

Targeting and Exploiting Android Devices via Hostile Networks



By Yakov Shafranovich
yakov@nightwatchcybersecurity.com
December 2nd, 2016



Disclaimer:

Don't do anything without talking to your mother and your lawyer first!



About Me

- Developer most of my professional career
- Security bug bounty hunter on the side
- Recently switched to application security full time **but I'm here personally, not on behalf of my employer**
- Was involved in some early anti-spam work:
 - Co-chaired IRTF's Anti Spam Research Group
 - Involved in IETF / pre-standards work for SPF and DKIM
 - Created the MARF protocol for exchanging spam reports (RFC 5965)
- Also did some non-security standards work:
 - RFCs 4180 (CSV files) and 6922 (SQL MIME type)
 - Participated in W3C's CSV for the Web WG



Part 1

Finding Vulnerable Devices



“One Ring to rule them all, One Ring to find them...”



Goals

- Find what firmware/security patches are on the device
- Different than fingerprinting – we don't care about identifying the user or phone model – we want to know how secure the user's device is
- Do it passively without user interaction
- Rinse and repeat on a massive scale



How?

User Agents!



How?

User Agents:

- Sent with every HTTP request
- Visible when SSL is not used
- Contains information about the user's device and software

From RFC 7231, section 5.5.3:

The "User-Agent" header field contains information about the user agent originating the request, which is often used by servers to help identify the scope of reported interoperability problems, to work around or tailor responses to avoid particular user agent limitations, and for analytics regarding browser or operating system use. **A user agent SHOULD send a User-Agent field in each request unless specifically configured not to do so.**



User Agents SHOULD NOT include device info, but they do anyway!

From RFC 7231, section 5.3.3:

A user agent SHOULD NOT generate a User-Agent field containing needlessly fine-grained detail and SHOULD limit the addition of subproducts by third parties. **Overly long and detailed User-Agent field values increase request latency and the risk of a user being identified against their wishes ("fingerprinting").**

From Mozilla:

Adding a device identifier to the Firefox OS User Agent (UA) string is STRONGLY DISCOURAGED by Mozilla.

...

Mozilla strives to provide greater privacy for users. Therefore, we have been working to reduce the level of "fingerprintability" of different browser configurations—that is to say, how uniquely identifiable a particular user's browser is to sites through detection methods of which the user is unaware. (i.e. server-side methods) **Adding e.g. hardware information to the UA reduces privacy by increasing fingerprintability.**



User Agents - Examples

Firefox on Ubuntu: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0

IE11 on Windows: Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko

Safari on MacOS: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11) AppleWebKit/601.1.39 (KHTML, like Gecko) Version/9.0 Safari/601.1.39

OkHttp (Java HTTP library): okhttp/2.3.0

Nexus 10 / Android 4.2: Mozilla/5.0 (Linux; U; Android 4.2; en-us; Nexus 10 Build/JVP15I) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Safari/534.30

Apple News on iOS 10: Mozilla/5.0 (iPhone; CPU iPhone OS 10_0 like Mac OS X) AppleWebKit/602.1.43 (KHTML, like Gecko) AppleNews/607 Version/2.0

Microsoft Lumia 950, Windows Phone 10: Mozilla/5.0 (Windows Phone 10.0; Android 4.2.1; Microsoft; Lumia 950) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2486.0 Mobile Safari/537.36 Edge/13.10586

IOS applications using NSURL: app/1 CFNetwork/758.0.2 Darwin/15.0.0



But?

Can user agents tell us about the security status of a given mobile device???

YES



According to the NSA (in 2010)

*“User-Agents can ... **identify CNE (Computer Network Exploitation) opportunities**”*

*“Mobile user agents **also usually give you the phone model** (Read: IMEI correlation opportunities)”*

*From the NSA User Agent Briefing (circa 2010), published by the Intercept:
<https://theintercept.com/document/2015/07/01/user-agents/>*



Anatomy of an Android User Agent



From Google's Webmaster Blog:

<https://webmasters.googleblog.com/2011/03/mo-better-to-also-detect-mobile-user.html>



Many Types of Android User Agents

- **Chrome web browser on Android**

- Mozilla/5.0 (Linux; Android 4.0.4; Galaxy Nexus Build/**IMM76B**)
AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.133 Mobile
Safari/535.19

- **Chrome web view - before Android 5.0**

- Mozilla/5.0 (Linux; Android 4.1.1; ALCATEL ONE TOUCH 5020X Build/**JRO03C**)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.59 Mobile
Safari/537.36

- **Chrome web view - after Android 5.0 - note the “wv” string**

- Mozilla/5.0 (Linux; Android 5.1.1; Nexus 5 Build/**LMY48B; wv**)
AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/43.0.2357.65
Mobile Safari/537.36

- **Android’s HTTP Client - used by default by most apps**

- Dalvik/2. 1. 0 (Linux; U; Android 5.0.2; **D5503** Build/14.5.A.0.270)

- **Other libraries:**

- Apache HttpClient on Android
 - Apache-HttpClient/UNAVAILABLE (java 1.4)
- OkHttp on Android
 - okhttp/2.3.0
- Etc.



About Android Builds

Android devices have a build-in MODEL and build ID, identifying the phone model and **Android build**. They are defined in `android.os.Build.MODEL` and **`android.os.Build.ID`** properties. (There is usually also a timestamp in the ROM image itself)

The ID is defined in the Android Compatibility Definition document as follows:

ID	An identifier chosen by the device implementer to refer to a specific release, in human-readable format. This field can be the same as <code>android.os.Build.VERSION.INCREMENTAL</code> , but SHOULD be a value sufficiently meaningful for end users to distinguish between software builds. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “ <code>^[a-zA-Z0-9._-]+\$</code> ”.
----	---



The Problem with Builds

- On desktop OSes like Windows and Linux, patches (and security patches) are granular and not “all in one”
- BUT on mobile devices OS patches are monolithic - packaging all patches into a single new build
- Every build on Android maps to a specific set of patches
- Builds can often be mapped to a specific phone model and carrier, but not always (and we don't care anyway)
- Recent Android builds can be mapped to a specific Android patch level - every monthly Android patch release results in one or more new builds



The Problem with Builds

Given a particular Android build, you can figure out what security patches it has, and what security patches it DOES NOT have, and the date it was made

...
BUT - someone has to go out and collect the information about builds to make it happen



Should be Trivial for Google Devices

Starting with Cupcake, individual builds are identified with a short build code, e.g. FRF85B.

The first letter is the code name of the release family, e.g. F is Froyo.

The second letter is a branch code that allows Google to identify the exact code branch that the build was made from, and R is by convention the primary release branch.

The next letter and two digits are a date code. The letter counts quarters, with A being Q1 2009. Therefore, F is Q2 2010. The two digits count days within the quarter, so F85 is June 24 2010.

Finally, the last letter identifies individual versions related to the same date code, sequentially starting with A; A is actually implicit and usually omitted for brevity.

The date code is not guaranteed to be the exact date at which a build was made, and it is common that minor variations added to an existing build re-use the same date code as that existing build.

From Google docs:

<https://source.android.com/source/build-numbers.html#platform-code-names-versions-api-levels-and-ndk-releases>



Example Android User Agents Mappings

IMM76B - Galaxy Nexus; v4.0.4; built on March 21st, 2012

LMY48B - Nexus 5; v5.1.1 Released May 21st, 2015

D5503 - Sony Xperia Z1; Android v5.1.1; released Nov 5th, 2014

MPA44G - Android 6.0 third developer preview for Nexus phones; released August 17th, 2015

MOB30M - BLU Life; v6.0.1; build on June 12th, 2016

N930AUCS1APH1 - Android v6.0.1; Galaxy Note 7 for AT&T; August 20th, 2016

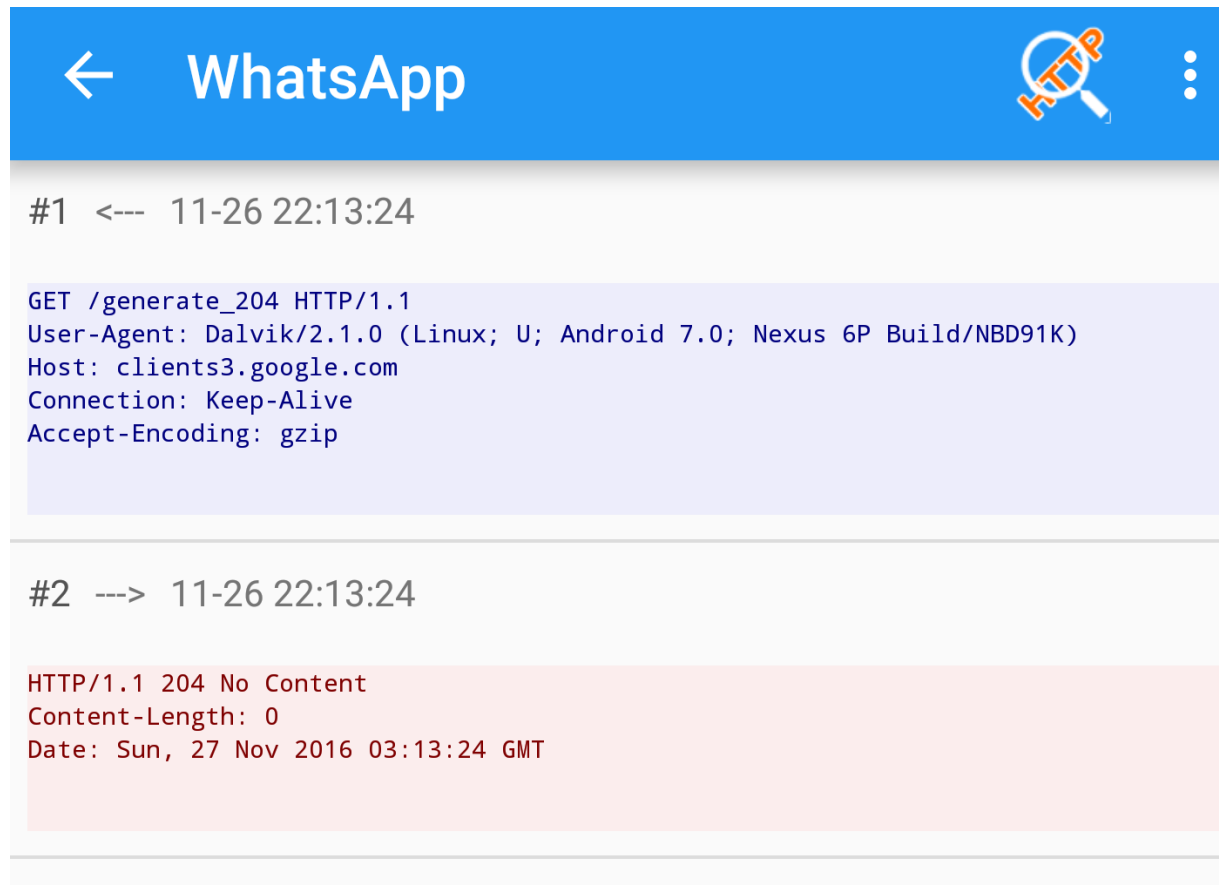


Things that make this worse


- Android apps and the OS will often start sending data right after connecting to a new network
- No user interaction needed
- Data will continue to be sent at regular intervals without user interaction
- Most apps use the default Android network library
- Some of that data will be unencrypted (unlike iOS which will soon mandate SSL)
- A lot of Android apps include third party libraries that do their own random network calls, often not encrypted



Examples - WhatsApp



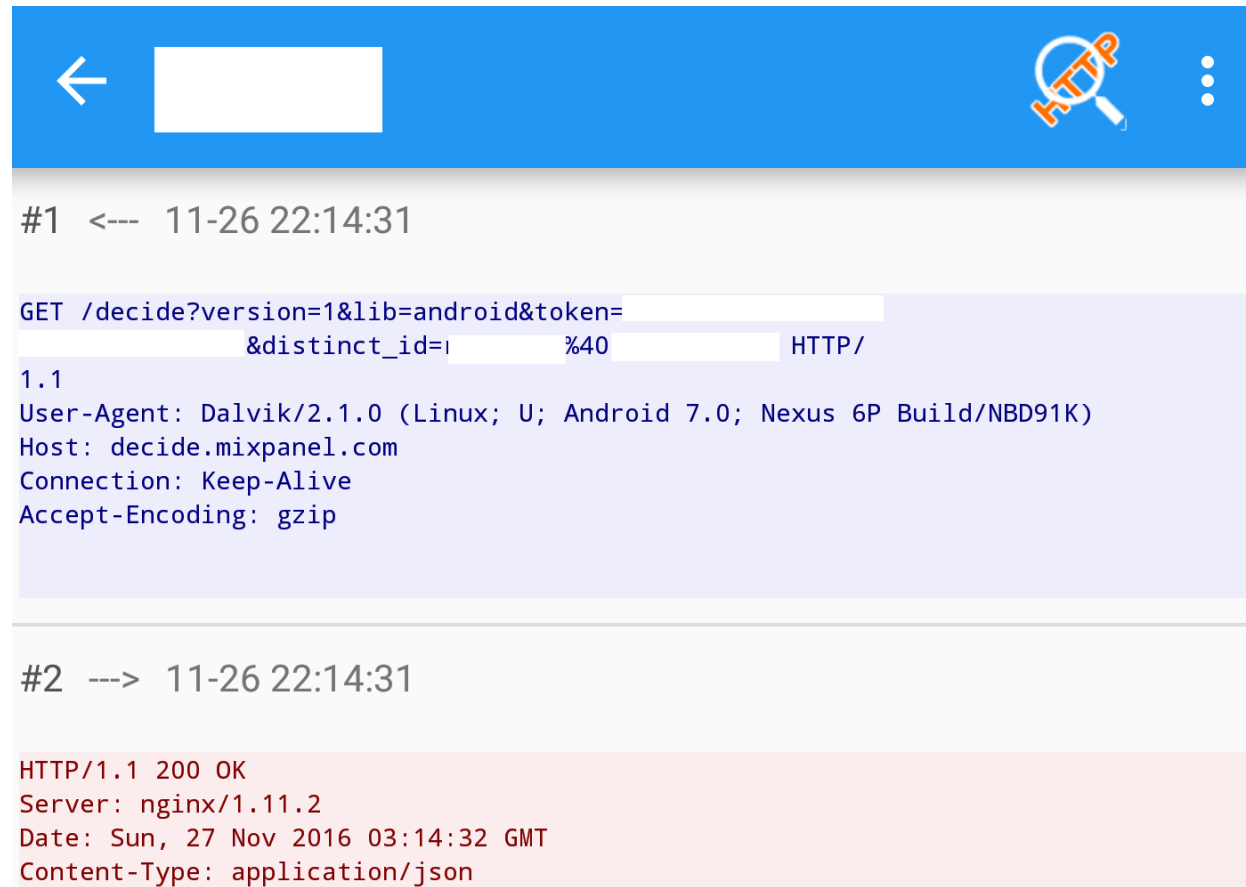
The screenshot shows the WhatsApp interface with a blue header bar containing a back arrow, the text "WhatsApp", a magnifying glass icon with "HTTP" written inside, and a three-dot menu icon. Below the header, the network log is displayed. The first entry, labeled "#1", is a request from the client to the server, timestamped "11-26 22:13:24". The request details are: "GET /generate_204 HTTP/1.1", "User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.0; Nexus 6P Build/NBD91K)", "Host: clients3.google.com", "Connection: Keep-Alive", and "Accept-Encoding: gzip". The second entry, labeled "#2", is a response from the server to the client, also timestamped "11-26 22:13:24". The response details are: "HTTP/1.1 204 No Content", "Content-Length: 0", and "Date: Sun, 27 Nov 2016 03:13:24 GMT".

```
← WhatsApp  ⋮  
  
#1 <--- 11-26 22:13:24  
  
GET /generate_204 HTTP/1.1  
User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.0; Nexus 6P Build/NBD91K)  
Host: clients3.google.com  
Connection: Keep-Alive  
Accept-Encoding: gzip  
  
#2 ---> 11-26 22:13:24  
  
HTTP/1.1 204 No Content  
Content-Length: 0  
Date: Sun, 27 Nov 2016 03:13:24 GMT
```

WhatsApp uses E2E encryption with the Signal protocol, yet still connects without encryption when a new network is detected!



Examples - IOT Control App



```
#1 <--- 11-26 22:14:31

GET /decide?version=1&lib=android&token=
&distinct_id= %40 HTTP/
1.1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.0; Nexus 6P Build/NBD91K)
Host: decide.mixpanel.com
Connection: Keep-Alive
Accept-Encoding: gzip

#2 ---> 11-26 22:14:31

HTTP/1.1 200 OK
Server: nginx/1.11.2
Date: Sun, 27 Nov 2016 03:14:32 GMT
Content-Type: application/json
```

Third-party service being called



Examples - Adobe AIR Runtime

Adobe security bulletin: ASPB16-31 My bug: CVE-2016-6936

Adobe AIR is a developer product which allows the same application code to be compiled and run across multiple desktop and mobile platforms. While monitoring network traffic during testing of several Android applications we observed network traffic over HTTP without the use of SSL going to several Adobe servers including the following:

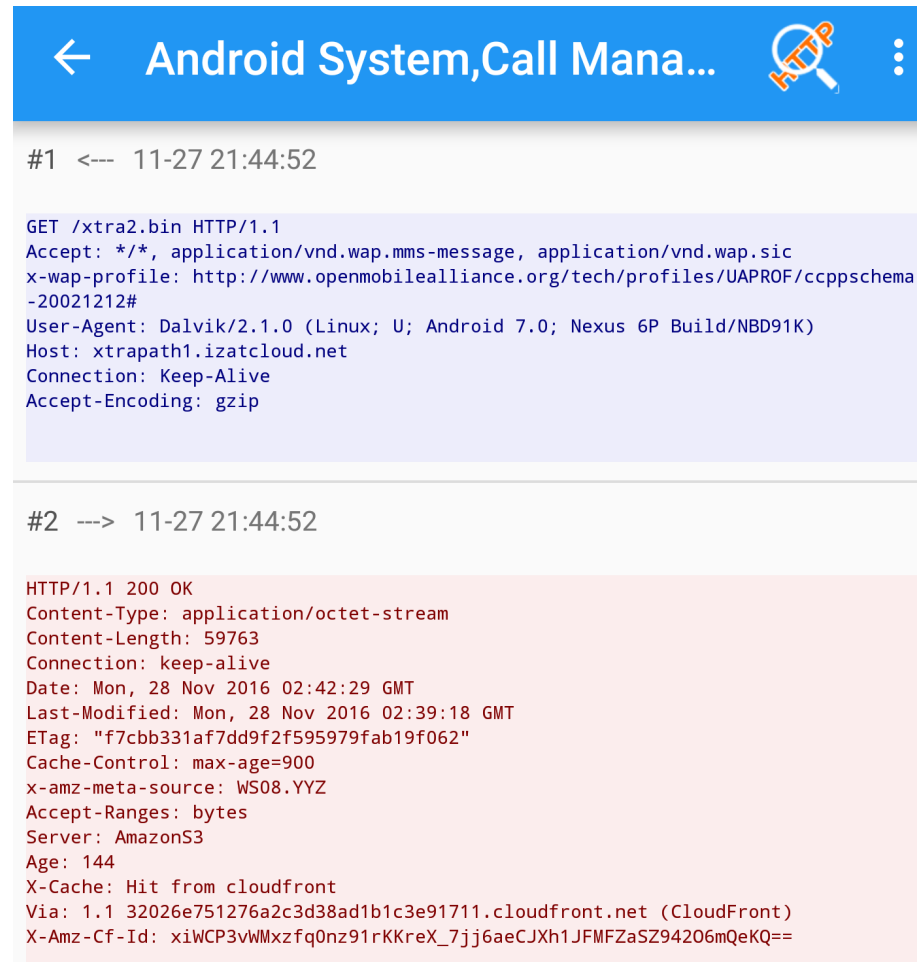
- `airdownload2.adobe.com`
- `mobiledl.adobe.com`

Because encryption is not used, this would allow a network-level attacker to observe the traffic and compromise the privacy of the applications' users.

This affects applications compiled with the Adobe AIR SDK versions 22.0.0.153 and earlier.



Examples - Android OS



The screenshot shows an Android system log with a blue header bar. The header contains a back arrow, the text "Android System, Call Mana...", a magnifying glass icon with "HTTP" written inside, and a three-dot menu icon. The log entries are as follows:

```
#1 <-- 11-27 21:44:52

GET /xtra2.bin HTTP/1.1
Accept: */*, application/vnd.wap.mms-message, application/vnd.wap.sic
x-wap-profile: http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-20021212#
User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.0; Nexus 6P Build/NBD91K)
Host: xtrapath1.izatcloud.net
Connection: Keep-Alive
Accept-Encoding: gzip

#2 ---> 11-27 21:44:52

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 59763
Connection: keep-alive
Date: Mon, 28 Nov 2016 02:42:29 GMT
Last-Modified: Mon, 28 Nov 2016 02:39:18 GMT
ETag: "f7cbb331af7dd9f2f595979fab19f062"
Cache-Control: max-age=900
x-amz-meta-source: WS08.YYZ
Accept-Ranges: bytes
Server: AmazonS3
Age: 144
X-Cache: Hit from cloudfront
Via: 1.1 32026e751276a2c3d38ad1b1c3e91711.cloudfront.net (CloudFront)
X-Amz-Cf-Id: xiWCP3vWMxzFq0nz91rKKreX_7jj6aeCJXh1JFMFZaSZ94206mQeKQ==
```

Retrieving some files without encryption - more in part 2!



Some additional notes

- Android apps can choose to override the user agent – for example, FireFox for Android does not reveal build numbers
- It is also possible to do this for iOS by correlating the CFNetwork ID numbers against builds
- I filed a bug with Google but it was rejected
- Chrome for Android, in Desktop mode DOES NOT leak build numbers
- **More Research is Needed**



Part 2

Exploiting Vulnerable Devices



... One Ring to bring them all and in the darkness bind them"



If we know what firmware is on the device,
how do we exploit it?



Autotarget exploits via Chrome: StageFright/CVE-2015-3864



[Home](#) [Exploits](#) [Shellcode](#) [Papers](#) [Google Hacking Database](#) [Submit](#) [Search](#)

Android 5.0 <= 5.1.1 - 'Stagefright' .MP4 tx3g Integer Overflow (Metasploit)

EDB-ID: 40436	Author: Metasploit	Published: 2016-09-27
CVE: CVE-2015-3864	Type: Remote	Platform: Android
Aliases: N/A	Advisory/Source: N/A	Tags: Metasploit Framework
E-DB Verified:	Exploit: Download / View Raw	Vulnerable App: N/A

[« Previous Exploit](#)

[Next Exploit »](#)

Inspect traffic, find matching devices to the exploit, and inject in HTTP traffic from Chrome browser on Android

```
##> ['Nexus 7 (Wi-Fi) (razor) with Android 5.0 (LRX21P)',
'Automatic', {} ],
#
# Each target includes information about the device, firmware
# how exactly to about exploiting it.
#
# Primarily, these targets are used to map a browser's User-
# exploit specifics for that device / build.
#
[
'Nexus 7 (Wi-Fi) (razor) with Android 5.0 (LRX21P)',
{
'Model' => 'Nexus 7',
'Build' => 'LRX21P',
'Release' => '5.0',
'Rop' => 'lrx',
'SprayAddress' => 0xb1508000
}
],
[
'Nexus 7 (Wi-Fi) (razor) with Android 5.0.1 (LRX22C)',
{
'Model' => 'Nexus 7',
'Build' => 'LRX22C',
'Release' => '5.0.1',
'Rop' => 'lrx'
}
],
[
'Nexus 7 (Wi-Fi) (razor) with Android 5.0.2 (LRX22G)',
{
'Model' => 'Nexus 7',
'Build' => 'LRX22G',
'Release' => '5.0.2',
'Rop' => 'lrx'
}
],
]
```



Or find an OS process that can be crashed or injected because it's trusting the network

(if you are like me and can't write binary exploits)

My bugs:

- [CVE-2016-6723](#) - crashing Android with large PAC files - patched in Nov 2016
- [CVE-2016-5348](#) - crashing Android with GPS XTRA files - patched in Oct 2016
- [CVE-2016-5341](#) - UNPATCHED

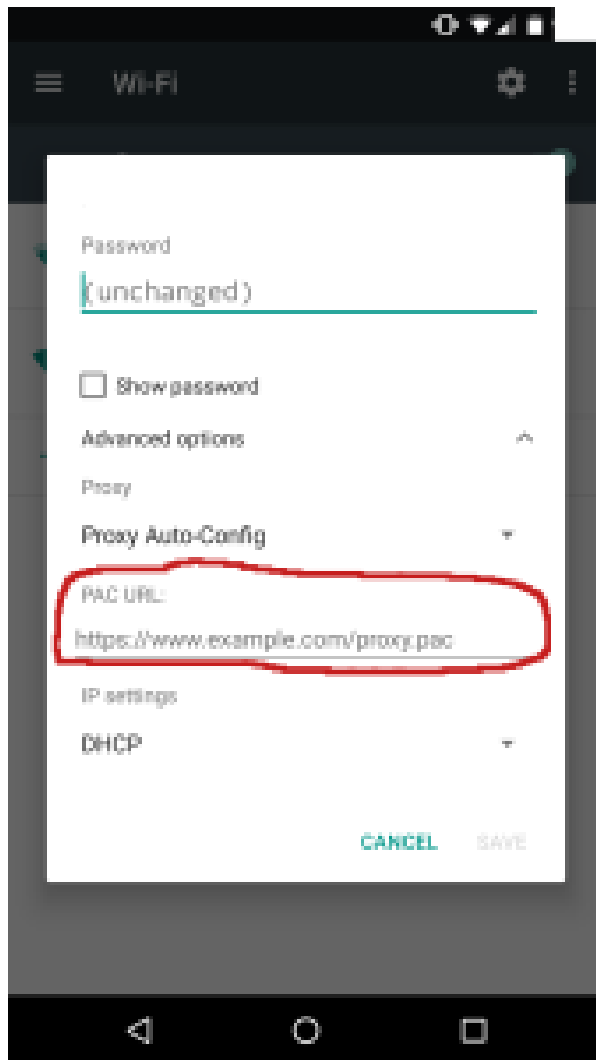


- **CVE-2016-6723 - Crashing Android with large PAC files**

- When connecting an Android device to a network using a proxy, sometimes you have to use a PAC file
- PAC files aren't always served over SSL since the local network is supposed to be secure
- What happens if you are on the same network and intercept the PAC call? CRASH!!! - device reboots
- Because Android doesn't support WPAD there is no way to fool a device into configuring itself with a PAC file - must be done manually
- **Scary thought for further research - PAC files are written in Javascript**



- **CVE-2016-6723 - Crashing Android with large PAC files**



Sample PAC file:

```
function FindProxyForURL(url, host) {  
    if (isResolvable(host))  
        return "DIRECT";  
    else  
        return "PROXY proxy.mydomain.com:8080";  
    }  
}
```



- **CVE-2016-6723 - Crashing Android with large PAC files**

- Caused by Java code in the Android platform which never checks the length of received packets
- Why is this running on the OS level???

PacManager.java, lines 120-127):

```
private static String get(Uri pacUri) throws IOException {
    URL url = new URL(pacUri.toString());
    URLConnection urlConnection =
url.openConnection(java.net.Proxy.NO_PROXY);
    return new
String(Streams.readFully(urlConnection.getInputStream()));
}
```



• CVE-2016-5348 - Crashing Android with large GPS XTRA files

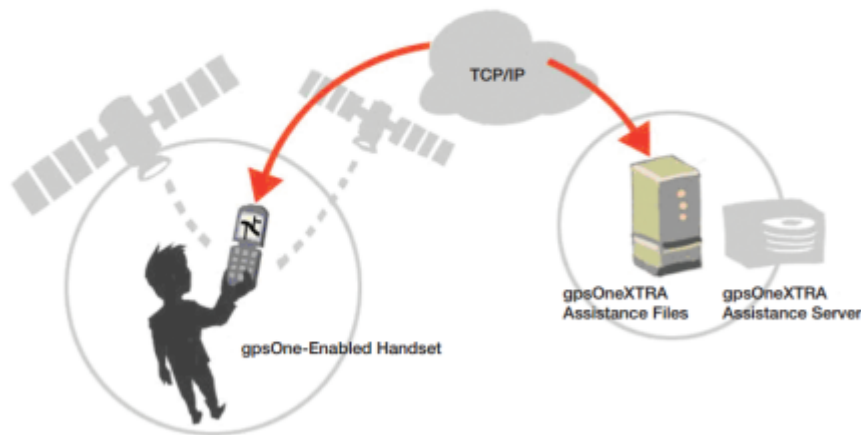
- When an Android phone connects to a new network, it will automatically retrieve a special file to help with GPS satellite resolution - GPS XTRA - xtra2.bin - NO SSL
- Contains almanac information about the locations of various GPS satellites for the next 7 days
- Other devices using Qualcomm chips may be affected
- The file is fetched by the Java layer, handed off to JNI C++ code, and then injected into the Qualcomm modem
- **If the request is intercepted and a large enough file is returned the device will crash**
- Further research on attacking the modem is needed



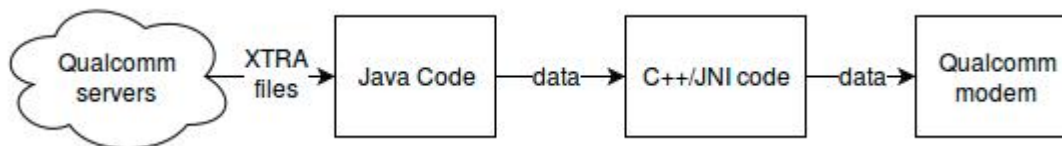
• CVE-2016-5348 - Crashing Android with large GPS XTRA files

gpsOneXTRA™ Assistance


A typical scenario for gpsOneXTRA Assistance usage is as follows:



1. Mobile connects to gpsOneXTRA Assistance server via Internet (thru PC connection) or wireless network
2. Mobile downloads small data file which contains forecasted satellite data valid for up to 7 days; data file automatically refreshed, if stale
3. Mobile uses downloaded data file to make fast(er) GPS fixes including tough environments



• CVE-2016-5348 - Crashing Android with large GPS XTRA files

```
← Android System,Call Mana...  ⋮
```

```
#1 <--- 11-27 21:44:52
```

```
GET /xtra2.bin HTTP/1.1
Accept: */*, application/vnd.wap.mms-message, application/vnd.wap.sic
x-wap-profile: http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschem
-20021212#
User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.0; Nexus 6P Build/NBD91K)
Host: xtrapath1.izatcloud.net
Connection: Keep-Alive
Accept-Encoding: gzip
```

```
#2 ---> 11-27 21:44:52
```

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 59763
Connection: keep-alive
Date: Mon, 28 Nov 2016 02:42:29 GMT
Last-Modified: Mon, 28 Nov 2016 02:39:18 GMT
ETag: "f7cbb331af7dd9f2f595979fab19f062"
Cache-Control: max-age=900
x-amz-meta-source: WS08.YYZ
Accept-Ranges: bytes
Server: AmazonS3
Age: 144
X-Cache: Hit from cloudfront
Via: 1.1 32026e751276a2c3d38ad1b1c3e91711.cloudfront.net (CloudFront)
X-Amz-Cf-Id: xiWCP3vWMxzfq0nz91rKKreX_7jj6aeCJXh1JFMFZaSZ94206mQeKQ==
```



• CVE-2016-5348 - Crashing Android with large GPS XTRA files

- Caused by Java code in the Android platform which never checks the length of received packets
- Also has C++ code that doesn't check either
- Once again, why is this running in the OS level???

GpsXtraDownloader.java, lines 120-127):

```
connection.connect();  
int statusCode = connection.getResponseCode();  
...  
Streams.readFully(connection.getInputStream());
```

com_android_server_location_GnssLocationProvider.cpp, lines 856-858):

```
jbyte* bytes = (jbyte *)env->GetPrimitiveArrayCritical(data, 0);  
sGpsXtraInterface->inject_xtra_data((char *)bytes, length);  
env->ReleasePrimitiveArrayCritical(data, bytes, JNI_ABORT);
```



CVE-2016-5341

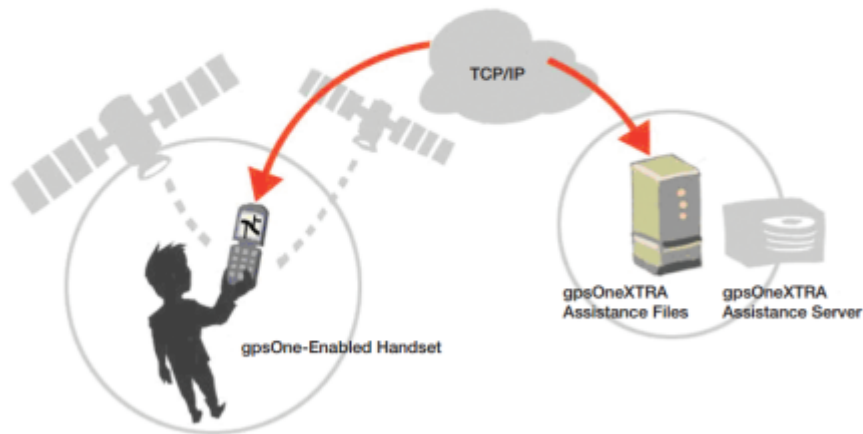
- Dates back to 2007
- Has been known to Qualcomm since 2014
- The root cause of CVE-2016-5348
- This is a Qualcomm, not Android issue
- Currently UNPATCHED
- Android patches to be released on Monday (12/5)
- Other Qualcomm partners have been notified in September



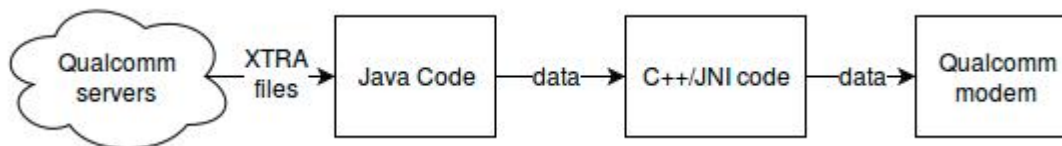
CVE-2016-5341

gpsOneXTRA™ Assistance

A typical scenario for gpsOneXTRA Assistance usage is as follows:



1. Mobile connects to gpsOneXTRA Assistance server via Internet (thru PC connection) or wireless network
2. Mobile downloads small data file which contains forecasted satellite data valid for up to 7 days; data file automatically refreshed, if stale
3. Mobile uses downloaded data file to make fast(er) GPS fixes including tough environments



CVE-2016-5341

- Qualcomm provides three types of data files:
 - Xtra.bin – has a simple checksum
 - Xtra2.bin – has a simple checksum
 - Xtra3.bin – digitally signed
- Includes almanac data for GPS, Glonass, Galileo, etc.
- Currently delivered without SSL, but SSL is now available
- Potentially used in any device with a Qualcomm GPS chip
- Apple and Microsoft devices not affected since they use a secure channel to deliver the files (unverified)

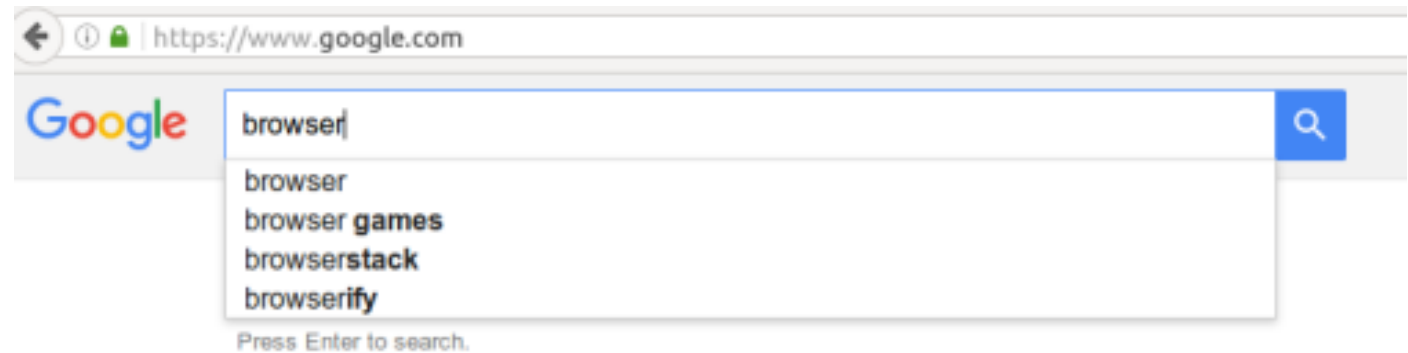
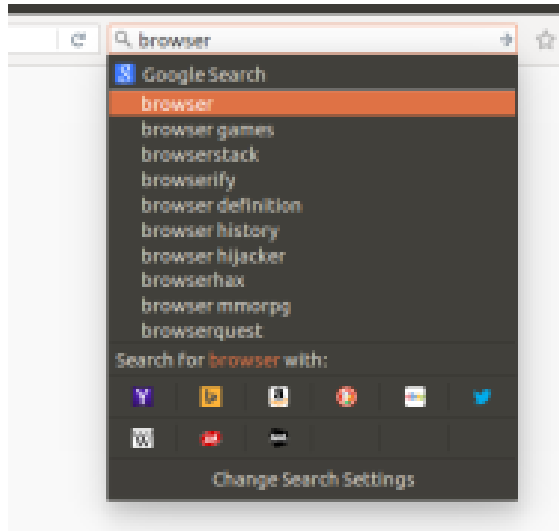


More fun - crashing search engines via search suggestions

- After finding the PAC and GPS bugs, I looked for other places in the Android source where **Streams.readFully** is used
- Found it in search suggestions in the Android v4 browser
- If the search engine doesn't support SSL, this will be done via HTTP
- Can be intercepted, malformed and make the browser crash; can also be used to inject fake results
- Also affects desktop browsers and will not be fixed; still a problem because 1 of the top 5 search engines in the US doesn't use SSL (it's AOL)
- Other countries also have search engines without SSL



More fun - crashing search engines via search suggestions



More fun - messing with unencrypted apps

- Some other ways to mess with Android apps that don't use SSL (must use arp spoofing or take over the router)
- If an app is doing a version check, MITM and return nothing:
 - Instead of **{“updateVersionAvailable”: true}**
 - Return “{}”/“” - blocks the user from knowing about updates
- If an app is retrieving static assets like images - inject your own!
- Return a malformed or very large packet and crash the app
- WhatsApp is doing a connection check - make it think its offline
- **The network should be considered hostile!**



What Can We Do?

- App developers:
 - Change the user agent or use okhttp, etc.
 - Use SSL, always and correctly
 - Protect your users' privacy
 - Do not trust the network, ever
 - Be cognizant of third party libraries/services and hold them to the same standards as your own app
- OS Developers:
 - Force app developers to always use SSL
 - Stop leaking data about users and devices
 - Do not trust the network, ever



Everything covered here is also published on my blog:
www.nightwatchcybersecurity.com



Questions? Comments?

Email: research@nightwatchcybersecurity.com

