Fake Conad Plus App: A Promise of 10× Loyalty Points Conceals Android Malware

By da Andrea Draghetti

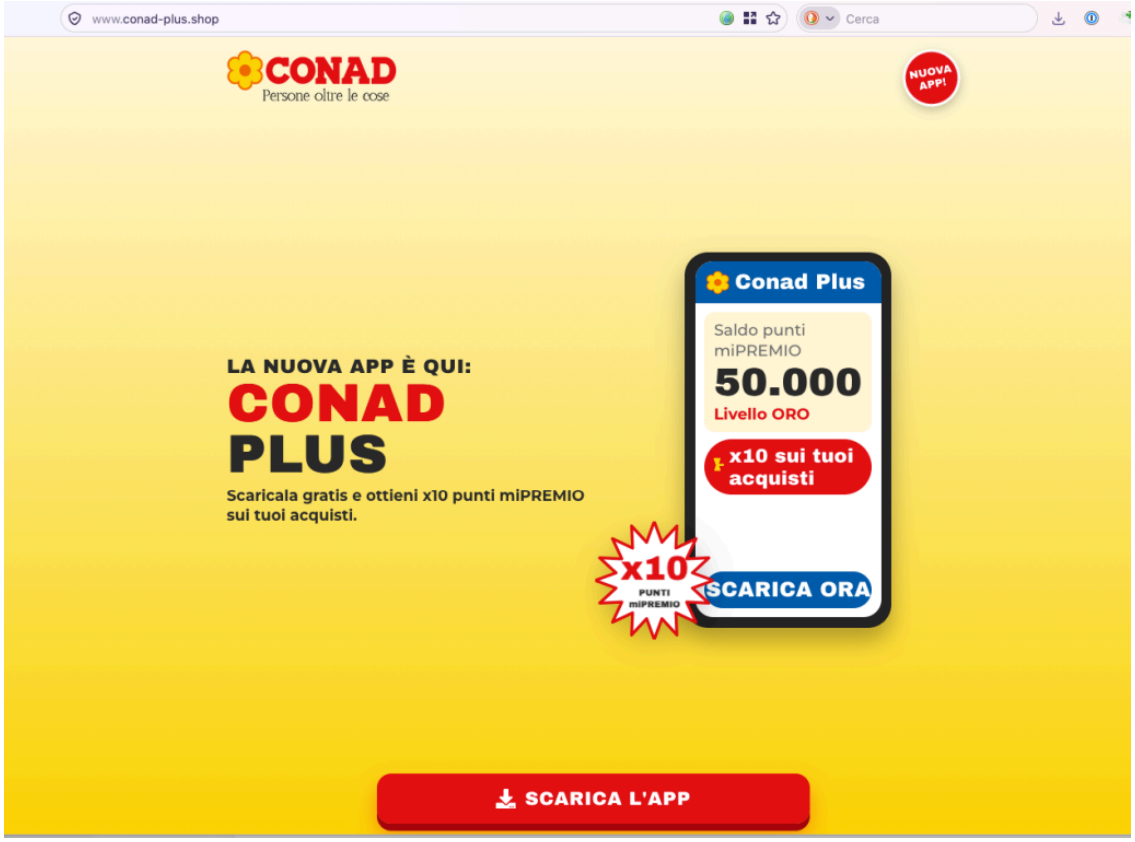
Published: 2026-11-18 · Archived: 2026-11-20 02:00:52 UTC

A fraudulent campaign targeting users in Italy impersonates Conad, promises to multiply miPREMIO loyalty points by ten, and invites users to download an application called **Conad Plus**. Behind the branding and the promised reward lies an installation chain that can give criminals control of an Android phone.

D3Lab identified `conad-plus[.]shop` through its **Brand Monitor** service, which detects domains that could be used to impersonate monitored brands.

The website reproduces Conad's logo, colors, and loyalty program references. It announces "The new app is here: Conad Plus" and offers a free download to earn 10× miPREMIO points. The download button leads to an Android App named `Apri_me.apk`, distributed directly from the website.

From a Promotion to Control of the Phone



The distribution page uses Conad branding and the promise of loyalty rewards to encourage installation.

Downloading the file alone does not give the malware control of the device. The dangerous steps are installing it and then approving requests to install an additional component and enable Accessibility services. The first APK acts as an intermediary: its code retrieves the resources needed to install the second component, which contains the remote-control functionality.

What Is at Risk for the Victim?

The analyzed application can send the command-and-control server **screenshots, text intercepted while applications are being used, information entered into forms displayed over other apps, an inventory of installed applications, and device information**, including the model, Android version, language, and battery status.

Criminals can also issue commands to interact with the interface, launch applications, and display deceptive screens over legitimate ones. The risk therefore includes both information theft and actions performed on the phone through applications already installed, including those handling personal or financial information. Success depends on the permissions granted, the Android version, and the protections implemented by individual applications.

The Koktevrat Malware

The Conad Plus campaign's payload is **Koktevrat**, the Android RAT described in CERT Orange Polska's analysis of the fake MandatGO application. We directly compared the reference samples with those from the Conad campaign. They share the DGA algorithm and seed, 46 commands, keylogging and form-data collection structures, and a byte-identical fake update page. This comparison establishes the malware family. Configuration and functionality differences distinguish the two variants, without establishing that they share the same operators.

The First APK: A Hidden Downloader

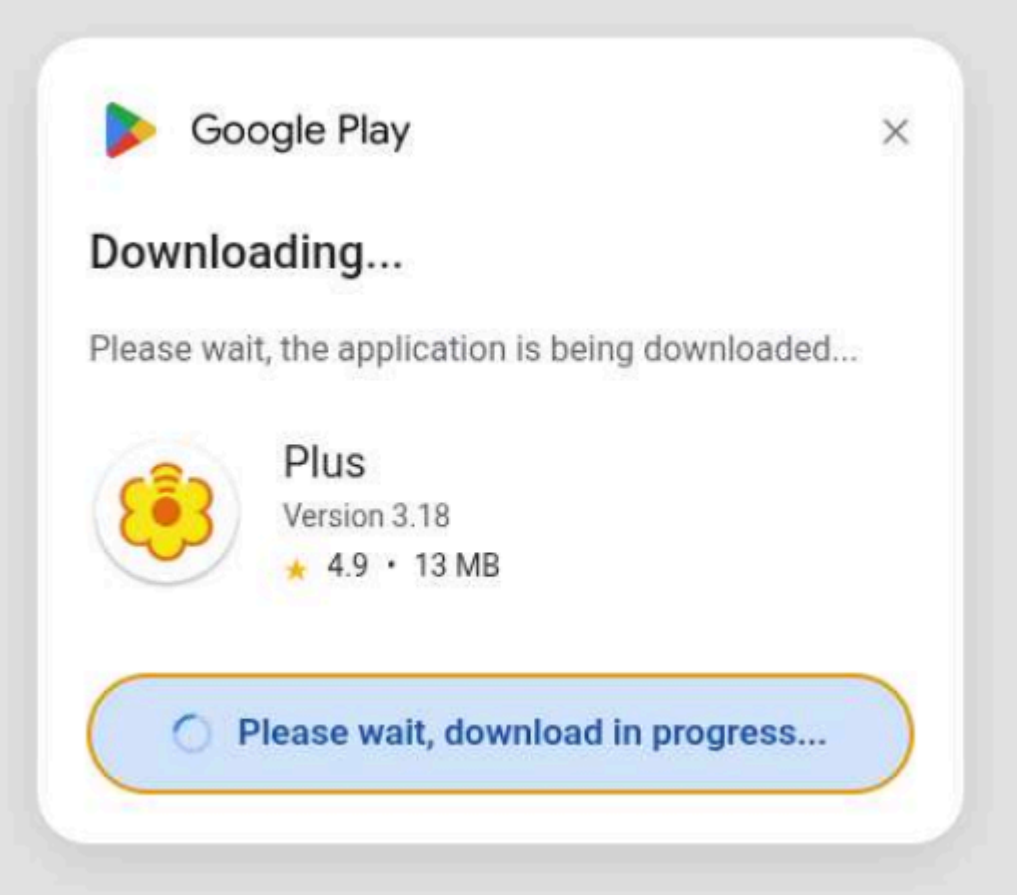
`Apri_me.apk` presents itself as Conad Plus and uses the package name `com.wehe.piteca`. This outer layer does not expose the RAT's operational code in readable form. Its Application class, `net.medal.among.Qquizsame`, initiates the retrieval of an asset hidden beneath a very long path containing Unicode components. The asset's final filename is `JLOEEFh.jpg`.

The image extension is misleading: the content is encrypted with **RC4 using the ASCII key** `ymn`. Decryption produces a ZIP archive containing a dynamically loaded DEX. This code implements the actual downloader and handles installation of the next component.

The downloader separately retrieves a password from a PHP endpoint and a password-protected ZIP archive, both hosted on `conad-plus[.]shop`. It then uses the password to extract the APK for installation. Separating the password, archive, and initial application adds a network dependency and prevents the full payload from being recovered by scanning the outer APK alone.

The sample declares `REQUEST_INSTALL_PACKAGES`, asks the user to allow installation from that source, and uses Android's standard installation flow. It also checks for the expected package and its Accessibility service. We did not identify silent installation or an exploit that automatically grants these permissions.

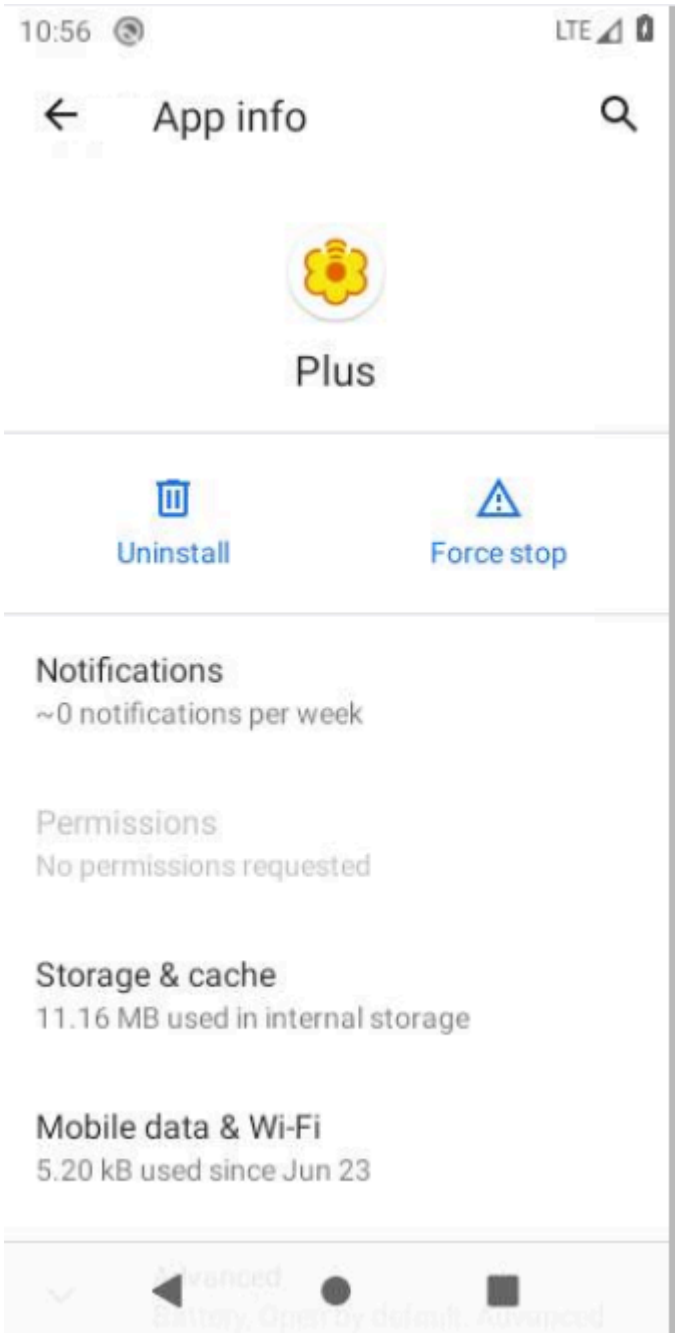
The second APK available for analysis, `cowaf.apk`, matches the package and components expected by the downloader. Code and local evidence support this relationship; no hash comparison with a remote response was performed to establish the exact bytes delivered by the website at a particular time.



The downloader imitates Google Play, displaying a rating, download size, and version 3.18.

The Second APK: The RAT and Its Obfuscation

`cowaf.apk` uses the package name `net.cowaf.xixibawo`, the display name “Plus,” and the declared version `3.19`. This component also conceals its operational code inside an encrypted asset. Its Application class, `net.security.canvas.Cproofancient`, initiates the loading of `PUOPNcZ.otf`, decrypted with **RC4 using the ASCII key** `cxMDYK`. The result is another ZIP container holding a DEX that implements Koktevrat.



Android's App info page identifies the installed payload as "Plus."

The reconstructed chain is: **Initial APK → RC4 asset → Downloader DEX → Remote password and ZIP → RAT APK → Second RC4 asset → Koktevrat DEX → C2 configuration**.

The SHA-256 hashes of the decrypted operational DEX files identify the code independently of its APK container:

Component	DEX SHA-256
Downloader	`e7b859f5e6560e441a00d18e03531f2e02532298246832574bd6220fa9b1942a`
Koktevrat	`21669e7c60f78a90c10c37b8c741e14efc0820c5e51725094dcb8b86767c6dca`

Obfuscation combines uninformative class and method names, reflection, misleading resource extensions, and dynamic code loading. The APKs also contain ZIP and Manifest anomalies, very long paths, and filler files. These techniques can hinder extraction and parsing with conventional tools. The C2 configuration adds another layer: it is encrypted with **AES-128-CBC** and Base64-encoded, with the IV prepended to the encrypted data and the key `7rnomEnDwJ7kFnIG`.

Permissions and Sensitive Services

The RAT declares `INTERNET`, `ACCESS_NETWORK_STATE`, and `QUERY_ALL_PACKAGES` to communicate and obtain visibility into installed applications. `RECEIVE_BOOT_COMPLETED`, `WAKE_LOCK`, foreground service permissions, and `REQUEST_IGNORE_BATTERY_OPTIMIZATIONS` support background execution and resumption of activity. It also declares `POST_NOTIFICATIONS`, `WRITE_SETTINGS`, and `ACCESS_NOTIFICATION_POLICY`. Manifest declarations do not mean that special access has actually been granted.

The central component is the Accessibility service `net.cowaf.xixibawo.KxlMliyRuy`. Its configuration requests all event types and declares the ability to retrieve window content, perform gestures, and take screenshots. `BIND_ACCESSIBILITY_SERVICE` protects the service's binding to the system; activation still requires user action in Settings.

The application also includes the notification listener `net.cowaf.xixibawo.KqnhHerqVy`, whose code can dismiss notifications associated with Google and Settings. This behavior does not establish general collection of notification content.

Commands and Data Collection

We identified **60 commands** in the Conad variant's code. The C2 service `QndglXgwfZk` handles communication and forwards instructions to execution components, including the Accessibility service. The table summarizes the main functional groups observed; it does not imply dynamic validation of every command.

Commands	Function in the Conad Sample
`get_apps`	Enumerate installed applications
`launch_app`	Launch an application
`click`, `swipe`, `now_swipe`, `key_press`	Interact with the interface through taps, gestures, and input actions
`action_home`, `action_back`, `action_recents`	Navigate using Android global actions
`set_text`, `clear_input`	Insert or clear text in interface fields
`clipboard`, `clear_clipboard`	Write text to the clipboard or clear its contents

Commands	Function in the Conad Sample
`inject_overlay`, `disable_overlay`	Enable or remove interfaces displayed over applications
`lock_screen`, `unlock_screen`	Lock-screen and unlock-interaction routines; these do not establish a PIN bypass
`olop_maybe_set`, `olop_app_set`, `olop_all_set`	Configure keylogging and its scope
`show_klock`, `hide_klock`, `Show_update_screen`, `hide_update_screen`	Display or remove cover screens
`protect_on`, `protect_off`, `gp_off`	Anti-removal logic and attempts to interfere with Google Play Protect
`proxy_open`, `proxy_data`, `proxy_close`	Open, relay data through, and close TCP connections via the device

Keylogging records contain the event type, package name, text, and timestamp. The buffer holds up to 300 records, with each record's text limited to 300 characters. Screenshots are JPEG-encoded with quality set to 50, then Base64-encoded for transmission over WebSocket. Data collected from overlay forms uses `overlay_data` messages, with a queue of up to 200 records.

Overlay templates and their associated application packages can be supplied by the server as XML, HTML, and configuration data. A complete target list therefore cannot be reconstructed from the APK alone. The `gp_off` command implements functionality intended to disable Play Protect through interface interaction, but its presence does not demonstrate success on every device.

One important difference from the MandatGO sample is that **the Conad variant does not contain** `READ_SMS` **or** `get_sms`. We therefore do not attribute the reference sample's direct SMS-reading capability to this variant. Similarly, the verified `clipboard` command writes to the clipboard; it does not establish clipboard reading or exfiltration.

WebSocket, Firebase, and Persistence

The static C2 is `wss://it-conad[.]sbs`, using port 443 and path `/`. The malware exchanges JSON messages. Commands, screenshots, intercepted text, application inventories, and form data travel over the WebSocket channel.

Firebase Cloud Messaging, associated with the project `tets12-a3f9a`, provides signaling. The service `MjncXdwjxNmvi` handles messages with `data.action` set to `connect` or `disconnect`. The FCM token is sent to the C2 in a message of type `fcm_token`, associating the installation with the push channel. The code also schedules a heartbeat through AlarmManager every **14,400,000 milliseconds, approximately four hours**.

The DGA: Date-Based Alternative Domains

Alongside its static C2, Koktevrat uses a **Domain Generation Algorithm** that produces two domains per period: one under `.icu` and one under `.cfd`. The reconstructed seed is `x7F9kQ2mZ8vB4nP1sD6gH3jL5rT0wY`, identical to the seed in the MandatGO reference sample.

The algorithm uses the UTC date and divides each month into three windows: days **1–9**, **10–19**, and **20–month end**, indexed as 0, 1, and 2 respectively. These periods are therefore not always exactly ten days long.

For each TLD, it constructs the string `seed|year|month|window_index|tld`, with the month represented as a number from 1 to 12. It computes SHA-256 once, then hashes the resulting digest bytes another 776 times, for **777 SHA-256 applications in total**. The first byte determines the hostname label's length, between 8 and 15 characters. The first character is a lowercase letter; subsequent characters are lowercase letters or digits, selected from subsequent digest bytes using modulo 26 and modulo 36 operations.

For November 10–19, 2026, the algorithm produces `nq9y7o5gd1zh[.]icu` and `z4t85uapc9skbvq[.]cfd`, which are added to the server list with the `ws://` scheme. DNS checks reported on November 18 returned NXDOMAIN. These are code-derived domains, rather than alternative C2 servers observed in operation. For threat hunting, the seed, transformations, and time windows allow analysts to reconstruct candidates without assuming that all of them are registered or active.

Indicators of Compromise

The following indicators relate to the Conad campaign. Domains are defanged using `[.]`.

Type	Indicator	Role
Domain	`conad-plus[.]shop`	Distribution and downloader resources
Hostname	`www.conad-plus[.]shop`	Fake promotion page and initial distribution
C2	`wss://it-conad[.]sbs`	WebSocket command and data-transfer channel
Package	`com.wehe.piteca`	Initial APK
Package	`net.cowaf.xixibawo`	RAT payload

Type	Indicator	Role
SHA-256	`6ac3cc61aefeb0c6a4600e03c35f72ca702b3b297ec72d1d844f19db76b7db9a`	`Apri_me.apk`
SHA-256	`ecb4590d7093295f3c3ea13c74b8c6e0c96b4d12ac977d5e4e8b14dc3f7534ec`	`cowaf.apk`

The promise of loyalty points persuades users to install the component that puts their phone at risk. Anyone who installed the APK and enabled Accessibility should consider information displayed or typed while the malware was active potentially exposed and seek assistance to clean the device. Sensitive accounts should be checked from another trusted device.

Conad is the brand impersonated in this campaign and is not associated with its operators. This research was conducted for defensive purposes.

Source: https://www.d3lab.net/fake-conad-plus-app-a-promise-of-10x-loyalty-points-conceals-android-malware/