논문 2016-11-38

어플리케이션의 가상 메모리 보호를 위한 연구

(A Study for Protecting the Virtual Memory of Applications)

김 동 율, 문 종 섭*
(Dong-Ryul Kim, Jong-sub Moon)

Abstract : As information technology advances rapidly, various smart devices are becoming an essential element in our lives. Smart devices are providing services to users through applications up on the operating system. Operating systems have a variety of rules, such as scheduling applications and controlling hardwares. Among those rules, it is significant to protect private information in the information-oriented society. Therefore, isolation task, that makes certain memory space separated for each application, should highly be guaranteed. However, modern operating system offers the function to access the memory space from other applications for the sake of debugging. If this ability is misused, private information can be leaked or modified. Even though the access authority to memory is strictly managed, there exist cases found exploited. In this paper, we analyze the problems of the function provided in the Android environment that is the most popular and opened operating system. Also, we discuss how to avoid such kind of problems and verify with experiments.

Keywords : Virtual memory protection, /proc/PID/mem, Ptrace system call, Android memory protection, Linux memory protection

Ⅰ. 서 론

현대의 삶은 스마트 장치와 깊은 유대 관계를 형성하고 있다. 스마트 폰으로 전화, 문자 등의 기본적인 기능 외에도 틈틈이 인터넷, 게임 등의 서비스를 즐기며, TV의 채널을 변경하기 위해 손동작이나 음성을 이용한다. 이러한 서비스는 스마트 장치가 어플리케이션 (이하 앱)을 통해서 사용자에게 제공한다. 앱은 운영체제에 의해 관리되는데, 운영체제의 여러 역할 중에서 근래에 더욱 강조되는 역할이 바로 정보 보호이다. 운영체제는 앱이 다루는 중요 데이터를 보호하기 위해 각 앱에게 독립된 메모리 공간인 가상 메모리 [1]를 제공하는데, 많은 운영체제가 디버깅 등의 편의를 위해 한 앱이 다른 앱의 가상 메모리에 접근할 수 있는 기능을 제공한다. 이러한 기능은 앱 개발을 위해선 반드시 필요한 기능이지만, 악의적인 의도로 악용될 경우엔 중요 정보의 유출 및 조작 등 막대한 피해를 초래할 수 있다. 따라서 운영체제는 이런 기능을 제공할 때 의도했던 용도로만 사용되도록 강력하게 보호를 하거나 적어도 앱에게 자신의 가상 메모리를 보호할 수 있는 기회를 제공해야 한다. 하지만 많은 사람들이 사용하는 운영체제에서도 가상 메모리를 보호하는 역할에 있어 결점이 존재하는 경우가 있다. 이에 본 논문에서는 안드로이드 오픈 소스 프로젝트 (Android Open Source Project, AOSP)에 의해 소스코드가 공개되어 있고, 2014년 1월 21일을 기준으로 한국의 스마트 폰 운영체제 시장에서 93.4%의 점유율을 차지한 안드로이드 [2]를 대상으로 다른 앱의 가상 메모리에 접근할 수 있는 기능의 문제점을 분석한 후에 해결 방안을 제시한다. 이후에는 제시한 방안을 적용하기 전후로 안드로이드에서 앱의 가상 메모리를 조작하는 실험을 수행함으로써 본 논문이 제시한 방안의 타당성을 검증한다.

본 논문은 7장으로 이루어져 있다. 2장에서는 안드로이드에서 앱의 메모리로부터 민감한 데이터를 추출한 기존 연구와 다른 앱의 가상 메모리에 있는 데이터를 조작하는 앱에 대해 소개한다. 3장에서는 다른 앱의 가상 메모리에 접근할 수 있는 기

*Corresponding Author (jsmoon@korea.ac.kr)
Received: Sep. 9 2016, Revised: Oct. 11 2016, Accepted: Oct. 12 2016.
D.R. Kim, J.S. Moon: Korea University

 2016 Dec. 11(6) 335-341
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2016.11.6.335

능을 소개하며, 4장에서는 해당 기능의 문제점을 분석한다. 5장에서는 문제점을 해결할 수 있는 방안을 제시하며, 6장에서는 실험을 통해서 본 논문에서 제안하는 방안의 타당성을 검증한다. 마지막 7장에서는 결론을 맺음으로써 논문을 마무리한다.

Ⅱ. 관련 연구

최근의 몇몇 연구는 대표적인 모바일 운영체제인 안드로이드에서 앱의 메모리 영역을 덤프하면 사용자의 민감한 데이터를 추출할 수 있음을 보이고 있다. 이는 만약 공격자가 악성 앱을 통해 다른 앱의 가상 메모리를 추출한다면, 연구에서와 마찬가지로 사용자의 민감한 데이터를 획득할 수 있음을 의미한다.

D. Apostolopoulos et al. 은 오픈 소스 도구인 Dalvik Debug Monitor Server (DDMS) [3]로 안드로이드 앱에서 가상 메모리를 덤프 한 후에 두 가지 실험을 수행하였다 [4]. 첫 번째 실험은 앱을 통해 전송한 아이디와 비밀번호를 메모리 덤프에서 찾는 것이다. 이 실험은 13개의 은행 앱, 쇼핑 앱, SNS 및 채팅 앱 중에서 12개의 앱이 메모리상에서 아이디를 보호하지 않고, 비밀번호 관리 앱을 포함한 30개의 앱 중 29개의 앱이 사용자의 비밀번호를 보호하지 않는다는 것을 증명한다. 이 실험 결과는 만약 공격자가 악성 앱으로 실험 대상 앱의 가상 메모리에 접근한다면 사용자의 아이디와 비밀번호와 같은 중요 개인 정보를 탈취할 수 있음을 의미한다. 두 번째 실험은 가상 메모리에서 개인정보가 위치하는 패턴을 파악하는 것인데, 사용자의 개인정보 주변에는 “password”, “@Paw”, “username” 등의 문자열이 위치하고 있어 메모리에서 개인정보를 찾는 것이 어렵지 않음을 증명한다.

P. Stirparo et al. 또한 안드로이드 앱의 가상 메모리를 덤프하면 사용자의 개인 정보를 탈취 할 수 있음을 보인다 [5]. 이들의 연구에서 특히 주목할 점은 앱의 가상 메모리를 덤프하기 위해 자체 제작한 트로잔 앱을 이용하는데, 이 앱은 표면상으로는 정상 앱과 차이가 없고 시스템에 미치는 성능상의 영향도 미비하여 사용자 입장에서 악성 앱임을 판단하기가 쉽지 않다는 것이다.

Z. Ding et al. 은 중국의 인기 메신저인 Wechat [6]을 대상으로 메모리를 덤프하여, 사용자의 중요 메시지나 이미 삭제한 메시지뿐만 아니라 SD 카드에는 암호화 되어 저장된 메시지를 평문 상태로 추출할 수 있음을 보인다 [7]. 이상의 기존 연구가 제시하는 사례를 방지하기 위해선 앱의 가상 메모리를 보호할 수 있는 연구를 반드시 수행해야 한다.

기존의 연구 외에도 안드로이드에선 앱의 가상 메모리를 위협하는 메모리 조작 앱이 존재한다. 앱에서 데이터는 개발자가 사용하는 방법에 따라서 스택 영역이나 힙 영역에 저장된다. 개발자는 데이터에 접근하기 위해 변수를 사용하는데, 변수를 사용할 때마다 재할당하지 않는 이상 데이터는 스택이나 힙 영역에서 동일한 주소에 위치한다. 메모리 조작 앱은 특정 데이터가 위치한 주소를 찾아서 조작할 수 있다. 대표적인 예로는 “Cheat Engine [8]”, “GAMEGUARDIAN [9]”, “SB Game Hacker [10]”가 있다.

안드로이드 사용자들이 메모리 조작 앱을 주로 사용하는 곳은 게임 앱이다. 게임 앱의 대부분은 수익을 거두기 위해 부분 유료화 정책을 사용한다. 사용자가 결제를 하면 게임 상의 금전을 제공하거나 아이템을 제공하는 방식인데, 악의적인 사용자는 메모리 조작 앱으로 주요 데이터를 조작함으로써 결제 없이 부당 이익을 취할 수 있다. 혹은 자신의 점수와 같은 데이터를 조작하기 위해 사용하기도 한다. 물론 게임 서버 측에서 중요 데이터에 대한 검증을 수행한다면 어느 정도 예방을 할 수는 있지만 모든 데이터를 검증하기란 쉽지 않을뿐더러 불가피하게 오버 헤드가 발생한다.

Ⅲ. 가상 메모리 접근 기능

1. 가상 메모리

가상 메모리란 앱에게 주어진 추상화 된 메모리 공간을 의미한다. 운영체제는 앱에게 독립된 가상 메모리를 제공함으로써 여러 앱이 동시에 동작하는 것처럼 보이게 하는 멀티태스킹 기능과 실제 주기억장치보다 큰 메모리 영역을 제공한다. 앞의 두 기능 외에도 운영체제가 가상 메모리 기법을 사용하는 중요한 이유는 가상 메모리 기법을 통해서 각 앱의 메모리 영역이 다른 앱에 의해 침해되지 않도록 보호하기 위해서이다. 안드로이드는 리눅스를 근간으로 하는 모바일 운영체제이며, 앱을 리눅스의 프로세스로 운영한다. 따라서 안드로이드에서 다른 앱의 가상 메모리에 접근할 수 있는 기능을 분석하기 위해선 리눅스를 분석해야 한다. 리눅스는 디버깅을 위해 다른 앱의 가상 메모리에 접근할 수 있도록 두 가지 방법을 지원한다.

2. 접근 방법 1 - ptrace 시스템 콜

리눅스에서 다른 앱의 가상 메모리에 접근할 수 있는 첫 번째 방법은 ptrace 시스템 콜 (이하 ptrace) [11]을 이용하는 것이다. ptrace는 process trace의 약자로, 다른 앱의 실행이나 시그널을 제어할 수 있고 메모리 및 레지스터에 존재하는 데이터를 확인하거나 수정할 수 있다. 이 시스템 콜은 리눅스의 대표적인 디버거인 GDB와 같은 디버거가 디버깅을 목적으로 주로 사용한다. ptrace는 인자로 전달하는 플래그에 따라 동작이 달라진다. 다른 앱의 가상 메모리에서 데이터를 읽기 위해선 PTRACE_PEEKTEXT나 PTRACE_PEEKDATA 플래그를 이용하며, 데이터를 수정하기 위해선 PTRACE_POKETEXT나 PTRACE_POKEDATA 플래그를 이용한다.

3. 접근 방법 2 - /proc/PID/mem 가상 파일

다른 앱의 가상 메모리에 접근할 수 있는 두 번째 방법은 /proc/PID/mem 가상 파일을 이용하는 것이다. 리눅스는 앱 및 시스템의 정보를 관리하기 위해 proc 가상 파일 시스템을 운영한다. 이 파일 시스템에는 리눅스가 앱을 관리하기 위해 부여한 프로세스 ID (PID)별로 디렉터리가 존재하는데, 디렉터리에는 앱의 가상 메모리를 보여주는 mem 파일이 존재한다. 다른 앱의 mem 가상 파일에 접근하기 위해선 일반 파일에 접근하는 것과 동일하게 open, read, write 와 같은 함수를 이용한다. 하지만 앱은 운영체제가 부여한 주소 공간에서 모든 영역을 사용하는 것이 아니기 때문에 사용하지 않는 영역에 대해 read나 write 함수를 사용하면 오류가 발생한다. 앱이 실제로 사용하는 가상 메모리 영역은 /proc/PID/maps 가상 파일에 기록되어 있다.

Ⅳ. 문제점 분석

리눅스가 제공하는 2가지 접근 방법 중 악성 앱이 첫 번째 방법을 이용하여 다른 앱의 가상 메모리에 접근하는 경우는 대처가 가능하다. ptrace는 앞서 소개한 플래그 외에도 PTRACE_TRACEME 플래그를 제공한다. 앱이 이 플래그를 이용해서 ptrace를 사용한다는 것은 부모 앱에 의해서 추적이 된다는 것을 의미한다. 이때 부모 앱이 정상적으로 자식 앱을 추적하기 위해선 CAP_SYS_PTRACE capability [12]를 가지고 있어야 한다. 안드로이드의 경우엔 모든 사용자 앱의 부모 프로세스인 zygote가 안드로이드 버전에 따라 CAP_SYS_PTRACE capability를 가질 수도 있고 가지지 않을 수도 있다. zygote가 CAP_SYS_PTRACE capability를 가지고 있는 경우엔 앱이 시작과 함께 ptrace를 PTRACE_TRACEME 플래그로 호출하면 악성 앱이 가상 메모리에 접근하는 것을 방지할 수 있다. 이 방법이 가능한 이유는 리눅스는 한 번에 하나의 앱이 다른 앱을 추적할 수 있도록 제한하기 때문이다. 다만 이 경우엔 시그널 처리에 대한 권한이 zygote 프로세스로 전달되는 등의 부작용이 발생할 수 있다. zygote 앱이 CAP_SYS_PTRACE capability를 가지지 않은 경우엔 메모리를 보호하려는 앱이 주기적으로 ptrace를 호출하면서 에러 코드를 확인해야 한다. zygote가 권한을 가지고 있지 않은 상태에서 앱이 ptrace를 호출하면 ptrace는 에러를 반환하면서 권한이 없다는 것을 의미하는 EACCESS 에러 코드를 함께 반환한다. 하지만 악성 앱이 정상 앱의 가상 메모리에 접근한 상태에서 정상 앱이 ptrace를 호출하면 EPERM 에러 코드를 반환한다. 이를 통해서 정상 앱은 강제 종료 등의 대처를 수행할 수 있다.

이에 반해 mem 가상 파일을 이용한 접근은 정상 앱이 능동적으로 대처할 수 있는 방안이 없다. mem 가상 파일을 이용하여 프로세스의 가상 메모리를 추출한 사례는 memfetch [13]에서 제시되었다. memfetch는 가상 메모리를 추출하기 위해 mem 가상 파일과 ptrace를 조합하여 사용하는데, 이런 경우엔 앞서 소개한 것처럼 ptrace 접근에 대해 대처를 하면 된다. 하지만 mem 가상 파일에 접근할 때 ptrace를 함께 사용하는 것은 데이터를 추출할 때 안정성을 더해줄 뿐이지 mem 가상 파일을 이용할 때 ptrace 시스템 콜이 반드시 필요한 것은 아니다. 만약 정상 앱이 메모리상에서 다른 앱이 추출하거나 조작할 만큼 충분한 시간동안 중요 데이터를 유지한다면 mem 가상 파일은 정보 보호의 관점에서 커다란 위협이 될 수 있다.

Ⅴ. 제안 사항

운영체제가 악성 앱으로부터 정상 앱의 가상 메모리를 지킬 수 있는 방법은 크게 2가지이다. 첫 번째는 악성 앱이 정상 앱의 가상 메모리에 접근하지 못하도록 해당 기능을 제거하는 것이다. 하지만 운영체제에서 동작하는 앱을 개발하기 위해선 디버깅 기능이 필수적이기 때문에 완전히 제거하는 것은

현실적으로 쉽지가 않다. 이에 대한 대안으로 대부분의 운영체제는 해당 기능에 대한 접근제어를 수행함으로써 기능이 남용되는 것을 막는다. 앞서 살펴본 리눅스도 다른 앱의 /proc/PID/mem 가상 파일에 접근할 때 kernel/fork.c 파일에 있는 mm_access 함수로 접근제어를 수행한다. 다른 앱이 해당 접근제어를 통과할 수 있는 경우는 부모 앱이 자식 앱의 가상 메모리에 접근하거나, 최고 권한인 루트 권한을 획득한 앱이 다른 앱의 가상 메모리에 접근하는 경우이다.

하지만 운영체제가 근본적으로 이 기능을 제거하는 것이 아니라면 접근제어를 강화하는 것만으로 앱의 가상 메모리를 완벽하게 보호하는 것은 힘들다. 안드로이드도 버전을 업그레이드 하면서 루트 권한을 획득할 수 있는 취약점들을 지속적으로 보완하였다. 그럼에도 불구하고 가상 메모리를 완벽하게 보호하기가 힘든 이유는 운영체제는 사용자와 상호작용하면서 시스템을 운영하기 때문이다. 안드로이드는 각 버전 별로 사용자가 루트 권한을 획득할 수 있는 방법이 존재한다. 이를 바탕으로 사용자는 안드로이드를 자신의 입맛에 맞게 변형하기 위해 루트 권한을 요구하는 앱이나 메모리 조작 앱을 설치한다. 메모리 조작 앱은 게임 앱과 같이 앱 개발자의 수익에 관련된 데이터를 조작하기 위해 사용할 수 있으며, 편의를 위해 설치했던 루트 권한을 요구하는 앱은 사용자 몰래 다른 앱으로부터 사용자의 개인정보를 탈취할 수 있다. 운영체제는 이러한 상황들에 대해서도 최소한의 내성을 갖추고 있어야 한다.

가장 이상적으로 가상 메모리를 보호할 수 있는 방법은 접근 제어를 강화하는 현대 운영체제의 추세에 본 논문이 제안하는 두 번째 방법을 결합하는 것이다. 두 번째 방법은 악성 앱이 정상 앱의 가상 메모리에 접근할 때 운영체제가 정상 앱에게 이를 통보함으로써 앱에게 허용 여부를 결정하게 하는 것이다. zygote가 CAP_SYS_PTRACE capability를 가지지 않은 환경에서 정상 앱이 ptrace를 PTRACE_TRACEME로 호출하는 경우가 이에 해당한다고 볼 수 있다. ptrace는 리눅스가 명시적으로 정상 앱에게 통보를 하는 것은 아니지만, 정상 앱이 에러 코드를 받아볼 수 있는 방법을 제공한다. 하지만 악성 앱이 정상 앱의 mem 가상 파일에 접근할 때에는 리눅스가 정상 앱에게 ptrace의 에러 코드와 같은 정보를 일절 제공하지 않는다. 만약 이러한 상황에서 리눅스가 악성 앱의 접근을 정상 앱에게 통보한다면, 정상 앱은 ptrace의 경우처럼 강제 종료와 같은 대응을 할 수가 있다.

본 논문이 안드로이드 환경에서 제안하는 사항은 악성 앱이 다른 앱의 /proc/PID/mem 가상 파일에 접근할 때, 안드로이드의 근간이자 앱의 가상 메모리를 보호해야 할 의무를 지니고 있는 리눅스 커널이 /proc/PID/mem 가상 파일을 소유하는 앱에게 시그널을 보냄으로써 가상 메모리에 접근 하는 앱의 존재 여부를 알리도록 하는 것이다. 이를 통해 자신의 가상 메모리를 보호하고자 하는 앱은 리눅스의 통보를 바탕으로 대책을 강구할 수 있다.

Ⅵ. 실 험

1. 실험 환경

본 장에서는 악의적인 사용자를 가장하여 앞서 설명한 문제점 중 하나를 직접 재현한다. 이후에는 제시한 방안을 적용함으로써 문제점을 해결할 수 있음을 보인다. 실험은 레퍼런스 폰인 Nexus 5를 위주로 진행한다. 레퍼런스 폰은 스마트 폰 제조업체가 안드로이드 운영체제를 자사 제품에 맞게 특화할 때 참고하는 장치이다. 따라서 레퍼런스 폰의 안드로이드 및 리눅스 커널 소스코드는 완전히 공개되어 있기 때문에 실험에 적합하다. 실험에 사용할 안드로이드의 버전은 4.4.4 (Kitkat)이며, 리눅스 버전은 3.4버전이다. 안드로이드 4.4.4는 Nexus 5에서 처음으로 소개된 버전으로써 2016년 9월 7일을 기준으로 가장 높은 점유율을 차지하고 있다[14]. 리눅스 3.4버전은 Nexus 5가 지원하는 가장 최신 버전인데, 근래에 출시된 스마트 폰은 리눅스 3.18버전을 사용한다. 다른 앱의 가상 메모리에 접근하기 위해 mem 가상 파일을 대상으로 open 함수를 호출하면, 리눅스는 fs/proc/base.c에 있는 mem_open 함수를 호출한다. mem_open 함수는 kernel/fork.c에 있는 mm_access 함수로 접근제어를 수행하는데, 리눅스 3.4버전과 3.18버전은 접근제어에서만 약간의 차이를 보일 뿐이지 수행하는 기능면에서는 별다른 차이가 없다. 따라서 본 논문의 실험 내용은 상위 버전의 리눅스에서도 동일하게 적용할 수 있다.

실험에서 사용할 앱은 두 가지이다. 첫 번째 앱은 2.2절에서 소개한 메모리 조작 앱으로써 안드로이드의 리눅스 커널을 수정하기 전후로 정상 앱의 가상 메모리를 조작하는 역할을 수행한다. 메모리 조작 앱이 다른 앱의 가상 메모리에 접근할 때 mem 가상 파일을 이용하는 것을 확인하기 위해

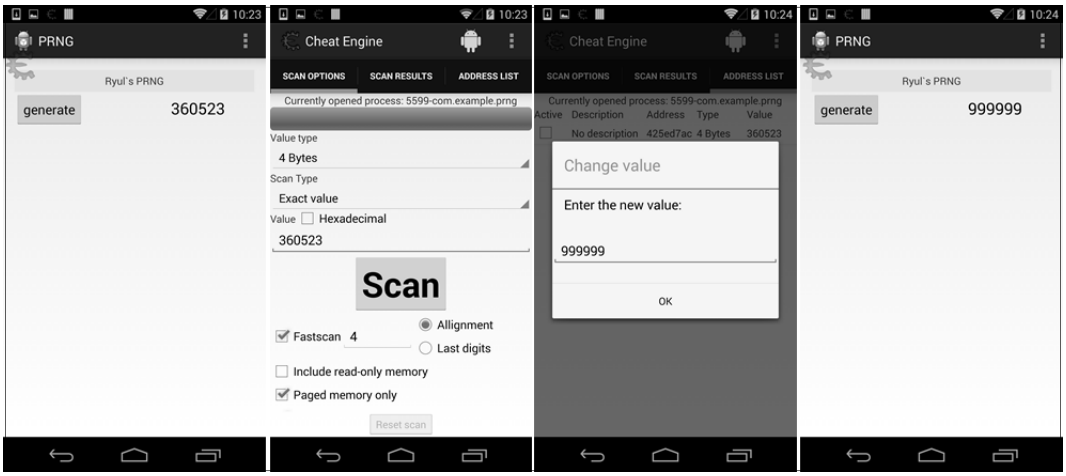

그림 1. 실험용 앱의 가상 메모리를 조작한 결과

Fig. 1 The result of modifying the virtual memory of the experimental app

mem_open 함수와 mem_read 함수를 수정하여 printk 함수로 mem 가상 파일을 소유하는 앱의 프로세스 ID를 로그로 출력한다. mem_read 함수는 mem 가상 파일에 대해 read 함수를 사용할 때 호출된다. printk가 출력하는 로그는 /proc/kmgs에서 확인할 수 있다. 두 번째 앱은 정상 앱으로써 실험을 위해 자체 제작한 앱이다. 이 앱은 버튼을 누르면 임의의 숫자를 생성해서 화면에 출력하며, 메모리 조작 여부를 판단할 수 있도록 초점이 돌아올 때 현재 메모리에 저장된 값으로 화면을 갱신한다.

2. 실험 결과

첫 번째 실험은 mem_open 함수와 mem_read 함수에서 로그만 출력하도록 수정하고 루트 권한을 획득한 후에 메모리 조작 앱으로 실험용 앱의 가상 메모리를 조작한다.

그림 1의 첫 번째 캡처 화면은 실험용 앱으로 generate 버튼을 터치하자 360523이라는 임의의 값을 출력한다. 두 번째 및 세 번째 캡처 화면은 메모리 조작 앱 중의 하나인 Cheat Engine을 이용하는 화면으로, 실험용 앱이 출력한 360523의 메모리 주소를 파악하여 이를 999999로 수정하는 화면이다. 마지막 캡처 화면은 초점이 실험용 앱으로 돌아왔을 때 실험용 앱이 화면을 갱신하자 메모리 조작 앱으로 수정한 999999가 출력되는 그림이다.

실험에서 사용한 메모리 조작 앱이 /proc/PID/mem 가상 파일을 사용하는지 확인하기 위해 /proc/kmsg의 커널 로그를 확인하면 그림 2와 같다. 그림 2의 로그에 따르면 프로세스 ID가 5599인 앱에 대해 mem_open 함수와 mem_read 함수가 호출되는 것을 확인할 수 있다. 실험을 진행 중인 Nexus 5에 adb [15]를 이용하여 쉘에 접근한 후, 프로세스 ID가 5599인 앱을 확인하면 그림 3과 같이 실험용 앱인 prng앱 임을 확인할 수 있다.

```
<1>[ 1450.222711] target_pid: 5599 (mem_open)
<1>[ 1450.224204] target_pid: 5599 (mem_read)
<1>[ 1450.224747] target_pid: 5599 (mem_read)
<1>[ 1450.226572] target_pid: 5599 (mem_read)
<1>[ 1450.226786] target_pid: 5599 (mem_read)
<1>[ 1450.227055] target_pid: 5599 (mem_read)
<1>[ 1450.227434] target_pid: 5599 (mem_read)
```

그림 2. /proc/kmsg 로그 확인 결과

Fig. 2 /proc/kmsg log messages

표 1. 다양한 환경에서의 실험 결과

Table 1. The result of experiments on other environments

Tested Devices	Android version	Linux kernel version	Memory manipulation
Nexus 5	5.0	3.4.0	Yes
Nexus 5	6.0	3.4.0	Yes
Samsung Galaxy S4	4.4.4	3.4.0	Yes
LG G Pro	4.4.4	3.4.0	Yes

정상 앱의 메모리를 조작하는 실험은 다양한 환경에서 적용이 가능하다. 표 1은 추가 실험을 진행한 환경과 그 결과를 나타낸다. 추가 실험은 Nexus 5에서 안드로이드 버전을 업그레이드 한 후와 국내의 유명 브랜드인 삼성과 LG 핸드폰에서 동일한 실험 환경을 구성한 후에 진행하였다.

```
root@hammerhead:/ # ps | grep 5599
u0_a78    5599  177   909768 40696 ffffffff 400a873c S com.example.prng
```

그림 3. PID가 5599인 앱을 확인한 결과
Fig. 3 The result of checking PID 5599

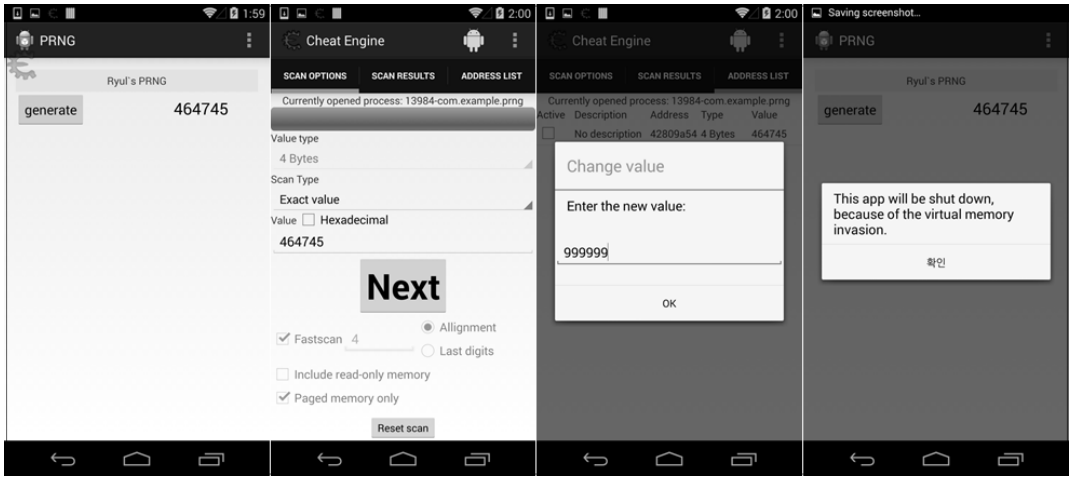

그림 4. 두 번째 실험에서 PRNG 앱의 가상 메모리를 조작한 결과
Fig. 4 The result of modifying the virtual memory of PRNG on the second experiment

두 번째 실험은 첫 번째 실험에서 두 가지를 수정한 후에 진행한다. 첫 번째 수정 사항은 리눅스 커널의 mem_open 함수에서 접근제어를 수행하는 mm_access 함수의 다음 줄에 특정 프로세스에게 시그널을 보내는 sys_kill 함수를 추가하는 것이다. sys_kill 함수를 mm_access 함수가 아닌 mem_open 함수에 추가하는 이유는 기존의 커널에 영향력을 최소화하기 위해서이다. 리눅스 커널 3.18 버전을 기준으로 mm_access 함수를 호출하는 다른 커널 함수는 존재하지만 [16], mem_open 함수를 호출하는 다른 함수는 존재하지 않다 [17]. sys_kill 함수를 추가하는 것은 악성 앱이 다른 앱의 mem 가상 파일에 대한 접근제어를 통과하더라도 운영체제가 악성 앱의 존재를 다른 앱에게 알려주는 역할을 한다. 현재의 리눅스 커널에는 디버깅 기능에 대한 통보를 위한 시그널이 따로 없기 때문에 실험에서는 SIGUSR2 시그널을 사용한다. 두 번째 수정 사항은 실험용 앱이 제안 사항을 적용한 리눅스 커널의 시그널을 처리할 수 있도록 시그널 핸들러를 등록하는 것이다. 안드로이드 앱은 자바로 작성된 앱이기 때문에 기본적으론 시그널을 처리할 수 없다. 따라서 시그널 핸들러를 네이티브 라이브러리로 작성한 후에 이를 NDK [18]과 JNI [19]로 실험용 앱에 이식해야 한다.

두 번째 실험도 실험 과정은 첫 번째 실험과 동일하다. 하지만 첫 번째 실험과는 달리 메모리 조작 앱이 실험용 앱의 가상 메모리에 접근할 때 리눅스 커널이 이를 통보해 주기 때문에 그림 4와 같이 실험용 앱을 강제 종료시키는 대응을 할 수가 있다.

Ⅶ. 결 론

본 논문에서는 안드로이드 운영체제를 예로 들어 앱의 가상 메모리가 운영체제로부터 제대로 보호받지 않을 수 있다는 것을 살펴보고, 이에 대한 대응책을 제시하였다. 물론 앱의 가상 메모리를 보호하는 가장 좋은 방법은 악성 앱이 다른 앱의 가상 메모리를 침해하기 전에 예방하는 것이다. 하지만 앞서 논의 했던 것처럼 운영체제엔 디버깅 기능이 필요하기 때문에 안드로이드를 비롯한 운영체제들은 디버깅에 필요한 권한을 강화하는 방향으로 진화하고 있다. 안드로이드의 경우에도 버전이 업그레이드될 때마다 이전 버전에 존재했던 루팅 방법이 불가능하도록 패치를 수행하고 있다. 그러나 앱의 가상 메모리를 보호하기 위해선 루트 권한을 막는 것에만 의존해선 안 된다. 안드로이드도 5.0 버

전을 출시할 때 루트 권한을 막기 위해 노력했었지만 결국 루팅 방법이 등장했으며, 6.0 버전도 부트 이미지를 수정하는 루팅 방법이 등장하였다. 따라서 각 운영체제는 가상 메모리에 대한 접근제어를 강화하는 것 외에도 본 논문에서 제안한 바와 같이 앱의 가상 메모리를 보호할 수 있는 최소한의 장치를 마련하는 노력을 해야 한다.

References

[1] A.S. Tanenabum, “Modern operating systems 3rd,” Pearson, pp. 186-187, 2014.

[2] http://koreajoongangdaily.joins.com/news/article/Article.aspx?aid=2983882

[3] https://developer.android.com/studio/profile/ddms.html

[4] D. Apostolopoulos, G. Marinakis, C. Ntantogian, C. Xenakis, “Discovering authentication credentials in volatile memory of android mobile devices,” Proceedings of Conference on International Federation for Information Processing, pp. 178-185, 2013.

[5] I.N. Fovino, M. Taddeo, I. Kounelis, “In-memory credentials robbery on android phones,” Proceedings of IEEE World Congress on Internet Security, pp. 88-93, 2013.

[6] http://www.wechat.com/en/

[7] F. Zhou, Y. Yang, Z. Ding, G. Sun, “Dump and analysis of android volatile memory on wechat,” Proceedings of IEEE International Conference on Communications, pp. 7151-7156, 2015.

[8] http://www.cheatengine.org/

[9] https://gameguardian.net/forum/

[10] http://sbgamehacker-apk.com/

[11] http://man7.org/linux/man-pages/man2/ptrace.2.html

[12] http://man7.org/linux/man-pages/man7/capabilities.7.html

[13] https://github.com/citypw/lcamtuf-memfetch

[14] https://en.wikipedia.org/wiki/Android_(operating_system)

[15] https://developer.android.com/studio/command-line/adb.html

[16] http://lxr.free-electrons.com/ident?v=3.18&i=mem_open

[17] http://lxr.free-electrons.com/ident?v=3.18&i=mm_access

[18] https://developer.android.com/ndk/index.html

[19] https://developer.android.com/training/articles/perf-jni.html

Dong-Ryul Kim (김 동 율)

He is received B.E. degree in computer engineering from Hongik University in 2015. He is currently enrolled in Graduate school of information security at Korea University. He interested in the field of Android O.S, Embedded device security, Vulnerability analysis.

Email: pixxi242@naver.com

Jong-Sub Moon (문 종 섭)

Professor Moon received his Ph.D. degree in computer science from Illinois Institute of Technology, U.S.A in 1991 and his M.S. degree and B.S. degree from Seoul University, Korea in 1983. He is currently a professor at information engineering, Korea University, Seoul, Korea. His research interests are system security, network security and pattern recognition, especially machine learning.

Email: jsmoon@korea.ac.kr