Attacking Nexus 6 & 6P Custom Bootmodes

Roee Hay & Michael Goberman
IBM Security

***Abstract*—We describe a high severity vulnerability (CVE-2016-8467) in Nexus 6 & 6P (with a lower impact on Nexus 6P) which allows enabling of hidden USB interfaces. This, together with another vulnerability can be combined for conducting several attacks. The first attack allows an adversary with USB access to the device, to practically own the Nexus 6 modem, by accessing its diagnostics interface. The second attack also works on Nexus 6P, and enables access to the modem's AT interface. The third and fourth attacks are against Nexus 6 only and allow for exfiltration of uninitialized kernel data and some network traffic via USB. The rest of the described attacks allow for the adversary to obtain an ADB session on a Nexus 6P device.**

I. Introduction

One of the significant physical attack vectors of mobile devices is their USB / Lightning port. Despite the immediate assumption that the adversary needs to posses the device in order to attack it, such an attack could also be conducted by malicious chargers (e.g. public chargers found in air ports) [19], also-known-as "Juice jacking" [18] or by PC malware waiting for the mobile device to be plugged-in [25]. Therefore, several security features exist in Android in order to block and lower the impact of such exploitation attempts. First, the bootloaders are locked by default. This means that the attacker should be incapable of doing any harm (such as replacing the platform operating system) by interacting with the device's *fastboot* interface. In addition, one cannot unlock the bootloader without enabling the "Allow OEM Unlocking" checkbox under Developer's Settings, which the attacker cannot reach if the device is protected with user credentials. Moreover, even if the attacker somehow managed to circumvent around that checkbox, unlocking the bootloader will trigger a factory reset, which should delete all user data. Factory-Reset Prevention (FRP) [9] will later make the device unusable. Other notable security features are Full Disk Encryption (FDE) [7] introduced in Android 5.0, and the recent File Based Encryption (FBE) [6] which enables Direct Boot [10], introduced in Android 7.0 – both features aim to prevent theft of personal data by physical attackers. Furthermore, in order to also protect against off-box attacks, the encryption keys are hardware-backed, implemented by the Trusted Execution Environment (TEE) [8].

Despite the protection mechanisms, mobile devices can still be compromised by exploiting vulnerabilities [14, 5, 4, 21].

II. Vulnerabilities

In this section we describe the new vulnerabilities that we discovered. It should be noted that all of them were responsibly disclosed to Google. Both vulnerabilities have been patched – see Section IV for more details.

The devices which we tested the vulnerabilities on are: Huawei Nexus 6P `ANGLER-ROW-VN1`, Huawei Nexus 6P `ANGLER-VN2` and two Motorola Nexus 6 `shamu XT1100 32GB P3A0` devices.

Vulnerability 1. *Nexus 6P/6 Custom Boot Modes USB Configs Override (CVE-2016-8467)*

The `androidboot.mode` kernel command line argument (with a default value of 'normal') propagates to the `ro.bootmode` system property which is later read by *UsbDeviceManager*. The latter maps, according to the `config.xml` file under the AOSP path `device/{huawei/angler,moto/shamu}/overlay/frameworks/base/core/res/res/values` (Figure 1 & 2), between the couple *(*`bootmode`*, current USB configuration)* and *(new USB configuration)*. The new USB configuration is then saved under the `sys.usb.config` system property which triggers an init *on property* event according to `device/{huawei/angler,moto/shamu}/init.{angler-/shamu}.usb.rc` (Figure 5). This implies that the attacker, capable of changing `androidboot.mode` can gain extra capabilities if a secure USB configuration can now be overridden to an insecure one. Indeed, as one can see, both on Nexus 6 and 6P the original configurations are overridden with some more capable ones. The added new interfaces for Nexus 6 are as follows: (1) `diag`: provides diagnostics access to the Snapdragon 805 SoC (`APQ8048`). We did not manage to conduct any attack by accessing this interface, although further research may prove otherwise. (2) `diag_mdm`: Provides diagnostics access to the to the modem (`MDM9x25`). Attack 1 describes what we managed to achieve by accessing it. (3) `serial_hsic`. Serial access to the device's modem's AT interface, covered by Attack 2. (4) `serial_tty`: Access to a `NMEA` interface. This interface should provide GPS data, however we do not cover it in this paper. (5) `rmnet_hsic`: Access to the RmNet interface. Not covered in this paper. (6) `usbnet`. a USB interface identified by "Motorola Test Command" – covered by Attack 3 & Attack 4.
As for Nexus 6P, the added new interfaces are: (1) `diag`: provides diagnostics access to the modem. Port

identified as `MSM8994`. Enabling this interface has no security impact, at least on our Nexus 6P test devices, because accessing the diagnostics data required flashing a custom radio image. (2) `serial_smd`: Serial access to the device's modem's AT interface, covered by Attack 2. (3) `adb`: Enables the 'Android Debug Bridge'. This added interface is problematic since it dishonors the 'Enable USB debugging' under the 'Developer Settings' menu, allowing the device to accept ADB connections from previously authorized hosts. Covered by Attacks 5 & 6. (4) `rmnet_ipa`: Access to the `RmNet` interface. Not covered in this paper. (5) `manufacture`: includes all of the above interfaces in addition to a mass-storage device interface, which seems to have no security impact.

The only open-question left is how the adversary changes `androidboot.mode`. It turns out that under the Nexus 6P/6 device's *fastboot* UI (which an unauthenticated physical attacker can boot into), two proprietary menu items exist. These menu items instruct, even on a locked bootloader, to change the `androidboot.mode` argument to either `bp-tools` or `hw/mot-factory`. Interestingly the ability to change the bootmode via the fastboot UI has long been known within the developers community [24], however its security impact seems to have been overlooked.

The situation is more severe, as an attacker with ADB access, such as PC malware or a malicious charger connected to an ADB-enabled device, can change the bootmode **permanently**, by issuing the following commands:

```
adb reboot bootloader
fastboot oem config bootmode bp-tools (N6)
fastboot oem bp-tools-on (N6, option 2)
fastboot oem enable-bp-tools (N6P)
fastboot reboot
```

Similarly, in order to boot with hw/mot-factory, the attacker can issue:

```
adb reboot bootloader
fastboot oem config bootmode factory (N6)
fastboot oem enable-hw-factory (N6P)
fastboot reboot
```

This means that the attacker does not even need to posses the device in order to attack it. Thus, the malware only needs to wait for the victim to enable ADB once, and then any future boot will have the dangerous bootmode enabled. It should be noted that an ADB authorization dialog may pop-up on the device. Another option for malware without ADB access, is to opportunistically wait for the device to be in the *fastboot* mode, and then just issue the relevant command.

Again, the ability to change the bootmode via a fastboot command has been known within the development / engineering community [16], yet security-wise, overlooked.

```
<string-array translatable="false" name="
    config_oemUsbModeOverride">
<item>"hw-factory:mtp:manufacture,adb"</item>
...
<item>"hw-factory:adb:manufacture,adb"</item>
<item>"bp-tools:mtp:diag,serial_smd,rmnet_ipa,
    adb"
</item>
...
<item>"bp-tools:rndis,adb:rndis,serial,adb"</
    item>
</string-array>
```

Figure 1. Nexus 6P USB port settings override

```
<string-array translatable="false"
             name="config_oemUsbModeOverride">
...
<item>"bp-tools:mtp:diag,diag_mdm,serial_hsic,
                    serial_tty,rmnet_hsic,usbnet"
</item>
<item>"bp-tools:adb:diag,diag_mdm,serial_hsic,
                    serial_tty,rmnet_hsic,usbnet,
                    adb"
</item>
<item>"mot-factory:rndis,adb:usbnet,adb"</item>
<...
<item>"mot-factory:adb:usbnet,adb"</item>
</string-array>
```

Figure 2. Nexus 6 USB port settings override

```
on property:sys.usb.config=diag,serial_smd,
                            rmnet_ipa,adb
stop adbd
write /sys/class/android_usb/android0/enable 0
write ...
write /sys/class/android_usb/android0/enable 1
start adbd
setprop sys.usb.state ${sys.usb.config}
```

Figure 3. `init.angler.usb.rc` of Nexus 6P

Vulnerability 2. *Nexus 6 usbnet Kernel Uninitialized Memory Leak Over USB (CVE-2016-6678)*

Motorola's f_usbnet kernel driver leaks 4-5 bytes of uninitialized kernel data for each frame it sends over USB, allowing the adversary to exfiltrate information out of the device.

The `usb_ether_xmit` function (Figure 4) receives the socket buffer (`skb`) and queues it on the USB endpoint. The function adds another 4-5 bytes to the socket buffer `len` field, in order to reserve space for the soon to be transmitted frame's CRC. Since the function does not compute and set the CRC on the reserved space, the field is sent uninitialized (and may contain data of previous allocations) over the USB wire. Figure 5 depicts a successful leak.

```
static int usb_ether_xmit(struct sk_buff *skb,
    struct net_device *dev) {
    struct usbnet_context *context = netdev_priv(
        dev);
    struct usb_request *req;
    unsigned long flags;
    unsigned len;
    int rc;
    req = usb_get_xmit_request(STOP_QUEUE, dev);
    ....
    /* Add 4 bytes CRC */
    skb->len += 4;

    /* ensure that we end with a short packet */
    len = skb->len;
    if (!(len & 63)|| !(len & 511))
        len++;
    req->context = skb;
    req->buf = skb->data;
    req->length = len;

    rc = usb_ep_queue(context->bulk_in, req,
        GFP_KERNEL);
    ....
    return 0;
}
```

Figure 4. f_usbnet's usb_ether_xmit function

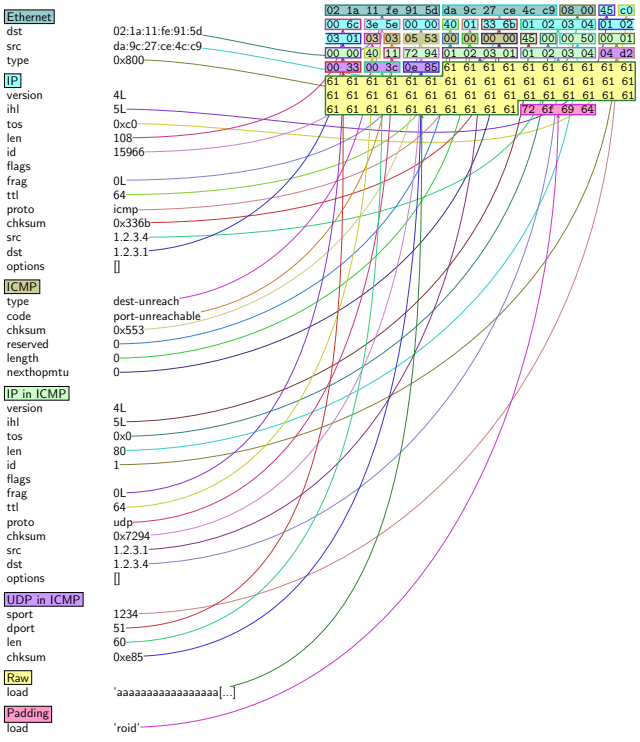

Figure 5. f_usbnet's Leak (the 'Padding' field)

III. Attacks

We describe a few attacks that can stem out of the vulnerabilities listed above. The following table summarizes the attack requirements and exploited vulnerabilities, for each of the described attacks.

Attack	Nexus 6	Nexus 6P	Vulnerability
1	$\chi \vee \varphi$	-	1
2	$\chi \vee \varphi$	$\chi \vee \varphi$	1
3	$\chi \vee \varphi$	-	1,2
4	$\chi \vee \varphi$	-	1
5	-	$\phi \wedge \varphi$	1
6	-	$M \wedge f$	1

Legend: ϕ - Physical access to the victim's ADB-authorized locked PC. φ- Physical access to the Android device. χ - Malware-infected PC / malicious charger with (at least) one time ADB / fastboot access to a device. M - Malware-infected ADB-authorized PC. f - device under fastboot mode.

Attack 1. *Unauthenticated Access to Nexus 6's Modem Diagnostics interface via USB*

Setting and Prerequisites: This attack exploits Vuln 1 and works on Nexus 6 only, thus it requires an attacker (PC malware / charger / physical attacker) who is able to reboot into one of the special bootmodes as explained above.

Attack Flow: (1) The attacker reboots the Nexus 6 device into one of the special modes in order to enable the `diag_mdm` interface. (2) The attacker can now access the USB interface. As for the physical adversary, It should be noted that FDE does not protect against this attack because accessing the diagnostics interface can be done prior to the victim's authentication, thus the attacker can cause some havoc before leaving the device.

It should also be noted that our owned model (`XT1100`) is the international one. The other model (`XT1103`) might prevent diagnostics access to the modem, and/or have a different Service Provider Code (SPC) & diag password.

Impact: Accessing the diagnostics interface allows the adversary to practically own the modem. We successfully managed to (1) Intercept phone calls. In our test environment, these were UMTS RX/TX vocoder frames with AMR_12.2 encoded audio [3]. We then assembled them into an AMR File [1], the result is displayed under Figure 6 as a waveform. (2) Sniff data packets. (Figure 7) (3) Find the exact GPS coordinates with detailed satellite information. (4) Get call information. (5) Initiate phone calls. (6) Access / Change NV items. (7) Access / Change the EFS.

Attack 2. *Unauthenticated Access to the AT interface of the device's modem via USB*

Setting and Prerequisites: Similar to Attack 1 but works on Nexus 6P as well.

Attack Flow: Similar to Attack 1.

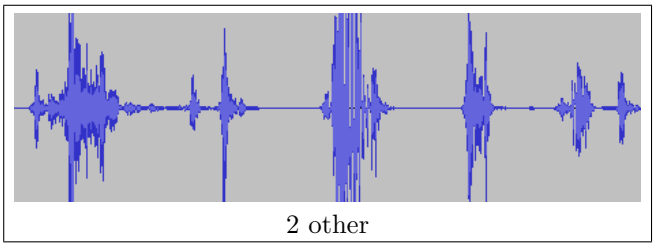

Figure 6. Intercepted 'IBM' (with Hebrew accent) waveform (UMTS RX)

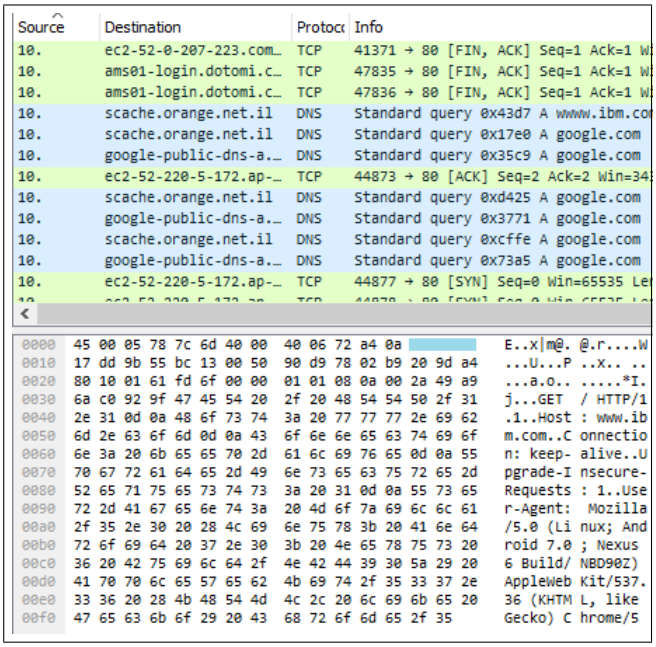

Figure 7. LTE data sniff

Impact: (1) The attacker has 'AT' access to the modem which allows him to steal sensitive information such seeing incoming call numbers. In addition, the attacker can retrieve the Physical Cell ID (PCI) and signal level. The attacker can also transparently read and send SMS messages on behalf of the victim, the attacker can make the device accept or place phone calls, which practically allows him to eavesdrop nearby conversations. Interestingly, under Nexus 6 with Android 6.0, if the device is under the Secure-startup authentication screen (FDE), a placed phone call (after enabling the modem functionality, using `AT^CFUN=1`) will have no UI indication, so the user will be unaware that the other side is listening. (In Android 7.0, the user sees a placed / incoming call, although an airplane mode is indicated.) Moreover, the attacker can permanently change various radio settings, such as disabling the circuit- or packet-switched services (with `AT^SYSCONFIG`), making the device unable to place/receive phone calls or data. Unfortunately, these changes survive Android Factory Resets. In addition, the attacker can downgrade the network connection to older protocols. Figure 8 depicts SMS sniffing via the AT interface on Nexus 6P, which may allow the attacker to

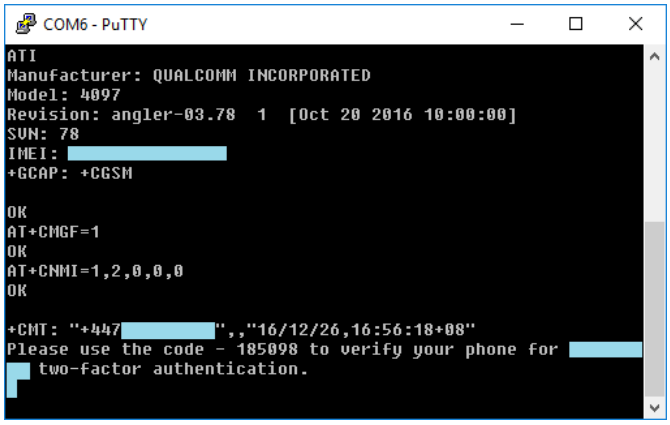

Figure 8. SMS sniffing via `AT` on Nexus 6P

AT	Description
+CLCC	Show current call
+VZWRSRP/Q (N6 only)	Get Physical Cell ID, RSRP and RSRQ
+CMGS	Send SMS
+CNMI=1,2,0,0,0	Sniff SMS
+CFUN=6 (N6 Only)	Reboot the device
^SYSCONFIG=13,0,2,4	Downgrade to GSM
^SYSCONFIG=2,0,2,0	Circuit Switching Only
^SYSCONFIG=2,0,2,1	Packet Switching Only

Figure 9. Various AT commands with security impact

bypass two-factor authentication. See Figure 9 for some AT commands with security impact. (2) The enabled interfaces unnecessarily increase the attack surface of the victim's device (Nexus 6 has 372 AT commands returned by `AT$QCCLAC`, while Nexus 6P has 316 commands), allowing for exploitation of other vulnerabilities, if such exist.

Attack 3. *Leaking Uninitialized Kernel Data via USB*

Setting and Prerequisites: This attack is against Nexus 6 only and targets Motorola's usbnet USB gadget [20].

Attack Flow: (1) The attacker reboots the phone into one of the special bootmodes, exploiting Vuln 1 (2) The attacker can now access the usbnet interface, which allows him to configure (i.e. change the IP settings) via USB the 'usb0' network interface on the victim's device (Figure 10 & 11). This allows him to capture network traffic flowing from/to that adapter via USB. (3) The attacker induces the device to send packets over the USB wire. For example, this can be achieved by sending UDP packets to closed UDP ports as it will cause the other end to transmit ICMP port unreachable replies. Due to Vuln 2 the reply packets contain the 4-5 leaked bytes (Figure 5).

Impact: The leaked kernel data may contain sensitive information and/or aid in conducting further exploitation.

```
ip = socket.inet_aton(IP_ADDR)
ipwords = struct.unpack("HH",ip)
sn = socket.inet_aton(SUBNET_MASK)
snwords = struct.unpack("HH",sn)
host = socket.inet_aton(HOST)
hostwords = struct.unpack("HH",host)

dev = usb.core.find()

dev.ctrl_transfer(0x40, 0x5, ipwords[1],ipwords[0])
dev.ctrl_transfer(0x40, 0x6, snwords[1],snwords[0])
dev.ctrl_transfer(0x40, 0x7, hostwords[1],hostwords[0])
```

Figure 10. Nexus 6 `usb0` configuration via USB

```
shell@shamu:/ $ ifconfig usb0
usb0      Link encap:UNSPEC
inet addr:1.2.3.4  Bcast:1.2.3.255
Mask:255.0.0.0
inet6 addr: fe80::dcb9:c8ff:fee0:e04b/64 Scope: Link
UP BROADCAST RUNNING MULTICAST  MTU:1500
Metric:1
RX packets:10 errors:96 dropped:2 overruns:0 frame:0
TX packets:394 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:176 TX bytes:50601
```

Figure 11. USB-configured Nexus 6's `usb0` adapter

Attack 4. *Limited Leakage of packets via USB*

Setting and Prerequisites: As of Attack 3.

Attack Flow: Similarly to Attack 3, the adversary interacts and configures, via USB, the usb0 network adapter, however this time with arbitrary network settings in order to exfiltrate/inject network traffic from/to the device.

Impact: (1) Our short analysis proved that we can capture broadcast and multicast packets (such as *Google-cast* discovery messages). We haven't managed to cause unicast packets with TCP/UDP ports below 1024 to be routed to the USB device except of those generated by *root*. (2) We have managed to capture unicast connections to any IP, targeting ports 1024 and above, by using a permission-less (except for the automatically-granted INTERNET permission) malicious app that listens on ports above 1024. Then, the attacker can configure the usb0 adapter with the IP of his target (e.g. www.ibm.com's IP). This will cause any traffic targeting that IP to bounce back to loopback, which the attacker listens on. This attack may allow for Universal XSS on sites without HSTS [15], or on never visited / aged HSTS sites not on the HSTS Preload List [2].

Attack 5. *Obtaining an ADB session from a Nexus 6P Device by a Physical Attacker*

Background: As part of the ADB handshake [22], *adbd*, running on the mobile device, generates a 20-byte random token (`adb_auth_generate_token` under `adb_auth_client.cpp`) which is then *RSA*-signed by the ADB host (`adb_auth_sign` under `adb_auth_host.cpp`). The former delivers the signed token back to adbd, which validates the signature (`adb_auth_verify` under `adb_auth_client.cpp`). One problem, as with Poison-Tap [17], is that the ADB host signs a given token, even on locked hosts. Another fundamental problem with the ADB protocol which enables attacks is the fact that only the handshake is secure. Thus, a Man-in-the-Middle (MiTM) attacker can place some special hardware between the victim's device and the host in order to hijack / monitor the ADB session.

Setting and Prerequisites: Our attack assumes a physical access to the victim's PC and Android device. The PC can be locked. The attack requires the victim's PC to be adbd-authenticated, however it does not require ADB to be running on the device prior to it. A feasible example for such a setting with the aforementioned prerequisites is an inside-attacker, targeting a fellow developer, that left his locked device unattended. The developer had permanently ADB-authorized his PC, but later disabled ADB on the device.

Attack Flow: (1) By exploiting Vuln 1, an unauthenticated attacker enables adbd on the device. (2) The attacker connects the device to through a USB proxy. (3) The victim's PC (the ADB HOST) blindly signs the ADB token generated by the device, even if it's locked. (3) The attacker now has an authenticated adb session with the victim's device.

Impact: The attacker now has a shell (running as the capable 'shell' user that is a member of multiple groups) on the device. He can thus install malicious apps, copy files from/to the device, generate a bug report and more. Note that Direct Boot [10] does not protect against malware installation.

Attack 6. *Obtaining an ADB session from a Nexus 6P Device by PC Malware*

Setting and Prerequisites: The attack requires the victim's PC to be adbd-authenticated, however it does not require ADB to be running on the device prior to it. A feasible example for such a setting is a developer that had permanently ADB-authorized his PC, but later disabled ADB on the device.

Attack Flow: (1) The PC malware waits for the victim to place the device in the *fastboot* mode, then it exploits Vuln 1 using the aforementioned fastboot command. (2) The attacker now has an authenticated adb session with the victim's device.

Impact: Similar to Attack 5.

IV. Coordinated Disclosure

All of the issues were responsibly disclosed to Google prior to the publication of this paper.

Google rated Vuln 1 (`CVE-2016-8467`) with **High** severity and mitigated it by forbidding a locked bootloader to boot with the dangerous boot modes. The first non-vulnerable bootloader version of Nexus 6 is 71.22 (released in the November 2016 Android Security Bulletin [11]). The first non-vulnerable bootloader version of

Nexus 6P is 03.64, released as part of the January 2017 Security Bulletin [13].

Google rated Vuln 2 (`CVE-2016-6678`) with **Moderate** severity and mitigated it by commit `3f3c8a8` [23]. The padding is now zeroed out so uninitialized bytes won't be leaked. The patch was released as part of the October 2016 Android Security bulletin [12].

References

[1] AMR File Format. http://hackipedia.org/File%20formats/Containers/AMR,%20Adaptive%20MultiRate/AMR%20format.pdf.

[2] HSTS Preload List Submission. https://hstspreload.org/.

[3] AMR Codec in UMTS, 2007. http://onlinelibrary.wiley.com/doi/10.1002/9780470612279.app1/pdf.

[4] Beniamini, G. QSEE privilege escalation vulnerability and exploit (CVE-2015-6639). http://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html.

[5] CodeAurora. Fastboot boot command bypasses signature verification (CVE-2014-4325), 2014. https://www.codeaurora.org/projects/security-advisories/fastboot-boot-command-bypasses-signature-verification-cve-2014-4325.

[6] Google. File-Based Encryption. https://source.android.com/security/encryption/file-based.html.

[7] Google. Full-Disk Encryption. https://source.android.com/security/encryption/full-disk.html.

[8] Google. Hardware-backed Keystore. https://source.android.com/security/keystore/index.html.

[9] Google. Make sure your device is protected. https://support.google.com/nexus/answer/6172890?hl=en.

[10] Google. Supporting direct boot. https://developer.android.com/training/articles/direct-boot.html.

[11] Google. Android Security Bulletin - November 2016, 2016. https://source.android.com/security/bulletin/2016-11-01.html.

[12] Google. Android Security Bulletin - October 2016, 2016. https://source.android.com/security/bulletin/2016-10-01.html.

[13] Google. Android Security Bulletin - January 2017, 2017. https://source.android.com/security/bulletin/2017-01-01.html.

[14] Hay, R. Undocumented Patched Vulnerability in Nexus 5X Allowed for Memory Dumping via USB, 2016. http://ibm.co/2i2HVlK.

[15] Hodges, J., Jackson, C., and Barth, A. RFC 6797: HTTP Strict Transport Security (HSTS), 2012. https://tools.ietf.org/html/rfc6797.

[16] Jimmy. ADB Commands. https://sites.google.com/site/jimmy1115kk/engineering/android/adb-commands.

[17] Kamkar, S. PoisonTap - siphons cookies, exposes internal router & installs web backdoor on locked computers, 2016. https://samy.pl/poisontap/.

[18] Krebs, B. Beware of Juice-Jacking, 2011. https://krebsonsecurity.com/2011/08/beware-of-juice-jacking/.

[19] Lau, B., Jang, Y., et al. MACTANS, Injecting Malware Into iOS Devices via Malicious Chargers. In *BlackHat USA* (2013), Georgia Institute of Technology. https://media.blackhat.com/us-13/US-13-Lau-Mactans-Injecting-Malware-into-iOS-Devices-via-Malicious-Chargers-WP.pdf.

[20] Motorola, Inc. Gadget Driver for Motorola USBNet, 2009. https://android.googlesource.com/kernel/msm.git/+/android-msm-shamu-3.10-marshmallow/drivers/usb/gadget/f_usbnet.c.

[21] Shen, D. Exploiting Trustzone on Android. In *BlackHat* (2015).

[22] Styan, C. ADB Protocol Documentation. https://github.com/cstyan/adbDocumentation.

[23] Tjin, P. usb: gadget: f_usbnet: zero out CRC padding. https://android.googlesource.com/kernel/msm/+/3f3c8a8313ff7995498d6e794f67650c8ba8072d.

[24] XDA-Developers. Brief Guide to Connect to diag port with QPST and QXDM for Nexus 6P, 2016. https://forum.xda-developers.com/nexus-6p/general/guide-biref-guide-to-connect-to-diag-t3354938.

[25] Xiao, C. DualToy: New Windows Trojan Sideloads Risky Apps to Android and iOS Devices, 2016. http://researchcenter.paloaltonetworks.com/2016/09/dualtoy-new-windows-trojan-sideloads-risky-apps-to-android-and-ios-devices/.