Joel Reardon

CVEs from CNN (extended abstract)

Introduction

In 2019 a method to find side and covert channels in mobile apps was presented along with some vulnerabilities it discovered [6]. Here we present two new vulnerabilities found using the same method and one of the earlier side channels being used again. In this abstract we explain how the vulnerabilities work and give further findings from our analysis of the libraries responsible for the exploits.

The method itself is as follows. First, apps are run in a dynamic analysis testbed where their network transmissions are collected; this is done at scale, in our case, tens of thousands. Data types that are protected by the permission system, such as location, connected router SSID, IMEI, etc., are sought in the network traffic. Findings of transmissions of such protected data are flagged for further analysis when the app does not have the corresponding permission. For example, an app sending the router's SSID without a location permission, or any user-space app sending the device's MAC address.

CNN hacks SSID

The app "CNN Breaking US & World News" is the flagship app for the Cable News Network, which is an American news-based television channel. This app included a third-party library from Vizbee, which claims to have patented "mobile-to-TV deep-linking technology [that] transforms existing mobile marketing channels into direct response paths" [9]. In a network transmission that it sent to `metrics.clasptws.tv`, it included the router's SSID under the key `WIFI_SSID`.

This transmission is noteworthy because the app did not hold a location permission. A location permission is required to collect information about the router because such information functions as a surrogate for location [1, 2, 5]. In the observed transmission, there were also key-value pairs for `GEO_LAT` and `GEO_LONG`, both of which had the value `UNKNOWN`, further indicating that location access was denied.

We traced the variable that set the `WIFI_SSID` value in the transmission, and observed that the Android Java bytecode still used the original, human-meaningful variable names instead of obfuscating them. Thus we saw that developers named the variable that stored the SSID `hackedSsid` and it was used as a specific workaround for not being permitted to use the standard API call. The side channel itself was a callback function `onCapabilitiesChanged`, which could be registered to be received through ConnectivityManager's NetworkCallback feature. This callback happened to included the SSID as an extra data field, which Vizbee caches and transmits later. This vulnerability is classified as CVE-2020-0454 [3].

Devtodev libcing the MAC

Devtodev claims to be "a comprehensive solution that analyzes your apps and games and gives you valuable insights". Beyond analyzing the apps, it also transmits the MAC address of the mobile phone. Unlike router MAC address and SSID—which acts as a form of location data—the phone's MAC address is an unresetable hardware serial number. It functions as an indelible supercookie to facilitate invasive user tracking that cannot be escaped. Android has officially disallowed collecting MAC address since version 6.0 in 2015 [2], though side channels exploiting this are well known [6, 7].

Devtodev provides another means to get this unresetable MAC address. They found that the libc system-call `getifaddrs` returned a linked list of information about the network interfaces, and the device MAC address was stored among this information.

Interestingly, Devtodev does not seem satisfied sending only this unresetable identifier. They further took an action that we have not previously seen being done so blatantly. First, they sent the device's Android Advertising ID (AAID), which is a resetable identifier used for the purposes of advertising. Users are able to reset this value to a new random value; this has the purpose of giving users a new identity unlinked to the old one. Devtodev, however, observes when users attempt privacy, and sends both the previous AAID and the new one in the same network transmission—they even call the previous value `prev` in the HTTP query arguments! This happens after rebooting the device, meaning that they store the old AAID persistently.

Measurelib Politely Asking

Measurelib is a Panamanian-based Internet measurement company about which details are hard to find. We observed that large amounts of data was being sent to `mobile.measurelib.com` by "Dub Music Player". It was nearly 30 KiB of gzip-compressed data, the bulk

Joel Reardon: University of Calgary and AppCensus, Inc.

```
String g() { return "Mea"; }
String h() { return "sure"; }
String i() { return "Move"; }
String j() { return "NzExOTc0NTE="; }
String k() { return "MTMxNTQxMjA="; }

String decode_string(String in) {
   String ret = base64_decode(in);
   String password = g() + h();
   String salt = g() + h() + i() + g() + h();
   int rounds = 10;
   int length = 128;
   byte[] key = PBKDF(v1, v2, rounds, length);
   String iv = base64_decode(j()) + base64_decode(k());
   return AES_CBC_128_decrypt(in, key, iv);
}
```

Fig. 1. Pseudocode representation of Measurement System's string decoding. Some variable names are replaced by our interpretation based on their ultimate use. String concatenation from pieces is faithfully represented from the implementation.

of which consisted of details of every app the user installed, including the list of permissions they required and where on the file system they were located.

Noteworthy for our purposes, however, was that it includes the router's MAC address despite not holding a location permission. Studying this was difficult, however, because string constant from the network traffic such as JSON keys and hostnames, were not seen anywhere in the decompiled code. By collecting a small set of apps that all communicated with the same domain, we found that they shared an obscure third-party library identified in Java as `coelib.c.couluslibrary`. With effort we found a developer's website detailing integration intructions [8], though searches directly for the code itself gives few results.

Examining the strings inside their library revealed the mechanism that was being used to obscure them. Each string was algorithmically decrypted at runtime by constructing an AES key through a password-based key derivation protocol. This derivation is done from scratch—for every string every time it is needed; thankfully, they only use ten rounds for the password-based key derivation. They also made the common mistake of using a fixed IV for AES in CBC mode [4]—though most of the strings they encrypted were smaller than a single AES block. Figure 1 gives a pseudocode representation of their string decryption routine.

After finding this library, we were able to confirm that it contains and execute code to perform a universal plug-and-play discovery on the local network. It issues a `M-SEARCH *` to `239.255.255.250:1900` stating `ssdp:discover`. We saw that our router eagerly provided its MAC address formatted as the 48-bit node id of an OSF UUID in its reply.

Discussion

All three of these findings show how basic assumptions and decisions can have long term security and privacy costs. Device MAC addresses were never meant to become an unresetable supercookie. Bluetooth MAC addresses must now be randomized to avoid being able to track people's location history. Router MAC addresses and SSIDs were never designed as surrogates for location, yet databases of router information and location are curated by third-party libraries running inside many popular apps.

The final example shows how the apps that we run on our mobile devices are running code from arbitrary places, in some cases from companies that are difficult to learn more details. These apps, and all the ads and analytics code therein, are running on the trusted side of the home or office router. The safeguards are off for this network traffic because the primary defense—a firewall—only protects against threats from outside, not from the worst SDK in the worst app that happens to be running on someone's phone that has access to your network.

References

[1] Jagdish Prasad Achara, Mathieu Cunche, Vincent Roca, and Aurélien Francillon. Short paper: Wifileaks: underestimated privacy implications of the ACCESS_WIFI_STATE Android permission. In *ACM conference on Security and privacy in wireless & mobile networks*, 2014.

[2] Android 6.0 changes. https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-hardware-id, 2015.

[3] CVE-2020-0454. Available from MITRE, 2020.

[4] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *ACM Conference on Computer and Communications Security (CCS)*, pages 73–84, 2013.

[5] United States of America. United States of America, Plaintiff v. InMobi Pte Ltd., Defendant; Case No.: 3:16-cv-3474, 2016.

[6] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX Security Symposium*, pages 603–620, 2019.

[7] StackOverflow. Getting mac address in android 6.0. https://stackoverflow.com/questions/33159224/getting-mac-address-in-android-6-0, 2016.

[8] Measurement Systems. Developer incentive program. https://measurementsys.com/program.php, 2021.

[9] Vizbee. https://www.vizbee.tv/.