Tool Chain Instantiation

Chapter 1 of this document contains the instantiation for the four tools in our toolchain. The toolchain depends on several open-source third-party packages (`wget`, `adb`, `html2text`, `tesseract`, etc.); Chapter 2 contains installation instructions for these prerequisites.
Our toolchain was developed and tested on Mac OS; since we use straight-up Unix utilities and no proprietary Mac software, Linux users should be able to install and use the toolchain with little to no effort.

Installation (no setup required, beyond installing the prerequisites)

```
$ git clone https://github.com/LeftoverAccountInformation/LAI.git
```

Notation:

./	run an executable bash script
< >	some user input
[X | Y]	choose either X or Y

Chapter 1. Tool Description

1.1. Account Deletion Analyzer

The Account Deletion Analyzer tool automatically determines whether a given APK or website has Account Deletion Functionality (ADF).

Accessing the tool

```
$ cd LAI/AccountDeletionAnalyzer/analyzer
```

Usage: The Account Deletion Analyzer tool uses the following commands:

- **./adf.sh [<apkfile> | <website>]**

  <apkfile>
  Single APK file.
  <website>
  Website which is downloaded, e.g., as a 'webpage complete' via Google Chrome.

- **./parallel.sh [<apk-directory> | <web-directory>] <max-num-of-cores>**

  <apk-directory>
  Directory which contains all downloaded APK files.
  <web-directory>
  Directory which contains all downloaded websites.
  <max-num-of-cores>
  Maximum number of CPU cores assigned to run this task in parallel.

- **./stats.sh [time | dot | svg | fp]**

  time
  Output time in seconds for each app (APK or website).
  dot
  Copy all valid *mapping_models.dot* files to *analyses/dot* directory (after running *adf.sh* or *parallel.sh*).
  svg

Convert those *mapping_model.dot* files to an *svg* file.

Description:

- APKs are collected automatically from Google Play Store website, using *auto.sh* script and uploaded to a dedicated server (please see *other useful tools* for detailed explanation of *auto.sh* script).
- With APK as an input to the command, *adf.sh* and *parallel.sh* use *Apktool* to reverse engineer APK files, *tesseract* to convert images to text and *semantics.py* to check the semantics of a phrase.
- The analyzer tool generates an output directory containing decompiled files. AD strings in *strings.xml* and callbacks associated with AD strings are used to find Account Deletion Functionality (ADF).
- With website input, *adf.sh* and *parallel.sh* use *html2text* to convert all html files to text.
- For example, 'delete account' (AD string) and 'delete password' (Not AD) are not similar, this semantic difference is identified by *semantics.py* script and returns the following as an output (TP-True positive, FP-False Positive).

  ('TP', '') means the paragraph's true meaning has been understood as intended by regex .<verb>.*<noun>.**

  ('FP', some reason) means the paragraph's true meaning has not been understood by regex .<verb>.*<noun>.**

Output:

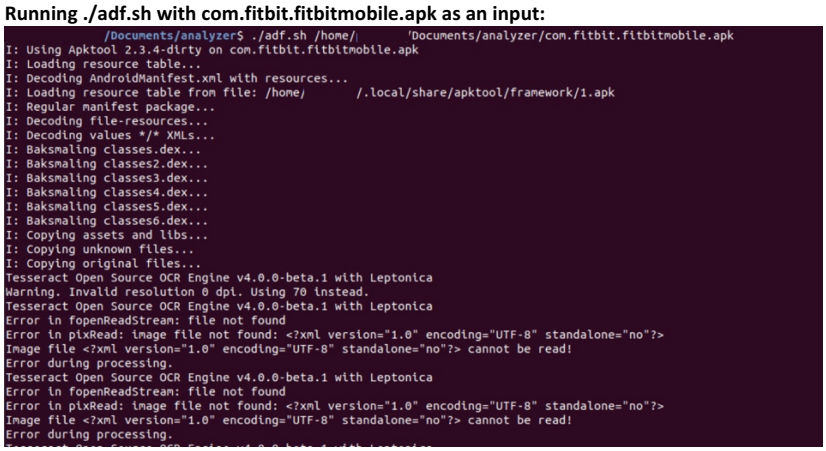
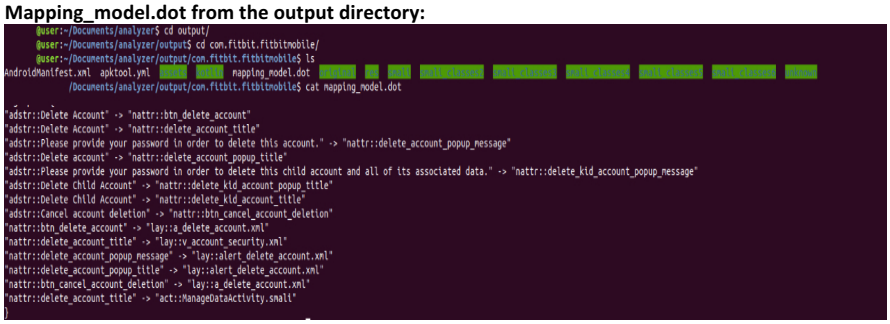
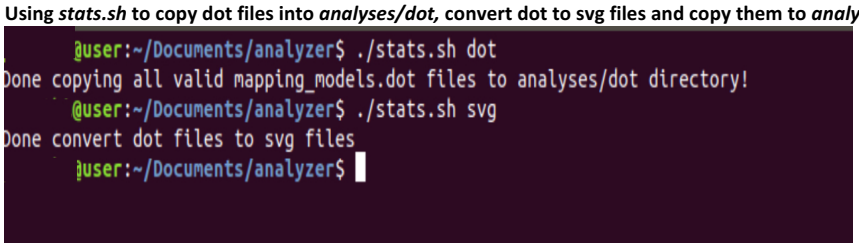
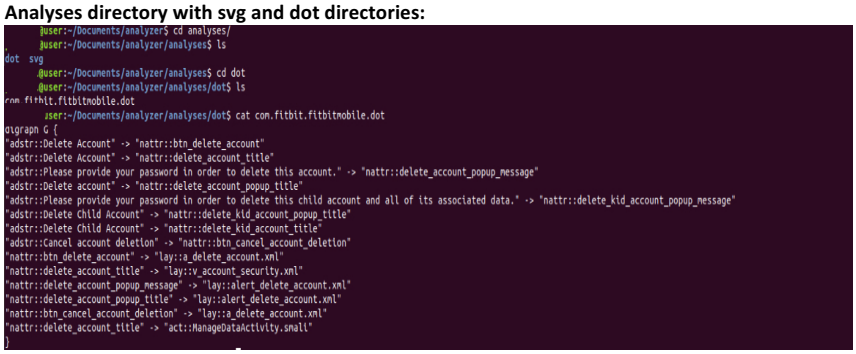
- The *adf.sh* and *parallel.sh* scripts generate an *output* directory for each APK or website. The output directory contains reverse-engineered files along with the mapping_model.dot file that builds a strings-to-screen mapping model.
- The *mapping_model.dot* file is available only when the APK or website has account deletion functionality (ADF); otherwise the *mapping_model.dot* file is empty.
- The *mapping_model.dot* file shows how AD strings (adstr) are connected to name attributes (nattr), attributes to layout (lay), layout to activity (act) inside an APK or website, and this mapping varies depending upon the design or structure of every APK or website.
- When running *stats.sh time* | *dot* | *svg* : the *time* option shows the output time of every APK or website in seconds; the *dot* option stores all dot files into *analyses/dot* directory; and *svg* converts all dot files to svg files (graphical representation of mapping_model.dot) and stores in *analyses/svg* directory.

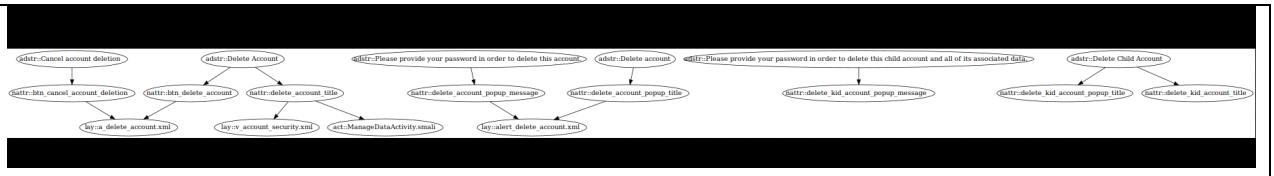
Result:

The output *mapping_model.dot* files generated by *Account Deletion Analyzer* tool are used by *Leftover Account Analyzer* tool (explained in next section) to automate the deletion process. So, it is important to run *Account Deletion Analyzer* to find ADF before *Leftover Account Analyzer.*

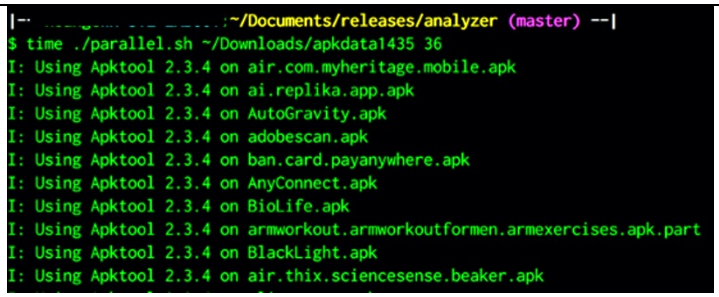
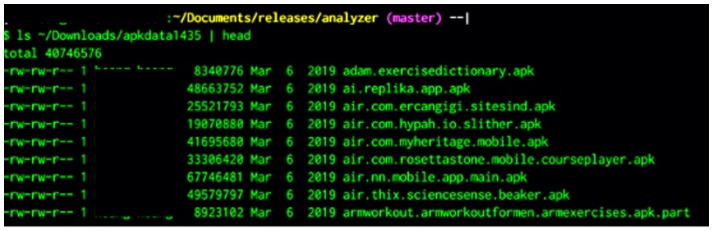
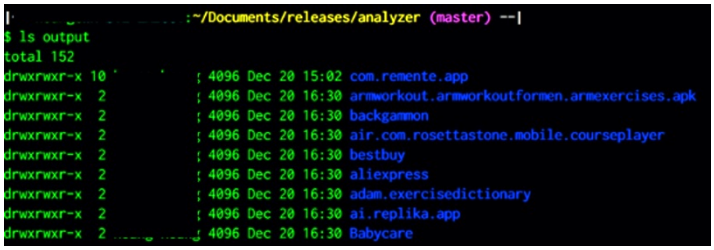
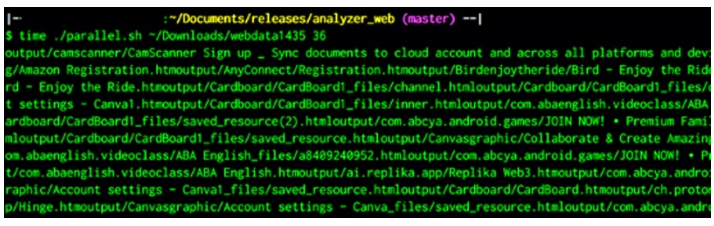
To run one single APK:

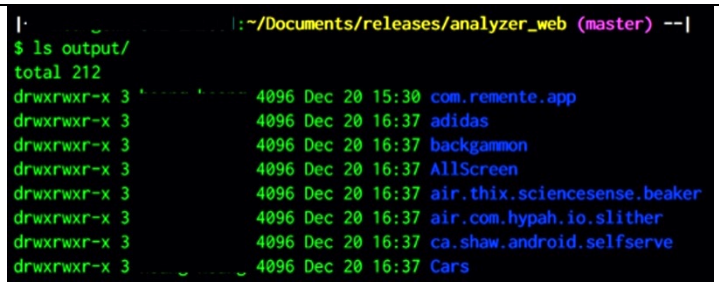
Synopsis	**./adf.sh [<apkfile> | <website>]**
Options	<apkfile> APK file. <website> Website which is downloaded as a webpage complete from Google Chrome.

Notes	Further analysis can be performed on *mapping_model.dot* file by using *stats.sh* script. Please check the *stats.sh* script below for detailed explanation.
Examples	**Running ./adf.sh with com.fitbit.fitbitmobile.apk as an input:**  **Mapping_model.dot from the output directory:**  **Using *stats.sh* to copy dot files into *analyses/dot,* convert dot to svg files and copy them to *analyses/svg***  **Analyses directory with svg and dot directories:**  ***svg* file- com.fitbit.fitbitmobile.apk**



To run adf.sh in a multiprocessing environment using parallel.sh:

Synopsis	**./parallel.sh [<apk-directory> | <web-directory>] <max-num-of-cores>**
Options	<apk-directory> Directory which contains all downloaded APK files. <web-directory> Directory which contains all downloaded websites. <max-num-of-cores> Maximum number of CPU cores assigned to run this task in parallel.
Notes	<max-num-of-cores> must be less than or equal to the actual number of CPU cores in the server.
Examples	   

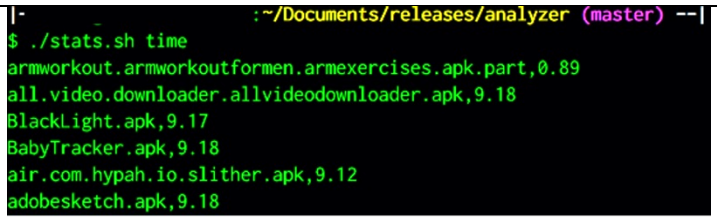
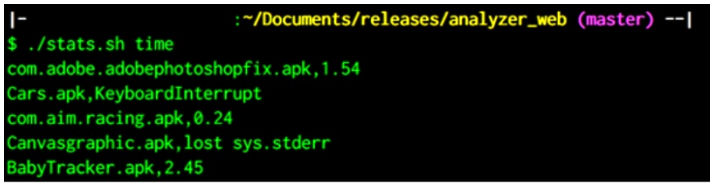
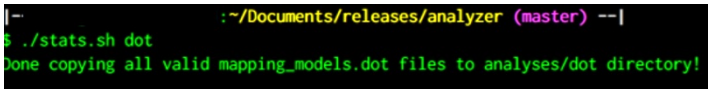
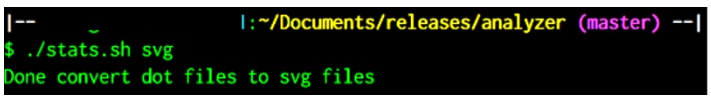


stats.sh:

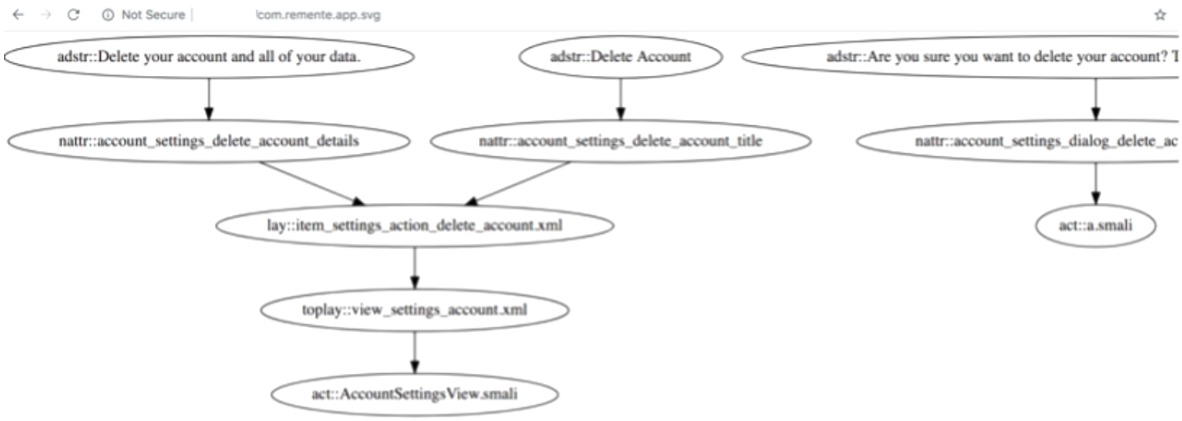
The *stats.sh time* generates time outputs of APK or website which can be redirected to a csv file for further analysis (as shown in the screenshot below).

After running *stats.sh dot*, dot files from *analyses/dot* can be viewed via terminal with *cat* mapping_mode.dot or with any text editor.

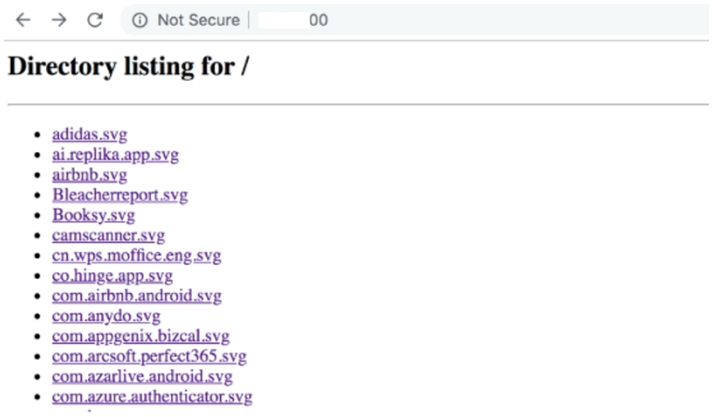
After running *stats.sh svg*, svg files from *analyses/dot* directory can be viewed using any browser or graphical editors.

Synopsis	**./stats.sh [time | dot | svg | fp]**
Description	A tool to extract statistical information from log files
Options	time Output time in seconds for each app (APK or website). dot Copy all valid *mapping_models.dot* files to *analyses/dot* directory (see *adf.sh* for more information). svg Convert those *mapping_model.dot* files to *svg* files. fp Copy all *not_ad_strings.txt* files to *analyses/fp* directory (see *adf_fp.sh* for more information from *software_manual* document).
Notes	Must run *adf.sh* or *parallel.sh* first before running this script. Time outputs can be redirected to a csv file and export to Microsoft Excel for further analyses.
Examples	   

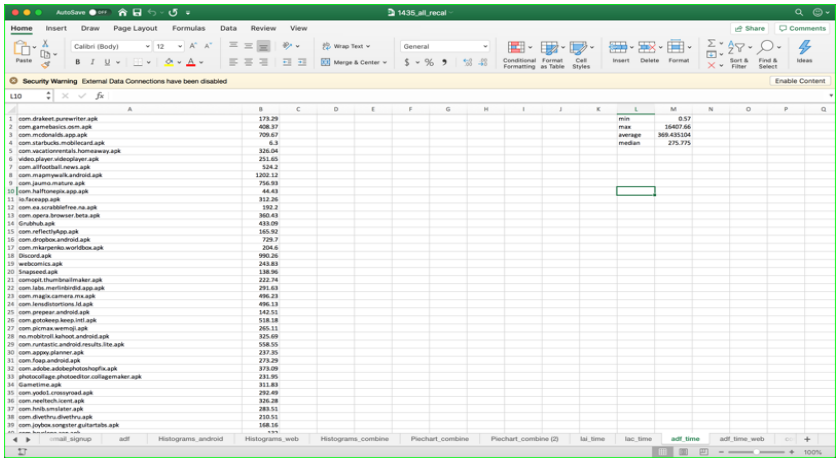
***svg* representation of com.remente.app**



***analyses/svg* directory shows a list of few svg files.**



Excel sheet showing time outputs redirected from *stats.sh* time script.



1.2. Leftover Account Analyzer

The Leftover Account Analyzer checks whether the user account is deleted from the corresponding APK or website.

Accessing the tool

```
$ cd LAI/LeftoverAccountAnalyzer/lai
```

Usage: Leftover Account Analyzer is implemented/run as follows.

Note: Please refer to the subsequent tables (followed by result) for detailed explanation of options.

To run single APK or website:

./single.sh <apkfile> [download | signup | install | uninstall | login | delete | launch | search | manual]

./single.sh <csv file > <web-directory> [signup | login | delete | manual]

To run APKs or websites in a multiprocessing environment using batch.sh:

APKs:

./batch.sh <apk-directory> [install | uninstall | remove | signup | login | delete | manual]

To extract statistical information from APK files:

./stats2.sh <log-directory>

Website:

./batch.sh <csv file > <web-directory> [signup | login | delete | manual | remove]

To extract information from webapps:

./stats2_web.sh <weblog-directory>

Description:

- The Leftover Account Analyzer checks if the user account is really deleted from the backend server. To achieve this, the analyzer *signs up* into the app, *logs in* and *deletes* the account; the again *sign up* using the same account information to find whether the account is genuinely deleted from backend server.
- If the account is deleted, the app allows us to sign up the second time using the same account information (information used to sign up first time), otherwise it displays an error message such as '*User already registered', 'Username taken'* etc.
- With the APK as input, *single.sh* and *batch.sh* scripts use Appium to automate the process based on the input options.
- For the download option, the *single.sh* and *batch.sh* scripts download APKs via SSH from the server storing APK files. With *install* option, the scripts install the APKs to the connected Android device (which is connected to the computer and runs apps automatically via Appium).
- When the script starts running, Appium triggers the Android device to open the corresponding app, and the app performs automation based on the given input options.
- For instance, if the input option is *delete*, the app automatically enters user credentials without manual intervention and logs into the account (user must sign up using *signup* option before login), uses

mapping_model.dot (output from *Account Deletion Analyzer* tool) to check if the current running app has ADF, and then deletes the user account.

- Given a website as input, the *single.sh* or *batch.sh* scripts (**./single.sh | ./batch.sh <csv file > <web-directory>[options])** use Selenium for automation in Chrome browser and ChromeDriver for launching Chrome. The input csv file has the website name in the first column and corresponding website address in the second column. The <*web-directory*> denotes the path of corresponding web app directory with options.

Output:

- The *batch.sh* script for both APKs and websites generates *_out* (output) and *_err* (error) directories in which every screen of an application (activity) is scraped during automation, and stored in the *_out* directory *(in the form of log files that contain app screens*). Errors are stored in the *_err* directory for each APK.
- If there are no errors encountered during automation of APK and website, then *_err* directory will be empty.
- The log files in the *_out and _err* directories can be used for further analyses on APKs and websites using *stats2.sh* script.

Result:

The *Leftover Account Analyzer* determines if user account from an app is deleted and uses this information for *Retention Period Analyzer* and *Leftover Account Cleaner.*

To run one single APK or website:

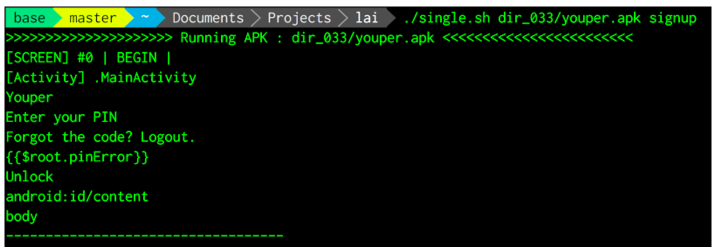
Synopsis	**APK:** **./single.sh <apkfile> [download | signup | install | uninstall | login | delete | launch | search | manual]** **Web:** **./single.sh <csv file > <web-directory> [signup | login | delete | manual]**
Description	A tool for testers to work with one single APK or website.
Options	APK: <apkfile> APK file. Choose one of the following options: download Download the <apkfile> from the server storing APK files. signup Automatically signup a user account for the <apkfile>. install Install the <apkfile> on the Android mobile phone. uninstall Uninstall the <apkfile> from the Android mobile phone. login Automatically login the user account for the <apkfile>. delete Automatically delete the user account for the <apkfile>. launch Launch the <apkfile> and fire random events with monkey tool. search Automatically search APK file for a given Android package name by automating Google web browser. manual Manually break into the ipdb console.

<table>
<tr><td></td><td>Web:

<csv file>
    csv file with a website name in first column and its equivalent website address in second column.

<web-directory>
    Directory containing webpages downloaded as ‘webpage-complete’ from Google Chrome.

Options: The operations done by web options are same as that of APK (signup, login, delete, manual).</td></tr>
<tr><td>Notes</td><td>Appium is required to run the script.
Selenium and Chrome drivers are required to run the script for websites.
Note: Run Appium-doctor to check if the dependencies and environment variables are set properly (especially JAVA_HOME and ANDROID_HOME), this allows the Appium to launch and run APKs.</td></tr>
<tr><td>Examples</td><td> 


</td></tr>
</table>

To run APKs or websites in batch mode:

<table>
<tr><td>Synopsis</td><td>APK:
./batch.sh <apk-directory> [install | uninstall | remove | signup | login | delete | manual]

Web:
./batch.sh <csv file > <web-directory> [signup | login | delete | manual | remove]</td></tr>
<tr><td>Description</td><td>A tool for testers to work with a directory containing APKs or websites (working in batch mode).</td></tr>
<tr><td>Options</td><td>APK:

<apk-directory>
    Directory which contains APK files.

Choose one of the following options:
    install
        Install all the APK files in the directory on the Android mobile phone.
    uninstall
        Uninstall all the APK files in the directory from the Android mobile phone.
    remove
        Remove the <apk-directory> and all the output directories _out and error directories _err, ready for next batch.
    signup
        Automatically signup a user account for each of the APK file in <apk-directory>.
    login
        Automatically login the user account for each of the APK file in <apk-directory>.
    delete
        Automatically delete the user account for each of the APK file in <apk-directory>.
    manual
        Manual intervention from testers, break into ipdb console and finish any of the options above.
        Testers can also use function dump_text built-in the tool to do screen scraping on the Android phone.</td></tr>
</table>

<table>
<tr><td></td><td>Web:
<csv file>
csv file with website names in first column and its equivalent website addresses in second column.

<web-directory>
Directories containing webpages downloaded as ‘webpage-complete’ from Google Chrome.
Options: The operations done by web options are same as that of APK (signup, login, delete, manual, remove).</td></tr>
<tr><td>Notes</td><td>Appium is required to run the script for APKs.
Selenium and Chrome drivers are required to run the script for websites.
Depending on the chosen option, the tool will save log files into _out and _err directories, from which statistical analysis can be performed.</td></tr>
<tr><td>Examples</td><td><pre>
base master ~ Documents Projects lai ./batch.sh dir_033 signup
>>>>>>>>>>>>>>>>>>>>>> Running APK : dir_033/zillow.apk <<<<<<<<<<<<<<<<<<<<<<<<
[SCREEN] #0 | BEGIN |
[Activity] com.zillow.android.re.ui.onboarding.view.LoginOnboardingActivity
Welcome!
Enter your email to get started
Email address
Continue
Or
Skip
googleplus
com.zillow.android.zillowmap:id/action_bar_root
android:id/content
com.zillow.android.zillowmap:id/fragment_container
com.zillow.android.zillowmap:id/login_layout_scroll
com.zillow.android.zillowmap:id/space
com.zillow.android.zillowmap:id/icon
com.zillow.android.zillowmap:id/login_title
com.zillow.android.zillowmap:id/login_explanation
com.zillow.android.zillowmap:id/zillow_login_layout
com.zillow.android.zillowmap:id/email_text_input_layout
com.zillow.android.zillowmap:id/label
com.zillow.android.zillowmap:id/input
com.zillow.android.zillowmap:id/login_button
com.zillow.android.zillowmap:id/third_party_signin_prefix
com.zillow.android.zillowmap:id/third_party_signin_layout
com.zillow.android.zillowmap:id/google_sign_in_button
com.zillow.android.zillowmap:id/facebook_login_button
com.zillow.android.zillowmap:id/register_terms_button
com.zillow.android.zillowmap:id/bottom_left_button
com.zillow.android.zillowmap:id/toolbar_as_actionbar

[SCREEN] #1 | IN REGISTER |
</pre><pre>
base master ~ Documents Projects lai ls dir_033_err/ dir_033_out/
dir_033_err/:
Zalo.log yelp.log youper.log yourphonecompanion.log

dir_033_out/:
Zalo.log yelp.log youper.log yourphonecompanion.log
</pre><pre>
base master ~ Documents Projects lai ./batch.sh dir_033 manual
>>>>>>>>>>>>>>>>>>>>>> Running APK : dir_033/yourphonecompanion.apk <<<<<<<<<<<<<<<<<<<<<<<<
--Return--
    s/Projects/lai/manual.py(22)<module>()
---> 22 import ipdb; ipdb.set_trace(context=1)

ipdb> dump_text()
[Activity] .Activity.WelcomeActivity
Update your phone to link
Your phone needs to be on Android version 7.0 or higher to link with your PC.
Learn more
Continue to Home Screen
Update your phone to link, Heading, Your phone needs to be on Android version 7.0 or higher to link with your PC.
Learn more, Link
Continue to Home Screen, Button
android:id/content
com.microsoft.appmanager:id/activity_welcome_root
com.microsoft.appmanager:id/error_view_root
com.microsoft.appmanager:id/error_layout
com.microsoft.appmanager:id/error_img
com.microsoft.appmanager:id/error_title
com.microsoft.appmanager:id/error_detail
com.microsoft.appmanager:id/learn_more
com.microsoft.appmanager:id/error_go_back_btn
----
ipdb>
</pre></td></tr>
</table>

```
master ~ Documents Projects lai ./batch.sh dir_033/ install
Installing dir_033//Zalo.apk
dir_033//Zalo.apk: 1 file pushed. 5.4 MB/s (41884948 bytes in 7.407s)
        pkg: /data/local/tmp/Zalo.apk
Success
Installing dir_033//yelp.apk
dir_033//yelp.apk: 1 file pushed. 6.1 MB/s (48800284 bytes in 7.591s)
        pkg: /data/local/tmp/yelp.apk
```

```
master ~ Documents Projects lai ls dir_033/
Zalo.apk  yelp.apk  youper.apk  yourphonecompanion.apk  zillow.apk  zomato.apk
```

stats2.sh:

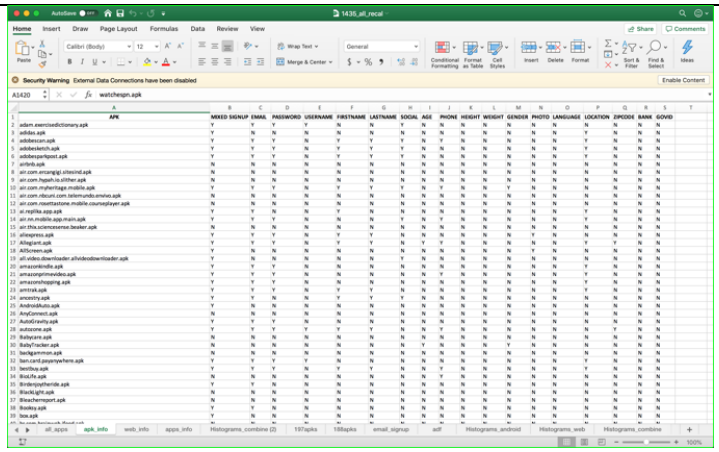
Running *stats2.sh <log-directory>* generates information about each APK inside the log directory.

For instance, if adidas.apk (see the subsequent output image and Excel sheet for adidas.apk, second app from the top) outputs Y, N, N, N, N, N……., the first Y (Y=Yes, required) in the sequence denotes that the Adidas app has mixed signup (refer the below column headings from Excel sheet), i.e., the Adidas app can be signed up using Google, Yahoo, or Facebook (that is, mixed signup options); the second N (N=No, not required) denotes that Adidas app doesn't require email for signup. This kind of information can be found from *stats2.sh script* for APKs.

Statistical analysis on the directories after running batch.sh script for APKs:
stats2.sh:

Synopsis	**./stats2.sh <log-directory>**
Description	Gather statistics for each Android app after running *batch.sh* script.
Options	<log-directory> Directory which contains all the log files stored under *_out* and *_err* directories.
Notes	The results can be redirected to a csv file and exported to Microsoft Excel for further analyses.

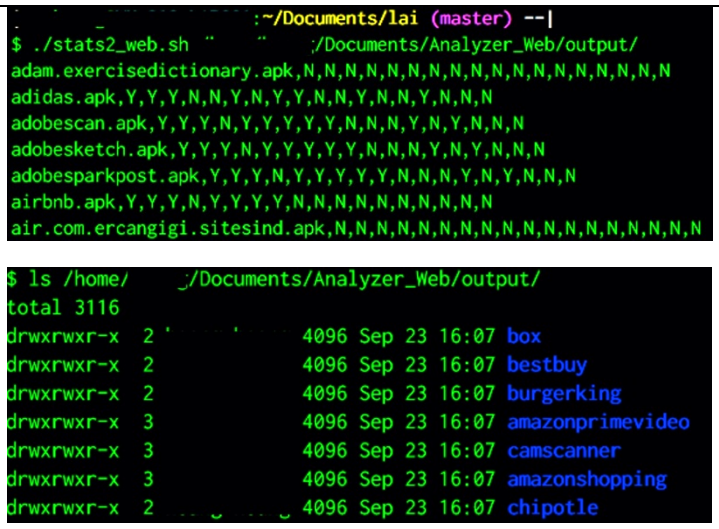
Examples	`$ ./stats2.sh ~/Downloads/chunks1435_lai_logs/`
	`adam.exercisedictionary.apk,Y,Y,Y,Y,N,N,Y,N,N,N,N,N,N,N,N,N,N`
	`adidas.apk,Y,N,N,N,N,N,N,N,N,N,N,N,N,Y,N,N,N`
	`adobescan.apk,Y,Y,Y,N,Y,Y,Y,N,Y,N,N,N,N,N,Y,N,N,N`
	`adobesketch.apk,Y,Y,Y,N,Y,Y,Y,N,N,N,N,N,N,N,Y,N,N,N`
	`adobesparkpost.apk,Y,Y,Y,N,Y,Y,Y,N,N,N,N,N,N,N,Y,N,N,N`
	`airbnb.apk,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N`
	`air.com.ercangigi.sitesind.apk,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N`
	`air.com.hypah.io.slither.apk,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N`
	`air.com.myheritage.mobile.apk,Y,Y,Y,N,Y,Y,Y,N,Y,N,N,Y,N,N,N,N,N,N`
	`air.com.nbcuni.com.telemundo.envivo.apk,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N`
	
	`$ ls ~/Downloads/chunks1435_lai_logs/`
	`total 268`
	`drwxrwxr-x 2 [redacted] 4096 Sep 2 22:22 dir_001_out`
	`drwxrwxr-x 2 [redacted] 4096 Sep 2 22:22 dir_001_err`
	`drwxrwxr-x 2 [redacted] 4096 Sep 3 16:59 dir_033_err`
	`drwxrwxr-x 2 [redacted] 4096 Sep 3 16:59 dir_033_out`
	`drwxrwxr-x 2 [redacted] 4096 Sep 4 11:40 dir_002_err`
	`drwxrwxr-x 2 [redacted] 4096 Sep 4 11:40 dir_002_out`
	`drwxrwxr-x 2 [redacted] 4096 Sep 4 18:14 dir_003_err`
	`drwxrwxr-x 2 [redacted] 4096 Sep 4 18:14 dir_003_out`



statst2_web.sh:

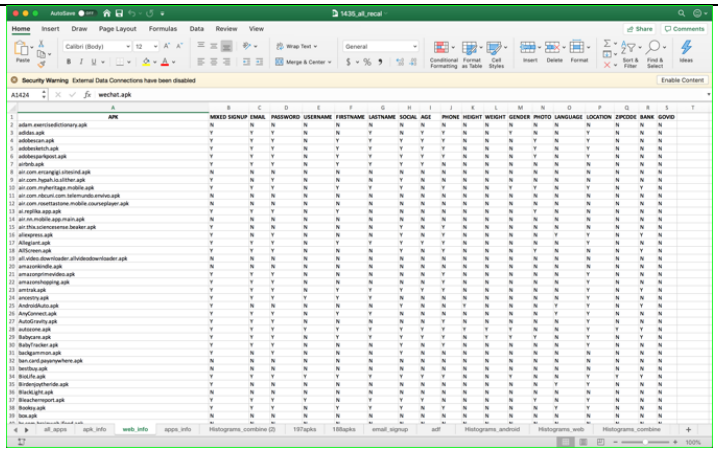
stats2_web.sh <log-directory> generates information about each website inside the log directory. (same as that of previous *stats2.sh* script)

Statistical analysis on the directories after running batch.sh script for websites
stats2_web.sh

Synopsis	**./stats2_web.sh <weblog-directory>**
Description	Gather statistics for each website after running *batch.sh* script.
Options	<weblog-directory> Directory which contains all the log files stored under *_out* and *_err* directories for websites.
Notes	The results can be redirected to a csv file and exported to Microsoft Excel for further analyses.
Examples	

```
:~/Documents/lai (master) --|
$ ./stats2_web.sh "     "     :/Documents/Analyzer_Web/output/
adam.exercisedictionary.apk,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N
adidas.apk,Y,Y,Y,N,N,Y,N,Y,Y,N,N,Y,N,N,Y,N,N,N
adobescan.apk,Y,Y,Y,N,Y,Y,Y,Y,Y,N,N,N,Y,N,Y,N,N,N
adobesketch.apk,Y,Y,Y,N,Y,Y,Y,Y,Y,Y,N,N,N,Y,N,Y,N,N,N
adobesparkpost.apk,Y,Y,Y,N,Y,Y,Y,Y,Y,Y,N,N,N,Y,N,Y,N,N,N
airbnb.apk,Y,Y,Y,N,Y,Y,Y,Y,N,N,N,N,N,N,N,N,N,N
air.com.ercangigi.sitesind.apk,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N
```

```
$ ls /home/     :/Documents/Analyzer_Web/output/
total 3116
drwxrwxr-x  2 '     '     4096 Sep 23 16:07 box
drwxrwxr-x  2             4096 Sep 23 16:07 bestbuy
drwxrwxr-x  2             4096 Sep 23 16:07 burgerking
drwxrwxr-x  3             4096 Sep 23 16:07 amazonprimevideo
drwxrwxr-x  3             4096 Sep 23 16:07 camscanner
drwxrwxr-x  3             4096 Sep 23 16:07 amazonshopping
drwxrwxr-x  2             4096 Sep 23 16:07 chipotle
```



1.3. Retention Period Analyzer

When a user deletes the user account, a message might pop up, indicating that the account can be deleted only after 7 days, or 1-month, etc.; this is called "retention period". It is required to run *Account Deletion Analyzer* and *Leftover Account Analyzer* before *Retention Period Analyzer* because this tool runs on only apps that have ADF.

Accessing the tool

```
$ cd LAI/RetentionPeriodAnalyzer/
```

Usage:

APK: **python py.py**

Website: **python testing.py <web-directory.apk>**

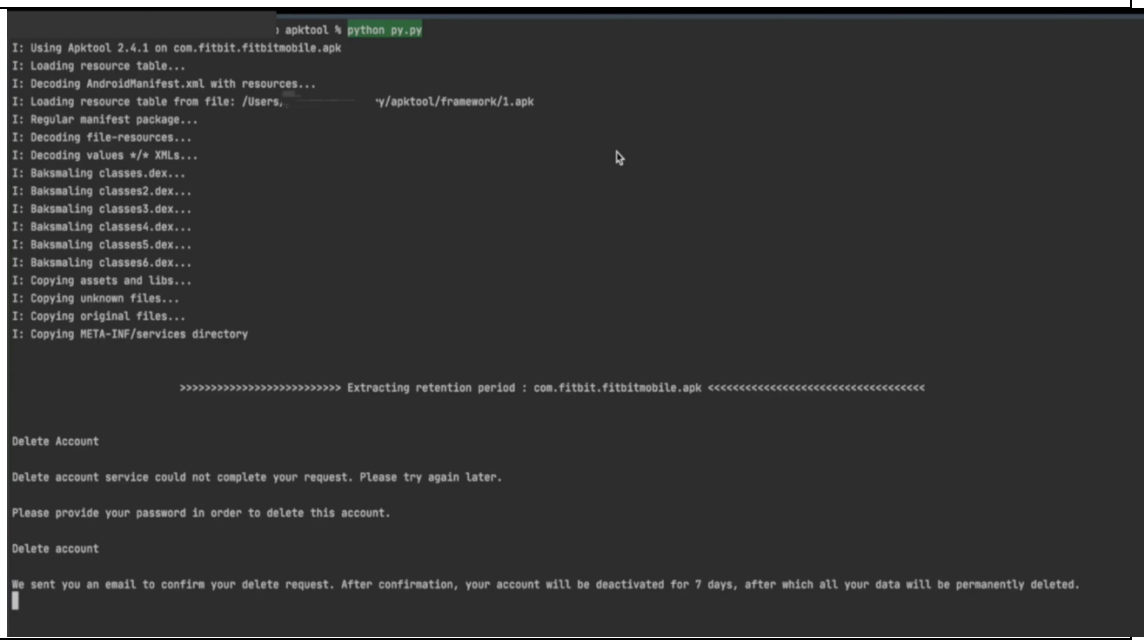
Description:

- Retention Period Analyzer for APK and websites; identifies whether an app has retention period after deletion or not.
- The Retention Period Analyzer for APK uses Apktool to reverse engineer files and depends on semantics.py to check semantics of a phrase using natural language processing techniques.
- The *testing.py* script converts HTML files to text files and depends on semantics.py to check semantics of a phrase using NLP techniques.

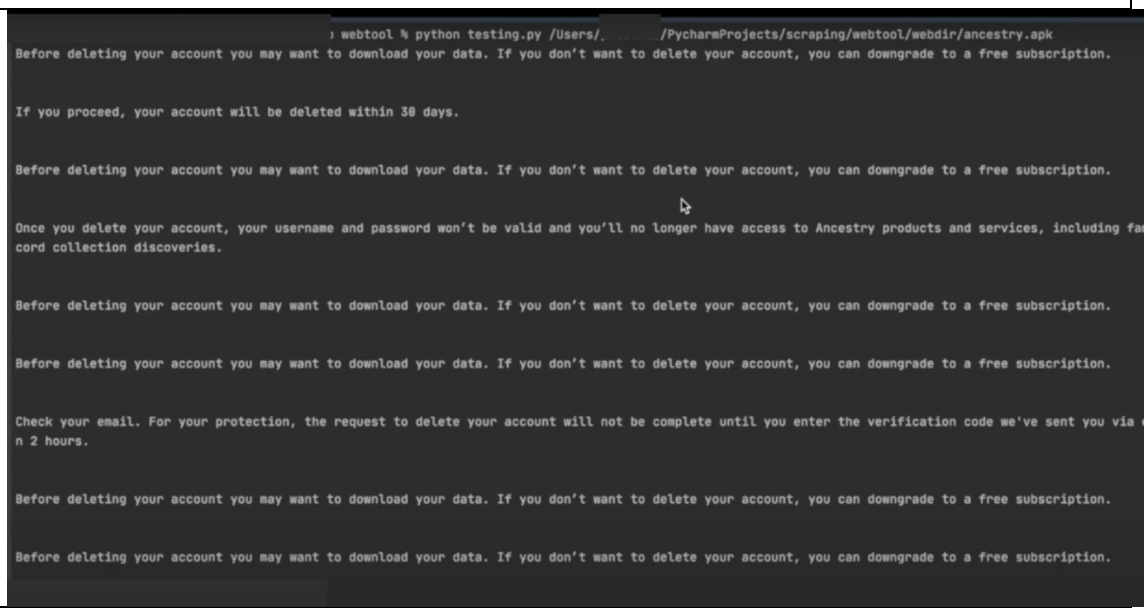
Output:

- The *py.py* script generates an output directory which contains the decompiled files and uses string.xml file from it to extract retention period patterns to check against semantics.py.
- The *testing.py* script generates an output directory that contains the converted text files and uses it to extract retention period patterns to check semantics using semantics.py.

To find retention period for APKs:

Synopsis	**python py.py** **Change corresponding app folder names in the source code and run.**
Options	None
Notes	Apktool is required to reverse engineer files.
Examples	

To find retention period for webapps:

Synopsis	**python testing.py <web-directory.apk>**
Options	None
Notes	This program converts folder containing webpages to texts and creates an output directory that contains converted text folders then uses these files to extract retention period patterns.
Examples	

1.4. Leftover Account Cleaner

If an app is uninstalled from the device when a user no longer needs it, but he/she forgets to delete the user account created for that specific application, then *Leftover Account Cleaner* can be used for automatically deleting the user accounts of these uninstalled apps.

Accessing the tool

```
$ cd LAI/LeftoverAccountCleaner/
```

Usage:

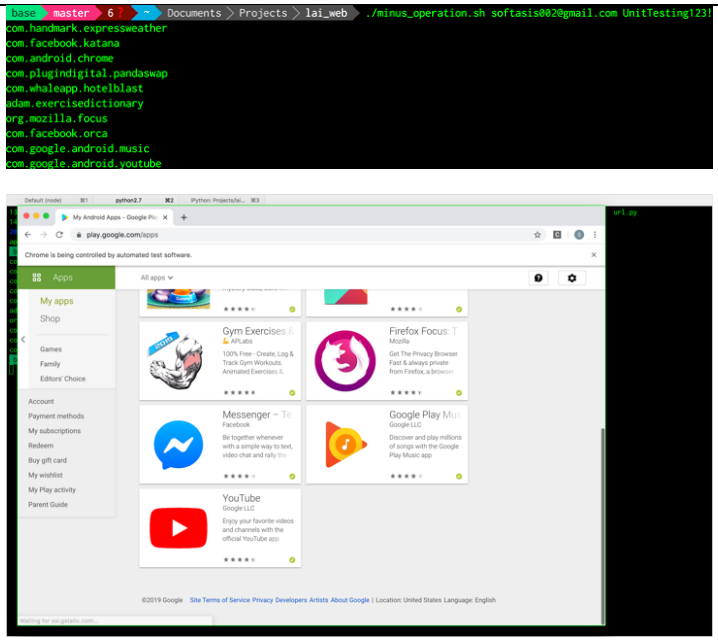
./lac_batch.sh [<gmail> <gpassword> minus (see *minus_operation.sh*)
| <./minus/laXX> install | ./batch.sh install
| <package-name> <app-email> <app-password> clean
| ./single.sh <delete> - To delete apps
| pull
| uninstall | ./single.sh uninstall
| <./minus/laXX> remove]

Description:

- *./lac_batch.sh* script is used for cleaning up the non-deleted user accounts from uninstalled apps.
- *./lac_batch.sh* offers the following options: minus, install, clean, delete, pull, uninstall, and remove.
- The Google Play Store website maintains a list of all installed apps (even if it is uninstalled on the device) that's been installed to a *particular* Gmail ID in *My apps* section.
- Then, the difference between all Android packages on Google Play Store website versus Android packages on the device will give the list of uninstalled apps using *minus_operation.sh* script.
- *minus_operation.sh* depends on *my_apps.py* script to auto login to the Google Play Store website to get the list of apps. A compatible *ChromeDriver* is required for using Chrome web browser.

Usage: ./minus_operation.sh <gmail> <gpassword>

Synopsis	**./minus_operation.sh <gmail> <gpassword>**
Options	<gmail> Google email address to login play store. <gpassword> Google password to login play store.
Notes	A compatible *ChromeDriver* is required for using Chrome web browser.

Examples	

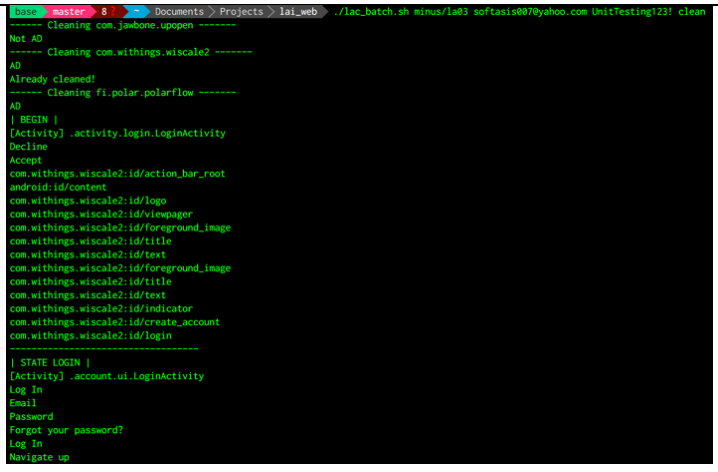
Output:

- The ./lac_batch.sh <gmail> <gpassword> minus outputs the list of apps to be cleaned and stores these apps in <*./minus/laXX*>.
- The directory <. */minus/laXX*>, contains a list of apps that needs to be installed using *install* option on Android device for deletion.
- Upon installation, the <package name> which is minus/laXX (containing the list of installed apps) needs to be cleaned, and Appium is required to run *lac_batch.sh* script for automatic deletion.
- The *./lac_batch.sh clean,* outputs the apps with AD (Account Deletion feature) and Not AD (No Account Deletion feature), then automatically deletes using clean or delete option to delete user accounts.
- Upon deletion,
  with *pull* option, APKs are pulled to the computer,
  with *uninstall,* uninstalled from Android device, and
  with *remove,* minus/laXX directory is removed for the next batch.

Result:

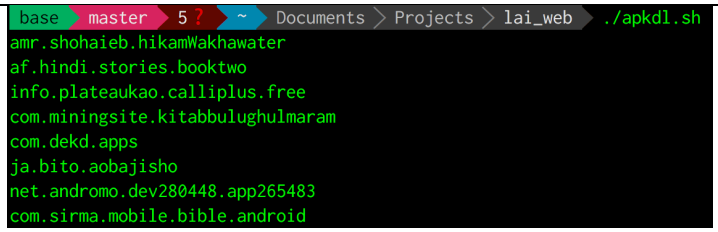
The account cleaner tool finds the list of not deleted, uninstalled apps, and cleans those automatically.

Synopsis	**./lac_batch.sh [<gmail> <gpassword> minus** (see minus_operation.sh) **| <./minus/laXX> install | ./batch.sh install** **| <package-name> <app-email> <app-password> clean** **| ./single.sh <delete> - To delete apps** **| pull** **| uninstall | ./single.sh uninstall** **| <./minus/laXX> remove]**
Options	minus See *minus_operation.sh*.

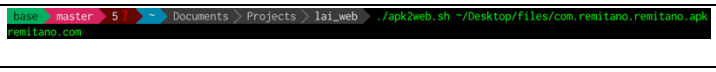
	<gmail> Google email address to login play store. <gpassword> Google password to login play store. install Automatically searching the play store for packages provided in <./minus/laXX> and install them on Android phone. <./minus/laXX> A text file which contains a list of Android package names. clean Given <app-email> and <app-password>, this outputs the apps that contain Account Deletion (AD) and apps that doesn't contain AD (Not AD), and will automatically delete the user account on a given <package-name> for apps with AD. <package-name> The package-name of an app currently residing on Android phone. <app-email> The email that user has provided when signing up with an app on Android phone. <app-password> The password that user has provided when signing up with an app on Android phone. or delete Deletes the user accounts of apps that contain AD. pull Pulls all non-system packages from Android phone to the connected computer. uninstall or ./single.sh <apk-file>| ./batch.sh <apk-directory> uninstall. remove Remove all log directories *_out* and *_err*, ready for the next batch. <./minus/laXX> A text file which contains a list of Android package names.
Notes	Appium is required to run the script.
Examples	

1.5. Other Useful Tools

apkdl.sh

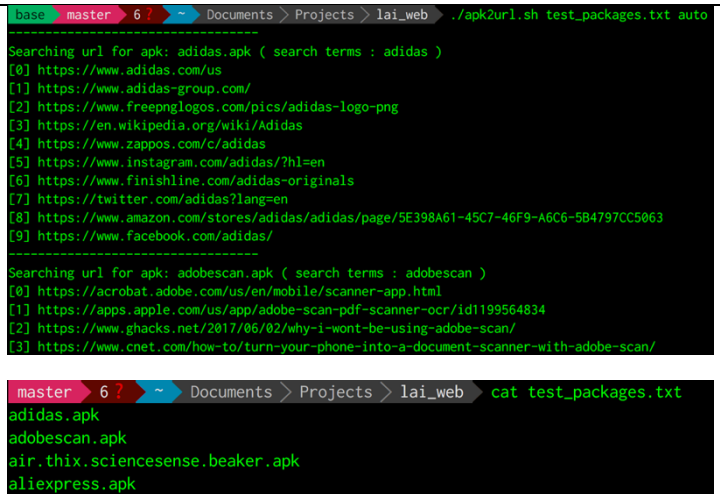
Synopsis	**./apkdl.sh**
Description	Scrape all the Android package names from a website.
Options	None The input website is already hardcoded as apk-dl.com. The script will return a list of Android package names.
Dependencies	curl awk
Packages	lai_web
Examples	

apk2web.sh

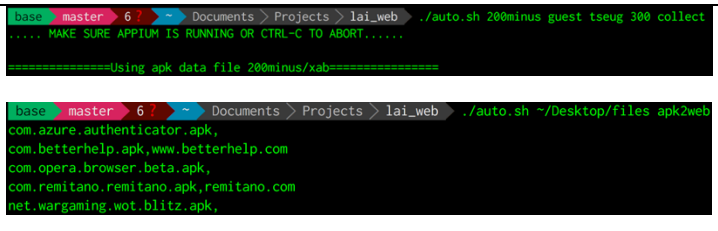
Synopsis	**./apk2web.sh <apkfile>**
Description	Find the equivalent website address for an APK file by searching the manifest.
Options	<apkfile> An APK file. Given an APK file, the script will return the equivalent website address.
Dependencies	aapt dos2unix grep
Packages	lai_web
Examples	

apk2url.sh

Synopsis	**./apk2url.sh <packagelist> auto**
Description	Finds the equivalent website addresses for a list of Android package names via Google.
Options	<packagelist> Text file containing Android package names. The auto flag is optional: If present, the script will automatically pick the first website address from Google candidate search results. If not present, the script will let testers pick a website manually from the Google candidate search results.
Dependencies	url.py
Packages	lai_web

Examples	

auto.sh

Synopsis	**./auto.sh [<apklist-directory> <server-username> <server-password> <wait-time-in-secs> collect | <apkdir> apk2web]**
Description	An automatic script to collect APKs and websites without any manual intervention.
Options	collect Automatically go to Google Play Store, download APKs, and upload them to a dedicated server. <apklist-directory> Directory which contains files whose contents are Android package names. <server-username> SSH username. <server-password> SSH password. <wait-time-in-secs> Time to wait between each batch of APK files finish uploading to the server. apk2web Automatically find equivalent websites from a directory containing all APK files pulled from the mobile phone. <apkdir> Directory which contains all APK files pulled from the Android mobile phone.
Dependencies	lac_batch.sh upload.sh apk2web.sh
Packages	lai_web
Notes	Appium is required to run the script.
Examples	

Chapter 2. Installation Instructions

2.1. Prerequisites

A laptop with Mac OS High Sierra or higher. Instructions for installing the required utilities/packages are provided in the subsequent tables.

Package	**Homebrew**
Command	/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
Notes	Enter your password to continue with the installation.
References	https://brew.sh/

Package	**wget**
Command	brew install wget
Notes	None
References	https://www.cyberciti.biz/faq/howto-install-wget-om-mac-os-x-mountain-lion-mavericks-snow-leopard/

Package	**adb**
Command	Install adb via Homebrew: brew install android-platform-tools. Start adb: adb devices
Notes	Setting up Android phone: • Settings->About device->Software info->To enable developer options, tap on build number 7 times. • Need to enable USB debugging, this allows Android Studio and other SDK tools to recognize the Android device when connected via USB. • Enable the Developer options on, which allows communication between device and the connected computer.
References	https://developer.android.com/studio/debug/dev-options

Package	**html2text**
Command	It can be installed via pip or homebrew: pip install html2text brew install html2text
Notes	None
References	https://pypi.org/project/html2text/ https://formulae.brew.sh/formula/html2text

Package	**tesseract**
Command	The easiest way to install tesseract is using Homebrew. brew install tesseract (for English language) brew install tesseract-lang (For all languages) To view the list of languages supported by tesseract: tesseract --list-langs
Notes	None
References	https://formulae.brew.sh/formula/tesseract https://guides.library.illinois.edu/c.php?g=347520&p=4121425

Package	**apktool**
Command	brew install apktool Try running apktool via cli.
Notes	The latest version will be installed in /usr/local/Cellar/apktool/[version]/ and linked to /usr/local/bin/apktool. We can check the location with 'which apktool'.
References	https://ibotpeaches.github.io/Apktool/install/

Package	**emacs, vscode, sublime**
Command	emacs: brew cask install emacs vscode: Download for Mac. Sublime: Download for Mac.
Notes	Any text editor can be used.
References	https://www.gnu.org/software/emacs/download.html https://code.visualstudio.com https://www.sublimetext.com

Package	**Eclipse IDE**
Command	Navigate to the references link: Download Open the Eclipse installer. Choose Eclipse IDE for Java Developers, choose the installation folder and Install. Select a directory as workspace->Launch.
Notes	
References	https://projects.eclipse.org/releases/kepler

Package	**ipdb**
Command	Run the following command to install ipdb via pip: pip install ipdb Run python

	import ipdb ipdb.set_trace()
Notes	
References	https://pypi.org/project/ipdb/

Package	**Android studio**
Command	
Notes	Download Android Studio for Mac.
References	https://developer.android.com/studio

Package	**Appium Python Client**
Command	pip install Appium-Python-Client
Notes	
References	https://pypi.org/project/Appium-Python-Client/

Package	**Appium, Appium-doctor**
Command	For Appium desktop: 'Download Appium' To download via Homebrew: brew install node # get node.js npm install -g appium # get Appium npm install wd # get Appium client appium & # start Appium node your-appium-test.js Appium-doctor: npm install -g appium-doctor appium-doctor
Notes	To verify that all of Appium's dependencies are met you can use Appium-doctor to verify that all of the dependencies are set up correctly. Set JAVA_HOME and ANDROID_HOME correctly to run Appium.
References	http://appium.io

Package	**Selenium**
Command	Use pip to install selenium: pip install selenium
Notes	
References	https://selenium-python.readthedocs.io/installation.html

Package	**Chrome browser**
Command	
Notes	'Download Chrome'
References	https://www.google.com/chrome/

Package	**Chrome driver: Web driver for chrome**
Command	
Notes	Choose and download your chrome driver according to the chrome browser version.
References	https://sites.google.com/a/chromium.org/chromedriver/downloads

Package	**nltk**
Command	pip install --user -U nltk run python then type: import nltk nltk.download()
Notes	A new window should open, showing the NLTK Downloader. Click on the File menu and select Change Download Directory. For central installation, set this to /usr/local/share/nltk_data (Mac). Next, select the packages or collections you want to download. It is recommended to download all packages. Following are the packages: All-corpora, all-nltk, book, popular, tests, third-party
References	http://www.nltk.org/install.html, http://www.nltk.org/data.html

The following subsections show the dependencies and packages for each tool.

2.2. Account Deletion Analyzer

Command	**./adf.sh**
Dependencies	apktool find awk xargs tesseract grep sed semantics.py
Packages	analyzer analyzer_web

Command	**semantics.py**
Dependencies	pystatparser nltk

Packages	nlp

Command	**./parallel.sh**
Dependencies	adf.sh xargs

Command	**./stats.sh**
Dependencies	find sed tail grep dot wc

2.3. Leftover Account Analyzer

Command	**./single.sh <apkfile> [download | signup | install | uninstall | login | delete | launch | search | manual]** **./single.sh <csv file > <web-directory> [signup | login | delete | manual]**
Dependencies	appium selenium adb monkey scp csv html2text re pyautogui chrome browser
Packages	lai lai_web

Command	**./batch.sh <apk-directory> [install | uninstall | remove | signup | login | delete | manual]** **./batch.sh <csv file > <web-directory> [signup | login | delete | manual | remove]**
Dependencies	Appium selenium adb aapt grep csv selenium html2text re pyautogui chrome browser
Packages	lai

	lai_web

Command	**./stats2.sh**
Dependencies	grep
Packages	lai

Command	**./stats2_web.sh**
Dependencies	grep
Packages	lai

2.4. Retention Period Analyzer

Command	**py.py**
Dependencies	semantics.py, apktool

Command	**testing.py**
Dependencies	semantics.py

2.5. Leftover Account Cleaner

Command	**./minus_operation.sh**
Dependencies	my_apps.py. Please refer to *software_manual* for detailed description of my_apps.py. awk dos2unix
Packages	lai_web

Command	**./lac_batch.sh [<gmail> <gpassword> minus** **| <./minus/laXX> install or ./batch.sh install** **| <package-name> <app-email> <app-password> clean or** **| ./single.sh or ./batch.sh <delete> - To delete apps** **| pull** **| uninstall or ./single.sh uninstall** **| <./minus/laXX> remove]**

Dependencies	minus_operation.sh gsplit store.py dos2unix adb grep lac_adf.sh cleanup.py
Packages	lai_web lai