker로 서명 검증을 요청한다.

4.2.3 Policy Writing Rule Monitor

변조된 정책은 Integrity Checker로 탐지할 수 있지만, 버전 다운그레이드 공격의 경우에는 Integrity Checker로는 방어할 수 없다. 낮은 버전의 정책 서명이라 하더라도 정당한 개인키로 서명되어 있기 때문이다. 따라서 신규 정책 번들의 버전과 기기에 적용되어 있는 번들 버전의 비교가 필요하다.

번들 버전 비교 방법으로 여러 기법을 생각할 수 있다. 기존에 적용되어 있는 것과 같이, 정책 번들 업데이트 프로세스 중 번들 버전을 비교하는 것이다. 그러나 공격자가 system_server의 권한을 탈취하여 /data/security/current에 존재하는 정책을 직접 변조한다면 회피 가능하다. Integrity Checker 구조를 기반으로, 버전 비교 과정을 실제 정책 적용 직전 단계로 이동시키는 것도 해결 방법이 될 수 없다. 인입된 번들 버전과 기존 번들 버전을 비교해야 하나, 공격자는 기존 버전이 저장되어 있는 /data/security/bundle/version을 변조할 수 있는 능력을 가지고 있으므로 신뢰할 수 있는 결과를 얻을 수 없다. 마지막으로, 기기 부팅 시 /data/security/bundle/version에 저장되어 있는 번들 버전을 안전한 메모리 공간에 미리 저장해 둔 후, 인입된 번들 버전과 비교하는 방법도 생각할 수 있다. 그러나 공격자가 번들 버전을 포함한 모든 정책에 대해 다운그레이드 버전으로 덮어쓴 후 즉시 재부팅시킨다면 부팅 시 이미 변조된 번들 버전을 읽어오게 되고, 해당 버전을 정상적인 것으로 신뢰하게 된다. Integrity Checker에서 번들 버전과 정책의 조합을 기반으로 이루어진 서명을 검증한다 하더라도 /data/security/bundle/version과 메모리에 저장된 번들 버전 또한 이미 변조된 이후이므로 공격을 감지할 수 없다.

따라서 이를 보완하기 위한 방법으로 PWRM (Policy Writing Rule Monitor)를 제안한다. PWRM은 정책 파일이 저장되어 있는 /data/security/current 디렉토리에 저장되어 있는 모든 정책 파일과 /data/security/bundle/version 파일을 실시간으로 감시하고 있으며, 감시 규칙은 다음과 같다. /data/security/current에 있는 파일을 변경하는 경우, 모든 정책 파일에 앞서 /data/security/bundle/version 내용이 먼저 쓰여져야 하며, 만약 version보다 다른 파일이 먼저 쓰여질 경우 공격으로 탐지한다.

4.2.4 SPPA 동작 절차

SPPA의 전체적인 동작은 다음과 같이 이루어진다. 기기 부팅 시 init 프로세스는 libselinux 라이브러리를 통해 /data/security/current/ bundle_version 파일을 읽어 커널로 전달한다. init 프로세스는 어떠한 권한으로도 변경 불가능하도록 SEAndroid 정책으로 보호된다. 커널의 selinux file system은 이 값을 /sys/fs/selinuxfs/bundle_version에 저장한다. 이 디렉토리는 selinux 파일 시스템으로서, 커널 메모리에 저장된

값을 유저 영역에서도 접근 가능하게 하며, 유저 영역에서는 이 값을 임의로 변경할 수 없다. 이로서 향후 번들 버전 비교 시, 비교 대상으로서의 신뢰성을 확보한다. PWRM는 부팅 시부터 데몬 프로세스로 동작하기 시작하며, 정책 및 번들 버전 파일 감시는 linux의 inotify API를 통하여 수행한다. inotify는 파일 변경을 감시하여 이벤트를 수신할 수 있는 API로서, 이벤트 중 IN_CLOSE_WRITE는 쓰기 동작을 위해 열렸던 파일이 닫힐 때 통보된다. PWRM은 정책 파일이 쓰여지는 순서를 위 이벤트 감시를 통해 동작한다.

공격자가 정책 다운그레이드 공격을 포함한 정책 변조 공격에 대한 대응은 다음과 같이 이루어진다.

- bundle_version 파일 내용을 먼저 변경하고 다른 정책 파일들을 변조하는 경우 : PWRM은 bundle_version 내용이 쓰여지고 IN_CLOSE_WRITE 이벤트를 수신하는 즉시 /sys/fs/selinux/bundle_version에 저장된 버전과 새로 쓰여진 버전을 비교한다. 만약 새로 쓰여진 버전이 더 낮다면 공격이라 판단한다. 새로 쓰여진 버전이 더 높다면 PWRM은 통과하며, 신규 번들 버전을 libselinux 라이브러리를 통해서 /sys/fs/selinux/bundle에 업데이트한다. 이는 향후 버전 비교와 Integrity Checker에서의 이용을 위함이다. 이후 실제 메모리에 정책 적용 시 Integrity Checker을 이용하여 서명 검증을 수행한다. Integrity Checker는 커널에 저장된 번들 버전과 정책 파일을 조합하여 서명을 검증하며, 여기까지 통과하면 현재보다 높은 버전의 정당한 정책이므로 정책은 보호된다.
- bundle_version 이외의 정책 파일을 먼저 변조하는 경우 : PWRM 규칙에 어긋나므로 즉시 공격이라 판단하고 필요한 조치를 취한다.

공격자가 /sys/fs/selinux/bundle_version의 값을 기존 번들보다 높은 버전으로 변조함으로써 PWRM에서 수행하는 버전 비교를 회피하려는 시도를 할 수 있는데, 제안하는 번들 구조의 정책 서명은 번들 구조와 정책의 조합으로 이루어져 있으므로 Integrity Checker의 서명 검증을 통해 이러한 공격을 효과적으로 탐지할 수 있다.

정책보다 버전을 먼저 쓰게 한 이유는 재부팅과 관련이 있다. 공격자가 낮은 버전의 정책을 모두 오버라이트하기 위한 목적으로 번들 버전을 먼저 변조할 경우, 다른 정책을 변조하는 시간 동안 변조된 버전과 메모리에 저장했던 번들 버전을 비교하여 사용자 경고 등의 조치를 취할 수 있다. 그러나 다른 정책을 모두 변조하고 마지막으로 번들 버전 변조를 허용한다면, 번들 버전 변조 직후 공격자는 재부팅을 할 수 있으며, 커널 메모리에 저장되는 번들 버전은 이미 변조된 버전이 된다. 이 경우 번들 버전과 정책 조합의 서명을 Integrity Checker에서 검증하더라도 통과할 수 있으므로 번들 버전을 먼저 쓰게 하는 규칙이 필요하다.

규칙에 맞게 정책을 변조했으나 버전 비교 시 실패, 혹은 규칙에 어긋난 정책 파일 변조 순서가 발생할 경우 조치할 수 있는 방법은 여러 가지가 존재한다. 사용자 UI로 경고, 정보 유출을 방지하기 위한 네트워크 제한, 보안 감사 서버로의 통보 등이 예시가 될 수 있으며, 이에 대해서는 본 논문의 범위를 벗어나므로 다루지 않기로 한다.

V. 구현 및 실험 결과

안드로이드 운영체제는 여러 분야에 급속도로 적용되고 있으며, 그 중 대부분은 스마트폰, 웨어러블, 그리고 IoT 등과 같은 소형 기기이다. 이러한 기기는 자원이 한정되어 있는 관계로, 제안하는 방법을 적용하는 경우 요구되는 자원 소모량 및 런타임시 기기 동작에 미치는 영향은 매우 중요한 이슈이다.

본 장에서는 4장에서 제안한 기법을 실제 구현하여 실제 단말의 성능에 미치는 영향을 측정하였다. 실험 기기로는 넥서스 5(Nexus 5)를 사용하였으며, 기저 코드로는 AOSP 6.0(빌드 넘버:MRA58K)를 기반으로 사용하여 본 논문의 기법을 구현하였다.

SPPA를 위한 신규 정책 번들을 생성하기 위해서는 각 정책 파일에 대한 서명을 포함할 수 있도록 정책 번들 생성 도구를 변경해야 한다. 번들 생성 도구는 buildsebundle이라는 이름으로 소개되어 있으나[16], 오픈소스(open source)가 아니므로 코드 변경이 불가능하다. 따라서 실험을 위해 정책 번들 생성 도구를 자체적으로 제작하였고, 서명 생성 및 검증을 위한 RSA 알고리즘은 OpenSSL 라이브러리의 함수를 사용하여 구현하였다.

서명 생성을 위한 서명키는 OpenSSL을 이용하여 자체 생성하였으며, 함께 생성된 공개키는

/etc/security 디렉토리에 policy_verify_key라는 파일로 저장하여 Integrity Checker에서 서명 검증에 이용할 수 있도록 하였다.

5.1 성능 영향 평가

SEAndroid 정책 업데이트는 긴급성에 비해 그리 자주 발생하는 것은 아니다. SPPA는 정책 번들의 서명 추가로 인한 크기의 증가, 패키지 분리를 위한 추가적인 파싱, 그리고 서명 검증 단계가 추가되며, 파일 모니터링을 수행한다. 따라서 기존 시스템에 비해 부하가 발생할 수 있으며, 중요한 작업 수행 시에는 작은 부하도 문제가 될 수 있다. 따라서 본 절에서는 기존 대비 CPU 부하와 정책 업데이트 시간 증가분을 측정하여 SPPA 적용에 무리가 없음을 보이고자 한다.

실험은 다음과 같은 과정으로 수행하였다. 기존과 같은 구조로 정책 번들을 구성하여 SD card에 저장 후, adb shell을 이용하여 인텐트를 system_server로 전송하였다. system_server가 정책 업데이트를 위해 동작하는 순간부터, 각 정책 파일 중 가장 마지막으로 업데이트되는 정책의 업데이트 완료 시간을 측정하였다. SPPA가 적용된 펌웨어에서도 동일하게 수행하였으며, 오차를 줄이기 위하여 각각 100번씩 반복 측정한 후, 평균값을 산출하였다. 〔Table 1〕은 측정한 정책 업데이트 시간을 나타낸다.

또한 해당 시간 중 top 명령을 사용하여 CPU 사용량의 최고값을 측정하고 그 평균값을 산출하였으며, 〔Table 2〕는 CPU 점유율 측정 결과를 나타낸 것이다. 정책 업데이트는 유저 영역의 system_server, 커널 영역의 LSM 등과 같은 여러 부분의 연계로 이루어지므로 시스템 전체 CPU 점유율을 측정하여 비교하였다.

우선 첫 번째 실험 결과에서, SPPA 적용 하의 정책 업데이트 시간은 AOSP 대비 219.745㎳ 가 증가하였음을 볼 수 있다. SPPA는 정책 패키지를 내용과 서명으로 분리하는 과정이 추가되어 있고, 또한 Integrity Checker를 이용하여 서명 검증하는 과정이 존재한다. 따라서 정책 업데이트 시간은 증가할 수 밖에 없다. 그러나 1초 이하의 시간 증가는 일반적인 기업 환경에서 허용 가능한 수치이다.

두 번째 실험 결과에서 SPPA 적용 하의 정책 업데이트 시 CPU 점유율은 AOSP 대비 1.6% 증가하였음을 볼 수 있다. 이 또한 SPPA의 추가적인 파싱과 서명 과정이 영향을 미쳤다 추정 가능하다. 그러나 첫 번째 실험에서 알 수 있듯, SPPA를 적용한다 하더라도 정책 업데이트 시간은 0.5초를 넘지 않는다. 이 시간 동안 기존 대비 1.6%의 추가적인 CPU 점유율은 일반적인 수준에서 허용 가능한 수치이다. 또한 이 값은 전체 CPU 점유율을 측정한 것이므로 시스템의 다른 부분의 영향이 있을 수도 있다. 즉, SPPA가 사용하는 추가 CPU 점유율은 최대 1.6%이며, 그 이하일 수도 있다.

이러한 결과를 통해, 제안하는 방법이 시스템 성능에 거의 영향을 미치지 않으면서 SEAndroid의 정책을 효율적으로 보호 할 수 있음을 확인할 수 있다.

Table 1. The comparison of Policy update time

	AOSP	SPPA	increment
Duration	198.994㎳	418.739㎳	219.745㎳

Table 2. The comparison of CPU usage

	AOSP	SPPA	increment
CPU usage	42.1%	43.5%	1.6%

5.2 자원 소모량 측정

PWRM은 별도의 데몬 프로세스로 동작하고 있다. 시스템 성능 측면에서 메모리에 상주하고 있는 프로세스가 추가된다는 것은 민감한 사항이 될 수 있다. 바이너리 사이즈의 증가는 물론, 프로세스 동작에 따른 메모리 소모량이 존재하기 때문이다.

〔Table 3〕은 PWRM의 바이너리 크기와 메모리 점유량을 측정한 것이다. 메모리 점유량은 정책 업데이트 프로세스 중 측정된 최고값을 기준으로 하며, 오차를 최소화하기 위해 100번의 실험을 하여 그 평균값을 제시하였다.

안드로이드가 적용되는 대표적인 기기인 스마트폰은 현재 시점으로 16~64GB의 내부 저장 공간, 그리고 1~4GB의 내장 메모리를 탑재하고 있다. PWRM의 바이너리 크기는 1.3KB로서 내부 저장

Table 3. PWRM resource usage

	Binary size	Memory usage
PWRM	1.3KB	141KB

공간 대비 미미한 수준으로 채택 가능한 수준이다.

또한, 제조 업체에 따라 1MB 이상의 메모리를 점유하는 상주 프로세스의 추가는 꺼리는 경향이 있으나 PWRM의 메모리 점유량은 141KB로서 적용에 전혀 부담이 없다 할 수 있다.

VI. 결 론

안드로이드는 현존하는 모바일 운영체제 중 가장 높은 점유율을 보이고 있으며, 적용 및 활용 분야 또한 급격히 확대되고 있다. 이에 안드로이드 보안 강화는 필수적인 요건으로 대두되고 있으며, 그를 위한 기술 중 하나로 SEAndroid를 채택하고 있다. SEAndroid의 보안 핵심 요소는 동작의 허용 여부를 결정하는 정책이지만, 이러한 정책을 보호하는 방법에 대한 연구는 진행되지 않고 있다.

본 논문은 SEAndroid의 정책을 변조로부터 보호하기 위한 방법인 SPPA를 제안하였다. SPPA는 정책의 신뢰성과 무결성을 보장하기 위해 제안하는 정책 번들 구조에 대한 서명 생성/검증 과정을 추가하였고, 버전 다운그레이드 공격을 방어하기 위해 PWRM이라는 정책 파일 모니터링 모듈을 포함한다. SPPA는 기존 시스템에 별도의 추가 요소로 설계되었기 때문에 기존 코드의 수정을 최소화하여 적용할 수 있으며, 추가적인 하드웨어 구성 요소가 필요치 않기 때문에 효율적으로 구성할 수 있다. 또한 향후 발생할 수 있는 정책 변조와 관련된 신규 취약점에 대해 유연하게 대응할 수 있기 때문에 SEAndroid가 적용된 안드로이드의 보안을 강화하기 위한 기법으로 폭넓게 활용될 수 있을 것으로 기대된다.

References

〔1〕 Egham, "Gartner Says Worldwide Smartphone Sales Grew 3.9 Percent in First Quarter of 2016", Gartner, May. 19, 2016. http:// www.gartner.com/newsroom/id/3323017

〔2〕 Wei, X., Gomez, L., Neamtiu, I., and Faloutsos, M., "ProfileDroid: Multi-layer Profiling of Android Applications", Proceedings of the 18th annual international conference on Mobile computing and networking, pp. 137- 148, Aug. 2012.

〔3〕 M Mohamed, B Shrestha, N Saxena, "SMASheD: Sniffing and Manipulating Android Sensor Data", Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, pp. 152-159, Mar. 2016.

〔4〕 Smalley, Stephen, Chris Vance, and Wayne Salamon, "Implementing SELinux as a Linux Security Module", NAI Labs Report, vol. 1, no. 43, pp. 139-213, May. 2002.

〔5〕 Smalley, Stephen, and Robert Craig, "Security Enhanced(SE) Android: Bringing Flexible MAC to Android", NDSS, vol. 310, pp.20-38, Feb. 2013.

〔6〕 S. Mutti, B. Enrico and P. Stefano, "An SELinux-based intent manager for Android." Communications and Network Security (CNS), 2015 IEEE Conference on IEEE, pp. 747-748, Sep. 2015.

〔7〕 P. Mateti, "NSA Security Enhanced Andr oid", Wright State University, Oct. 2014. http://cecs.wright.edu/ ~pmateti/Courses/4900/Lectures/Security/NSA-SE-Android/nsa-se.html

〔8〕 Jann Horn, "CVE-2014-7911: Android <5. 0 Privilege Escalation using ObjectInput Stream", Nov. 15. 2014. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-7911

〔9〕 O Peles, R Hay, "One class to rule them all: 0-day deserialization vulnerabilities in android", WOOT'15 Proceedings of the 9th USENIX Conference on Offensive Technologies, pp. 5-5, Aug. 2015.

〔10〕 Android, "Android XRef Marshmallow 6.0.1_r10", Google, http://androidxref.com/6.0.1_r10/xref/external/sepolicy/domain.te#221

〔11〕 "Android Wear", Wikipedia, Mar. 2014. https://en.wikipedia.org/wiki/Android_Wear

〔12〕 "Android Auto", Wikipedia, June 2015. htt

ps://en.wikipedia.org/wiki/Android_Auto
〔13〕 Android, "Brillo", Google, May 2015. https://developers.google.com/brillo/
〔14〕 Android, "Android: Android Compatibility", Google. https://source.android.com / compatibility/
〔15〕 Android, "Android 4.4 Compatibility Definition", Google, https://static.googleusercontent.com/ media/source.android.com/ko//compatibility/4.4/android-4.4-cdd.pdf
〔16〕 R. Haines, "SELinux Project Wiki: NBSEforAndroid 2, Build Bundle Tools", SELinux Project Wiki, http://selinuxproject.org/page/NB_SEforAndroid_2#Build_Bundle_Tools
〔17〕 Android, "Android: Implementing SELinux#Use cases", Google, http://source.android.com/security/ selinux/implement.html#use_cases
〔18〕 Samsung Knox, "Samsung Knox: Real-time Kernel Protection(RKP)", Samsung, https://www2. samsungknox.com/en/blog/real-time-kernel-protection-rkp

〈저자소개〉

유 석 만 (Seok-man Yoo) 정회원
2004년 8월: 고려대학교 전기전자전파공학부 졸업
2004년 7월~현재: 삼성전자 무선사업부 재직
2015년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 암호프로토콜, 모바일 보안, 드론 보안

박 진 형 (Jin-Hyung Park) 학생회원
2010년 2월: 건국대학교 컴퓨터공학과 졸업
2012년 2월: 고려대학교 정보보호학과 석사
2012년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 암호프로토콜, 모바일 보안, 암호 알고리즘, 스토리지 보안

이 동 훈 (Dong-Hoon Lee) 종신회원
1983년 8월: 고려대학교 경제학사 졸업
1987년 12월: Oklahoma University 전산학과 석사
1992년 5월: Oklahoma University 전산학과 박사
1993년 3월~1997년 2월: 고려대학교 전산학과 조교수
1997년 3월~2001년 2월: 고려대학교 전산학과 부교수
2001년 3월~2015년 2월: 고려대학교 정보보호대학원 교수
2016년 3월~현재: 고려대학교 정보보호대학원 원장
〈관심분야〉 암호프로토콜, 암호이론, 자동차 보안, 난독화, PET 기술