ANDROID MALWARE IN DONOT APT OPERATIONS - CYFIRMA

Archived: 2026-04-02 12:41:21 UTC

Published On : 2025-01-17

EXECUTIVE SUMMARY

The research team at CYFIRMA collected a sample attributed to the Indian APT group known as ‘DONOT’, which appears to serve Indian national interests, and additionally seems to have been designed for intelligence gathering against internal threats and uses an innocent customer engagement platform for malicious purposes.

INTRODUCTION

The application is named “Tanzeem” and “Tanzeem Update”, which in Urdu translates to “organization.” Terrorist organizations and several Indian law enforcement agencies use this term to refer to groups they are associated with, such as Jaish-e-Mohammad and Lashkar. We collected two samples at different times, one from October and another from December, and found both apps nearly identical, with only slight changes to the user interface.

Although the app is supposed to function as a chat application, it does not work once installed, shutting down after the necessary permissions are granted. The app's name suggests that it is designed to target specific individuals or groups both inside and outside the country.

TECHNICAL ANALYSIS

OneSignal is a popular platform that provides tools for sending push notifications, in-app messages, emails, and SMS, which are widely used in mobile and web applications. In this instance, however, we believe the library is being misused to push notifications containing phishing links, as we have observed the OneSignal library being used in both applications. The techniques employed are similar to those seen in other applications used by the group in the past, however, this is the first time we have observed this APT group utilizing it.

PROCESS OVERVIEW

Once installed, the app takes you to a landing page that says, "Tanzeem App," possibly referring to an application used by members of terrorist organizations.

Figure 1. Landing page of the application.

The application then loads the second page where the user is shown the fake chat functions:

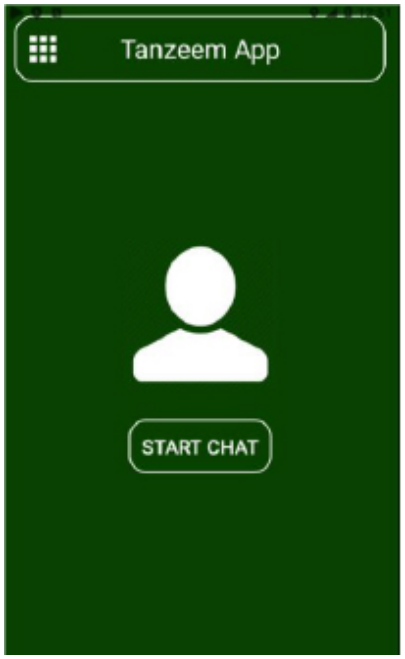


Figure 2. Fake chat page.

Upon clicking “START CHAT”, a pop-up message asks the user to turn on accessibility access for the Tanzeem App.

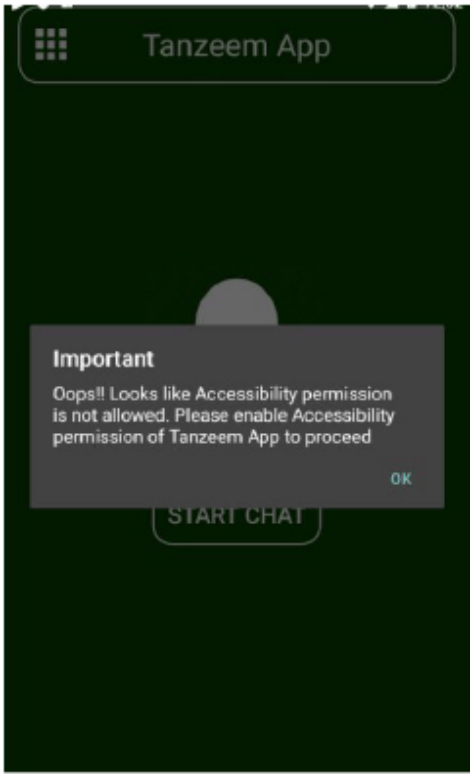


Figure 3. Pop up after clicking on ‘start chat’.

The user is then directed to the accessibility settings page.

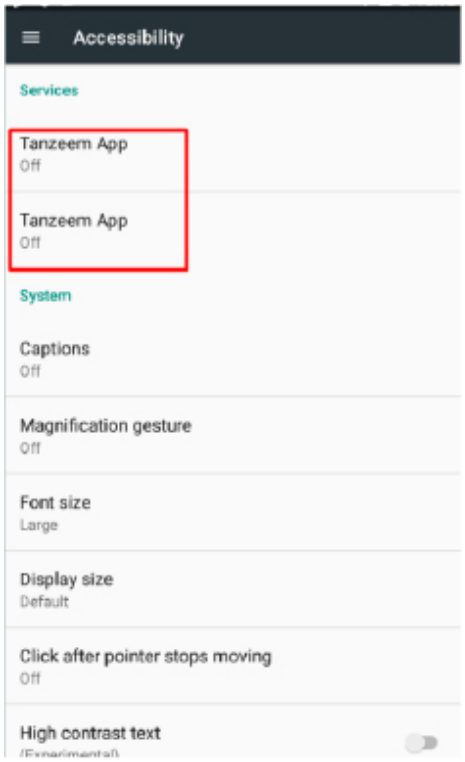


Figure 4. Accessibility setting once clicked on 'ok' on the pop-up.

The snippet below from another sample shows slight differences from the other applications, but the functions remain the same except for the color change.

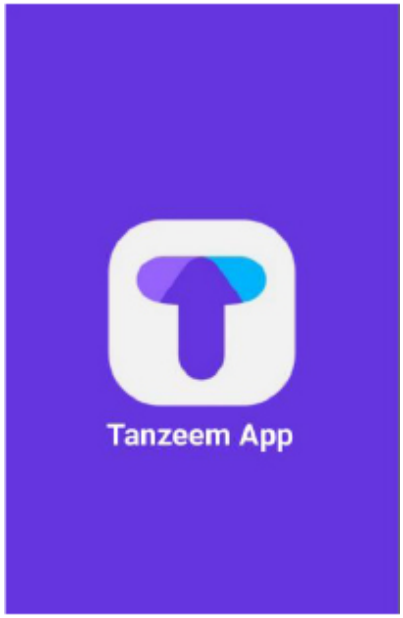


Figure 5. Second applicaion.

CODE OVERVIEW

The snippet is from the extracted Android Manifest file of the app.

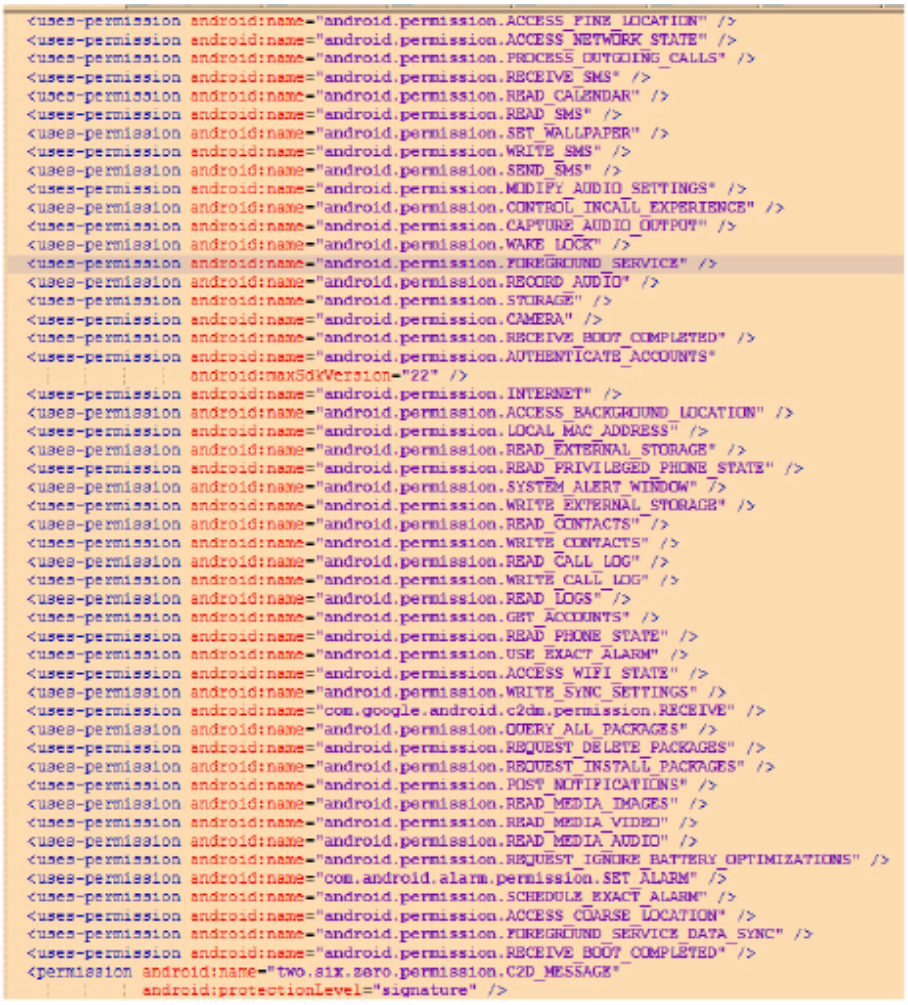


Figure 6. Snippet from Android Manifest file.

Below are a few dangerous permissions described that malicious Android app accesses:

Sr.no	Permissions	Descriptions
1.	READ_CALL_LOG	This permission enables threat actors to read and fetch call logs.
2.	READ_CONTACTS	Permission allows TA to read and fetch contacts.
3.	READ_EXTERNAL_STORAGE	Allows threat actors to explore and fetch data from the file manager.
4.	WRITE_EXTERNAL_STORAGE	Allows threat actors to delete and move files.
5.	READ_SMS	This allows attackers to delete and read outgoing and incoming SMSs
6.	STORAGE	This gives access to mobile internal storage to view and access files.
7.	ACCESS_FINE_LOCATION	Threat actors are able to extract precise locations and monitor the live movement of the device.

8.	GET_ACCOUNTS	This allows the threat actor to extract emails and usernames used for logging into various internet platforms.

The URL shown below serves as a command-and-control server for the app.

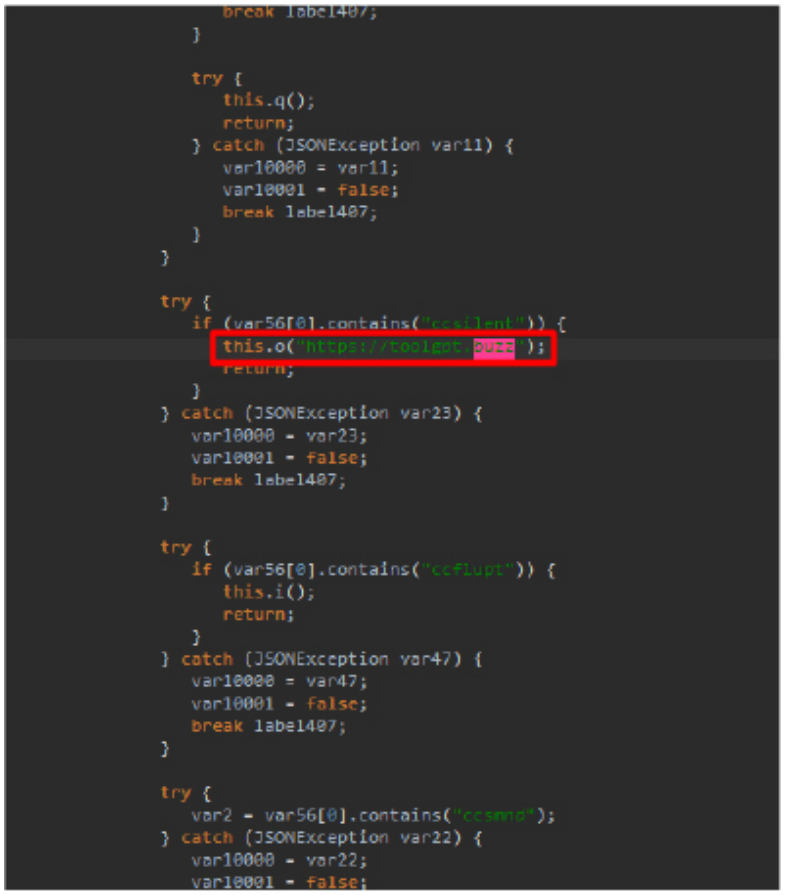


Figure 7. The Module is part of handling communication with the C2 server.

The snippet below shows the OneSignal library after decompiling the Android package. Another snippet displays the Appspot domains used for communication with the OneSignal library.

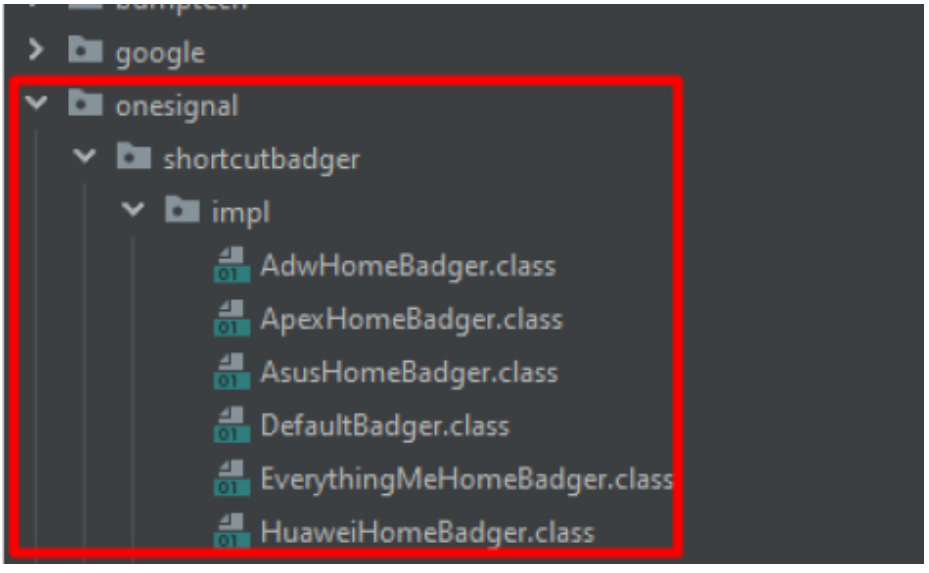


Figure 8. SignalOne library

Figure 9. Snippet from strings file.

This image below shows part of the module that fetches information about the permissions gained after application installation.

```
            var2 = true;
        }

        String var14;
        if (var2) {
            var14 = "enable";
        } else {
            var14 = "disable";
        }

        try {
            var16.put("accb", var34);
            var16.put("notif", var39);
            var16.put("sms", var7);
            var16.put("call", var8);
            var16.put("cont", var9);
            var16.put("storg", var10);
            var16.put("locs", var13);
            var16.put("gp", var41);
            var16.put("drwap", var40);
            var16.put("cmr", var14);
            var16.put("micr", var12);
            var15.put("did", b.q(this.getApplicationContext()));
            SharedPreferences var36 = this.getApplicationContext().getSharedPreferences("shared_pref", 0);
            var36.edit();
            var15.put("Key", b.L(var36.getString("fcm_token", "")));
            var15.put("perms", var16);
            c var37 = new c(this.getApplicationContext(), this);
            var37.l(var15.toString());
            this.moveTaskToBack(true);
            break label1250;
        } catch (Exception var18) {
            var10000 = var18;
            var10001 = false;
        }
    }

    Exception var35 = var10000;
    var35.printStackTrace();
}
```

Figure 10. The module handles the fetching of basic information from the device.

The below code handles accessibility for the application.

```
@SuppressLint({"NewApi"})
public class ChatUi extends h {
   public static final int I = 0;
   public Button C;
   public AlertDialog.Builder D;
   public a E;
   public k0 F;
   public Context G;
   public int H = 101;

   public final void B() {
      AlertDialog.Builder var2 = new AlertDialog.Builder(this);
      this.D = var2;
      StringBuilder var1 = b.j("Oops!! Looks like Accessibility permission is not allowed. Please enable Accessibility permission of ");
      var1.append(this.getResources().getString(2131689499));
      var1.append(" to proceed");
      var2.setMessage(var1.toString()).setCancelable(false).setPositiveButton("Ok", new c(this));
      AlertDialog var3 = this.D.create();
      var3.setTitle("Important");
      var3.setCancelable(false);
      var3.show();
   }

   public final void C() {
      AlertDialog.Builder var2 = new AlertDialog.Builder(this);
      this.D = var2;
      StringBuilder var1 = b.j("Please enable Draw Over Other apps permission of ");
      var1.append(this.getResources().getString(2131689502));
      var1.append(" to proceed");
      var2.setMessage(var1.toString()).setCancelable(false).setPositiveButton("Ok", new b(this));
      AlertDialog var3 = this.D.create();
      var3.setTitle("Important");
      var3.setCancelable(false);
      var3.show();
   }

   public final void onCreate(Bundle var1) {
      super.onCreate(var1);
      j.a(this);
      this.G = this.getApplicationContext();
      this.setContentView(2131427356);
      this.C = (Button)this.findViewById(2131231189);
      this.E = new a(this);
      this.G.getSharedPreferences("shared_pref", 0).edit();
      if (VERSION.SDK_INT >= 26) {
         this.G.startForegroundService(new Intent(this.G, TempServ.class));
      } else {
```

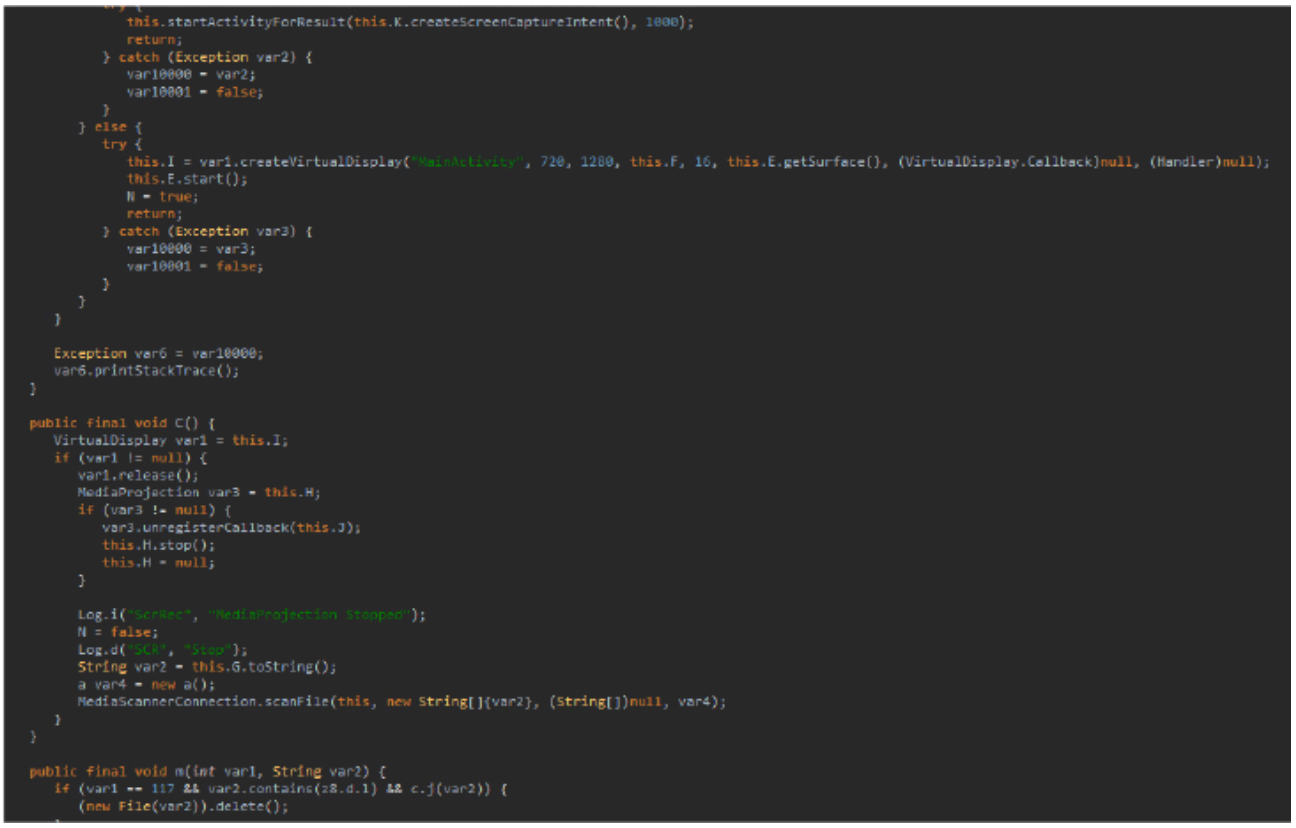
Figure 11. The module handles accessibility permissions for the application.

The code below helps applications handle permissions.

```
   // $FF: Couldn't be decompiled
}

public final void a(int var1) {
}

public final void onCreate(Bundle var1) {
   super.onCreate(var1);
   this.setContentView(2131427361);
   this.F = new e(this.getApplicationContext());
   Chamber_計算.getOnInstance(this.getApplicationContext());
   new f(this.getApplicationContext());
   ImageView var4 = (ImageView)this.findViewById(2131230982);
   this.D = (Button)this.findViewById(2131230832);
   this.C = new z8.a(this);
   new c(this.getApplicationContext(), this);
   this.D.setOnClickListener(new a(this));
   z8.a var5 = this.C;
   int var2 = d0.a.a(var5.a, "android.permission.READ_EXTERNAL_STORAGE");
   boolean var3 = false;
   boolean var7;
   if (var2 == 0 && d0.a.a(var5.a, "android.permission.WRITE_EXTERNAL_STORAGE") == 0 && d0.a.a(var5.a, "android.permission.READ_CONTACTS") == 0 && d0.a.a(var5.a, "android.permission.WRITE_CONTACTS
      var7 = true;
   } else {
      var7 = false;
   }

   if (!var7) {
      c0.a.c(this.C.a, new String[]{"android.permission.READ_EXTERNAL_STORAGE", "android.permission.WRITE_EXTERNAL_STORAGE", "android.permission.READ_CONTACTS", "android.permission.WRITE_CONTACTS",
   }

   this.getApplicationContext().startService(new Intent(this.getApplicationContext(), TokSer_ｻｰﾋﾞｽ.class));
   if (!this.F.h().equals("")) {
      this.B();
   }

   e var6 = b.f;
   if (!"google".equals(Build.BRAND.toLowerCase())) {
      var7 = var3;
      if (Secure.getInt(this.getApplicationContext().getContentResolver(), "adb_enabled", 0) == 1) {
         var7 = true;
      }

      if (!var7) {
         this.getApplicationContext().startService(new Intent(this.getApplicationContext(), TokSer_ｻｰﾋﾞｽ.class));
```

Figure 12. The module that handles permissions.

The snippet shows a code from the module that helps applications record screens.

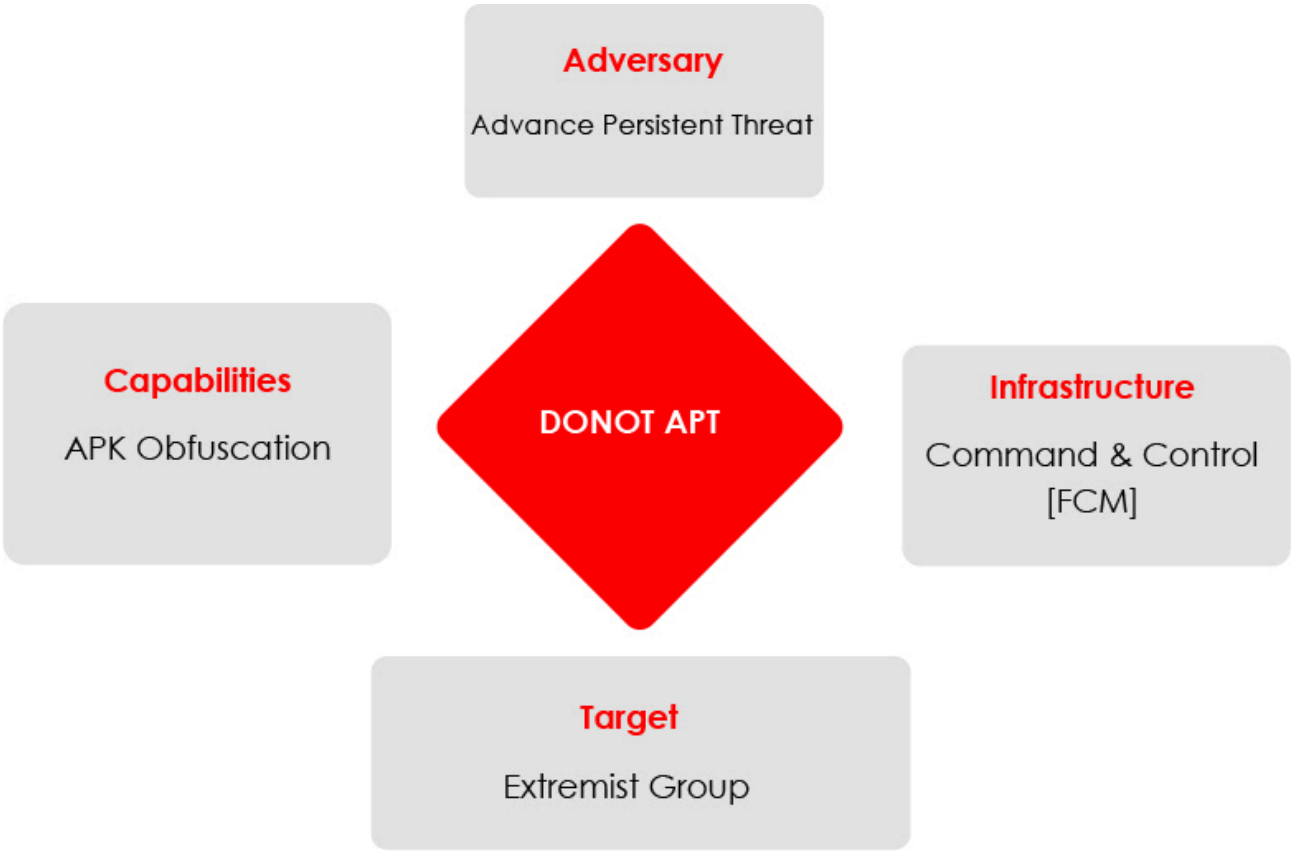


```
            this.startActivityForResult(this.K.createScreenCaptureIntent(), 1000);
            return;
         } catch (Exception var2) {
            var10000 = var2;
            var10001 = false;
         }
      } else {
         try {
            this.I = var1.createVirtualDisplay("MainActivity", 720, 1280, this.F, 16, this.E.getSurface(), (VirtualDisplay.Callback)null, (Handler)null);
            this.E.start();
            N = true;
            return;
         } catch (Exception var3) {
            var10000 = var3;
            var10001 = false;
         }
      }
   }

   Exception var6 = var10000;
   var6.printStackTrace();
}

public final void C() {
   VirtualDisplay var1 = this.I;
   if (var1 != null) {
      var1.release();
      MediaProjection var3 = this.H;
      if (var3 != null) {
         var3.unregisterCallback(this.J);
         this.H.stop();
         this.H = null;
      }

      Log.i("ScrRec", "MediaProjection Stopped");
      N = false;
      Log.d("SCR", "Stop");
      String var2 = this.G.toString();
      a var4 = new a();
      MediaScannerConnection.scanFile(this, new String[]{var2}, (String[])null, var4);
   }
}

public final void m(int var1, String var2) {
   if (var1 == 117 && var2.contains(z8.d.1) && c.j(var2)) {
      (new File(var2)).delete();
```

EXTERNAL THREAT LANDSCAPE MANAGEMENT

The ongoing efforts by the notorious DONOT APT extend beyond gathering intelligence on internal threats; they have also targeted various organizations in South Asia to assist India with strategic intelligence collection. The collected samples reveal a new tactic involving push notifications that encourage users to install additional Android malware, ensuring the persistence of the malware on the device. This tactic enhances the malware's ability to remain active on the targeted device, indicating the threat group's evolving intentions to continue participating in intelligence gathering for national interests. The group's relentless efforts suggest that their operations are far from over.

Diamond Model



MITRE AT&CK FRAMEWORK

MITRE ATT&CK framework for Android malware payload in a table format

Tactics	Technique ID	Description
Defense Evasion	T1406 – Obfuscated Files or Information	Uses obfuscation techniques to hide malicious code within the APK.
Discovery	T1420 – File and Directory Discovery	Enumerate files and directories on the device to locate valuable information.
Credential Access	T1417 – Input Capture	Captures keystrokes to steal sensitive credentials like usernames and passwords.
Discovery	T1426 – System Information Discovery	Collects device information, such as device model, and user details.
Collection	T1533 – Data from Local System	Extracts data such as contacts, messages, photos, and videos from the infected device.
Collection	T1513 – Screen Capture	Takes screenshots and records video of the infected device to capture sensitive information.
Exfiltration	T1646 – Exfiltration Over C2 Channel	Sends stolen data (e.g., contacts, messages, credentials) to the C2 server.

INDICATORS OF COMPROMISES

Indicator	Type	Remarks
8689D59AAC223219E0FDB7886BE289A9536817EB6711089B5DD099A1E580F8E4	SHA-256	File Hash
D512664DF24B5F8A2B1211D240E3E767F5DD06809BB67AFA367CDC06E2366AEC	SHA-256	File Hash
toolgpt[.]buzz	Domain	Command and Control
Updash[.]info	domain	Command & Control
Solarradiationneutron[.]appspot[.]com	Sub-domain	Command & Control
saturn789454[.]appspot[.]com	Sub-domain	Command & Control

CONCLUSION

The cybersecurity community is well aware that the DONOT group is actively targeting organizations and individuals across the South Asia region. The group persistently employs similar techniques in their Android malware. Recently, we observed the implementation of OneSignal in their latest attack, further demonstrating their efforts to maintain persistence. As the group continues to evolve, we can expect further modifications in their tactics, aiming to strengthen their ability to maintain persistence in future cyberattacks using Android malware.
