Mark V. Lese

[lesemv01@students.ipfw.edu](mailto:lesemv01@students.ipfw.edu)

CS59000-09

Dr. Elish

# Progress Report #2

March 9, 2016

## General Requirements

This semester’s project is to create at least two colluding Android applications that exhibit malicious activity when working together unbeknownst to the operator. Each application shall have at least two components. Each application shall have allocated permissions suitable for the application that masks the malicious intent. The allocated permission for the malicious intent shall be different in the applications. The malicious activity shall be a result of a user action, as malicious activity due to user activity is more difficult to detect than a background process activity.

Recall, the two applications are a Basketball Statistics editor (BBStat) and a NASA web service client (NASAClient). BBStat allows a user to enter game activity in real-time, accumulating the statistics as the game proceeds. The user may share the game results to those in his Contact list. The application requests access to the Contact list, thus exposing it to the other colluding application, NASAClient. NASAClient is simply a UI that displays information from one of NASA’s available web services. When the user requests this data, the application retrieves the Contact list from BBStat and transmits the malicious data to a web service in the cloud know only to the application’s developer. This can occur because the NASAClient has specific INTERNET permission.

## Summary

I have application shells that perform malicious activity between the colluding Android applications. The applications’ user-interfaces currently are simply default views that allow for the malicious activity to occur. Work will begin soon to create the UI’s that will mask’s the applications true intent. I am using Android Studio for development and GITHUB for source control. Source code and documentation may be found here:

<https://github.com/mvlese/CS590-MD>

Note the code is still in its infancy and is not yet properly documented.

The contact data that has already been uploaded during testing may be viewed here:

<http://cs590.leseonline.net>

This website is an interface to the **JOT** website Dustin Wolf and I developed for our **Software Engineering (ACS56000)** project. More on this is to follow.

## System Design

The system design is fairly simple as illustrated in Figure 1‑1 below.



Figure 1-1

This shows **NASAClient** sends a message to the **BBStat RemoteService** initiating the malicious activity. **BBStat RemoteService** retrieves the contact list and sends the list via messaging to the **NASAClient RemoteService**. This **RemoteService** then transmits the list to the **jot.leseonline.net** web service.

### BBStat

Since BBStat must save data from game to game, it shall use the internal Android system-provided SQLite database as shown in Figure 1‑2 below.



Figure 1-2

The design and coding to use this database is forthcoming in the next half of the project.

Figure 1‑3 on the following page shows **BBStat’s** general class diagram. Note, in particular the relationships between **RemoteService**, **Contact**, and **ContactsReader**. The **handleMessage** method receives an initiation request from **NASAClient** and calls **doWork**. The **doWork** method calls **ContactsReader** to retrieve the contacts from the system. The **doWork** method then calls **sendContacts** to the send the contact list (as a JSON string) to **NASAClient RemoteService**.

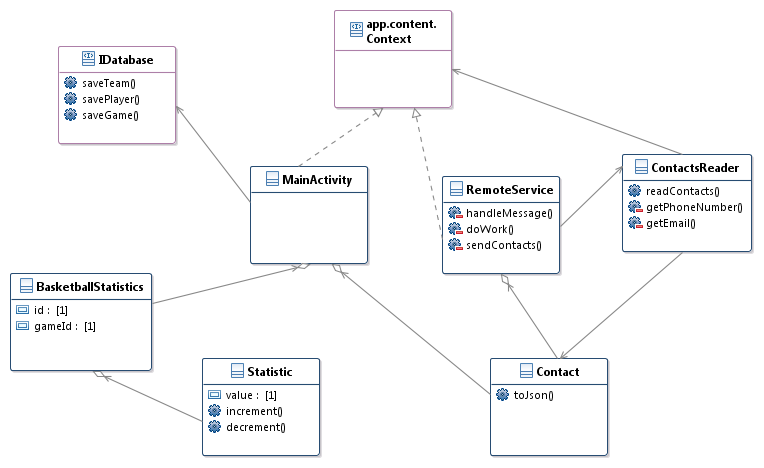


Figure 1-3

### NASAClient

Figure 1-4 below shows **NASAClient’s** basic class diagram. Note the relationship between the two packages which exists in the separate applications. The **MainActivity**, upon startup, starts **BBStat’s** RemoteService for background processing by calling **startRemoteService**. Upon a user activity, the **MainActivity** calls **getRemoteContacts** which sends a message request to the **BBStat RemoteService**. If all goes well in **BBStat**, its remote service sends a message with the contact list to the **NASAClient RemoteSevice** which then calls the **HttpUtils.post** method to send the list to the **JOT** web service.

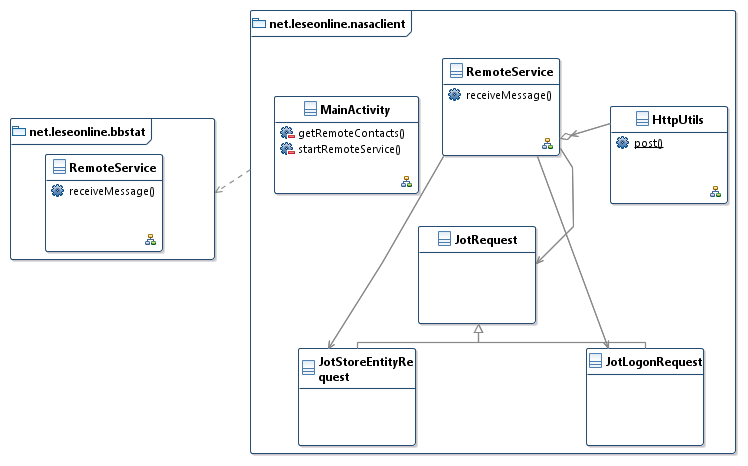


Figure 1-4

Sending the contact list to JOT is a two-step operation. First the application must login to receive a token. The token is used for all subsequent operations. Second the application sends the contact list along with the token to JOT for storage. This requires simple state activity as illustrated in Figure 1‑5 on the following page.

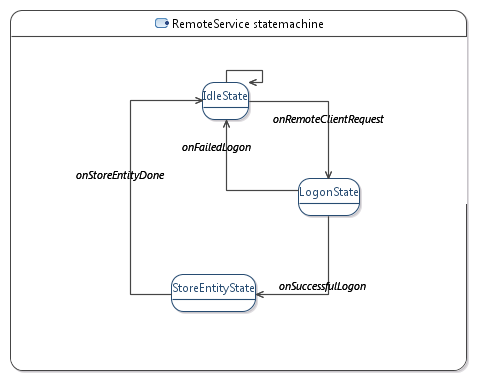


Figure 1-5

## JOT

JOT is a web service developed for another class, ACS56000. JOT provides a WSDL here:

<http://jot.leseonline.net/service2.php?wsdl>

This was the original interface, but we developed a simplified interface requiring a single parameter that was a JSON string. The JSON string encapsulates the same parameters as required by the original WSDL. The new interface is:

<http://jot.leseonline.net/capi.php>

It requires a **POST** message with the following parameter:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| api | This is a JSON representation for messaging.  For the **logon** method  { “method” : “logon”, “args: {“username”: <username>, “password”: <password>}}  The logon method returns a token, if successful.  For the **storeEntity** method  { “method” : “storeEntity”, “args: {“token”: token, “entity”: {“key”:<key>, “items” : [  {“itemid” : “1”, “itemtype” : “text”, “annotation” : <contact list as JSON string> }  ]}}}  The **key** uniquely identifies this entry. Although **items** is an array, it has only one array element. |

The contact list has the following JSON structure. The contacts element is an array and may contain one or more contacts consisting of the name, phone number, and email of the contact.

{ “contacts” : [

{ “name”:<name>, “phoneNumber”:<phoneNumber>, “email”:<email> },

…

{ “name”:<name>, “phoneNumber”:<phoneNumber>, “email”:<email> }

]}

### cs590.leseonline.net

The web site <http://cs590.leseonline.net> provides a view of the uploaded contacts lists. Figure 1‑6 below shows the current output. The key is they **key** element sent and contains the timestamp of the data.

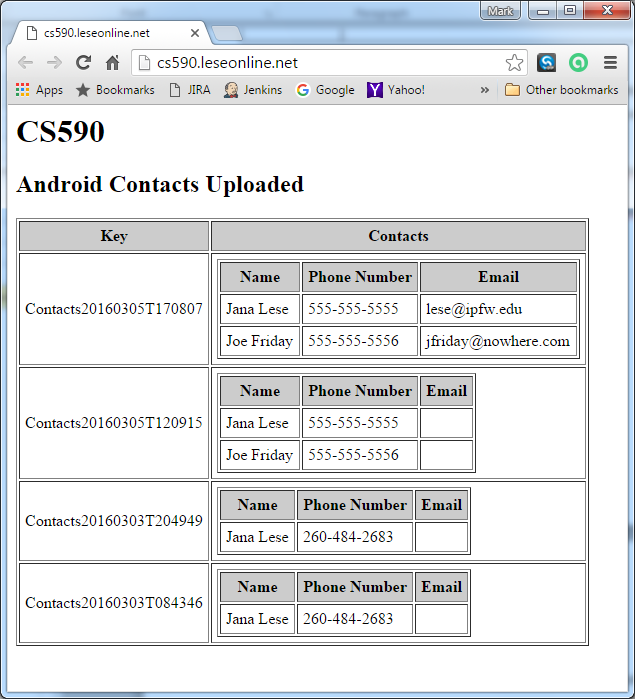


Figure 1-6

The code I slapped together for this web site is in **Appendix A** at the end of this document.

## Work to be Done

I have at least the following to get done.

### NASA Client UI

This UI should be simple to complete. I have not decided what NASA web service to use, yet. I may use several. One shows a satellite view of a given location. Another gets data on near-Earth asteroids. Others can be found here:

<https://api.nasa.gov/index.html>

### BBStat UI

This UI is ambitious for the time remaining due to the level of user interaction in a grid view. This will pose the biggest challenge to complete by the end of the semester.

## Appendix A

Note this code uses existing **JOT** code to gather the stored data for the **cs590** user. **JOT** methods used are **logon**, **getAllKeys**, and **getEntity**.

<?php

require\_once "/home/lese/www/jot/logger.php";

require\_once '/home/lese/www/jot/methods2.php';

header('Access-Control-Allow-Origin: \*');

// Comparison function (DESC by key)

function cmp($a, $b) {

$result = strcmp($a['key'], $b['key']);

$result = ($result == 1) ? -1 : (($result == -1) ? 1 : 0);

return $result;

}

echo "<h1>CS590</h1>";

echo "<h2>Android Contacts Uploaded</h2>";

$token = logon("cs590", <password remove>);

if (strlen($token) > 0) {

//echo $token . "<br>";

$searchKeyResult = getAllKeys($token);

$items = $searchKeyResult['searchKeyItems'];

//echo print\_r($items, true) . "<br><br>";

// Sort descending by key.

uasort($items, 'cmp');

//echo print\_r($items, true) . "<br><br>";

echo "<table border=\"1\" cellpadding=\"5\">\n";

echo "<tr><th style=\"background: #ccc;\">Key</th><th style=\"background: #ccc;\">Contacts</th></tr>\n";

foreach($items as $value) {

echo "<tr>";

//echo print\_r($value) . "<br><br>";

$key = $value['key'];

echo "<td>$key</td>";

echo "<td>";

//"$name</td><td>$phoneNumber</td></tr>\n;

$entityResult = getEntity($token, $key);

//echo print\_r($entityResult ) . "<br><br>";

$entities = $entityResult['entities'];

echo "<table border=\"1\" cellpadding=\"5\">\n";

echo "<tr><th style=\"background: #ccc;\">Name</th><th style=\"background: #ccc;\">Phone Number</th><th style=\"background: #ccc;\">Email</th></tr>\n";

$item = $entityResult['entities'][0]['items'][0]['annotation'];

//echo $item . "<br><br>";

$o = json\_decode($item);

foreach($o->contacts as $contact) {

//echo print\_r($o->contacts[0]->name, true) . "<br><br>";

$name = $contact->name;

$phoneNumber = $contact->phoneNumber;

$email = $contact->email;

echo "<tr><td>$name</td><td>$phoneNumber</td><td>$email</td></tr>\n";

}

echo "</table>\n";

echo "</td>";

echo "</tr>";

}

echo "</table>\n";

}

?>