Security Evaluation of Android APP HackAlert

Chi Zhang

Abstract

As HackAlert is an open source android app developed by Zero Personally Identifiable Information (0pii), I did a thorough code inspection on all available files from Bitbucket, where they uploaded the source code and open to public for download.

The app was written quite considered. With restricted user input, and a matter of fact it is a mobile app accessed by user, for most of times the owner of the very device running this app or even scarcely other people, mostly with buttons, overflow and injection is not a problem. The app HackAlert has two major vulnerabilities found during inspecting the code.The first one is loggings and tracings were used or even abused as it is aeployed version. Developer mode is specified in code, but still loggings and tracings were not deactivated. The second one is about developers' attitudes toward internet communication.

Introduction

As a homework assignment for course Application Security taken from NYU Polytechnic School of Engineering, the initial motivation to do such a job was purely academical. With a mindset that ordinary developers know how to have devices do works efficiently, And security people know what are the bad ways to get the job done, I chose this android application as it could be considered as a security related app. Also, as I only have some familiarity to java, compared to all other mainstream programming languages, and the fact that this app is a free open source project, I chose this android app available in Google Play.

During the assessment process, I could clearly feel that developers of this slim app are quite clear about what they are doing, and have some understanding of what their product could do to the user's device if things went wrong. But they seems to be careless toward internet communication generated from their product. And also, not very cautious for possibilities that the device might be handled by other people rather than its owner.Also, highly limited processing resources for mobile platforms seems not be considered very seriously.

Background

HackAlert is developed based on the reality, as explained in the web page for this app in Google Play, that on Sept 9, 2014 a list of 5 million e-mail addresses, among which most were gmail accounts, were released with their password on a Russian Bitcoin forum. Though based on either developers research or already

existing claim, 50% of a small sample from that list was inaccurate as a matter of password, that they were either never used or very ancient. Though gmail should be considered a highly secure email service, existing of that particular list might cause some curiosity and little panic among its users. In helping users to know whether their own accounts or accounts belonging to people they know have been exposed, 0pii developed this app.

The app is fairly simple with a single purpose. Also, codes are short. All source codes in combine is less than 1000 lines. And user input is highly restricted as well.

Communications were handled with http protocol, which is prone to Man In the Middle attack especially when running in 802.11 networks, which in most cases are poorly configured. To make situation worse, there was no verification or authentication mechanism implemented to check any income packets. Thus it could be a used as an entry point for arbitrary codes.

As a matter of fact, number of reviews of this app is very very small, and I doubt if there would be any competitor in this merely existing market. Submissions seems to be handled by directly contacting one of the developers via email as he/she posted on product web page in Google Play.

Architecture

HackAlert has a single goal that is to see if user's or any of local contacts' email account is mentioned in the list posted on that Bitcoin forum. This job is done by two part, Get first two characters of user input and check if any exposed account starts with those two continuous characters. And check if any local contact's email address starts with those two character. If match, the app provides the option to send them a prewritten email informing them that their email account may have been exposed. To do so, all local contacts' email addresses are gathered locally. And a subset of database, which contains all 500 million may exposedaddresses, held by 0pii is downloaded. Comparing and checking are done locally. All user could do are executing the app, typing in a string greater than or equal to two characters, push the button to scan, wait, select the contact(s) (if any) he/she wants to alert, and decided whether or not to send that prewritten message.

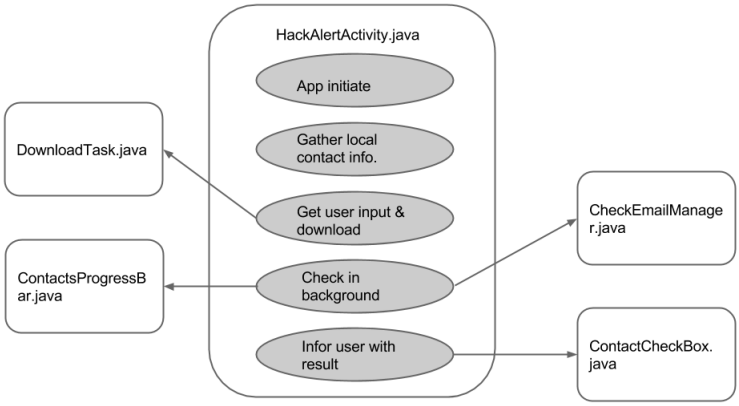

Figure: Functionality of HackAlert project.

As shown above, in package com.zeropii.hackalert, there were five classes been implemented by developers. Arrows mean calling. Class HackAlertActivity was implemented as the activity file, which is required for android platforms. This class provides all methods and subclasses to use when the app is been executed and have direct interaction with user.

CheckEmailManager was implemented to handle email checks when called in activity file. ContactCheckBox does exactly what the name reveals. And ContactsProgressBar was for handling graphic changes on the progress bar when the app is gathering all local contacts' email address.

The most serious vulnerability in this app lies in DownloadTask, a class handles database downloading. In this class, only http is used for handling internet communications. Http itself suffers from plaintext communication and poor verification mechanism. What made this app more vulnerable to a hijacked network is that, as mentioned previously, there is no verification or authentication mechanism or any inspection on what coming in from the available network.

Vulnerability Assessment

Petential Vulnerabilities

Based on source codes of this app, there could be possible vulneratilities:

Permission Abusing

App interacts with android resources. Permission granted to the app might be abused to access excessive resources rather than needed. It may put user's privacy at rist.

Privacy Breaking

Contacts' information is gathered in executing this app. How does the app handle gathered information is vital to user's privacy. If it is leaked, this app can do only one thing of great importance that is leaking user privacy.

I/O Abusing

During developing, logs and tracings would be used to facilitate debugging. If they are not deactivated or deleted in distributed version, new vulnerability could be introduced.

Executing Arbitrary Codes

Contact information is accessed, if there is a thread of contact information containing malicious code somehow obtained for some source, it might be executed. And a portion of vendor held database is downloaded in executing this app. Using a insecure protocol for such network communication could put residing device at risk by allowing arbitrary code from executing as well.

Code Sample of Interest - 1

Line 425 till 471 in HackAlertActivity.java:

```
private Map⟨String, EmailGroup⟩ getEmaiGroups() {
    Map⟨String, String⟩ emlRecs = new HashMap⟨String, String⟩();
    Map⟨String, EmailGroup⟩ emlGrp = new HashMap⟨String, EmailGroup⟩();

        ContentResolver cr = this.getContentResolver();
        String[] PROJECTION = new String[] { ContactsContract.RawContacts._ID,
            ContactsContract.Contacts.DISPLAY_NAME,
            ContactsContract.CommonDataKinds.Email.DATA};
        String order = "CASE WHEN "
            + ContactsContract.Contacts.DISPLAY_NAME
            + " NOT LIKE '%%' THEN 1 ELSE 2 END, "
            + ContactsContract.Contacts.DISPLAY_NAME
            + ", "
            + ContactsContract.CommonDataKinds.Email.DATA
            + " COLLATE NOCASE";
        StringBuilder filter = new StringBuilder();
        for(MailServer s: MailServer.values()) {
            filter.append(ContactsContract.CommonDataKinds.Email.DATA).append("
LIKE '%").append(s.getName()).append("'").append(" OR ");
        }
        filter.delete(filter.length()-4, filter.length()); delete last " OR "

        Cursor cur = cr.query(ContactsContract.CommonDataKinds.Email.CONTENT_URI,
PROJECTION, filter.toString(), null, order);
        if (cur.moveToFirst())
            do
                names comes in hand sometimes
                String name = cur.getString(1);
                String emlAddr = cur.getString(2);
                keep unique only
                if (!emlRecs.containsKey(emlAddr))
                    emlRecs.put(emlAddr, name);

                    String canonicalEmail = emlAddr;  todo
                    String key = canonicalEmail.substring(0, 2).toLowerCase();
                    EmailGroup grp = emlGrp.get(key);
                    if (grp == null) {
                        grp = new EmailGroup(key);
                        emlGrp.put(key, grp);
                    }
                    grp.add(canonicalEmail, name);

                }
            } while (cur.moveToNext());
        }

    cur.close();
```

```
    return emlGrp;
}
```

Permission Abusing

First as defined in enum MailServer in class HackAlertActivity on line 53 till 75:

```
public static enum MailServer {
    we can add more domains in the future and contacts will be filtered by these domains
    GMAIL("gmail.com"), YAHOO("yahoo.com"), LIVE("live.com"), HOTMAIL("hotmail.com"), AOL("aol.com"), YANDEX("yandex.ru");

    private String name;

    MailServer(String name) {
        this.name = name;
    }
    public String getName(){
        return name;
        }
    }
```

And the advanced for-loop showed above, only contacts' email information using these servers are gathered. getEmaiGroups() is the only method used to get contacts' email information, and as showed above, it is done by appending ContactsContract.CommonDataKinds.Email.DATA into a stringBuilder, which in fact is a kind of string stored in a stack. Then use this stringBuilder to make a local database query to get contacts' email addresses. So, Only contact information is accessed.

Privacy Breaking

Objects involved in getting contacts' email addresses are of ContentResolver, stringBuilder and casted string, and Cursor. All of them are locally defined variables within method getEmaiGroups(), thus they are all stored in a stack on the residing device. No network communication is involved on this step. Using this method alone may not cause any user privacy breaking.

I/O Abusing

Though developer mode is considered in overriding method doInBackGroud() under subclass CheckContactsTask which is also a subclass of standard class AsynTask, which is defined and used to do background works in android platforms, 6 Log.d()s are used in activity file alone for debugging and system output. Logging is very useful while developing android apps, but in runtime on real devices, it will write to dev/log/main file in device. Though it is a circular file that new data written in it will overwrite old data when it is full, with code like line 374 in activity file (Log.d(TAG, grp.emails.get(index) + " [HACKED]");) local contacts' email could still possibly be observed when device owner decided to clean up all contacts. As for mobile platforms, I/O is quite expensive. And

these not yet deactivated loggings will generate unnecessary I/O operations. And probably slow down user's device. From the code, developer mode is considered already, so it is a justifiable assumption that this demo is in fact the deployed version. If it is tue, then it directly violates android app developing guidelines. Considering real life scenario, it may not be the worst or even normal case as a vulnerability. But it should still be considered as a break of privacy as this app is claimed and intended to protect user privacy. Even though it is a demo version.

Code Sample of Interest - 2

In DownloadTask.java,
Line 33-35:

```
private static final String TAG = "DownloadTask";

private static final String WS_BASE_URL = "http:alert.0pii.comh";
```

Line 70-87:

```
public void run() {
    Set¡String¿ emails = new HashSet¡String¿();

    String httpParam = String.format("%02x%02x", (int) key.charAt(0), (int)
key.charAt(1));
    final String url = WS_BASE_URL + httpParam;

    try {
        HttpClient httpclient = new DefaultHttpClient();
        HttpResponse response;

        try {
            response = httpclient.execute(new HttpGet(url));
            StatusLine statusLine = response.getStatusLine();
            if (statusLine.getStatusCode() == HttpStatus.SC_OK) {

                ByteArrayOutputStream out = new ByteArrayOutputStream();
                response.getEntity().writeTo(out);
                BufferedReader bufferedReader = new BufferedReader(new In-
putStreamReader(new ByteArrayInputStream(out.toByteArray())));
```

Executing Arbitrary Codes

As shown in the first code sample, contact information is grated full trust. So whatever in the contact information is used to perform query directly. No check is performed to such information. Thus if somehow there is one/multiple malicious code has already been embeded in it, it might be executed. It could be regarded as vulnerability for code jection, but a bit tricky. As it needs malicious code to be embeded in contact information first somehow. But based on personal using experience of using mobile devices, it is not impossible.

As you can see in the second code sample, only http is used for network communication. The most vulnerable case is useing 802.11 networks, which is

very common. But 802.11 networks are not safe. And by design they are not secure. Developers of this app failed to regard the boundary of local device and the network as a trust boundary. And inherent the trust to local device to network communications. Though database url is predefined, by using http as the only means to download database thread without confirming authentication and verification of downloaded contents, once arp table of the router which the device is connected to got polluted, things could go wild. And as what we all have seen, most 802.11 networks nowadays are very poorly configured. To be realistic, most of such networks' accessing points have default username and password that never got changed, which is admin and admin. This makes hijacking a such local network very easy. With some simple analysing of captured packages, as http protocol uses plain text for communication, and arp pollution, hijacker could send users of this app pretty whatever he/she wants. Thus an attack using code injection could easily break all other protection on the device. And in the worst case, rootkit could be planted.

Recommendations

First, implement a mechanism to check if there are arbitrary codes in contact information. If there is, pop up alarms and terminate all processes related to this app.

Second, deactivate all Log.d() methods. If information during runtime needs to be reveal to the user. Use android.text class instead.

Third, implement an architecture using https for downloading database threads. And implement content inspection architectures to make sure that downloaded contents has no malicious things in it.

Conclusion

HackAlert is a simple software for a simple purpose. As for codes itself and how user input is handled, it is free from injections or overflows. Also, by sending only two characters, user's privacy could be protected. But because developers had either limited time or other reason, this version has directly violated a basic requirement from android, that is to deactivate all loggings and tracings when app is to deploy in Google Play platform. And developers failed to recognize trust boundary between app itself and stored data, and between local environment and network. Thus by executing this app, user might be directly exposed to a hijacked network.

With more time and a real android device, I will try observe this app's activity in situation that the process has been paused for a long time. As it is declared on line 398 in the activity file (checkEmailManager.awaitTermination(300);) to pause 5 minutes waiting for all other processes to be finished. Whether a incomplete list of contacts been hacked will be returned or the app will simply crash is to beexamined. And as many synchronization was used, I am wonder if such a hold could cause the app to crash, or cause even more severe damage to OS or device itself.