Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.34, NO.4, Aug. 2024 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2024.34.4.651

LockPickFuzzer: ADB 기반 퍼징 기법을 활용한 안드로이드 잠금 화면 메커니즘의 취약점 탐색*

고 대 훈,[1†] 김 형 식[2‡]
[1,2]성균관대학교 (대학원생, 교수)

LockPickFuzzer: Exploring Vulnerabilities in Android Lock Screen Mechanisms through ADB-Based Fuzzing*

Daehoon Ko,[1†] Hyoungshick Kim[2‡]
[1,2]Sungkyunkwan University (Graduate student, Professor)

요 약

안드로이드 디바이스는 다양한 인증 방식을 제공하는 잠금 화면으로 사용자 데이터를 보호한다. 그러나 잠금 화면이 활성화된 상태에서도 Android Debug Bridge(ADB)를 통해 디바이스에 접근할 수 있다. 본 연구에서는 ADB의 특성을 활용하여 잠금 화면 보안 메커니즘을 우회할 수 있는 방법을 탐색하고자 한다. 이를 위해 ADB 명령어를 분석하고, 잠금 화면 보안을 무력화할 수 있는 명령어 조합을 자동으로 탐색하는 퍼징 테스트 도구인 LockPickFuzzer를 제안한다. LockPickFuzzer의 성능을 평가하기 위해 안드로이드 14를 탑재한 갤럭시 S23과 픽셀 8을 대상으로 실험을 진행하였다. 실험 결과, 잠금 화면의 인증 정보를 탈취하거나 우회할 수 있는 두 가지 ADB 명령 조합을 발견하였다. 이 발견된 취약점에 대해 삼성 보안팀에 리포트를 제출하였고, 한 가지 ADB 명령어 조합에 대해 삼성전자에서 공식적으로 인정받았다 (SVE-2023-1344). LockPickFuzzer는 자동으로 작동하며, 안드로이드 디바이스에서 ADB 명령어 조합으로 인한 보안 취약점을 효과적으로 탐지하는 데 기여할 것으로 기대된다.

ABSTRACT

Android devices employ lock screens with various authentication methods to protect user data. However, even with the lock screen active, the device can be accessed via the Android Debug Bridge(ADB), a powerful development tool that controls devices connected through USB. In this paper, we explore methods to bypass the lock screen security mechanism by leveraging the characteristics of ADB. To achieve this, we analyze ADB commands to categorize those that can severely impact the Android system and propose LockPickFuzzer, a fuzzing test tool that automatically explores ways to combine these commands to disable lock screen security. To demonstrate LockPickFuzzer's ability to detect security vulnerabilities using ADB, we conducted experiments on the Galaxy S23 and Pixel 8, both running Android 14. The results revealed two ADB command combinations that could either steal authentication information or bypass the lock screen. We submitted a report on these discovered vulnerabilities to the Samsung security team and received official acknowledgment (SVE-2023-1344) from Samsung Electronics for one ADB command combination that can be reproduced on user devices. LockPickFuzzer is a practical tool that operates automatically without user intervention and is expected to contribute to the effective detection of security vulnerabilities caused by ADB command combinations on Android devices.

Keywords: Android Security, Lock Screen Bypass, Fuzzing, ADB, Authentication Vulnerabilities

Received(06. 03. 2024), Accepted(07. 09. 2024)
* 이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기술평가원의 지원을 받아 수행된 연구임 (RS-2018-II180532, 고등급(EAL6 이상) 보안 마이크로커널 개발).
† 주저자, kodh21@gmail.com
‡ 교신저자, hyoung@skku.edu(Corresponding author)

I. 서 론

현대 사회에서 스마트폰은 개인 및 업무 생활의 핵심 도구로 자리 잡았으며, 이에 따라 스마트폰 내 데이터 보호의 중요성 또한 급증하고 있다. 사이버 공격의 빈도와 정교함이 지속적으로 증가함에 따라, 모바일 장치의 취약점을 악용하여 민감한 정보를 탈취하는 사례가 빈번해지고 있다. 안드로이드 디바이스는 이러한 위협으로부터 데이터를 보호하기 위해 잠금 화면 기능을 제공한다. 잠금 화면은 무단 접근을 방지하는 첫 번째 방어선으로, 디바이스가 도난을 당하거나 분실된 경우 중요한 개인정보와 업무 데이터를 보호하는 역할을 한다.

안드로이드 잠금 화면[9]은 사용자 데이터를 보호하기 위한 목적으로 설계되었다. 잠금 화면은 핀, 패턴, 비밀번호, 생체 인식 등 다양한 인증 방식을 제공하며, 디바이스가 분실되거나 도난을 당했을 때 무단 접근을 방지하는 기능을 한다. 그러나 잠금 화면이 활성화된 상태에서도 일부 인터페이스를 통해 디바이스에 접근할 수 있는 가능성이 있으며, 이를 악용하여 공격자들은 잠금 화면을 우회하거나 인증 정보를 탈취하려는 시도를 할 수 있다. 가장 일반적인 접근 방식 중 하나는 USB 포트를 이용한 접근이다. 특히 Android Debug Bridge(ADB)[7]를 활용하여 디바이스에 접근함으로써 잠금 화면을 우회하여 데이터를 추출할 수 있다. 초기 안드로이드 버전에서는 ADB를 통해 디바이스의 잠금 화면 설정 파일을 삭제하여 잠금 화면을 해제하는 방법[17]이 있었으나 최신 버전에서 동작하지 않는다. 또한 터치스크린을 통한 상호작용도 공격자들에게 잠금 화면 우회 가능성을 제공한다. 예를 들어, 안드로이드 5.x 버전에서는 긴급 통화 인터페이스를 통한 잠금 화면 우회가 가능했으나, 안드로이드 6.0 이후로는 이 취약점이 보완되었다[8]. 안드로이드는 이러한 취약점을 해결하기 위해 정기적으로 보안 패치를 제공하고 있으나, ADB와 같은 인터페이스는 개발 및 디버깅 용도로 여전히 사용되고 있다.

본 연구의 목적은 ADB 명령어의 조합을 통해 안드로이드 장치의 잠금 화면 보호 메커니즘을 무력화할 수 있는 경로를 발견하는 것이다. 시스템에 영향을 미칠 수 있는 ADB 명령어들을 Critical ADB 명령어 집합으로 분류하고, 이를 바탕으로 잠금 화면 보안을 해제할 수 있는 경로를 탐색하는 퍼징 테스트를 설계하고 자동으로 수행할 수 있는 퍼징 테스트 도구를 개발하였다. 유의미한 테스트 케이스를 생성하기 위해 분류된 ADB 명령어의 템플릿을 분석하고 검사 대상 디바이스의 정보를 기반으로 초기 입력 데이터를 생성하고 활용한다. 퍼징 테스트 과정에서 잠금 화면의 인증 정보를 추출하거나 잠금 화면이 우회되었다고 판단된 경우, 누적된 전체 명령어 집합을 최적화하여 최소 명령어 집합을 도출한다.

이 도구를 사용하여 안드로이드 버전 14을 탑재한 갤럭시 S23과 픽셀 8에서 인증 정보를 탈취하거나 잠금 화면을 우회할 수 있는 두 가지 경로를 발견하였다. 발견된 취약점에 대해 삼성 보안팀에 리포트를 제출하였고, 사용자 단말에서 재현 가능한 취약점에 대해 SVE-2023-1344를 할당받았다. 이러한 도구는 안드로이드 스마트폰 제조사가 디바이스를 출시하기 전에 잠재적인 공격을 식별하는 데 큰 도움을 줄 수 있다.

II. 배 경

2.1 안드로이드 잠금 화면

안드로이드 장치의 잠금 화면은 사용자의 개인 정보 보호를 위해 필수적인 보안 메커니즘으로, 초기 버전에서는 핀, 패턴, 패스워드 등의 기본적인 보안 수단을 제공하였다. 핀은 숫자로 설정된 코드, 패턴은 점을 연결하여 만든 도형, 패스워드는 문자와 숫자의 조합으로 구성된다. 안드로이드 5.0 에서는 도입된 스마트 락 기능은 사용자의 위치나 연결된 블루투스 장치와 같은 신뢰할 수 있는 요소를 기반으로 장치를 자동으로 잠금 해제할 수 있는 확장된 보안 옵션이 추가되었다. 이후 안드로이드 6.0 에서는 지문 인식이, 9.0 파이에서는 얼굴 인식 기능이 추가되었다.

잠금 화면 기능의 발전에도 불구하고, 잠금 화면의 보안 메커니즘이 우회하는 여러 취약점[10,11,12,18,19]이 발견되었다. Schutz는[11] 심카드의 PUK(Personal Unblocking Key)를 이용하여 인증없이 잠금 화면을 해제할 수 있는 취약점을 발견했다. 이 방법은 사용자가 심 카드 핀 코드를 세 번 잘못 입력한 후 PUK 코드로 심카드를 잠금 해제하면 안드로이드 디바이스의 잠금 화면을 우회할 수 있다. 또한, Rodriguez는[12] 구글 어시스턴트의 번역 모드를 이용한 잠금 화면 우회 취약점을 발견하였다. 사용자는 구글 어시스턴트를 통해 번

역 모드에 진입한 후, 키보드를 활용하여 Google Maps 링크를 입력하고 몇 가지 단계를 거쳐 Google Maps로 이동함으로써 잠금 화면을 우회할 수 있다. Potocky 등 [10]은 HID((Human Interface Device)공격을 이용하여 비생체 잠금 화면을 우회하는 연구를 진행했으며, 이 방법은 USB OTG와 Android-PIN-Bruteforce[23]를 사용하여 자동으로 핀 입력을 시도하는 방법으로, 이 방법은 안드로이드 장치에만 적용되며, 핀에만 한정된다. 또한, Ertam 등[13]의 연구에서는 경량 딥러닝 모델을 활용하여 안드로이드 장치의 패턴 잠금 화면을 감지하는 방법을 제안하였다. 이 방법은 효율적인 패턴 식별을 가능하게 하지만, 연구에서 사용된 특정 타입의 데이터에만 최적화되어 있어 다양한 장치나 새로운 안드로이드 버전에서의 범용성과 확장성이 제한적일 수 있다는 한계가 있다. 또한, 복잡한 패턴이나 빈번하게 변경되는 보안 설정에 대해선 정확도가 떨어질 가능성이 있다.

스마트폰은 다양한 기능과 휴대성 때문에 분실 또는 도난 가능성이 높다. 이로 인해 잠금 화면의 보안 메커니즘이 약화될 경우, 사용자 개인정보의 유출, 금융 정보의 도용, 중요한 데이터의 무단 접근 등 심각한 보안 문제가 발생할 수 있다.

2.2 ADB (Android Debug Bridge)

ADB는 안드로이드 디바이스와 에뮬레이터를 관리하고 상호 작용하기 위해 설계된 다목적 명령줄 도구이다. ADB는 서버, 클라이언트, 그리고 데몬(adbd)의 세 가지 주요 구성 요소로 이루어져 있으며, 이를 통해 개발자는 호스트 컴퓨터에서 안드로이드 디바이스와 직접적으로 통신할 수 있다. 이 도구를 사용함으로써, 개발자들은 안드로이드 디바이스 내의 애플리케이션 및 서비스를 관리하고, 필요한 설정 변경이나 파일 주입을 수행함으로써 소프트웨어 개발 및 테스트 과정을 최적화할 수 있다. 또한, 시스템 로그 수집과 다양한 성능 지표의 모니터링을 통해 디바이스의 상태를 정확히 파악하고 시스템 오류를 진단하며, 이를 통해 디바이스의 성능을 분석하고 개선할 수 있다.

이러한 ADB의 기능들은 안드로이드 어플리케이션 테스트 자동화[6] 및 디바이스 내 악성 도구 탐지 방법론[5] 개발에 널리 활용되었다. 하지만, ADB의 강력한 기능은 동시에 악의적인 목적으로도

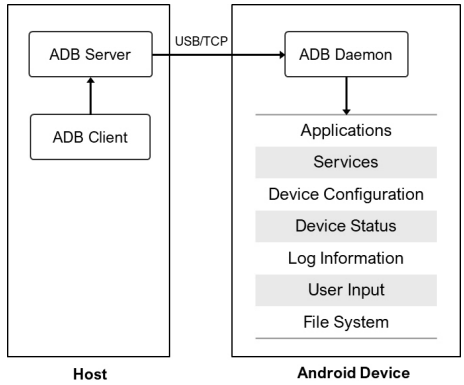

Fig. 1. Android Debug Bridge.

활용될 수 있는 잠재력을 가지고 있다. 예를 들어, Lin 등 [1]은 ADB를 이용하여 화면 정보를 추출하여 민감한 정보를 수집할 수 있는 Screenmilker 프레임워크를 개발하였다. 또한, Mohamed 등[3]은 ADB를 이용하여 안드로이드 디바이스의 센서 데이터를 스니핑하고 조작할 수 있는 Smashed라는 악성 어플리케이션을 개발하였으며, 이 어플리케이션은 센서 데이터의 조작을 통해 개인 정보를 유출하거나 인증 시스템을 무력화할 수 있다. Yang 등[4]은 ADB를 활용하여 시스템 애플리케이션의 로그 데이터를 확보하고 로그인을 요구하는 애플리케이션의 활동을 철저히 모니터링하는 기법을 개발하여, 사용자의 계정 정보를 무단으로 탈취하는 공격 시나리오를 구현하였다. 마지막으로, Hwang 등[2]은 ADB를 통해 가능한 다양한 공격 유형을 정리하고, 이러한 공격들이 안드로이드 디바이스 보안에 어떤 위협을 가할 수 있는지 분석하였다. 이와 같은 연구들은 ADB가 악의적으로 사용될 수 있음을 보여준다

III. 위협 모델

공격자의 목표는 도난 또는 분실된 잠금 화면이 설정된 안드로이드 디바이스에서 사용자 데이터를 탈취하고 기기를 제어하는 것이다. 이를 위해 공격자는 잠금 화면 보안 메커니즘을 무력화하려 한다. 이 시나리오에서 공격자는 물리적으로 잠금 화면이 설정된 안드로이드 디바이스에 접근할 수 있으며, ADB를 사용할 수 있는 상황을 가정한다.

일반적으로 ADB를 사용하기 위해서는 개발자 모드를 통해 USB 디버깅 모드를 활성화해야 한다. 그러나 다양한 보안 취약점을 통해 비정상적인 경로로

ADB가 활성화될 수 있다. 예를 들어, Salaxy[20]는 접근성 서비스를 악용하여 사용자가 인지하지 못한 상태에서 USB 디버깅 모드를 자동으로 활성화하는 방법을 제시하였다. Tian 등[21]은 특정 안드로이드 디바이스에서 AT 명령어를 이용해 USB 디버깅을 직접 활성화할 수 있음을 보여주었다. 이는 디바이스 제조사가 보안을 충분히 고려하지 않은 구성으로 인한 것이다. 추가적으로, 공격자는 사용자를 속여 USB 디버깅 모드를 활성화하도록 유도하는 방법[22]을 사용할 수 있다. 예를 들어, 공격자는 USB 디버깅을 활성화하도록 유도하는 안드로이드 디바이스 관리용 컴퓨터 프로그램을 제공하고, 해당 프로그램의 기능을 이용하기 위해 사용자가 USB 디버깅을 활성화하도록 만들 수 있다. 이러한 상황을 상정하여, 공격자는 대상 안드로이드 디바이스의 ADB 접근 권한을 확보할 수 있다고 가정한다.

IV. Critical ADB 명령어 집합

Critical ADB 명령어 집합은 안드로이드 디바이스의 시스템에 중대한 영향을 미칠 수 있는 ADB 명령어들로 구성된다. 이 명령어들은 시스템의 안정성, 보안, 성능에 큰 영향을 줄 수 있으며, 잘못 사용될 경우 시스템의 정상적인 작동을 방해하거나 보안 취약점을 유발할 수 있다. Table 1.과 같이 Critical ADB 명령어를 하기 기준에 따라 세 가지로 분류한다.

(1) 외부에서 데이터를 주입할 수 있는 명령어

이 범주의 명령어들은 외부 데이터를 시스템에 주입할 수 있으며, 악의적인 목적으로 어플리케이션 및 파일을 삽입할 수 있다.

예를 들어, adb install 명령어는 디바이스에 APK 파일을 설치하는 데 사용된다. 로컬 컴퓨터에 있는 testapp.apk 파일을 디바이스에 설치하려면

Table 1. Critical ADB Command Set.

Type	Command
(1) External Data Injection	adb push [file] [file-path]
	adb install [apk-file]
	adb shell setprop [property] [value]
(2) Changing System Component States	adb shell pm grant [package] [permission]
	adb shell pm enable [package]
	adb shell pm disable [package]
	adb shell pm clear [package]
	adb shell pm uninstall [package]
	adb shell pm uninstall-system-updates
	adb shell am start -n [component]
	adb shell am broadcast -a [action]
	adb shell am startservice -n [service]
	adb shell am stopservice -n [service]
	adb shell service call [service] [code] [args...]
(3) Modifying System Information	adb shell content insert --uri [content-uri] --bind [column]:[type]:[value]
	adb shell content update --uri [content-uri] --bind [column]:[type]:[value]
	adb shell content delete --uri [content-uri]
	adb shell content query --uri [content-uri]
	adb shell content call --uri [content-uri] --method [method] --arg [arg]
	adb shell device_config put [namespace] [key] [value]
	adb shell settings put [namespace] [key] [value]
	adb shell cmd [service] [command] [arguments]

```
adb install testapp.apk
```

Fig. 2. Example of External Data Injection Command: adb install.

다음과 같은 명령어를 사용할 수 있다[Fig. 2.].

(2) 시스템 컴포넌트의 실행 상태를 변경시킬 수 있는 명령어

이 범주의 명령어들은 시스템 컴포넌트의 실행 상태를 변경하여 시스템 동작에 영향을 줄 수 있다.

예를 들어, adb shell am startservice 명령어는 디바이스에서 특정 서비스를 시작하는 데 사용된다. 위치 기반 서비스를 수행하는 com.example.locationservice 패키지가 있다고 가정할 때, 서비스를 시작하면서 위치 업데이트 간격을 전달하려면 다음과 같은 명령어를 사용할 수 있다[Fig. 3.].

이 명령어는 com.example.locationservice 패키지의 LocationService를 시작하고, 업데이트 간격을 나타내는 키 값인 update_interval에 5000 밀리초의 정수 값을 설정한다. 이를 통해 서비스의 동작을 제어할 수 있다.

```
adb shell am startservice -n com.example.locationservice/.LocationService
 --ei update_interval 5000
```

Fig. 3. Example of Changing System Component Execution State: adb shell am startservice.

(3) 시스템 컴포넌트의 정보를 변경시킬 수 있는 명령어

이 범주의 명령어들은 시스템 설정이나 속성을 변경하여 시스템의 동작 방식에 영향을 미칠 수 있다.

예를 들어, adb shell content 명령어는 콘텐츠 프로바이더를 통해 시스템 컴포넌트의 데이터를 읽거나 수정할 수 있는 명령어이다. 앞서 언급된 위치 기반 서비스인 LocationService에 위치 데이터를 삽입하려면 다음과 같은 명령어를 사용할 수 있다[Fig. 4.].

이 명령어를 통해 latitude, longitude, timestamp 값을 가진 새 위치 데이터를 삽입할 수 있다. 이를 통해 시스템 컴포넌트의 데이터베이스의 특정 정보를 변경할 수 있다.

Table 2.는 Critical ADB 명령어 집합에서 각 명령어 별 파라미터와 그 설명을 나타낸다. 각 명령어를 정상적으로 실행하기 위해선 정의된 명령어 템플릿

Table 2. Command Parameters and Descriptions.

Command	Parameter	Description
push	file	Path to the local file to be transferred
	file-path	Destination path on the device
install	apk-file	Path to the APK file to be installed on the device
setprop	property	Key of the system property to be modified
	value	Value to set for the system property
pm	package	Name of the package
	permission	Specific permission to be granted to the package
am	component	Name of the component (activity) to start
	action	Action to perform in the broadcast
	service	Name of the service to start or stop
service	service	Name of the system service to be called
	code	Method code to execute
	args	Arguments to pass to the service method
content	content-uri	URI of the content provider
	column	Column where data will be manipulated
	type	Type of data to be handled
	value	Data value to insert, update, or match in delete/query
	method	Method to call on the content provider
	arg	Argument for the method call

```
adb shell content insert --uri content://com.example.locationservice/locations
--bind latitude:d:37.7749 --bind longitude:d:-122.4194 --bind
timestamp:l:1627891234567
```

Fig. 4. Example of Modifying System Component Information: adb shell content.

에 맞춰 유효한 파라미터 값이나 파일이 필요하다.

V. LockPickFuzzer

이 장에서는 제안된 퍼징 도구인 LockPickFuzzer의 과정을 설명한다.

Fig. 5.는 잠금 화면이 설정된 안드로이드 디바이스에서 다양한 ADB 명령어의 조합을 통해, 장치에서 잠금 화면의 인증 정보를 추출하거나 잠금 화면을 우회하는 명령어 시퀀스를 탐색하는 퍼징 도구 LockPickFuzzer의 실행 흐름도를 나타낸다. 이 도구는 데스크톱 PC에서 실행되며, 안드로이드 디바이스와 USB로 연결되어 안드로이드 SDK에서 제공하는 ADB 인터페이스를 통해 퍼징 작업을 수행한다. LockPickFuzzer의 주요 과정은 다음과 같다.

첫째, 도구는 초기 ADB 명령어 실행을 위한 테스트 케이스를 준비한다. 둘째, 무작위로 선정된 ADB 명령어를 실행하며, 실행 결과로부터 로그와 메모리 덤프를 수집한다. 셋째, 수집된 정보를 분석하여 인증 정보를 확보하거나 잠금 화면이 우회된 경우, 수집된 명령어 집합을 최적화하여 결과 보고서를 작성한다. 이러한 각 단계는 뒤이어 설명할 절에서 자세히 다룬다.

5.1 Preprocessing

Fig. 6.은 퍼징 테스트를 효과적으로 수행하기 위해 테스트 케이스를 구성하는 초기 입력 데이터 셋을 생성하는 단계이다. 테스트 케이스 구성은 사전에 분류된 Critical ADB 명령어들의 구조화된 명령어 템플릿과 명령어를 구성하는 파라미터에 크게 의존한

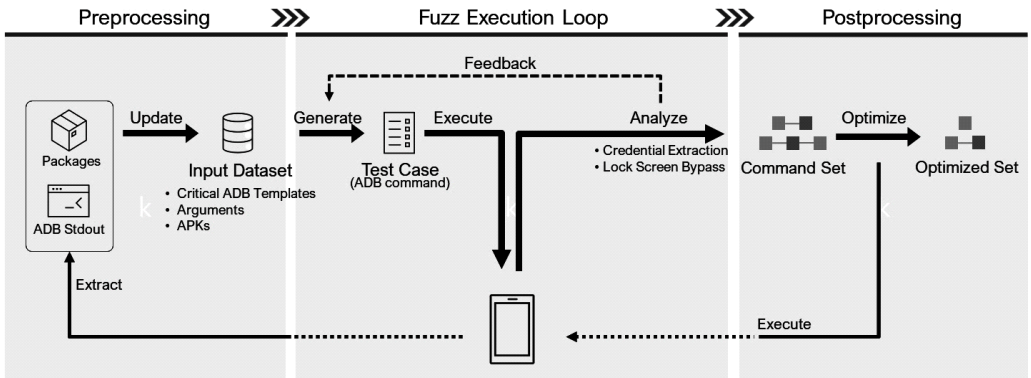

Fig. 5. Overview and workflow of LockPickFuzzer.

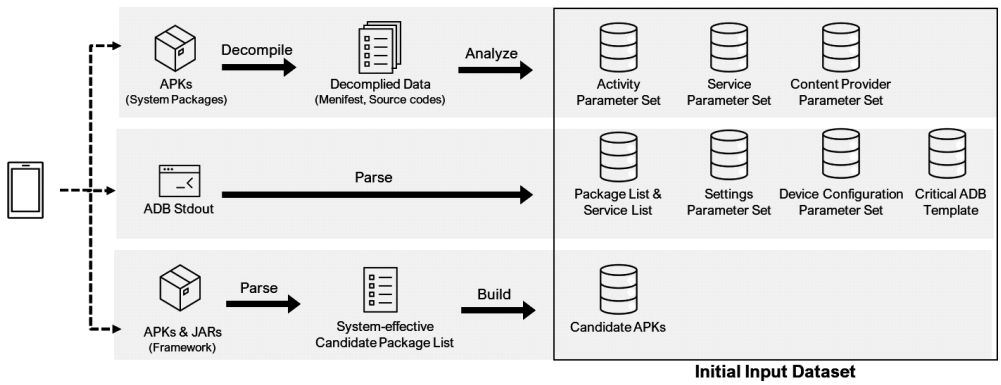

Fig. 6. Generating the Initial Input Dataset in LockPickFuzzer's Preprocessing.

다. 이러한 파라미터 값들은 대부분 테스트 디바이스의 구체적인 정보를 반영하는 값들로 구성되어 있으며, 이는 테스트 케이스의 유효성에 직접적인 영향을 미친다. 따라서 퍼징 테스트의 효율성을 높이기 위해 디바이스로부터 데이터를 수집하고, 이를 통해 적절한 파라미터 초기값을 확보하는 과정은 필수적이다.

초기 입력 데이터 셋의 생성이 완료되면, 테스트 디바이스를 초기화하고 잠금 설정을 수행한다.

5.1.1 디바이스 정보 수집

ADB 명령어를 활용하여 퍼징 대상 디바이스의 시스템 애플리케이션과 프레임워크 리스트와 산출물, 디바이스 구성 정보 및 설정 정보를 수집한다.

시스템 애플리케이션 정보 수집

장치에 설치된 모든 애플리케이션의 구조적 세부 사항은 adb shell pm list packages -f 명령을 통해 얻어진다. 이 명령은 각 애플리케이션의 APK 파일 경로와 패키지 이름을 반환하며, adb pull 명령을 이용하여 해당 APK 파일들을 로컬 환경으로 추출할 수 있다. 추출된 APK 파일은 후속 분석을 위한 기초 자료로 활용된다.

시스템 프레임워크 정보 수집

장치에 활성화된 모든 시스템 서비스는 adb shell service list 명령을 통해 식별된다. 안드로이드 시스템의 핵심 라이브러리와 서비스 구현은 /system/framework 내의 JAR 파일로 존재하며, adb pull 명령으로 추출되어 후속 분석에 활용된다.

디바이스 구성 정보 및 설정 정보 수집

디바이스의 구성 요소 및 파라미터 정보는 adb shell device_config list 명령을 통해 수집한다. 그리고 adb shell settings list system/secure/global 명령을 통해 시스템, 보안, 글로벌 설정에 대한 상세 파라미터 정보를 수집한다.

5.1.2 컴포넌트 정보 수집

시스템 애플리케이션과 프레임워크의 컴포넌트 정보를 수집하는 과정은 다음과 같다. 먼저 디바이스 정보 수집 단계에서 추출된 APK 파일과 JAR 파일들을 jadx[14]를 사용하여 디컴파일한다. 이후 디컴파일된 AndroidManifest.xml 파일 및 소스 코드를 자세히 분석하여 각 컴포넌트를 식별하고, 이 컴포넌트들이 수용하는 파라미터 정보를 수집한다.

AndroidManifest.xml 분석을 통한 컴포넌트 식별

Fig. 7.는 AndroidManifest.xml의 예시를 보여준다. AndroidManifest.xml에서 컴포넌트 태그를 조사하고 각 태그 내의 android:name 속성을 통해 컴포넌트 이름을 식별한다. 컴포넌트 태그는 안드로이드에 사전에 정의되어 있으며, 액티비티는 〈activity〉, 서비스는 〈service〉, 컨텐트 프로바이더는 〈provider〉, 인텐트 필터는 〈intent-filter〉로 정의되어 있다. 추가적으로 adb를 통해 접근 가능한 컴포넌트를 식별하기 위해 android:exported 속성이 true로 설정되어 있거나 인텐트 필터가 설정된 컴포넌트 이름만 수집한다.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.testapp">
    <application ...>
        <activity
            android:name="com.example.testapp.MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service
            android:name="com.example.testapp.ExampleService"
            android:exported="true" />
        <provider
            android:name="com.example.testapp.ExampleProvider"
            android:authorities="com.example.testapp.provider"
            android:exported="true" />
        ...
    </application>
</manifest>
```

Fig. 7. Example of AndroidManifest.xml.

액티비티 및 서비스에 대한 파라미터 정보 수집

ADB를 통해 액티비티와 서비스 컴포넌트에 데이터를 전달하는 과정은 인텐트를 사용한다. 이 방법을 통해 액티비티와 서비스는 다양한 기본 데이터 타입 및 배열 또는 리스트 형태의 데이터를 수용할 수 있다. Fig. 8.는 액티비티를 시작하거나 서비스를 호출할 때 데이터를 전달하는 명령어 예시이다.

액티비티나 서비스에서 getExtras()와 같은 메

```
adb shell am start -n com.example.app/.MainActivity -es "key" "value"
adb shell am start-service -n com.example.app/.ExampleService -ei "num" 123
```

Fig. 8. Example ADB Commands for Data Transmission to Activity and Service.

```
Public class ExampleActivity extends AppCompatActivity {
…
    Intent intetn =getIntent();
    String extraData = intent.getStringExtra("key");
…
}
```

Fig. 9. Sample for Extracting String Data from an Intent in an Activity.

소드를 사용하여 인텐트 데이터를 수용한다. Fig. 9.는 액티비티에서 인텐트를 받아 문자열 데이터를 추출하는 코드의 예시를 보여준다.

Table 3.과 같이 인텐트에서 데이터를 추출하는 메소드는 다양하며, 각 메소드는 특정 데이터 타입을 반환한다.

앞서 설명한 인텐트 데이터 전달 및 처리 방법을 바탕으로, 데이터 타입과 키 값을 수집하는 과정은 다음과 같이 진행된다. 먼저 AndroidManifest.xml에서 식별된 각 컴포넌트의 이름을 기반으로 소스 코드 내에서 해당 컴포넌트에 해당하는 클래스를 찾는다. 찾은 클래스의 소스 코드에서 인텐트 객체에서 데이터를 추출하는 코드 부분을 찾아 데이터 타입과 키 값을 수집한다. 인텐트에서 데이터를 추출하는 코드는 상속이나 유틸리티 클래스를 통해서 처리될 수 있으므로, 각 컴포넌트 클래스의 부모 클래스와 호출되는 모든 클래스에 대해서도 동일한 방법으로 데이터 타입과 키 값을 수집한다.

Table 3. Methods for Extracting Data from an Intent.

Method	Data Type	ADB Option
getStringExtra	String	--es
getIntExtra	Integer	--ei
getBooleanExtra	Boolean	--ez
getByteExtra	Byte	--eb
getShortExtra	Short integer	--es
getLongExtra	Long integer	--el
getFloatExtra	Floating point	--ef
getDoubleExtra	Double floating-point	--ed
getCharExtra	Character	--ec

콘텐츠 프로바이더에 대한 파라미터 정보 수집

콘텐츠 프로바이더에서 URI를 통해 데이터를 전달하는 과정에서 URI와 그를 통해 접근할 수 있는 데이터의 타입과 키 값은 필수적인 파라미터 정보이다. URI, 즉 Uniform Resource Identifier는 특정 데이터 자원을 식별하는 데 사용되며, 콘텐츠 프로바이더의 URI는 authority와 path로 구성된다. Authority는 콘텐츠 프로바이더를 유일하게 식별하는 부분이며, path는 콘텐츠 프로바이더 내에서 특정 데이터를 찾기 위해 사용되는 경로이다.

URI의 정확한 식별을 위해 먼저 AndroidManifest.xml 파일에서 〈provider〉 요소를 조사하여 콘텐츠 프로바이더의 authorities 속성을 확인한다. 이 속성은 콘텐츠 프로바이더의 고유 식별자 역할을 하며, URI의 기반을 형성한다. Fig. 7.에서는 provider의 authority는 com.example.testapp.provider이다.

이와 관련된 path 정보를 파악하기 위해서는 디컴파일된 소스 코드를 분석한다. 소스 코드에서 ContentProvider 클래스 내에서 정의된 URI 패턴을 검토하는 것이 중요하다. 일반적으로 UriMatcher 객체를 사용하여 URI 패턴을 관리하는 방법과 하드코딩 방식으로 URI를 정의하는 두 가지 방법이 존재한다.

UriMatcher는 URI 요청을 라우팅하기 위해 사용되며, 다음과 같은 방식으로 URI를 추가한다. Fig. 10.는 그 UriMatcher를 사용하여 URI 패턴을 구성한 예시 코드를 보여준다.

또한 Fig. 11.은 URI가 소스 코드 내에서 직접 정의되어 처리되는 코드를 보여준다.

LockPickFuzzer는 소스 코드에서 정의된 URI 패턴을 이용해 authority와 관련된 path 정보를 수집한다.

추가적으로 테이블을 대상으로 하는 URI의 경우, path가 테이블 이름을 나타내므로 소스 코드에서 해당 테이블 이름으로 검색하여 CREATE TABLE {path}와 같은 SQL 구문을 찾을 수 있다. 이 구문을 통해 해당 테이블의 컬럼 정보를 수집한다.

```
public static final UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
static {
    uriMatcher.addURI("com.example.testapp.provider", "restable", 1);
}
```

Fig. 10. URI Patterns Using UriMatcher.

```
public Cursor query(Uri uri, …) {
    if (uri.toString().equals("content://com.example.testapp.provider/restable"))
    { … }
}
```

Fig. 11. Hard-Coded URI.

System-effective 패키지 리스트 추출

시스템 컴포넌트에 영향을 줄 수 있을 것으로 예상되는 패키지 리스트를 식별한다. 디컴파일된 데이터에서 추출된 문자열 중 패키지 명명 규칙에 부합하는 문자열을 후보 패키지 리스트로 분류한다. 패키지 명명 규칙에 대한 정규식 `[a-z][a-z0-9_]*(?:\.[a-z0-9_]+){2,}`를 사용한다.

5.1.3 설치 패키지 생성

앞 장에서 추출한 System-effective 패키지 리스트를 기반으로 adb install 을 통해 주입할 APK를 생성한다.

사전에 준비된 더미 안드로이드 애플리케이션 프로젝트 파일에서 'AndroidManifest.xml' 파일을 로드하고 해당 XML의 구조를 분석하여, 루트 요소에 명시된 기존 패키지 이름을 새로운 패키지 이름으로 수정하고, 파일 내에서 모든 권한, 사용 권한 및 공급자 요소에 대해 원본 패키지 이름이 참조된 부분을 새로운 패키지 이름으로 치환한다. 수정된 'AndroidManifest.xml' 파일과 함께 프로젝트 디렉토리 내에서 APK를 재구성한다. 최종적으로, 생성된 APK 파일은 Zipalign[15]과 APKSinger[16]를 통해 서명 과정을 한다. 이 서명된 APK는 최종적으로 변경하고자 하는 패키지 이름을 가진 상태로 안드로이드 디바이스에서의 설치 및 실행이 가능하다.

5.1.4 디바이스 잠금 화면 설정

검사 대상 디바이스의 잠금 화면을 설정하는 단계이다. adb shell locksettings {set-pin/set-password/set-pattern} {credential} 명령을 통해 사용자가 원하는 잠금 방식과 자격증명(credential)을 설정할 수 있다. LockPickFuzzer는 퍼징 테스트를 시작하기 전에 이러한 설정을 완료하며, 설정된 자격증명은 테스트 결과 분석 과정에서 인증 정보의 추출 여부를 검증하는 데 활용된다.

5.2 Fuzz Execution Loop

퍼즐 실행 루프는 테스트 케이스를 생성하고 디바이스에서 실행한 후, 잠금 화면이 우회되었거나 인증 정보가 추출 가능한지 검사하는 과정을 반복하는 것이다. 문제가 발견되지 않는 한, 이 과정을 지속적으로 반복한다.

우선, Critical ADB 명령어 중 하나를 임의로 선택한다. Preprocessing 과정에서 생성한 초기 입력 데이터 셋을 사용하여 선택된 명령어 템플릿에 맞춘 적절한 테스트 케이스를 생성하고 실행한다. 이 과정에서 테스트 케이스가 정상적으로 동작했는지 판단하기 위해 표준 출력을 검사한다. 표준 출력이 실패 응답이 아닌 경우, 해당 명령어를 실행 명령어 집합에 추가한다. 만약 표준 출력이 실패 응답인 경우, 해당 테스트 케이스에 '비사용' 플래그를 설정하고 실행 명령어 집합에 추가하지 않는다. 만약 명령어 결과에 대한 응답이 1분 이상 없는 경우에도 실패 응답으로 처리하고, 지속적인 퍼징 테스트를 유지하기 위해 디바이스를 재부팅한다.

5.2.1 잠금 화면 보호 우회 검사

테스트 케이스가 정상적으로 수행된 경우, 잠금 화면 보호 메커니즘이 우회되었는지 확인한다. 이 검사는 두 단계로 이루어진다. 먼저 잠금 화면 설정 여부를 확인하고, 설정된 경우 잠금이 해제된 상태인지를 확인한다.

잠금 설정 여부를 확인하기 위해 adb shell dumpsys lock_settings 명령어를 사용한다. 표준 출력에서 CredentialType 값이 NONE으로 설정된 경우, 잠금 화면이 해제된 것으로 간주한다. CredentialType 값이 NONE이 아닌 경우, 이는 PIN, PATTERN, PASSWORD 중 하나로 설정된 것이며 잠금 화면이 활성화된 상태이다. 이 경우, 잠금 화면은 해제된 상태인지를 확인하기 위해 adb shell dumpsys window 명령어를 사용한다. 표준 출력에서 mDreamingLockscreen 값이 false로 설정된 경우, 잠금 화면이 해제된 것으로 간주한다. 위 검사 절차에서 잠금 화면이 해제된 것으로 판정되는 경우, 실행 명령어 집합을 최적화하는 단계로 넘어간다.

5.2.2 잠금 화면 인증 정보 유출 검증

잠금 화면 보호 우회 검사가 실패한 경우, 잠금 화면 해제를 시도하고 인증 정보를 추출할 수 있는지 확인한다. 퍼징 과정에서 잠금 화면 인증 정보를 사전에 알고 있기 때문에, adb shell input 명령어를

사용하여 잠금 해제를 수행한다. PIN과 PASSWORD타입의 경우 adb shell input text {credential} 명령어를 통해 입력할 수 있으며, PATTERN 타입의 경우 adb shell input swipe {x1, y1, x2, y2} 명령어를 사용하여 패턴을 입력할 수 있다.

잠금 해제를 시도한 후, 시스템 로그와 메모리 덤프를 수집하여 인증 정보가 노출되었는지 확인한다. 시스템 로그는 adb logcat -b main -b kernel 명령어를 통해 추출할 수 있으며, 메모리 덤프는 adb shell am dumpheap {process id} 명령어를 통해 추출할 수 있다. 이때, 프로세스 ID는 adb shell ps -ef 명령어를 통해 현재 실행 중인 프로세스 정보를 이용하여 확인한다. 커널 프로세스는 추출 범위에서 제외한다. 수집된 로그 파일과 메모리 덤프 파일에서 사전에 정의된 인증 정보가 노출되었는지 검사한다. 인증 정보가 노출된 것으로 판정되는 경우, 실행 명령어 집합을 최적화하는 단계로 넘어간다.

5.3 Postprocessing

퍼징 테스트를 통해 생성된 실행 명령어 집합에서 실제로 잠금 화면 보안을 무력화하는 데 영향을 주는 명령어들만 추출하는 최적화 과정이 필요하다. 퍼징 테스트는 방대한 양의 명령어 집합을 생성하게 되는데, 이 중 실제로 보안 메커니즘을 무력화하는 데 기여하는 명령어는 일부에 불과하다. 따라서, 이러한 불필요한 명령어를 제거하고 중요한 명령어만을 식별하는 최적화 과정이 필수적이다. 이를 통해 연구자는 효율적으로 보안 취약점을 분석하고, 명확한 대응 방안을 제시할 수 있다.

Fig. 12.는 실행 명령어 집합을 최적화하는 알고리즘이다. 퍼징 테스트의 결과물로 생성된 실행 명령어 집합 중 마지막 명령어를 키 명령어로 설정한다. 이 키 명령어는 잠금 화면 보안 메커니즘을 무력화하는 데 중요한 역할을 하며, 최적화 과정의 모든 단계에서 고정된 상태로 사용된다. 초기 실행 명령어 집합은 키 명령어를 제외한 나머지 명령어들로 구성되며, 이는 최적화 과정의 시작점이 된다. 또한, 사용자는 지정한 최대 명령어 수(maxCommands)를 지정하여 최종적으로 반환할 수 있는 명령어 리스트의 최대 길이를 제한할 수 있다.

최적화 과정의 첫 단계에서는 초기 실행 명령어

Require: A list of commands C, maximum number of commands $maxCommands$
Ensure: Optimized subset of commands that effectively deactivates screen lock security

```
function FindOptimalCommands(C, maxCommands)
    keyCommand ← C[end]
    initialCommands ← C[1 : end − 1]
    return RecursiveOptimization(initialCommands, keyCommand, 2, maxCommands)
end function

function RecursiveOptimization(commands, keyCommand, n, maxCommands)
    if length(commands) = 0 then
        return ∅
    end if
    if length(commands) = 1 then
        if TestBypass(commands + [keyCommand]) then
            return commands + [keyCommand]
        else
            return ∅
        end if
    end if
    subsets ← SplitIntoSubsets(commands, n)
    k ← n − 1
    for combination in GenerateCombinations(subsets, k) do
        combinedSubset ← CombineSubsets(combination)
        if TestBypass(combinedSubset + [keyCommand]) then
            if length(combinedSubset) + 1 ≤ maxCommands then
                return combinedSubset + [keyCommand]
            end if
            return RecursiveOptimization(combinedSubset, keyCommand, 2, maxCommands)
        end if
    end for
    return RecursiveOptimization(commands, keyCommand, n + 1, maxCommands)
end function
```

Fig. 12. Commands Optimization Algorithm.

집합을 n개의 서브 리스트로 분할한다. 이때, n은 2로 시작하며, 각 서브 리스트는 가능한 한 균등하게 나누어진다. 분할된 서브 리스트 중 k개의 서브 리스트를 조합하여 새로운 명령어 집합을 생성하고, 이 명령어 집합에 키 명령어를 추가하여 테스트를 수행한다. 테스트는 testBypass 함수를 통해 이루어지며, 주어진 명령어 집합을 실행하여 잠금 화면 보안이 무력화되는지 확인한다. 테스트가 성공하면 수행된 명령어 집합을 반환한다. 이때, 테스트가 성공한 명령어 집합의 길이가 maxCommands 이하인 경우, 해당 리스트를 반환하고 최적화 과정을 종료한다.

만약 테스트가 성공하지 못하면, n 값을 증가시켜 n+1개의 서브 리스트로 분할한 후 다시 조합하여 테스트를 반복한다. 이 과정은 서브 리스트가 더 이상 분할되지 않을 때까지 계속된다. 모든 테스트가 실패하면 최적화 과정을 종료하고, 최적화된 명령어 리스트를 반환한다. 이 재귀적 접근 방식을 통해, 최적화 과정은 점진적으로 명령어 집합을 축소하여, 실제로 잠금 화면 보안 메커니즘을 무력화하는 데 기여하는 명령어를 식별할 수 있다.

VI. 평 가

LockPickFuzzer는 Android 14를 탑재한 Samsung 갤럭시 S23과 픽셀 8에서 테스트되었다. 실험은 Userdebug 이미지에서 모든 adb 명령어를 사용할 수 있는 환경에서 퍼징을 수행한 후, 생성된 명령어 집합을 User 이미지에서 재현하여 검증하는 방식으로 진행되었다. 본 실험을 통해 LockPickFuzzer는 갤럭시 S23 디바이스에서 잠금 화면 보안 메커니즘을 무력화시킬 수 있는 두 가지 시나리오에 대해 각각 ADB 명령어 조합을 발견했다. 이 중 User 이미지에서 재현된 취약점은 삼성전자 보안팀에 보고되었으며, 삼성 취약점 식별 번호인 SVE-2023-1344를 할당받았다.

6.1 잠금 화면 인증 정보 추출

Fig. 13.과 같이 시스템 UI 앱의 메모리 덤프를 통해 잠금 화면 인증 정보가 유출될 수 있는 문제점이 확인되었다. 이 문제는 갤럭시 S23에서 특정 애플리케이션이 설치된 상황에서 발생했으며, Fig. 14. 는 문제를 재현하는 ADB 명령어 집합을 보여준다. User 이미지에선 시스템 어플리케이션의 힙 메모리 덤프를 추출할 수 없으나, 이 명령어 집합을 사용하여 시스템 어플리케이션 중 하나인 시스템 UI의 힙 메모리 덤프를 추출할 수 있었으며 잠금 화면의 인증 정보인 핀 코드를 얻을 수 있었다. 수동 분석 과정에서 확인한 바에 따르면, 잠금 화면에서 홈 화면으로의 전환 후 약 20분 동안 핀 코드는 힙 메모리에 남아 있었다. 이는 ActivityManager서비스에서 특정 패키지가 설치된 경우, 시스템 어플리케이션에 대한 힙 메모리를 추출할 수 있도록 구현된 것이 원인이다. 이 취약점은 삼성전자 공식 취약점 식별 번호

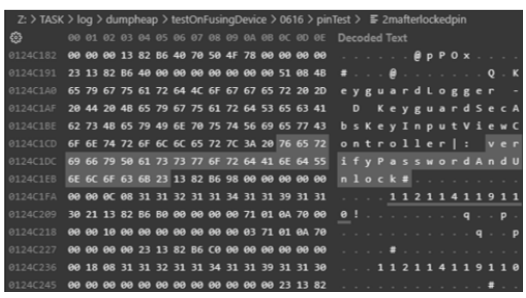

Fig. 13. Heapdump of SystemUI in Device with PIN Lock.

```
[
{"command": "adb install com.salab.act.apk"},
{"command": "adb shell ps -ef"},
{
"command": "adb shell am dumpheap 3774
/data/local/tmp/3774_com.android.systemui.prof"},
{
"command":adb pull
/data/local/tmp/3774_com.android.systemui.prof"}
]
```

Fig. 14. Command Set Inducing Lock Screen Credential Extraction Vulnerability.

(SVE-2023-1344)가 부여 되었다. 현재 ActivityManager서비스에서시스템 UI와 같은 시스템 애플리케이션의 메모리 덤프 추출을 제한하는 보안 패치가 적용되었다.

6.2 인증 없이 잠금 화면 해제

갤럭시 S23에 탑재된 '모드 및 루틴' 시스템 애플리케이션의 기능을 이용하여 인증 과정 없이 잠금 화면을 임의로 해제가 가능한 문제점을 발견하였다. content 명령어와 service 명령어를 사용하며, '모드 및 루틴' 어플리케이션과 TrustAgent 서비스를 조작함으로써 잠금 화면의 잠금 설정을 스와이프로 변경할 수 있다.

Fig. 15.는 이를 재현할 수 있는 명령어 집합이다. 먼저 routineprovider에 content 명령어을 통해 'Driving' 모드를 '모드 및 루틴'에 추가하고, 해당 모드가 활성화되는 경우 잠금 화면을 해제되도록 설정한다. 그리고 service 명령어를 통해 잠금 화면에서 활성화되는 TrustAgents 목록에 '모드 및 루틴'을 등록하고 변경 사항을 적용하면 잠금 타입이 스와이프로 변경되어 잠금 화면이 해제된다.

그러나 이 명령어 집합은 User 이미지에서 재현되지 않으며, 분석 결과 TrustAgent 서비스를 조작할 때 system 권한이 필요한 것으로 확인되었다. 하지만 이 문제점은 시스템 애플리케이션의 TrustAgents 목록에 등록할 수 있는 권한이 남용될 경우 발생할 수 있는 보안 문제를 드러낸다. 제조사는 이러한 분석을 통해 보안 점검 체크 리스트를 강화하고, 잠재적 위험을 사전에 차단할 수 있는 시스템 개선을 고려할 수 있다.

```
[
{"command": "adb shell content insert --uri
content://com.samsung.android.app.routines.routinepro
vider/routine --bind _uuid:s:101 --bind is_running:s:1 --
bind name:s:Driving --bind routine_extra:s:'state&0' --
bind icon:s:9 --bind color:s:-11813903 --bind
tag:s:'mode\\/driving' --bind
attributes:s:'{\\\"type\\\":\\\"lifestyle_mode\\\"}' --bind
is_show_notification:s:1 --bind toggle_time:s:0 --bind
manual_routine_running_time:s:'{\\\"timeout\\\":7200000,
\\\"type\\\":\\\"UNTIL_STOP_MANUALLY\\\"}'"
},
{"command": "adb shell content query --uri
content://com.samsung.android.app.routines.routinepro
vider/routine --where \"name='Driving'\""},
{"command": "adb shell content insert --uri
content://com.samsung.android.app.routines.routinepro
vider/action_instance --bind
_uuid:s:4915100277853143657 --bind
package:s:'com.samsung.android.app.routines' --bind
action_tag:s:trust_lock --bind routine_id:s:9 --bind
is_negative:s:0 --bind timestamp:s:0 --bind
valid_state:s:1 --bind version:s:1"},
{"command": "adb shell content update --uri
content://com.samsung.android.app.routines.routinepro
vider/routine --bind routine_extra:s:'state&1' --where
\"_id='9'\""},{
"command": "adb shell content insert --uri
content://com.samsung.android.app.routines.routinepro
vider/preference --bind key:s:boot_time --bind
value:s:'1799083149509'"},
{"command": "adb shell content insert --uri
content://com.samsung.android.app.routines.routinepro
vider/preference --bind key:s:grant_required --bind
value:s:true"},
{"command": "adb shell service call lock_settings 3 s16
'lockscreen.enabledtrustagents' s16
'com.samsung.android.app.routines/.preloadproviders.s
ystem.actions.trustlock.SepRoutineTrustAgentService'
i32 0"},
{"command": "adb shell service call trust 5 i32 0"},
{"command": "adb shell service call trust 1 i32 1 i32 0"}
]
```


Fig. 15. Command Set Causing Arbitrary Lock Screen Bypass.

6.3 탐색 시간

LockPickFuzzer의 탐색 시간과 최적화 시간을 평가하였다. 퍼즈 테스트 과정에서는 한 개의 테스트 케이스를 실행하는 시간보다 실행 결과를 검사하는 시간이 더 오래 소요된다. 테스트 케이스인 ADB 명령어들은 일반적으로 2초에서 5초 이내로 수행되지만, 잠금 화면 우회 검사는 평균 7초, 잠금 화면 인증 정보 유출 검사는 5분이 소요된다. 특히 잠금 화면 인증 정보 유출 검사에서는 프로세스의 메모리 덤프를 생성하고 추출하는 데 약 7초가 걸리며, 약 40개의 프로세스를 추출할 경우 총 280초로 검사 시간의 90% 이상을 차지한다. 평균 테스트 케이스 수행 시간이 3.5초라면, 시간당 11개의 테스트 케이스밖에 실행할 수 없다. 따라서 시간당 수행 가능한 테스트 케이스 수를 높이기 위해 천 개의 테스트 케이스당 한 번 검사를 수행하였다.

Fig. 16.은 잠금 화면 우회 취약점을 발견하기까지 소요된 일자와 유효한 테스트 케이스 수와 유효하지 않은 테스트 케이스 수의 비율을 나타낸다. 취약점을 발견하기까지 18일이 소요되었으며, 평균적으로 하루에 약 22,000개의 테스트 케이스가 실행되었다. 그러나 모든 테스트 케이스가 검사 디바이스에서 유효하지 않으며, 퍼징 테스트 첫날에는 테스트 케이스의 약 27%가 실패했다. 하지만 테스트를 반복함에 따라 실패하는 테스트 케이스의 빈도는 점차 감소하는 추세를 보였다. 테스트 케이스가 실패하는 이유는 Preprocessing 단계에서 얻은 데이터만으로 검사 디바이스에 유효한 테스트 케이스를 완벽하게 구성할 수 없기 때문이다. 예를 들어, 초기 데이터셋의 컴포넌트에 대해 am 명령어를 통해 실행했을 때 검사 디바이스에서 존재하지 않거나 인식하지 못하는 경우가 있다. service 명령어의 경우 지정된 서비스

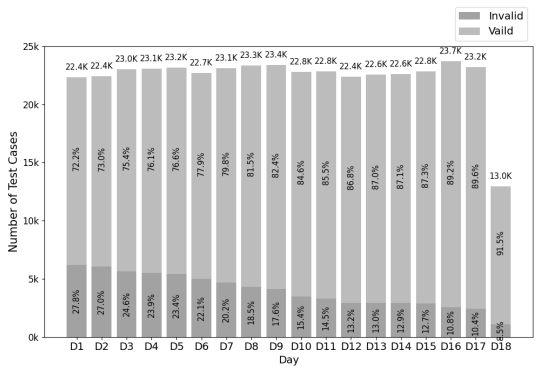

Fig. 16. Daily Ratios of Valid and Invalid Test Cases in the Search for Lock Screen Bypass Vulnerability.

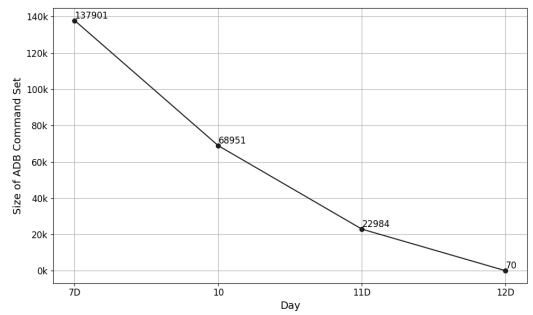

Fig. 17. Duration to Optimize ADB Commands to Below 100 for Lock Screen Bypass Vulnerability.

가 단말에 존재하지 않거나 호출된 서비스의 코드를 인식할 수 없거나 필요한 파라미터가 포함되지 않은 경우가 있다. content 명령어의 경우 URI가 존재하지 않거나 쿼리 형식이 잘못되었거나 잘못된 데이터 처리를 요청하는 경우가 있다. 이러한 실패를 유발한 데이터는 재사용되지 않도록 플래그가 설정되며, 테스트가 반복됨에 따라 실패율은 점차 감소하게 된다.

Fig. 17.은 잠금 화면 우회 취약점을 발생시키는 ADB 명령어 집합을 100개 이하로 최적화하는 데 걸린 기간을 보여준다. 최적화된 명령어 집합을 도출하기 위해 수행해야 할 모든 경우의 수를 고려하면, 원본 집합보다 수행해야 할 테스트 케이스가 수배 더 많아질 수 있다. 따라서 최적화 시간이 탐색 시간보다 수배 더 소요된다. 이에 최적화 과정에서는 동일한 길이의 서브 명령어 집합들에 대해 여러 안드로이드 디바이스를 병렬 연결하여 테스트 횟수를 증가시켜 수행 시간을 최적화하였다. 이를 통해 탐색 시간보다 더 적은 시간으로 최적화를 수행할 수 있었다.

VII. 논 의

이 장에서는 LockPickFuzzer의 구현 및 실험 결과를 바탕으로 퍼징 기법의 효율성 저하 요인과 위협 모델에 대한 주요 한계점에 대해 고찰한다.

7.1 퍼징 효율성 저하

초기 데이터셋 구축 과정에서 난독화된 정보로 인해 수집하지 못하는 경우가 발생하면, 관련된 테스트 케이스를 수행할 수 없다. 또한, 잘못된 데이터가 수집될 경우 테스트 케이스의 실패율이 높아져 퍼징의 성능이 저하된다. 디바이스 제조사는 자신들이 탑재하는 시스템 애플리케이션 코드에 직접 접근할 수 있으므로, 코드 레벨에서 정보를 수집하면 더 많은 유효한 테스트 케이스를 만들 수 있다.

무작위 퍼징 기법은 취약점을 유발할 수 있는 명령어 조합을 찾는 데 적합하지만, 긴 탐색 시간으로 인해 효율이 저하될 수 있다. 이를 해결하기 위해 잠금 화면 문제와 관련이 없는 테스트 케이스를 사전에 분류해낼 수 있다면, 불필요한 테스트 케이스 수를 줄여 퍼징 효율을 높일 수 있다.

최적화 과정에서는 원본 명령어 집합에서 단순히 나누고 조합하는 방식으로 부분 집합을 구성하여 최적화가 진행되므로, 시간이 많이 소요된다. 이 과정을 개선하기 위해 부분 집합을 구성하는 과정에서 명령어 간의 상관 관계를 분석하여 좀 더 유효한 부분 집합을 구성하는 방안을 고려하면 최적화 시간을 단축할 수 있다.

7.2 위협 모델의 현실적 한계

위협 모델에서는 공격자가 비정상적인 경로를 통해 USB 디버깅 모드를 활성화할 수 있다고 가정하지만, 최신 안드로이드 디바이스에서는 USB 디버깅 모드 활성화에 여러 보안 장치가 도입되어 있어 공격자가 이를 우회하는 것이 어렵다.

그럼에도 불구하고, ADB는 디버깅 및 개발 용도로 필수적이기 때문에 원천적으로 봉쇄할 수 없으며, 비정상적으로 USB 디버깅 모드를 활성화한 사례들이 다시 발생하지 않는다는 보장을 할 수 없다. 이러한 이유로 이 연구는 USB 디버깅 모드가 활성화된 환경이나 개발 중인 기기에서 발생할 수 있는 보안 취약점을 사전에 발견하고 대응할 수 있는 가능성을 열어준다.

VIII. 결 론

본 연구는 ADB를 활용하여 안드로이드 장치의 잠금 화면 보안 메커니즘에 존재할 수 있는 취약점을 탐색하고, 그 우회 가능성을 평가하는 데 초점을 맞추었다. 연구의 핵심은 복잡하게 얽힌 ADB 명령어의 조합을 자동화된 방식으로 테스트하여 보안 취약점을 식별하는 과정에 있다. 이는 인간의 손을 거치지 않고 대규모로 수행할 수 있는 효율적인 검증 작업을 요구했으며, 이에 따라 LockPickFuzzer라는 자동화 퍼징 도구를 개발하였다.

LockPickFuzzer를 활용한 실험을 통해, 안드로이드 14를 운영 체제로 사용하는 Samsung 갤럭시 S23에서 잠금 화면 보안 메커니즘을 무력화시킬 수 있는 두 가지 ADB 명령어 조합을 발견하였다. 이 명령어 조합들은 잠금 화면의 인증 정보를 추출하거나, 인증 없이 잠금 화면을 임의로 해제할 수 있는 능력을 제공하며, 이러한 결과는 안드로이드의 보안 기능이 지속적으로 강화되는 현 시점에서도 ADB가 시스템의 보안 메커니즘에 심각한 영향을 미칠 수 있는 가능성을 드러내고 있다.

LockPickFuzzer는 안드로이드 디바이스의 보

안 검증 작업에 있어 필수적인 도구로 자리 잡을 잠재력을 보여주며, 시스템의 보안 메커니즘을 강화하는 데 중요한 기여할 것으로 기대한다. 향후 연구에서는 이 도구를 다양한 제조사의 디바이스에 적용하고, 추가적인 잠금 화면 보안 메커니즘의 우회 가능성을 탐구하여 보다 견고하고 신뢰할 수 있는 안드로이드 보안 환경을 구축하는 데 기여하고자 한다.

References

[1] Chia-Chi Lin, Ting-Fang Yen, Hsin-Kuo Lin, and Yu-Chih Chen, "Screenmilker: How to milk your android screen for secrets," NDSS, Feb. 2014.

[2] Sungjae Hwang, Youngsik Kim, and DaeHun Nyang, "Bittersweet adb: Attacks and defenses," Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, pp. 97-108, Apr. 2015.

[3] Manar Mohamed, Babins Shrestha, and Nitesh Saxena, "Smashed: Sniffing and manipulating android sensor data for offensive purposes," IEEE Transactions on Information Forensics and Security, vol. 12, no. 4, pp. 901-913, Apr. 2016.

[4] Li Yang, Lijun Wang, and Dongdong Zhang, "Malicious behavior analysis of Android GUI based on ADB," IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), pp. 596-599, Jul. 2017.

[5] Chuck Easttom and Willie Sanders, "On the efficacy of using android debugging bridge for android device forensics," IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), pp. 395-399, Oct. 2019.

[6] Pingfan Kong, Yu Hao, and Xuanzhe Liu, "Automated testing of android apps: A systematic literature review," IEEE Transactions on Reliability, vol. 68, no. 1, pp. 45-66, Mar. 2018.

[7] Developer Android, "Android Debug Bridge" https://developer.android.com/tools/adb, Apr. 2024.

[8] The Guardian, "Android lockscreen can be bypassed by overloading with massive password" https://www.theguardian.com/technology/2015/sep/16/android-lockscreen-password, Apr. 2024.

[9] Geumhwan Cho, Hyunsoo Kim, Seungjoo Kim, and Kwangjo Kim, "On the security and usability implications of providing multiple authentication choices on smartphones: The more, the better?," ACM Transactions on Privacy and Security (TOPS), vol. 23, no. 4, pp. 1-32, Oct. 2020.

[10] Sebastian Potocky and Jozef Stulrajter, "The human interface device (HID) attack on android lock screen non-biometric protections and its computational complexity," Science & Military Journal, vol. 17, no. 1, pp. 29-36, Jan. 2022.

[11] TechCrunch, "A simple Android lock screen bypass bug" https://techcrunch.com/2022/11/14/android-lock-screen-bypass-google-pixel, Apr. 2024.

[12] CybersecurityNews, "Bypassed Android Lock Screen using Driving mode Assistant" https://cybersecuritynews.com/researchers-bypassed-android-lock-screen, Apr. 2024.

[13] Fatih Ertam, Omer Faruk Yakut, and Turker Tuncer, "Pattern lock screen detection method based on lightweight deep feature extraction," Neural Computing and Applications,

vol. 35, no. 2, pp. 1549-1567, Sep. 2023.
[14] GitHub, “jadx - Dex to Java decompiler” https://github.com/skylot/jadx, Jan. 2024.
[15] Android Developers, “Zipalign” https://developer.android.com/tools/zipalign, Jan. 2024.
[16] Android Developers, “APKSigner” https://developer.android.com/tools/apksigner, Jan. 2024.
[17] Nerdschalk, “How to Bypass Pattern Lock, Fingerprint, Password Lockscreen Security on Android via ADB” https://nerdschalk.com/bypass-pattern-lock-fingerprint-password-lockscreen-security-android-via-adb, Apr. 2024.
[18] Panagiotis Andriotis, Theo Tryfonas, and Zhaoqian Yu, "Poster: breaking the android pattern lock screen with neural networks and smudge attacks," Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'14), pp. 261-262, Jul. 2014.
[19] V. Venkateswara Rao and A. S. N. Chakravarthy, "Analysis and bypassing of pattern lock in android smartphone," IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1-3, Dec. 2016.
[20] Hui Lu, Zhiqiang Lin, Zhiyun Qian, and Haixin Duan, "Salaxy: Enabling usb debugging mode automatically to control android devices," IEEE Access, vol. 7, pp. 178321-178330, Dec. 2019.
[21] Dave Jing Tian, Kevin Butler, and Patrick Traynor, "ATtention spanned: Comprehensive vulnerability analysis of AT commands within the Android ecosystem," 27th USENIX Security Symposium, pp. 23-39, Aug. 2018.
[22] Mingzhe Xu, Weiqing Sun, and Mansoor Alam, "Security enhancement of secure USB debugging in Android system," 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), pp. 146-152, Jan. 2015.
[23] GitHub, “Android-PIN-Bruteforce” https://github.com/urbanadventurer/Android-PIN-Bruteforcer, Apr. 2024.

〈저자소개〉

고 대 훈 (Daehoon Ko) 정회원
2013년 2월: 인하대학교 컴퓨토공학과 졸업
2023년 3월~현재: 성균관대학교 DMC공학과 석사 과정
〈관심분야〉 보안공학, 모바일 보안

김 형 식 (Hyoungshick Kim) 종신회원
1999년 2월: 성균관대학교 정보공학부 학사
2001년 2월: KAIST 컴퓨터 과학과 석사
2012년 2월: University of Cambridge 컴퓨터공학과 박사
2013년 3월~현재: 성균관대학교 소프트웨어학과 부교수
〈관심분야〉 보안공학, 모바일 보안