Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.2, Apr. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.2.227

ARM 캐시 일관성 인터페이스를 이용한 안드로이드 OS의 스크린 잠금 기능 부채널 공격*

김 영 필,[1†] 이 경 운,[2] 유 시 환,[3‡] 유 혁[4]
[1]인천대학교 (교수), [2]경북대학교 (교수), [3]단국대학교 (교수), [4]고려대학교 (교수)

Side-Channel Attack of Android Pattern Screen Lock Exploiting Cache-Coherent Interface in ARM Processors*

Youngpil Kim,[1†] Kyungwoon Lee,[2] Seehwan Yoo,[3‡] Chuck Yoo[4]
[1]Incheon National University (Professor), [2]Kyungpook National University (Professor),
[3]Dankook University (Professor), [4]Korea University (Professor)

요 약

안드로이드 OS의 패턴 스크린 잠금 기능은 가장 일반적으로 사용되는 사용자 인증 기법이다. 현재 사용되는 패턴 스크린 잠금 기법은 패턴의 종류에 따라 약 39만개의 조합이 가능하며, 잘못된 입력 시 입력 지연 등의 기법이 적용되어 무작위 대입 공격으로는 쉽게 공격하기 어렵다. 이 논문에서는 ARM 기반의 멀티코어 시스템에서 사용하는 하드웨어의 구성 요소 중 캐시 일관성 인터페이스가 패턴의 종류를 파악할 수 있는 부채널이 될 수 있음을 보인다. 이러한 하드웨어 부채널을 이용하여 스크린 잠금 패턴의 꺾임 횟수와 전체 길이를 유추할 수 있으며, 이를 통해 공격의 효율성이 매우 높아질 수 있음을 제시한다.

ABSTRACT

This paper presents a Cache-Coherency Interconnect(CCI)-based Android pattern screen lock(PSL) attack on modern ARM processors. CCI has been introduced to maintain the cache coherency between the big core cluster and the little core cluster. That is, CCI is the central interconnect inside SoC that maintains cache coherency and shares data. In this paper, we reveal that CCI can be a side channel in security, that an adversary can observe security-sensitive operations. We design and implement a technique to compromise Android PSL within only a few attempts using the information of CCI in user-level applications on Android Nougat. Further, we analyzed the relationship between the pattern complexity and security. Our evaluation results show that complex and simple patterns would have similar security strengths against the proposed technique.

Keywords: Pattern screen lock, Cache-coherency interconnect, Side channel attack

Received(02. 09. 2022), Modified(03. 04. 2022), Accepted(03. 04. 2022)

* 본 연구는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대 공학 연구자 육성사업의 지원(No. NRF-2019H1D8A2105513), 2022년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No. NRF-2021R1A6A1A13044830), 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No. 2015-0-00280, (SW 스타랩) 성능 및 보안 SLA 보장이 가능한 차세대 클라우드 인프라SW 개발)을 받아 수행된 연구임.

† 주저자, ypkim@inu.ac.kr
‡ 교신저자, seehwan.yoo@dankook.ac.kr(Corresponding author)

I. 서 론

패턴 스크린 잠금 기법은 안드로이드 OS의 기본적인 사용자 인증을 위한 보호 기법이다. 비교적 단순한 9개의 점으로 구성된 한붓그리기 패턴은 대략 39만개의 패턴을 생성하며, 연속된 실패 시에 입력 지연이나 추가 사용자 인증을 요구하므로, 무작위 대입 공격(brute-force attack)에는 비교적 강건한 기법으로 알려져 있다.

스크린 잠금 패턴에 대한 공격은 꽤 많이 이루어져 왔다. 스크린에 남겨진 지문을 이용하거나, 비디오 녹화 등을 이용한 잠금 패턴의 탈취 공격이 지속적으로 보고되어 왔다[1,2,3,4,5]. 하지만, 현재까지 제시된 공격 기법들은 몇 가지 제한점이 있다. 외부 카메라 장치를 사용하는 경우, 카메라의 각도와 빛의 세기에 따라 오차가 발생할 수 있으며, 스크린에 보호 필름을 붙이는 등의 작업으로 비교적 쉽게 방해할 수 있다. 지문을 이용한 공격 역시 터치스크린의 필름 상태에 따라 지문이 남아있지 않은 상태에서는 정보를 추정하기 어려운 단점이 있다.

본 논문에서는 스크린 잠금 패턴을 보다 정교하게 추적할 수 있는 하드웨어 부채널을 제시한다. ARM 멀티코어 프로세서 널리 활용되는 캐시 일관성 인터페이스(Cache-Coherency Interconnect, CCI)[6]는 고성능/저전력 코어가 통합된 빅-리틀 구조에서 빅 코어 캐시와 리틀 코어 캐시 간의 일관성 유지를 위해 필수적으로 사용된다.

논문에서 제안하는 공격 모델은 안드로이드 패턴 스크린 잠금을 공격하는데 활용되었으나, CCI와 관련한 공격 기법은 일반적으로 ARM 기반의 멀티코어 시스템에서 적용이 가능하며, CCI가 연결되는 다양한 장치와 펌웨어에서도 공격 벡터를 찾을 수 있다.

이 논문의 기여점은 다음과 같다. 첫째, 이 논문은 저수준 하드웨어 캐시 일관성 인터페이스를 이용해 고수준의 사용자 동작을 추론할 수 있는 부채널을 만들 수 있음을 보여준다. 둘째, CCI를 이용한 부채널을 이용하여 안드로이드(Android) OS의 스크린 잠금 패턴을 효율적으로 공격할 수 있는 기법을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 배경지식을 설명한다. 3, 4장에서 논문의 주요한 공격 벡터인 캐시 일관성 인터페이스와 이를 이용한 사용자의 행동 패턴 분석 기법을 제시하고, 5장에서 구현 내용을 다룬다. 6장에서는 성능 평가를 통해 공격의 유효성을 보여준다. 7장에서는 관련 연구를 소개하고, 8장에서 결론을 맺는다.

II. 배경지식

2.1 ARM 캐시 일관성 인터페이스

ARM 프로세서에서는 고성능 애플리케이션을 위한 빅 코어와 저전력 애플리케이션을 위한 리틀 코어의 두 가지 코어가 있는 빅리틀(big.LITTLE) 디자인을 채택한다. 각 코어에는 고유한 L1 캐시가 있으며 동일한 종류의 코어는 L2 캐시를 공유한다. 따라서, 빅리틀 프로세서에는 두 개의 서로 다른 L2 캐시 도메인이 존재하며, 최신 ARM 프로세서 설계에서 L2 캐시는 일반적으로 마지막 수준 캐시(last level cache, LLC)로 여러 코어에서 캐시를 사용할 때 메모리와 캐시 내용의 일관성을 유지해야 한다.

일관성을 유지하기 위해 ARM은 캐시를 관리하는 세 가지 방법을 제공한다. 첫 번째로, 사용자는 메모리 영역을 캐시 불가능으로 정의할 수 있는데, 이 경우 사용자 데이터는 캐시에 저장되지 않고 사용자가 메모리에 직접 액세스할 수 있다. 두 번째로, 사용자는 캐시 관리 작업을 명시적으로 호출할 수 있다. 예를 들어, ARM에서는 캐시 제어 레지스터를 정의하며 사용자는 특정 명령어를 사용하여 캐시라인을 명시적으로 정리/무효화할 수 있다. 셋째, ARM은 CCI 하드웨어를 제공하는데, 이를 이용하여 크고 작은 코어뿐만 아니라 GPU 및 NIC 장치 캐시 간의 캐시 도메인 간의 캐시 콘텐츠를 관리할 수 있다.

ARM의 CCI는 MOESI 기반 캐시 일관성 프로토콜을 사용하며, 캐시라인은 5가지 상태 중 하나로 각 상태의 의미는 다음과 같다.

- Modified: 캐시라인이 수정된 고유한 복사본.
- Owned: 캐시라인이 수정되었으며, 데이터 수정 권한을 가지고 있음. 다른 캐시에 복사본이 있더라도 해당 캐시라인의 데이터만 수정 가능.
- Exclusive: 캐시라인이 수정되지 않은 고유한 복사본.
- Shared: 캐시라인이 수정되지 않았으며, 다른 캐시 간에 공유. 즉, 다른 캐시에 동일한 데이터의 여러 복사본이 있음.
- Invalid: 무효화된 캐시라인.

MOESI는 MSI(modified-shared-invalid) 캐시 일관성 프로토콜의 변형으로, 캐시 일관성 버스

트래픽을 효율적으로 줄이고 '쓰기 가능한 캐시라인'과 '공유만 가능한 캐시라인'을 구별할 수 있다는 특징을 가진다. 또한, ARM CCI[6]는 ARM Cortex CPU의 두 코어 클러스터 간에 완전한 캐시 일관성을 제공한다. 예를 들어, CCI-400은 AMBA 프로토콜을 사용하는 ARM CCI의 네 번째 버전으로, 최신 ARM 기반 모바일 장치에 사용되어 코어 캐시, 메모리 및 GPU 메모리 간의 일관성을 유지한다. CPU가 GPU와 메모리 영역을 공유할 때 메모리 영역은 CCI에 의해 공유되고 업데이트된다. OpenGL과 같은 사용자 수준의 그래픽 라이브러리는 그래픽 버퍼를 할당하고 버퍼를 업데이트하여 화면을 만든다. CPU 캐시에 사용자 데이터가 있으며 메모리와 동기화되고, 그래픽 버퍼가 업데이트되면 버퍼가 GPU 메모리로 전송된다. 또한, GPU에는 공유 메모리 영역이 있기 때문에 직접 액세스할 수 있다.

캐시 일관성을 유지하기 위해 CCI는 일정 기간 동안 캐시 메모리를 모니터링하고 '읽기 또는 쓰기 요청'과 관련된 이벤트를 카운터(counter)로 알리는 것과 같은 버스 스누핑(bus snooping) 인터페이스를 제공한다. 이를 이용하여, Performance Monitoring Unit(PMU)를 통해 실행시간에 CCI의 다양한 통계를 얻을 수 있다. 이때, PMU를 사용하려면 세 가지 작업이 필요하다. 첫 번째는 대상 PMU 이벤트를 설정하는 것으로, CCI에서는 PMU 이벤트를 최대 4개까지 선택할 수 있다. PMU 이벤트는 버스 마스터(master)와 슬레이브(slave) 간에 읽기/쓰기 요청이 발생할 때의 버스 트래픽 이벤트를 의미한다. 두 번째로 모니터링 클럭 사이클(clock cycles)의 길이를 설정해야 한다. 설정된 모니터링 클럭 사이클 동안 PMU 이벤트가 생성되면, 마지막 설정으로 모니터링 클럭 사이클이 끝날 때 카운터 레지스터(counter register)를 읽는 것이다. 카운터 레지스터는 PMU 결과가 저장되는 CCI 레지스터로, 3개의 주요 CCI 레지스터가 있다: '이벤트 선택 레지스터(ESR)', '이벤트 카운트 레지스터(ECR)', 그리고 '성능 모니터 제어 레지스터(PMCR)'. ESR은 첫 번째 작업에서 대상 PMU 이벤트를 선택하는 데 사용되며 ECR은 최종 작업에서 대상 PMU 이벤트의 발생 횟수를 계산하는 데 사용된다. PMCR은 PMU 모니터링을 활성화하거나 비활성화하는 데 사용되며 모니터링을 시작하도록 설정해야 한다. 이 CCI 레지스터는 전역 주소 맵(global address map)에 있으며 기본 주소에서 오프셋 0x90000을 통해 쉽게 액세스할 수 있다.

2.2 안드로이드 패턴 스크린 잠금

안드로이드 기기에서 패턴 스크린 잠금 (pattern screen lock, PSL)은 기기 액세스를 위한 기본 보안 메커니즘으로, 사용자가 올바른 패턴을 그릴 때까지 안드로이드 기기를 잠근 상태로 유지한다. 패턴은 3x3 그리드(grid)에서 9개의 점을 연결하는 선의 조합으로, 사용자는 인증을 위한 그래픽 암호로 패턴을 그리는 점을 자유롭게 선택할 수 있다. 사용자가 저장된 패턴을 입력하면 안드로이드 장치가 잠금 해제된다. 본 논문에서는 패턴을 표현하기 위해 3 x 3 그리드의 시퀀스로 왼쪽 상단의 점은 0, 수평으로 인접한 점은 1, 다음 점은 2, 오른쪽 아래 점은 8로 설명한다. 예를 들어 'L' 모양의 패턴은 '0 3 6 7 8'로 설명된다.

기존 연구[7]에 따르면 안드로이드 PSL은 유효한 패턴에 대해 세 가지 규칙을 적용한다: 1) 패턴에는 최소 4개의 점, 최대 9개의 점을 사용해야 한다, 2) 접촉점은 패턴에서 한 번만 사용해야 한다, 3) 패턴에 선에 포함된 중간 점이 있는 경우 항상 패턴에 포함되어야 한다. 따라서, 9개의 도트에 대해 총 982,800개의 패턴이 존재할 수 있지만, 위에서 언급한 3가지 규칙을 적용하게 되면 패턴의 수가 389,112[7]로 줄어든다. 또한, 안드로이드 기기 관리 정책[1]에 따라서 최대 시도 횟수나 잠금 실패 횟수에 따른 대기 시간 중첩 혹은 구글 계정을 통한 추가 인증 등 다양한 패널티가 부여될 수 있다. 본 논문에서는 실험 안드로이드 기기에 부여된 관리 정책인 최대 시도 횟수 20회 실패시 구글(Google) 계정을 통한 추가 인증을 관리 정책 기준으로 정하였다.

III. 새로운 공격 벡터: 캐시 일관성 인터페이스

3.1 안드로이드 PSL 위협 모델

안드로이드 PSL은 안드로이드 OS의 기본적인 사용자 인증 기법으로서 많은 공격이 시도되었다[1,2,3,4,5]. 각 공격들은 다양한 위협 모델을 가정

1) https://developer.android.com/guide/topics/admin/device-admin

하고 있으며, 근거리에서 카메라 등의 외부 관찰 장치를 이용한 경우와 피해(victim) 장치에 대한 물리적인 접근을 가정한다.

우리는 이 논문에서 안드로이드 PSL과 관련하여 다음의 위협 모델을 가정한다. 첫째, 공격자는 CCI와 관련한 값들을 읽어올 수 있다. 이는 일반적으로 개발자 모드(developer mode)의 핸드폰에서는 비교적 쉽게 가능하다. 보다 강력한 공격자의 경우, 펌웨어 인터페이스를 통해 CCI 이벤트 값을 직접 읽어올 수 있다. 다만, 상용으로 출시되는 핸드폰에서는 직접적으로 CCI 이벤트에 관련한 값들을 읽어올 필요가 없으므로, 권한이 제한된 경우가 많다. 이 경우, 캐시 부채널 공격 등을 통해 CCI 이벤트의 값을 간접적으로 추정할 수 있다.

둘째, 공격자는 백그라운드로 동작하는 소프트웨어(background process)를 실행할 수 있다. 안드로이드 OS에서 서비스 형태로 동작하는 소프트웨어들은 백그라운드에서 지정된 이벤트를 받으면 실행이 시작된다. 안드로이드 PSL의 경우 시작하는 시점과 종료 시점에 명시적으로 인텐트(intent)를 발생시키므로, 해당 이벤트가 발생하는 시점에 백그라운드로 동작하는 공격을 시도할 수 있다.

본 논문에서는 이외의 외부적인 장치나 터치스크린의 물리적인 접촉을 가정하지 않는다.

3.2 CCI를 이용한 PSL 공격 벡터

캐시 일관성 인터페이스 CCI는 CPU의 메모리 접근에 따른 동작을 정의하는 매우 저수준의 인터페이스를 가진다. 일반적으로 이러한 저수준 인터페이스에서 고차원의 OS 동작을 추출하는 것은 쉽지 않다. 이번 절에서는 CCI를 이용하여 사용자 동작과 관련한 이벤트를 추출할 수 있는 부채널을 생성하는 방법을 제시한다.

3.2.1 PMU를 이용한 캐시 일관성 정보 추출

ARM의 CCI는 고성능인 빅코어 간 공유되는 빅코어-캐시(빅캐시) 와 저전력을 소모하는 리틀코어 간 공유되는 리틀코어-캐시(리틀캐시) 간의 일관성을 유지하는데 사용된다.

CCI는 캐시 일관성유지 뿐만 아니라 서로 다른 캐시 간 빠른 데이터의 이동을 지원한다. 즉, 빅캐시에 저장된 데이터가 리틀캐시에서 요청되는 경우, 캐시라인을 메모리에서 가져오지 않고, 빅캐시에서 리틀캐시로 직접 전달할 수 있다.

CCI는 다양한 성능 모니터링 값을 추출할 수 있는 프로파일링 인터페이스를 제공한다. 우리는 다양한 성능 모니터링 이벤트 중에서 '리드 데이터 라스트 핸드쉐이크(read data last handshake)' 이벤트를 활용하였다[2]. 해당 이벤트는 메모리 요청이 발생한 캐시의 데이터를 메모리가 아닌 다른 도메인의 캐시로부터 데이터를 전송하는 경우 발생하는 이벤트이다.

이러한 이벤트는 빅코어(또는 리틀코어)에서 실행되던 작업이 리틀코어(또는 빅코어)로 마이그레이션 되거나, 빅코어(또는 리틀코어)에서 접근된 데이터가 다른 코어에서 사용되는 작업과의 통신 등으로 리틀코어(또는 빅코어)에서 사용되는 경우에 빈번하게 발생한다.

3.2.2 멀티코어 환경의 안드로이드 패턴 스크린 잠금

빅코어와 리틀코어는 사용되는 용도가 다르다. CPU 활용율이 낮은 경우, 리틀코어는 사용자와 인터랙션이 많은 입출력을 담당하는 코어로 활용된다. 사용자의 인터랙션이 많은 입출력 작업들은 상대적으로 CPU 활용율이 높지 않지만, 빠른 응답시간을 요구하는 짧은 태스크 실행을 위해 활용된다. 반면, 빅코어는 갑자기 많은 작업이 요청되거나, CPU 활용율이 일정 수준 이상으로 올라간 경우 활용된다.

안드로이드 OS의 PSL과 관련한 프로세스는 'SurfaceFlinger'와 'SystemServer' 이다. 스크린 잠금 화면(lock screen)은 SystemServer에 의해 실행되는 안드로이드 OS의 SystemUI의 구성요소로서, 화면의 최상위 레이어에서 활성화되는 시스템 위젯의 일종이다. 스크린 잠금은 최상위 레이어에서 활성화되므로, 모든 사용자 입력은 스크린 잠금 위젯으로 전달되며, 잠김 화면에서 일반 사용자 응용들은 사용자 인증이 되기 전이므로, Table 1의 API나 접근권한 설정을 통해 화면에 응용 프로세스와 관련한 내용을 출력하거나 입력을 받지 않는다.

스크린 잠금이 실행되면 SystemServer는 시스템으로 입력된 모든 입력 이벤트를 처리하는 EventThread를 통해 터치 스크린의 화면 터치 이

2) https://developer.arm.com/documentation/ddi0470/k/functional-description/performance-monitoring-unit/pmu-event-list

Table 1. API for screen access control when the screen is locked.

No.	API
1	setShowWhenLocked(disable)
2	android:showOnLockScreen="false"

벤트를 입력 받는다. 스크린 잠금 위젯은 업데이트되는 터치 스크린 화면의 손가락 위치에 따라 화면의 선을 그린다. 화면의 선을 그리는 작업은 'SurfaceFlinger' 프로세스가 담당한다. 만약 화면의 선이 3x3 그리드를 지나게 되면 패턴이 생성되어 실제 사용자의 인증 패턴과 동일한지 확인한다.

SystemServer는 사용자의 터치입력을 화면에 그리도록 화면에 선을 그리고, 업데이트 인증을 처리하는 작업을 수행한다. 이 때, 실제 화면을 업데이트하는 작업은 SurfaceFlinger에서 수행하며, 두 프로세스는 안드로이드의 바인더 IPC를 이용하여 그래픽 객체를 넘겨주고 받는다.

현재 안드로이드 OS의 프로세스 스케줄링과 코어 할당 정책은 특정 작업을 빅/리틀 코어로 고정하지는 않지만, 대개의 구현에서 모든 하드웨어 인터럽트는 0번 코어로 전달된다. 따라서, 입력을 담당하는 프로세스인 SystemServer는 0번 리틀코어에서 실행된다. 반면, 그래픽처리를 담당하는 SurfaceFlinger는 사용자의 입력 지연을 최소화하기 위해 별도의 코어에서 실행된다. 그래픽의 해상도와 프레임속도가 높아질수록 CPU작업의 부하가 커지며, 많은 경우 빅코어에서 실행된다.

이 때 리틀 코어에서 생성된 그래픽 객체는 빅코어에서 실행되는 SurfaceFlinger에서 접근된다. 리틀 코어의 그래픽 객체는 리틀 코어의 캐시에 올라오게 되며, 빅코어에서 참조됨에 따라 빅코어의 캐시에도 공유된다. ARM 멀티코어 구조는 MOESI 프로토콜을 구현하며, 서로 분리된 빅코어-캐시와 리틀코어-캐시가 데이터를 공유하는 경우, 캐시라인의 상태를 Modified에서 Owned로 변경하며 메모리 업데이트 없이 캐시-캐시 간 통신을 통해 데이터를 빠르게 복사한다. 이 과정에서 CCI 이벤트가 발생하게 된다. 즉, CCI 이벤트는 화면의 업데이트와 관련한 빅리틀 구조의 ARM 프로세서에서 발생한다. 안드로이드의 사용자 인증에 활용되는 스크린 잠금 위젯 역시 화면 업데이트를 하는 과정에서 CCI 이벤트를 발생시키므로, 이를 이용하면 안드로이드 사용자 인증을 진행하는 공격의 부채널로서 활용이 가능하다.

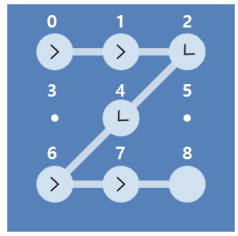

Fig. 1. An example of pattern '0124678' in 3x3 grid. The symbol 〉 indicates the drawing direction, and the empty circle indicates the last dot.

IV. 안드로이드 스크린 잠금 패턴 공격

본 논문의 주요 아이디어는 CCI 이벤트를 이용하여 방향 전환(direction change, #DC)과 완료 시간(time to finish, TTF) 정보를 수집하여, 개별 패턴을 그룹으로 분류하는 것이다. #DC는 패턴의 방향 전환 횟수를 나타내는 패턴의 그래픽 특성이고, TTF는 패턴을 그리는 데 걸리는 시간을 나타내는 패턴의 동작 특성이다. 모든 패턴은 #DC, TTF 특성을 가지기 때문에 이 두 가지를 패턴 분류 기준으로 활용한다.

패턴의 분류를 통해 전체 패턴을 하위 그룹으로 나누고, 그 그룹의 크기를 알게 되면, 그룹에 속한 특정 패턴은 유한한 시행 안에서 찾을 수 있다. 이는 해당 그룹에 대한 '추측 공격'(guessing attack)이 가능하다는 것을 의미한다. 공격에 필요한 추측의 수를 #G라고 하면 해당 그룹의 크기도 #G가 된다. 특히, #G가 20 미만인 경우, 2,2장에서 언급한 추가 인증 전에 공격이 가능한 취약한 패턴이라고 할 수 있다.

4.1 방향 전환 횟수

본 장에서는 3x3 그리드에서 패턴이 그려질 때 방향 전환 총 횟수인 #DC를 계산하는 방법을 설명한다. 예를 들어, Fig. 1과 같이 '0124678' 패턴의 경우[3], 1 → 2 → 4 및 4 → 6 → 7일 때 방향이 변경되기 때문에 #DC는 2가 된다. 이 때, 서로 다른 패턴이 같은 #DC 값을 가질 수 있기 때문에 #DC를 이용하여 여러 패턴을 그룹화할 수 있다. #DC를 통해 패턴 그룹화하여 예측함으로써 개별 패턴 예측

3) 본 논문에서는 오픈소스 웹앱인 patternLock.js (http://ignitersworld.com/lab/)을 이용하여 패턴 화면을 캡쳐한다.

보다 용이하게 패턴 예측이 가능하다.

본 논문에서는 벡터 덧셈을 이용하여 #DC를 계산한다. 그리드에 A, B, C라는 세 개의 점이 있는 경우 A에서 B로, B에서 C로의 두 방향을 나타내기 위해 두 벡터 $\overrightarrow{AB}$와 $\overrightarrow{BC}$ 를 만들 수 있다. 두 벡터의 합은 $\overrightarrow{AB}+\overrightarrow{BC}$로, 평행사변형 법칙[8]에 따르면 $|\overrightarrow{AB}+\overrightarrow{BC}|$의 최대 벡터 길이는 $2\times|\overrightarrow{AB}|$ 또는 $2\times|\overrightarrow{BC}|$ (동일한 방향)이고 최소 값은 0 (반대 방향)이다. 벡터 덧셈을 이용한 #DC 계산은 다음과 같다. 2차원 데카르트 좌표에서 3x3 격자에 9개의 점을 넣고 3x3 격자의 중심을 점 4 (0,0)로 만든다. 이 때, 패턴은 좌표에서 일련의 점으로 표시되는데, 예를 들어, 패턴 '0 1 2'는 (-1, 1), (0, 1) 및 (1, 1)의 시퀀스로 표현된다.

이전 벡터(V_P)와 현재 벡터(V_C)는 다음과 같이 정의된다. 세 점을 d_i, d_{i+1}, d_{i+2}라고 하고 xy 좌표를 (x_i, y_i), (x_{i+1}, y_{i+1}), (x_{i+2}, y_{i+2})라고 할 때, V_P, V_C, 그리고 V_P+V_C 는 각각 ($x_{i+1}-x_i$, $y_{i+1}-y_i$), ($x_{i+2}-x_{i+1}$, $y_{i+2}-y_{i+1}$), ($x_{i+2}-x_i$, $y_{i+2}-y_i$)로 표현된다. 이 때, V_P, V_C의 방향이 다를 때에만 방정식이 참이기 때문에 $|V_P|+|V_C|>|V_P+V_C|$ 가 참인지 확인한다. 따라서, 아래의 함수 f가 0보다 큰 경우, 방향 전환이 발생한다.

$$f=\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2}+\sqrt{(x_{i+2}-x_{i+1})^2+(y_{i+2}-y_{i+1})^2}-\sqrt{(x_{i+2}-x_i)^2+(y_{i+2}-y_i)^2} \quad (1)$$

#DC는 i=0일 때 아래와 같이 계산 된다.

1) 패턴의 모든 점 시퀀스에 대해 먼저 세 개의 점 d_i, d_{i+1}, d_{i+2}을 순서대로 선택한다.
2) 세 개의 점에 대해 f>0일 때마다 카운터 c를 증가시킨다.
3) 이를 i가 마지막 점에 도달할 때까지 i를 1씩 증가시켜 반복한다.

이 때 카운터 c는 0으로 초기화 하고, 최종 c의 값이 #DC가 된다.

위 알고리즘을 모든 안드로이드 PSL 패턴에 적용하여 모든 패턴의 #DC를 계산한 다음 모든 패턴을 #DC 값에 따라 그룹으로 분류한다. 동일한 #DC 값을 가지는 패턴들은 동일한 그룹으로 분류된다.

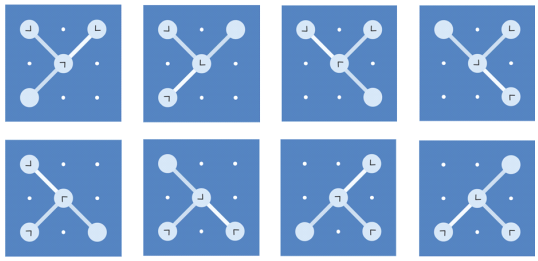

Fig. 2. Patterns in 5.66 TTF group: 0426 0462 2408 2480 (top), 6408 6480 8426 8462 (bottom).

#DC의 최대 값은 모든 패턴에서 7이므로 총 그룹 수는 7이 된다.

4.2 패턴 완료 시간

다음으로, 패턴 그리기를 완료하는 데 경과된 시간인 패턴 완료 시간(time-to-finish, TTF)를 알아낼 필요가 있다. 본 연구진은 패턴을 완료 시간에 따라 그룹으로 분류될 수 있다는 것을 확인하였다. 즉, 두 패턴이 완료 시간이 다른 경우 서로 다른 그룹에 속하게 된다. 각각의 패턴을 그리는 속도에 따라 패턴 완료 시간이 달라지므로 정규화된 TTF를 도입해야 한다. 패턴마다 그리는데 소모되는 시간은 매우 크거나 작을 수 있어 정규화를 통해 고정된 범위의 기준 TTF(baseline TTF)를 계산하고 그룹화 할 필요가 있다. 또한, 사용자에 따라 TTF가 달라질 수 있기 때문에, 이를 고려하기 위해 TTF 오류(TTF error)를 반영해야 한다. 따라서, 본 연구에서는 기준 TTF와 TTF 오류를 모두 고려하여 TTF를 고안하였다.

1) 기준 TTF: 기준 TTF는 일정한 속도로 패턴을 그린다고 가정한다. #DC와 마찬가지로 직교 좌표의 각 점에 9개의 점을 매핑하고, 유클리드 거리[9]를 사용하여 모든 라인의 총 길이로 기준선 TTF를 계산한다. 또한, 패턴을 그리는 시간을 추정하기 위해 총 길이를 사용한다. 그 이유는, 기존 연구[10]에서 올바른 패턴을 잠금 해제하는 시간(= 패턴을 그리는 시간)이 패턴 길이에 따른 회귀 모델(regression model)로 표현될 수 있음을 보였기 때문이다. 회귀 모델은 $y=-53+147\times x$ 로, 이는 패턴에 사용되는 추가 점(dot) 당 잠금 해제 시간이 평균 147ms씩 증가함을 의미한다. 그러나, 실제로는 CCI가 점의 개수를 제공하지 않기 때문에 이 값을

Table 2. The results of TTF 5.66 of pattern '0426'

Line	P		Q		d(P, Q)
	x	y	x	y	
0 → 4	-1	1	0	0	1.41
4 → 2	0	0	1	1	1.41
2 → 6	1	1	-1	-1	2.83
Total TTF					5.66

직접 사용하기는 어렵다. 대신, CCI 정보로부터 얻을 수 있는 '패턴을 그리는데 소모되는 시간'에 초점을 맞춘다. 따라서, 본 연구에서는 TTF를 추정하기 위해 '패턴을 그리는데 소모되는 시간'을 기반으로 회귀 함수를 구축하였다(5.1장 참조).

다음으로 모든 패턴을 TTF 그룹으로 분류한다. 모든 안드로이드 PSL 패턴(총 389,112개)을 생성하고 기준 TTF를 계산한다. 그리고, 동일한 TTF를 갖는 패턴을 그룹(TTF 그룹이라고 함)으로 분류하는데, 이때 TTF 그룹의 크기는 그룹의 패턴 수가 된다. 취약한 TTF 그룹 (그룹 크기 〈 20)을 식별하기 위해 모든 TTF 그룹을 크기의 오름차순으로 정렬한다. 그런 다음 일정한 속도로 모든 안드로이드 패턴을 그리는 스크립트를 실행하여 기준선을 분석하였다. 그 결과, 모든 안드로이드 패턴에 대해 총 273개의 TTF 그룹을 찾았고, 24개의 TTF 그룹이 취약한 것을 확인하였다.

Fig. 2는 5.66 TTF의 그룹을 보여주며, 예제 패턴 '0426'을 사용하여 5.66을 계산하는 방법을 설명한다. 이 패턴을 그리는 경우, 0 → 4, 4 → 2, 2 → 6 로 4개의 선이 순차적으로 생성된다. 각 선의 길이는 유클리드 거리 공식으로 계산할 수 있다. 선의 끝점이 #DC 계산에 사용된 동일한 2차원 데카르트 좌표에 매핑되기 때문이다 (4.1장 참조). 2차원에 대한 유클리드 거리의 공식은 다음과 같다.

$$P(p_1, p_2),\ Q(q_1, q_2) \qquad (2)$$
$$d(P, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

위 수식에서 P와 Q는 선의 각 끝점에 해당하고 네 개의 점 0, 4, 2, 6의 좌표는 (-1, 1), (0, 0), (1, 1), (-1, -1)에 매핑된다. 수식의 결과는 Table 2와 같다.

2) TTF 오류: 실제로 사용자는 다양한 속도로 패턴을 그리기 때문에 TTF 계산에 이를 고려하기 위해 TTF 오류를 고려한다. TTF 오류는 패턴을 그리는데 소모되는 시간의 평균 값과 기준 TTF 값의 차이를 백분율로 계산하는 것이다. 예를 들어, TTF 그룹 5.66에 1% TTF 오류가 있는 경우 오류의 크기는 0.0566이다. 따라서, 1% 오류가 있는 TTF 5.66의 범위는 5.6034(5.66 - 0.0566)에서 5.7166(5.66 + 0.0566)이 된다. 기준 TTF에 따라 5.65 및 5.66의 두 TTF 그룹을 가지는데, 분석 결과 TTF 5.65 그룹은 1,312개의 패턴을 가지고, TTF 5.66 그룹은 8개의 패턴을 가짐을 알아냈다.

4.3 추측 엔트로피

#G는 공격자가 시도할 수 있는 횟수(최악의 경우)를 나타내어, 패턴을 추측하는데 어려움의 정도를 나타내는 일종의 '추측 엔트로피(guessing entropy)'이다. 추측 엔트로피를 설명하려면 먼저 확률 문제로 패턴을 찾을 가능성을 정량화해야 한다. 패턴을 선택하는 모든 추측의 집합을 Ω, 모든 추측의 수를 $k = n(\Omega)$ 라고 할 때, 안드로이드 PSL에서 k는 389,112이다.

그 다음, i 패턴을 추측할 확률을 p_i 라고 할 때, 모든 패턴에 대해 $\forall i \in \Omega, 0 \le p_i \le 1$인 경우 $\sum p_i = 1$가 된다. 또한 p_i의 확률분포를 χ라고 할 때, 완전히 임의적인 조건에서 χ는 $p_i = \frac{1}{n(\Omega)}$ 인 균일 분포를 나타낸다. PSL 취약점을 정량화하기 위해, 본 논문에서는 패턴 또는 비밀번호 강도를 평가하는 데 널리 사용되는 '추측 엔트로피(또는 간단히 엔트로피)[11,12,13]'를 사용한다. 추측 엔트로피는 아래와 같이 임의 변수인 목표 값 X [14,15]를 찾기 위한 예상 추측 수로 정의된다.

$$Entropy(\chi) = E\left[\#\, G(X \xrightarrow{R} \chi)\right] = \sum_{i=1}^{N} p_i \bullet i \qquad (3)$$

위 수식에서 $X \xrightarrow{R} \chi$ 는 에서 무작위로 뽑는 χ에 대해 X 패턴을 선택하는 확률 모델을 의미한다.

'추측 공격'은 일반적으로 공격할 후보 패턴을 포함하는 패턴 리스트를 만든 후, 완전 무작위 선택 또

는 편향 선택에 의해 후보 패턴을 선택한다. 완전히 무작위 선택인 경우 분포는 (1/389,112)가 되며, 많은 패턴 잠금 연구[4,7,11,16,17]에서는 사용자 설문조사를 기반으로 한 편향 선택 분포를 사용한다. 본 논문에서 제시하는 CCI 기반 안드로이드 PSL 공격은 무작위 선택과 편향 선택 여부에 관계없이 공격이 성공함을 보여준다.

제안하는 기법에서는, 알 수 없는 패턴 집합 (X_1, X_2, … , X_k)이 주어지는 경우, 제안하는 기법에서는 최악의 경우 알 수 없는 패턴 X_i의 #G를 계산한다. 본 논문의 핵심 아이디어는 #DC와 TTF를 모두 이용하여 #G를 줄이는 것인데, 이는 다음과 같이 두 단계로 이루어진다. 먼저 #DC를 활용한다. 예시로, #DC=1인 패턴을 포함하는 #DC=1 그룹과 #DC=2 그룹($1 \leq$ #DC ≤ 7)을 생성할 수 있다. 즉, #DC를 활용하여 특정 패턴들을 분류할 수 있기 때문에, 전체 패턴들을 하위 집합으로 좁힐 수 있다. 다음으로, TTF를 이용하여, 첫 번째 단계의 하위 집합을 TTF의 필터를 통과시킨다. 그 결과 하위 집합의 하위 집합이 생성되고 패턴 그룹의 크기가 훨씬 작아지게 된다. 예를 들어, #DC=1인 경우 #DC=1의 크기는 168이다. 이때, #DC=1 패턴 그룹에 TTF=4를 적용하면 패턴 그룹의 크기가 8로 줄어든다. 이는 두 집합($\Omega_{TTF=4}$ 및 $\Omega_{\#DC=1}$)의 교집합이 각 집합보다 작기 때문이다. 따라서, n($\Omega_{TTF=4}$)=136, n($\Omega_{\#DC=1}$)=168의 크기가 n($\Omega_{TTF=4}$, $\Omega_{\#DC=1}$)=8로 줄어들게 된다. 본 연구에서는 #DC와 TTF를 적용할 때, #DC를 먼저 활용하는 경우 하위 집합의 크기를 감소시키는데 더 효과가 있음을 확인하였고, #DC와 TTF의 순서로 하위 패턴 집합을 선택한다.

V. 안드로이드 PSL 공격 구현

5.1 전체 구조

Fig. 3은 CCI를 이용한 안드로이드 PSL 공격의 전체 구조를 보여준다. 구현 환경은 ARM Cortex A53/A57 멀티코어 보드(Juno board)를 기반으로 리눅스 커널(Linux kernel) 4.4.49 및 안드로이드 7.1.2(Nougat)를 사용하였다. 사용자가 유효한 패턴을 그리면 관련 리눅스 프로세스(Liunx process)(예: 시스템 서버의 SurfaceFlinger 및 Lockscreen 작업)가 안드로이드 장치에서 실행된

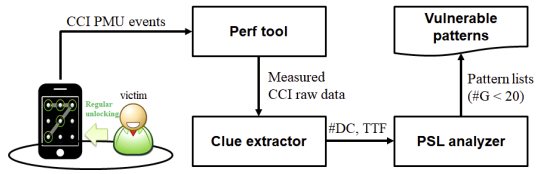

Fig. 3. CCI-based side-channel attack of Android PSL

다. 이때, 리눅스 CPU 스케줄러는 프로세스에 적절한 CPU 코어를 할당하고, 이로 인해 ARM MPcores에서 공유하는 캐시 메모리의 상태를 변경된다. ARM CCI(2.1장 참조)는 로컬 캐시 상태의 변경으로 인해 발생하는 CCI PMU 이벤트를 모니터링한다. 이를 이용한 안드로이드 PSL 공격의 순서와 세부 내용은 다음과 같다.

1) 프로파일링 도구(perf tool): 프로파일링 도구는 리눅스 커널 소스에 포함된 사용자 수준 프로파일러이고, 리눅스 커널이 CCI 버스 드라이버를 지원하는 경우 사용자 수준에서 CCI PMU 이벤트를 캡처할 수 있다. 이를 이용하여 본 기법에서는 CCI PMU 이벤트를 수집하고 CCI 데이터가 포함된 일반 텍스트 기반 출력 파일을 생성한다. CCI 데이터는 프로파일링 도구에 의해 생성되는 텍스트 기반 로그 파일이며 샘플링 기간(기본 값 100ms) 동안 측정된 시간과 누적된 CCI 카운터의 수치로 구성된다. 실제 CCI 데이터는 5.2장과 Fig. 8에서 설명한다.

2) 단서 추출기(clue extractor): 단서 추출기는 CCI 데이터를 기반으로 #DC 및 TTF의 두 가지 값을 계산한다. 단서 추출기는 다음 관찰을 기반으로 한다. CCI 데이터 그래프는 패턴을 그릴 때 봉우리와 계곡이 포함된 산 모양을 보여 준다(Fig. 5 참조). 이 때, 계곡의 수가 #DC와 관련되어 있음을 파악하였고(Table 3의 Avg. Valleys 참조), CCI 데이터에 노이즈가 있음을 확인하였다.

실제 예시를 통해 두 값을 도출하는 방법은 다음과 같다. 먼저 기존 연구[4]에서 '단순(simple)', '중간 복잡(median complex)' 및 '복잡(complex)' 범주에 사용된 세 가지 대상 패턴('24678', '0485762', '456183270')을 선택하였다. 이 세 가지의 패턴(Fig. 4)에 대해 12번 반복하여 그렸을 때의 CCI 데이터를 총 4,400개 이상의 샘플을 수집하여 평균값을 계산하였다. 수집한 CCI 데이터는 타임스탬프(timestamp), CCI 카운터로 구성된 시계열 데이터로, Fig. 5에 나타나 있다.

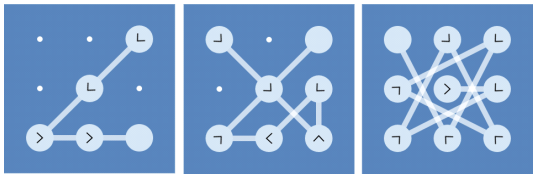

Fig. 4. Test patterns: '24678','0485762' and '456183270'

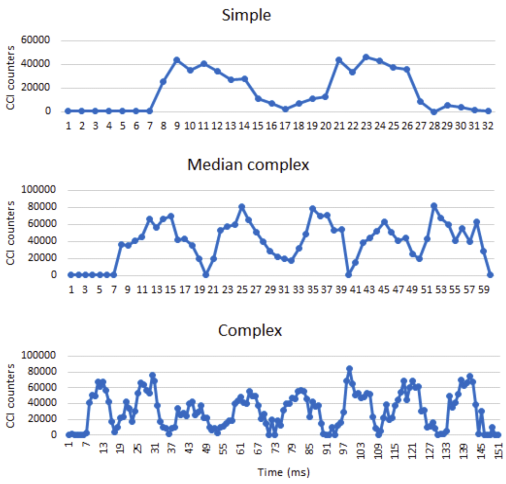

Fig. 5. CCI data for the test patterns

Fig. 5의 세 그래프에서 x축은 샘플링 시간(100ms)을 나타내고 y축은 측정된 CCI 카운터의 양이다. 각 패턴의 그래프는 하나 이상의 봉우리와 계곡이 있는 산 모양을 보여준다. 흥미로운 점은 패턴의 계곡 수가 #DC와 연관이 있다는 것이다. 예를 들어, Fig. 4의 '24678' 패턴의 방향 전환 횟수는 1

Table 3. Data statistics for the test patterns

Pattern	Avg. valleys (std)	Avg. drawing time (std)	Thres hold	# of samples
Simple (#DC=1, TTF=4.83)	1.16 (0.58)	1,783 ms (152.75)	15,000	960
Median complex (#DC=4, TTF=9.07)	4 (1.28)	5,258 ms (287.49)	15,000	1,400
Complex (#DC=7, TTF=16.65)	7.08 (1.78)	12,175 ms (973.40)	6,500	2,080
Total				4,440

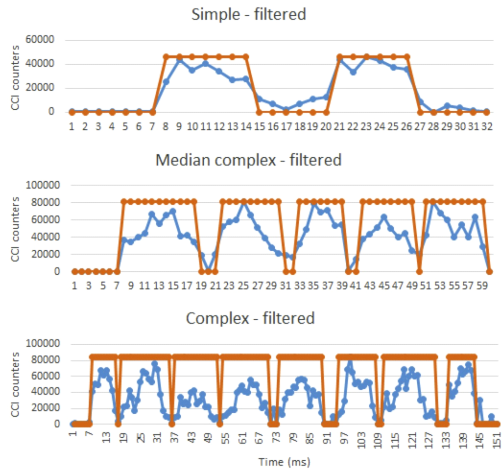

Fig. 6. Filtered CCI data for the test patterns

회로 #DC=1이고 Fig. 5의 그래프는 하나의 계곡이 있다. Fig. 4의 '0485762', '456183270' 패턴은 #DC=4와 #DC=7이고, Fig. 5의 그래프는 각각 4개와 7개의 계곡 형태를 보여주고 있다.

본 연구에서는 기존 연구[18]에서 소개된 계곡 감지 알고리즘을 사용하여 다음과 같이 계산하며 아래 수식을 이용하면 #DC를 계산하기 위한 계곡의 숫자를 도출할 수 있다. 즉, 시계열 데이터

$$T = (t_1, x_1), (t_2, x_2), \cdots, (t_n, x_n)$$

에서 봉우리의 집합 P는 다음과 같이 계산한다.

$$P = \left\{ (t_i, x_i) \mid \begin{array}{l} (x_{i-1} < x_i > x_{i+1}) \\ \vee (x_1 > x_2) \vee (x_n > x_{n-1}) \end{array} \right\} \quad (4)$$
$$\forall i = 2, \cdots, n-1$$

또한, 계곡의 집합 V는 다음과 같이 계산한다.

$$V = \left\{ (t_i, x_i) \mid \begin{array}{l} (x_{i-1} > x_i < x_{i+1}) \vee (x_1 < x_2) \\ \vee (x_n < x_{n-1}) \end{array} \right\}$$
$$\forall i = 2, \cdots, n-1 \quad (5)$$

또한, CCI 데이터에서 #DC를 계산하는 과정에서 정확도를 감소시킬 수 있는 얕은 계곡 형태의 노이즈가 있다는 것을 확인하였다. 이러한 노이즈를 완화하기 위해 본 논문에서는 경험적 임계값으로 데이

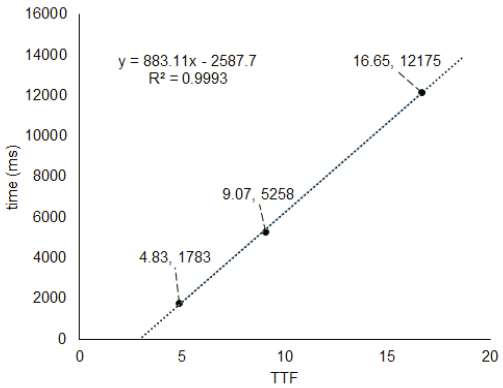

Fig. 7. Linear regression for estimating TTF

터에 대한 필터링을 수행한다. 필터링을 수행하는 방법은, 먼저 단서 추출기에서 로그 파일을 읽고 내용을 시간 및 카운터 목록(X)에 로드 한다. 그 다음 필터 함수 Z를 목록에 적용하여 CCI 값의 차이를 증폭하고 노이즈 효과를 완화한다. 필터 함수는 X의 요소인 샘플 데이터(t_i)를 다음 연산에 의해 필터링 된 출력으로 변환한다.

z_i = Z(t_i, thresh) 일 때,

$$Y(z_i) = \begin{cases} 0 & \text{if } z_{i-1}, z_{i+1} = 0,\ z_i > 0 \\ z_i & otherwise \end{cases} \quad (6)$$

Y(Z(t_i, thresh))를 이용하여 Fig. 6의 주황색 선과 같은 같이 필터링 수행을 통해 노이즈나 급격한 봉우리가 없는 그래프를 얻을 수 있다. 이처럼 필터링 된 그래프의 계곡의 수를 이용하여 #DC 값을 도출한다. Table 3은 CCI 데이터의 요약이며 'Avg. 계곡'은 계곡 수의 결과인데, 이 값이 #DC와 거의 일치하였다.

다음으로, TTF 값 계산을 수행한다. 본 연구진은 CCI 데이터 분석을 통해 CCI 카운터를 이용하여 그 값이 패턴을 그리기 시작할 때(첫 번째 피크 시작 시) 급격히 증가하고 끝날 때(마지막 피크 끝에서) 감소하는 것을 관찰하였다. 따라서, 필터링 된 그래프에서

(End_time_of_last_peak - Start_time_of_first_peak)

를 계산하여 패턴을 그리는 데 소요된 시간을 얻을 수 있다. Table 3의 'Avg. drawing time'은 목표 패턴을 그리는 데 평균적으로 소요되는 시간을 나타낸다. TTF는 패턴을 그리는 정규화 시간이기 때문에 패턴을 그리는데 소요된 시간은 TTF와 직접적인 관련이 있다. 측정된 드로잉 시간에서 TTF를 추정하기 위해 '(TTF, 평균 드로잉 시간)'의 데이터 포인트에 선형 회귀를 적용한 결과는 Fig. 7과 같다.

'Avg. drawing time'에 적합한 선형 회귀 방정식 y=883.11x-2587.7 로, Fig. 7을 통해 측정된 'Avg. drawing time'을 이용하여 높은 정확도로 TTF를 계산할 수 있음을 보여준다. 이때, R-제곱(R^2) 값은 0.9993으로 나타났다. 이로써, CCI 데이터를 이용하여 계곡의 수를 세고 패턴을 그리는데 소요되는 시간을 측정하면 PSL 공격에 사용하기 위한 #DC 및 TTF를 도출할 수 있게 되었다.

3) PSL 분석기 (PSL analyzer): PSL 분석기는 도출한 #DC 및 TTF 값을 사용하여 'all-pattern-case' 파일에서 취약한 패턴을 생성한다. 'all-pattern-case' 파일은 #DC 및 TTF가 있는 모든 패턴 시퀀스 조합을 포함하는 일반 텍스트 파일이다. 텍스트 파일의 각 줄에는 '패턴 도트 번호 시퀀스', #DC 및 TTF(예: 0123, 1, 4.24)가 포함된다. PSL 분석기는 리눅스의 기본 문서 편집 도구인 awk와 grep을 사용하여 파일을 분석한다. Awk는 특정 열 데이터를 선택하는 데 사용되며 grep은 특정 값을 일치시키는 데 사용된다. 이를 통해, PSL 분석기에서 값이 일치하는 결과의 숫자를 계산할 수 있다. 예를 들어, #G 〈 20인 패턴의 경우 PSL 분석기가 취약한 패턴으로 판단하고, 이를 이용하여 공격자는 20회 시도 이내에 안드로이드 장치의 잠금 해제가 가능해진다.

5.2 CCI 프로파일링

리눅스 커널 버전 3.11 이후에서는 'arm-cci'라는 기본 버스 장치 드라이버를 이용하여 CCI를 지원한다. 드라이버는 CCI 카운터를 매핑하고 CCI 포트를 노출시킨다. CCI 포트는 CCI 데이터에 대한 액세스 포인트이며 포트 번호로 참조된다. 포트 번호는 드라이버 모듈이 초기화될 때 'arm-cci'에 의해 할당된다. 프로파일링 도구에서는 포트 번호를 설정하여 CCI 이벤트 모니터링 대상을 나타낼 수 있다. 예를 들어 "-e CCI_400/cci_event,source=3/"와 같이 프로파일링 도구 이벤트 옵션에서 포트 번호 3을 할당할 수 있다.

그런 다음, 프로파일링 도구를 이용하여 모니터링

```
0.100336960,1246,,CCI_400/si_r_data_last_hs_snoop,source=3/
0.200772000,774,,CCI_400/si_r_data_last_hs_snoop,source=3/
0.301289580,1644,,CCI_400/si_r_data_last_hs_snoop,source=3/
0.401814500,82,,CCI_400/si_r_data_last_hs_snoop,source=3/
0.502332940,868,,CCI_400/si_r_data_last_hs_snoop,source=3/
0.602851840,81,,CCI_400/si_r_data_last_hs_snoop,source=3/
0.703372300,82,,CCI_400/si_r_data_last_hs_snoop,source=3/
0.803948660,79,,CCI_400/si_r_data_last_hs_snoop,source=3/
0.904528000,247,,CCI_400/si_r_data_last_hs_snoop,source=3/
1.005109520,257,,CCI_400/si_r_data_last_hs_snoop,source=3/
1.105689920,262,,CCI_400/si_r_data_last_hs_snoop,source=3/
1.206255760,333,,CCI_400/si_r_data_last_hs_snoop,source=3/
1.306817120,985,,CCI_400/si_r_data_last_hs_snoop,source=3/
1.407395300,253,,CCI_400/si_r_data_last_hs_snoop,source=3/
...
```

Fig. 8. An example of the CCI profiling: complex pattern '456183270'

주기마다 CCI 카운터를 수집하고 그 결과를 출력 파일에 저장한다. CCI 프로파일링의 경우 "perf stat -i 100 -a -e CCI_400/cci_event/ -o *outputfile*" 명령을 사용한다. 프로파일링 도구는 '-a -e CCI_400/cci_event/' 옵션으로 전역 CCI 이벤트를 수집한다. 프로파일링 도구에서 안정적이고 가장 짧은 샘플링 시간은 100ms이므로 옵션 ('-i 100')로 설정한다.

Fig. 8은 패턴 '456183270'(Complex)의 CCI 데이터로, 각 줄에는 타임스탬프, CCI 카운터 및 소스(source)가 있는 CCI 이벤트 이름이 포함된다. 이 예시는 캐시 읽기 업데이트 카운터('si_r_data_last_hs_snoop')의 CCI 이벤트로, 소스는 CCI 이벤트 모니터링을 위한 포트 번호를 나타낸다.

VI. 성능 평가

본 장에서는 제안하는 CCI 기반 안드로이드 PSL 공격의 성능 평가를 위해 총 3가지 실험 결과를 설명한다. 먼저, #DC와 TTF를 통해 공격 가능한 안드로이드 PSL 패턴이 얼마나 되는지를 알아보고, TTF 값 계산에 있어서의 오류가 공격 정확도에 미치는 영향을 평가한다. 그 다음, 복잡한 PSL 패턴이 실제로 공격에 안전한지 평가한다.

6.1 #DC 및 TTF를 고려한 안드로이드 PSL 취약점

#DC, TTF 값 계산을 통한 안드로이드 PSL 공격의 영향을 평가하기 위해 먼저 전체 패턴 시퀀스(총 389,112개의 패턴)를 생성하고 동일한 #DC를 갖는 패턴 그룹을 생성한다. 그리고, 각 그룹에 대해 #G를 계산한다. 그 다음, 모든 패턴에 대해 TTF를 계산하여 각 패턴이 속한 #DC 그룹에 따라 TTF 그룹을 분류하였다.

Table 4. #G calculated from both #DC and TTF

#DC	# of TTF groups	#G			Vulnerable patterns	
		min.	max.	avg.	# of patterns	avg. #G
1	7	8	48	24	56	14
2	27	8	336	95	48	9.6
3	46	8	1,152	241	72	12
4	74	16	2,632	503	64	16
5	109	8	5,976	830	56	14
6	155	8	8,376	914	136	12.36
7	163	8	5,496	652	216	11.37
Total					648	avg. 12.76

Table 4는 전체 패턴을 #DC와 TTF 그룹으로 나눈 후 각 패턴 그룹의 #G 범위를 표시한 결과이다. Table 4에서 모든 패턴에 대해 #G의 범위는 8에서 8,376까지로 나타났으며, 이 중 20 이하의 값을 가지는 패턴들을 취약한 패턴으로 분류할 수 있다. 그 결과 총 648개의 패턴이 취약한 것으로 나타났으며, 취약한 패턴의 평균 #G는 12.76 이다. 즉, 평균 13회 정도의 공격을 통해 PSL 패턴 해제가 가능한 것을 의미한다.

6.2 TTF 오류와 공격의 정확도

실제 환경에서는 사용자가 매번 동일한 패턴을 다른 방식으로 그릴 수 있기 때문에, TTF의 정확한 추정이 어려울 수 있다. 따라서, TTF 추정의 오류를 고려하기 위해 제안하는 기법에서는 TTF 값을 최소 및 최대 값을 가지는 범위에서 추정한다. 따라서, TTF 그룹의 크기가 클수록, TTF 오류가 커질 수 있다. 예를 들어, TTF의 1% 오류는 TTF의 99%와 101% 사이의 TTF 그룹을 포함하는 것으로 통계적으로 이 1%는 TTF 평균값의 2% 표준편차에 해당한다. 4.2장에서 살펴보았듯이 5.66 TTF 그룹의 크기는 8로, 1% 오차를 적용하면 TTF 값의 범위는 최소 5.60에서 최대 5.71이 된다. 따라서,

범위 내 패턴의 수는 8개(5.66 TTF 그룹)에서 1,320개(5.65 TTF 그룹의 1312 패턴 + 5.66 TTF 그룹의 8 패턴 = 1,320 패턴)로 증가한다.

안드로이드의 PSL 해제 시간에 대한 기존 연구[10]에서 보고된 내용에 따르면, 패턴 잠금 해제 시간은 통계적으로 평균 910ms로 표준편차는 625ms이다. 표준편차 625ms가 910ms의 약 68.6%이기 때문에 잠금 해제 시간이 -34.3%에서 +34.3%까지 다양할 수 있다. 따라서, TTF 시간의 범위에 따른 안드로이드 PSL 공격의 정확도를 평가하기 위해 Table 5와 같이 TTF 오류가 #G 계산에 미치는 영향을 분석한다.

먼저, TTF 값에 오류가 없을 경우, 제안하는 기법을 통해 공격 가능한 취약 패턴은 총 648개이다. 만약, TTF 오류가 0.1%에서 5%로 증가하는 경우 취약한 패턴의 개수는 16개로 감소한다. 특히, TTF 오류가 1% 발생하는 경우 취약 패턴의 개수가 80% 수준 감소한다. 그 이유는 TTF 오류가 TTF 그룹의 범위를 증가시켜 취약 패턴의 추측을 어렵게 만들기 때문이다. 그 결과, TTF 오류가 10% 이상으로 높아지게 되면 제안하는 기법을 이용했을 때 공격 가능한 취약 패턴이 0개로 나타난다.

본 연구진은 실제 환경에서 TTF 오류의 발생 빈도를 파악하기 위해 안드로이드 응용앱(app)인 '패턴 잠금 트레이너(Pattern Lock Trainer)'를 개발하였다. 본 앱은 점의 개수에 따라 임의의 패턴을 생성하고 패턴의 TTF를 반복적으로 측정한다. 사용자는 시도횟수 또는 점의 수를 선택할 수 있으며 기본 설정은 3번의 시도와 4개의 점이다. 이 정보를 이용하여 패턴을 반복적으로 그려서 평균 TTF와 TTF 오류를 평가한다. 결과를 생성한 후 앱은 사용자 데이터 저장소인 구글 스프레드시트에 데이터를 생성한다. 앱을 통해 익명 사용자에 대한 공개 평가를 위해 공개 저장소[4]를 통해 앱을 배포하였다. 3주간의 테스트 기간 동안, 총 207개의 샘플과 69개의 테스트 패턴을 수집했다. 수집한 데이터에서 평균 TTF 및 TTF 오류는 각각 1937.61ms 및 6.08%로 나타났다. 수집한 데이터에서 단순한 패턴의 경우 평균 TTF 오류의 4.69%를 보여주었고, 이보다 복잡한 패턴들도 평균 TTF 오류의 7.04% 및 6.15%로 유사한 양상을 보였다.

4) https://github.com/saintgodkyp/PatternLockChallenge, 베타 릴리스 2019년 1월 24일.

Table 5. #G with TTF errors

TTF error (%)	#G (min.)	# of vulnerable patterns (#G<20)
0 (baseline)	8	296
0.1	8	256
0.5	8	104
1.0	8	40
5.0	48	0
≥10.0	72	0

Table 6. TTF error ratio in user data

Avg. TTF error (%)	# of patterns	Ratio (%)
≤1.50	3	4.35
≤2.00	7	10.14
≤3.00	14	20.29
≤4.00	32	46.38
≤5.00	38	55.07
≤10.00	60	86.96
≥10.00	9	13.04

Table 6은 전체 테스트된 패턴의 결과를 정리한 것으로 전체 패턴의 86.96%가 10% 미만의 TTF 오류를 보이는 것을 확인하였다.

6.3 복잡한 패턴의 안전성 평가

본 논문에서 제안한 CCI 기반의 안드로이드 PSL 공격 기법에 대한 안전성을 평가하기 위해 패턴 강도 측정기(pattern strength meter)[12]를 이용하여 서로 다른 복잡도 및 강도를 가지는 3개의 패턴을 추출하였다. Table 7은 기존 연구를 기반으로 세 가지 패턴의 특성을 분석한 결과이다.

Table 7에서 주어진 세 개의 패턴들의 패턴 강도 점수(PS score)[12] 범위는 10에서 41.68 사이의 범위를 가진다. 패턴 강도 점수는 패턴 길이(L), 교차(I), 겹침(O)과 같은 패턴 그리기 복잡도에 따라 증가하는 점수이다. 패턴 강도 측정기(PS meter)[19]는 패턴 길이, 비반복 세그먼트 비율, 교차점 수 등을 기반으로 한 점수이다. 기존 연구[19]의 저자는 점수를 세 가지 범주(weak, medium, strong)로 나누어 점수로 0에서 1까지로 차등화 하였다. 유사한 메트릭으로 패턴 복잡도

Table 7. Pattern strength comparison

Pattern	PS score[12]	PS meter (score) [19]	Pattern complexity [4]	L	I	O	#DC, TTF	#G
04678	10	Medium (0.44)	10 (Simple)	5	0	0	1, 4	8
426780	20.81	Medium (0.47)	18.68 (Medium)	6	1	1	3, 9.07	8
462350817	41.68	Strong (0.83)	42.68 (Complex)	9	6	1	7, 17.78	8

(Pattern complexity)[4]는 기존 연구[19]에서 수집된 120개의 패턴에 대한 패턴 강도 메트릭[12]을 기반으로 한 점수이다. 기존 연구들[12,19]을 기반으로 그 점수를 계산하면 6.340에서 46.807까지 다양하게 나타난다.

위의 세 가지 패턴에 대해 제안하는 기법을 활용하여 #DC와 TTF를 모두 계산했을 때, 세 가지 패턴의 #G는 모두 같은 값인 8로 나타났다. 첫 번째 패턴 '03678'은 #DC=1 및 TTF=4로 분류되어 우리의 기법을 이용하면 8회 시도 이내에 PSL을 해제할 수 있다. 또한, '426780'과 '462350817' 같은 복잡한 패턴의 경우에도 동일한 시도 횟수(8회)로 해제할 수 있음을 확인하였다. 이는 기존에 복잡한 패턴으로 분류된 것이라도 단순한 패턴과 비슷한 수준의 노력으로 해제할 수 있음을 의미하며, 기존 패턴 복잡성의 메트릭을 재검토해야 함을 시사한다.

VII. 관련 연구

본 장에서는 안드로이드 PSL 공격에 대한 기존 연구들에 대해 설명한다. 안드로이드 PSL 공격에 대한 연구들은 그 방법에 따라 크게 추측(guessing) 공격과 캡쳐(capture) 공격으로 나눌 수 있다.

추측 공격은 검색 공간의 모든 요소를 무자위 탐색(exhaustive search) 또는 지능적인 방법으로 검색하는 방법이다. 무작위 탐색 추측 공격은 빠른 테이블 검색을 위해 메모리와 사전 계산 시간을 교환하는 레인보우 테이블[20]을 사용한다. 사전 기반 추측 공격[21]은 사전 컴파일된 비교적 짧은 확률의 후보 암호 목록(dictionary)을 사용한다. 이는 사용자 행동에 대한 경험적 데이터 또는 가정을 기반으로 한다. 스마트 사전 공격[22]은 무작위 공격의 시간-메모리 트레이드오프를 고려하며, 특정한 경우를 반영한 알고리즘 기반의 우선순위 목록을 생성하면 성공 확률이 더 높아짐을 보였다.

캡처 공격은 사용자의 입력이나 사용자가 노출한 데이터를 직접 가로채서 암호를 얻는 방식으로 숄더 서핑(shoulder surfing)과 재구축(reconstruction)으로 나눌 수 있다. 숄더 서핑[23,24]은 그래픽 암호의 시각적 측면을 이용한 공격이고, 재구축[25]은 직접 암호를 캡처하는 대신 암호를 재구축하는 공격으로, 한 연구[26]에서는 사용자 로그인을 여러 번 관찰하면 몇 초 안에 사용자 암호를 재구축할 수 있음을 보였다.

또한, 안드로이드 PSL을 포함한 그래픽 암호에 대한 부채널 공격이 제안되었다. WiPass[27]는 WIFI 신호 간섭이 그래픽 암호의 부채널로 악용될 수 있음을 보여주었으며, 스머지 공격(Smudge attack)[7]은 터치스크린에 남겨진 기름진 얼룩(smudge)으로 사용자의 패턴을 추측하는 것이 가능함을 보였다. 다른 스머지 공격[16]은 패턴 설정에 대한 인간 행동 요인을 고려하였다. 인간이 자주 사용하는 시작 점이나 자주 쓰는 패턴 등에 초점을 두었다. 또 다른 스머지 공격 변형[17]은 자주 사용하는 점에 기반한 마르코프 모델[28]을 이용하면 스머지 공격이 더 효과적일 수 있음을 보였다. 최근에는 스마트폰의 충전 케이블을 이용하여 사용자의 터치 정보를 탈취[5]하는 부채널 공격이 제안되기도 하였다.

이 외에도 시각(vision) 기반 공격이 있다. 예를 들어, 한 연구[29]에서는 그리드의 크기를 사용하여 비디오 기반 PIN 공격이 가능하다고 제안하였다. 또 다른 연구[30]에서는 화면 밝기를 분석하여 PIN 기반 비밀번호 공격이 가능함을 보였다. 또한, 손가락 끝 추적 공격[3,4]이 제안되어 좋은 공격 성공률과 추측 비용을 보여주었다. 하지만 일반적으로 시각 기반 공격은 시야가 좋지 않은 상황에서는 공격이 효과적이지 않은 한계가 존재한다.

VIII. 결 론

본 논문에서는 빅리틀 구조 ARM 프로세서의 캐시 일관성 인터페이스인 ARM CCI를 이용한 안드로이드 PSL 공격의 설계 및 구현을 소개하였다. 본 공격은 2015년 5월 발표된 안드로이드 마시멜로 버전을 포함한 이후의 모든 안드로이드 기반 스마트폰에 적용할 수 있으며, 저수준의 하드웨어 인터페이스를 이용하여 중요한 사용자 인증 정보와 관련한 부채널을 성공적으로 생성할 수 있다. 또한, 본 논문은 PSL 공격을 통해 안드로이드 스크린 잠금 패턴을 효율적으로 식별할 수 있고 8번의 시도만으로도 해제 가능한 패턴을 찾아낼 수 있음을 보였다.

References

[1] Man Zhou, et al., "Patternlistener: Cracking android pattern lock using acoustic signals," Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1775-1787, Oct. 2018.

[2] Man Zhou, et al., "Stealing your android patterns via acoustic signals," IEEE Transactions on Mobile Computing, vol. 20, no. 4, pp. 1656-1671, Apr. 2021.

[3] Guixin Ye, et al., "A Video-based Attack for Android Pattern Lock," ACM Transactions on Privacy and Security, vol. 21, no. 4, pp. 1-31, Nov. 2018.

[4] Guixin Ye, et al., "Cracking Android Pattern Lock in Five Attempts," Proceedings of the 2017 Network and Distributed System Security Symposium, pp. 1-15. Feb. 2017.

[5] Patrick Cronin, et al., "Charger-Surfing: Exploiting a Power Line Side-Channel for Smartphone Information Leakage," Proceedings in 30th USENIX Security Symposium, pp.681-698, Aug. 2021.

[6] ARM Developer site, "Corelink cci-400 cache coherent interconnect technical reference manual," http://infocenter.arm.com/help/topic/com.arm.doc.ddi0470c/DDI0470C_cci400_r0p2_trm.pdf, Mar. 2022.

[7] Adam J. Aviv, et al., "Smudge attacks on smartphone touch screens," Proceedings of the Workshop on Offensive Technologies, pp. 1-7, Aug. 2010.

[8] Wolfram Mathworld site, "Parallelogram law," http://mathworld.wolfram.com/ParallelogramLaw.html, Mar. 2022.

[9] Michel Marie Deza and Elena Deza, Encyclopedia of distances, Springer, Berlin, Heidelberg, pp. 1-583, 2009.

[10] Marian Harbach, et al., "The anatomy of smartphone unlocking: A field study of android lock screens," Proceedings of the 2016 CHI conference on Human Factors in Computing Systems, pp. 4806-4817, May. 2016.

[11] J. Bonneau, "The science of guessing: analyzing an anonymized corpus of 70 million passwords," Proceedings of the 2012 IEEE Symposium on Security and Privacy, pp. 538 - 552, Jul. 2012.

[12] C. Sun, et al., "Dissecting pattern unlock: The effect of pattern strength meter on pattern selection," Elsevier Journal of Information Security and Applications, vol. 19, no. 4-5, pp. 308 - 320, Nov. 2014.

[13] S. Uellenbeck, et al., "Quantifying the security of graphical passwords: the case of android unlock patterns," Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security. pp. 161 - 172, Nov. 2013.

[14] James L. Massey, "Guessing and entropy." Proceedings of the 1994 IEEE International Symposium on

Information Theory, pp. 204-204 , Jul. 1994.
[15] Christian Cachin, "Entropy measures and unconditional security in cryptography," doctorial dissertation, ETH Zurich, 1997.
[16] Panagiotis Andriotis, et al., "A pilot study on the security of pattern screen-lock methods and soft side channel attacks," Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks, pp. 1-6, Apr. 2013.
[17] Seunghun Cha, et al., "Boosting the guessing attack performance on android lock patterns with smudge attacks," Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 313-326, Apr. 2017.
[18] Roger Schneider, "Survey of peaks/valleys identification in time series," Student project 2011, Department of Informatics, University of Zurich, Aug. 2011.
[19] Youngbae Song, et al., "On the effectiveness of pattern lock strength meters: Measuring the strength of real world pattern locks," Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, pp. 2343-2352, Apr. 2015.
[20] Philippe Oechslin, "Making a faster cryptanalytic time-memory trade-off," Annual International Cryptology Conference, LNCS 2729, pp. 617-630, Aug. 2003.
[21] Darren Davis, et al., "On user choice in graphical password schemes, " Proceedings of the USENIX security symposium, pp. 151-164, Aug. 2004.
[22] Arvind Narayanan and Vitaly Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff, " Proceedings of the 12th ACM conference on Computer and communications security, pp. 364-372, Nov. 2005.
[23] Volker Roth, et al., "A PIN-entry method resilient against shoulder surfing," Proceedings of the 11th ACM conference on Computer and communications security, pp. 236-245, Oct. 2004.
[24] Michael Backes, et al., "Compromising reflections-or-how to read LCD monitors around the corner," Proceedings of the 2008 IEEE Symposium on Security and Privacy, pp. 158-169, May 2008.
[25] Baris Coskun and Cormac Herley, "Can "something you know" be saved? " International Conference on Information Security, LNCS 5222, pp. 421-440, Sep. 2008.
[26] Philippe Golle and David Wagner, "Cryptanalysis of a cognitive authentication scheme," Proceedings of the 2007 IEEE Symposium on Security and Privacy, pp. 66-70, May 2007.
[27] Jie Zhang, et al., "Privacy leakage in mobile sensing: Your unlock passwords can be leaked through wireless hotspot functionality," Hindawi Mobile Information Systems, vol. 2016, no. 8793025, Apr. 2016.
[28] Claude Castelluccia, et al., "Adaptive password-strength meters from markov models," Proceedings of the 19th Annual Network and Distributed System Security Symposium, pp. 1-14, Feb. 2012.
[29] Diksha Shukla, et al., "Beware, your hands reveal your secrets!," Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp.

904-917, Nov. 2014.
[30] Qinggang Yue, et al., "Blind recognition of touched keys on mobile devices," Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1403-1414, Nov. 2014.
[31] M. Egele, et al., "A survey on automated dynamic malware-analysis techniques and tools," ACM computing surveys, vol. 44, no. 2, pp. 1-42, Feb. 2012.

〈저 자 소 개〉

김 영 필 (Youngpil Kim) 정회원
2002년 2월: 고려대학교 컴퓨터학과 졸업
2004년 2월: 고려대학교 컴퓨터학과 석사
2015년 2월: 고려대학교 컴퓨터학과 박사
2015년 3월~2019년 2월: 고려대학교 정보대학 융합소프트웨어 연구소 연구교수
2019년 3월~2021년 8월: 세명대학교 컴퓨터학부 조교수
2021년 9월~현재: 인천대학교 정보통신공학과 조교수
〈관심분야〉 운영체제, 시스템보안, 클라우드컴퓨팅

이 경 운 (Kyungwoon Lee) 정회원
2010년 2월: 경북대학교 전자전기컴퓨터학부 졸업
2015년 2월: 고려대학교 임베디드소프트웨어학과 석사
2020년 8월: 고려대학교 컴퓨터학과 박사
2020년 9월~2022년 2월: 고려대학교 정보대학 융합소프트웨어 연구소 연구교수
2022년 3월~현재: 경북대학교 IT대학 전자공학부 조교수
〈관심분야〉 운영체제, 서버 가상화, 클라우드컴퓨팅

유 시 환 (Seehwan Yoo) 정회원
2002년 2월: 고려대학교 컴퓨터학과 졸업
2004년 2월: 고려대학교 대학원 이학석사
2013년 2월: 고려대학교 대학원 이학박사
2014년 3월~현재: 단국대학교 모바일시스템공학과 부교수
〈관심분야〉 운영체제, 시스템보안, 모바일컴퓨팅

유 혁 (Chuck Yoo) 종신회원
1984년 2월: 서울대학교 전자공학과 졸업
1986년 8월: University of Michigan at Ann arbor 석사
1990년 8월: University of Michigan at Ann arbor 박사
1990년 8월~1995년 2월: Sun Microsystems Lab 연구원
1995년 3월~현재: 고려대학교 정보대학 교수
〈관심분야〉 운영체제, 소프트웨어정의네트워크, 디지털헬스