DEKRA

App
com.protectstar.antivirus
MASA L1
2025-03-17

Summary of the application

Target of Evaluation	**App Version: 2.2**
Model and/or type reference	**com.protectstar.antivirus**
Other identification of the product	**com.protectstar.antivirus**
Features	
Manufacturer	**Protectstar Inc.**
Test Method Requested	**Security evaluation based on limited set of evaluation procedures from OWASP Mobile Application Security Verification Standard established by ADA.**
Validation Type	**Level 1 - Verified Self**
Validated By	**Jose María Santos López – Cybersecurity Engineer**
Platform	**Android**
Date of Issue	**2025-03-17**

Nomenclature

Below you can find the verification type considered for MASA L1 as well as the description of each of them.

Verification Type

- **Self Declare:** developer provides Yes/No response to verify they meet the requirement
- **Documentation:** developer provides artifacts / evidence to verify they meet the requirement
- **NMI:** lab provides output from static analysis for developers to respond to
- **Scan Verified:** lab performs static analysis against fully automatable requirements to assign a Pass / Fail for each requirement
    - For scan failures the developer is expected to resolve the issue or provide a written justification explaining how they meet the requirement including supporting evidence such as scan output, screenshots or supporting documentation.

Summary Table

Requirement	Description	Validation Type	Status	Dev Action
MSTG-STORAGE-1	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys	Self Declare	Pass	N
MSTG-STORAGE-2	No sensitive data should be stored outside of the app container or system credential storage facilities.	Self Declare	Pass	N
MSTG-STORAGE-3	No sensitive data is written to application logs.	NMI	Pass	N
MSTG-STORAGE-5	The keyboard cache is disabled on text inputs that process sensitive	Scan Verified	Pass	N

	data.			
MSTG-STORAGE-7	No sensitive data, such as passwords or pins, is exposed through the user interface.	NMI	Pass	N
MSTG-STORAGE-12	The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.	Self Declare	Pass	N
MSTG-CRYPTO-1	The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.	Scan Verified	Pass	N
MSTG-CRYPTO-2	The app uses proven implementations of cryptographic primitives.	Scan Verified	Pass	N
MSTG-CRYPTO-3	The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.	Scan Verified	Pass	N
MSTG-CRYPTO-4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	Scan Verified	Pass	N

MSTG-CRYPTO-5	The app does not re-use the same cryptographic key for multiple purposes.	Self Declare	Pass	N
MSTG-CRYPTO-6	All random values are generated using a sufficiently secure random number generator.	Scan Verified	Pass	N
MSTG-AUTH-1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	Self Declare	Pass	N
MSTG-AUTH-2	If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user`s credentials.	Self Declare	Pass	N
MSTG-AUTH-3	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.	Self Declare	Pass	N
MSTG-AUTH-4	The remote endpoint terminates the existing session when the user logs	Self Declare	Pass	N

	out.			
MSTG-AUTH-5	A password policy exists and is enforced at the remote endpoint.	Self Declare	Pass	N
MSTG-AUTH-6	The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.	Self Declare	Pass	N
MSTG-AUTH-7	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.	Self Declare	Pass	N
MSTG-NETWORK-1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	Scan Verified	Pass	N
MSTG-NETWORK-2	The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.	Scan Verified	Pass	N
MSTG-NETWORK-3	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established.	Scan Verified	Pass	N

MSTG-PLATFORM-1	The app only requests the minimum set of permissions necessary.	Scan Verified	Pass	N
MSTG-PLATFORM-2	All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.	Scan Verified	Pass	N
MSTG-PLATFORM-3	The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.	Scan Verified	Pass	N
MSTG-PLATFORM-4	The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.	Self Declare	Pass	N
MSTG-CODE-1	The app is signed and provisioned with a valid certificate, of which the private key is properly protected.	Scan Verified	Pass	N
MSTG-CODE-2	The app has been built in release mode, with settings appropriate for a release build (e.g.	Scan Verified	Pass	N

	non-debuggable).			
MSTG-CODE-3	Debugging symbols have been removed from native binaries.	Scan Verified	Pass	N
MSTG-CODE-4	Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.	Scan Verified	Pass	N
MSTG-CODE-5	All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.	Self Declare	Pass	N
MSTG-CODE-9	Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.	Scan Verified	Pass	N

Self-Declare

MSTG-STORAGE-1

System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys

Does your app (or library/SDK) use cryptographic keys to encrypt all data that may be considered sensitive, such as PII?

A. Yes
B. **No**

Does your app (or library/SDK) use Android Keystore API to store user credentials?

A. Yes
B. **No**

Does your app (or library/SDK) use Android Keystore API to store cryptographic keys?

A. Yes
B. **No**

MSTG-STORAGE-2

No sensitive data should be stored outside of the app container or system credential storage facilities.

Does your application or library/SDK exclusively store sensitive data within the app container or use system credential storage facilities?

A. **Yes**
B. No

MSTG-STORAGE-12

The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.

Do you educate users about the types of personally identifiable information (PII) it processes, as well as security best practices they should follow when using the app?

A. **Yes**
B. No

Does your app (or library/SDK) provide a clear and easily accessible link to your privacy policy within the app?

A. **Yes**
B. No

MSTG-CRYPTO-5

The app does not re-use the same cryptographic key for multiple purposes.

Do you have mechanisms in place to audit and verify that each cryptographic key is used exclusively for its designated purpose within the app (or library/SDK) ?

A. **Yes**
B. No

Do you verify that each cryptographic key in your app (or library/SDK) is assigned a single, specific purpose to avoid key reuse?

A. **Yes**
B. No

MSTG-AUTH-1

If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.

Does your app (or library/SDK) implement authentication at the remote endpoint?

A. **Yes**
B. No
C. N/A (app does not have authentication)

MSTG-AUTH-2

If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user`s credentials.

Does your app`s (or library/SDK) remote endpoint use unique and unpredictable session identifiers to authenticate client requests without transmitting the user's credentials?

A. **Yes**
B. No
C. N/A (app does not have authentication)

MSTG-AUTH-3

If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.

If stateless token-based authentication is utilized by your application, does your app server (or library/SDK) provide a token protected using a secure algorithm such as HMAC-SHA256?

A. **Yes**
B. No
C. N/A (app does not have authentication or app does not use stateless token-based authentication)

MSTG-AUTH-4

The remote endpoint terminates the existing session when the user logs out.

Does your app (or library/SDK) terminate the existing session at the remote endpoint when the user logs out?

A. **Yes**
B. No
C. N/A (app does not have authentication)

MSTG-AUTH-5

A password policy exists and is enforced at the remote endpoint.

Does your app (or library/SDK) prevent users from setting passwords they have already used before at the remote endpoint?

A. **Yes**
B. No
C. N/A (app does not have authentication)

Does your app (or library/SDK) enforce a password policy (e.g., minimum length, complexity) on the server/backend side?

A. **Yes**

B. No
C. N/A (app does not have authentication or password is never sent to server (e.g. federated auth or zero-knowledge password proof))

MSTG-AUTH-6

The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.

Does your app (or library/SDK) implement a mechanism at the remote endpoint to protect against the submission of credentials an excessive number of times?

A. **Yes**
B. No
C. N/A (app does not have authentication or password is never sent to server (e.g. federated auth or zero-knowledge password proof))

MSTG-AUTH-7

Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.

Does your app (or library/SDK) implement a mechanism at the remote endpoint (server-side) to automatically invalidate user sessions after a predefined period of inactivity?

A. **Yes**
B. No
C. N/A (app does not have authentication)

Do access tokens used for authentication and authorization within your app (or library/SDK) have a set expiration time, after which the server will reject them as invalid?

A. **Yes**
B. No
C. N/A (app does not have authentication)

MSTG-PLATFORM-4

The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.

Does your app (or library/SDK) expose sensitive functionality through inter-process communication (IPC) mechanisms?

A. Yes
B. **No**

Does your app (or library/SDK) have security measures like access controls, data validation, and encryption in place to protect sensitive functionality exposed through IPC mechanisms in your app or library/SDK?

A. **Yes**
B. No
C. N/A (app does not have IPC mechanisms)

MSTG-CODE-5

All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.

Does your app (or library/SDK) use third party libraries?

A. **Yes**
B. No

1. Provide a list of 3P libraries to labs

net.dongliu:apk-parser:2.6.10
com.squareup.okhttp3:okhttp:4.11.0
com.zsoltsafrany:needle:1.0.0
org.apache.tika:tika-core:2.1.0
com.github.PhilJay:MPAndroidChart:v3.1.0
com.squareup.retrofit2:retrofit:2.9.0
com.squareup.retrofit2:converter-gson:2.9.0
com.airbnb.android:lottie:6.6.3
org.greenrobot:eventbus:3.3.1
com.sothree.slidinguppanel:library:3.4.0
com.ogaclejapan.smarttablayout:library:2.0.0@aar

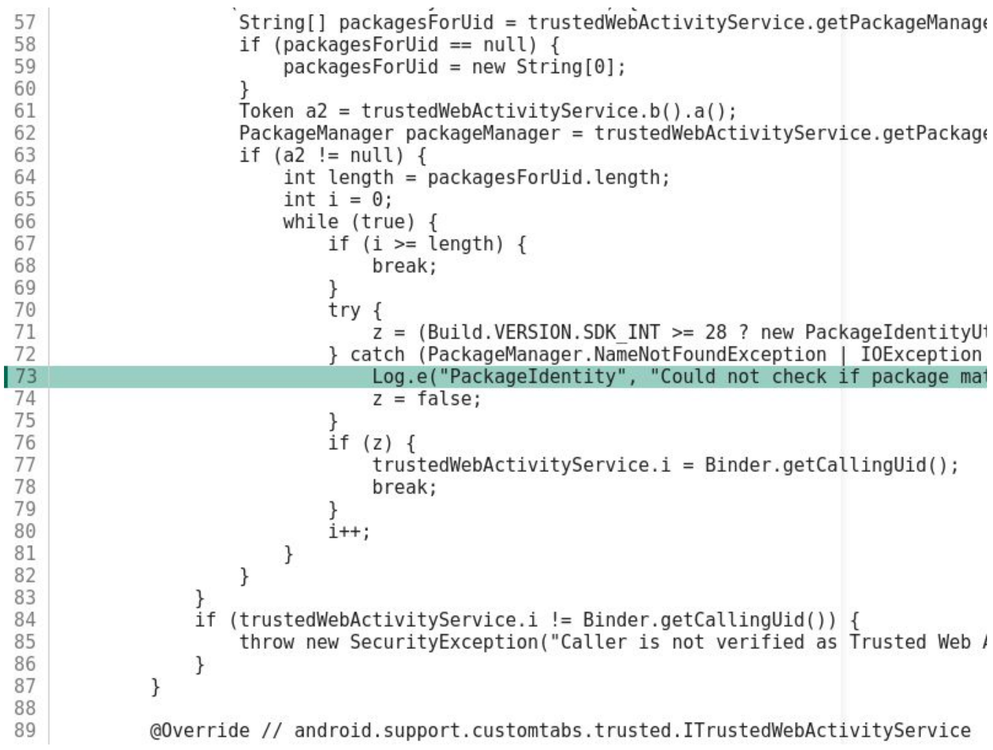
com.futuremind.recyclerfastscroll:fastscroll:0.2.5
com.github.bumptech.glide:glide:4.14.2
commons-codec:commons-codec:1.15
commons-lang:commons-lang:2.6
com.android.volley:volley:1.2.1
com.android.billingclient:billing:7.0.0
com.google.code.gson:gson:2.10.1
com.google.android.play:review:2.0.2
com.google.android.flexbox:flexbox:3.0.0
com.google.android.material:material:1.12.0
com.google.firebase:firebase-messaging:24.1.0
com.google.guava:guava:30.1.1-android
androidx.browser:browser:1.8.0
androidx.cardview:cardview:1.0.0
androidx.work:work-runtime:2.9.1
androidx.appcompat:appcompat:1.7.0
androidx.preference:preference:1.2.1
androidx.recyclerview:recyclerview:1.3.2
androidx.concurrent:concurrent-futures:1.2.0
androidx.swiperefreshlayout:swiperefreshlayout:1.1.0
androidx.lifecycle:lifecycle-extensions:2.2.0
androidx.annotation:annotation:1.8.2

NMI

MSTG-STORAGE-3

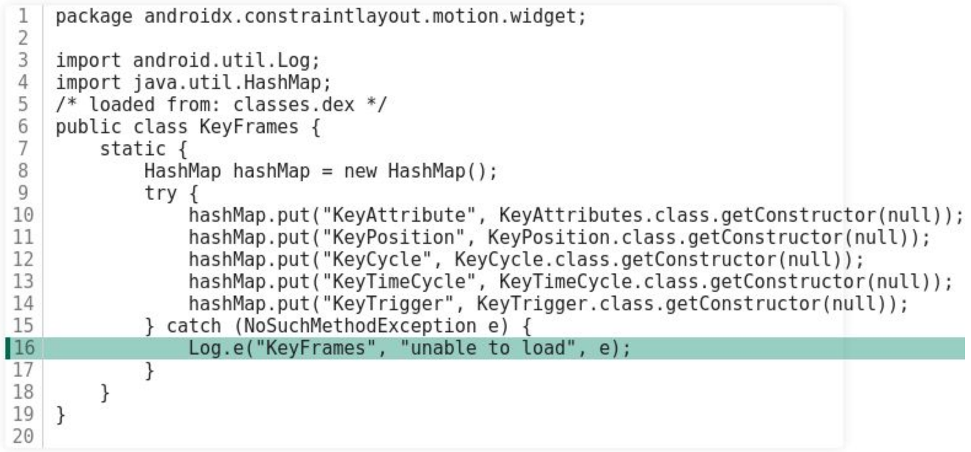
No sensitive data is written to application logs.

Finding 1



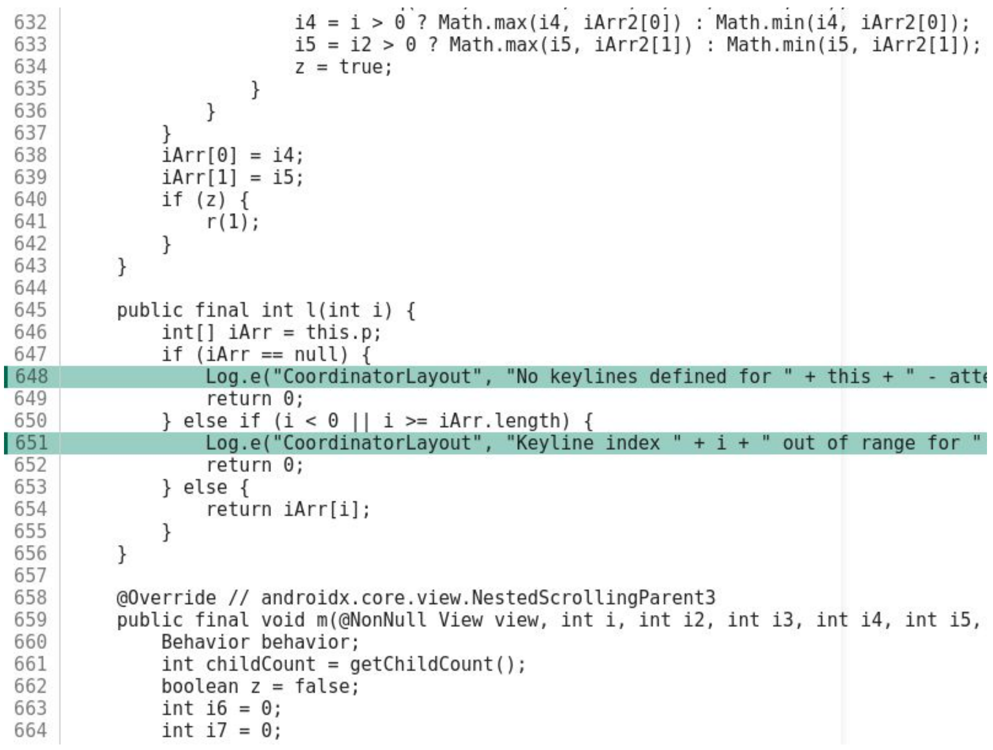
```
                String[] packagesForUid = trustedWebActivityService.getPackageManag
                if (packagesForUid == null) {
                    packagesForUid = new String[0];
                }
                Token a2 = trustedWebActivityService.b().a();
                PackageManager packageManager = trustedWebActivityService.getPackage
                if (a2 != null) {
                    int length = packagesForUid.length;
                    int i = 0;
                    while (true) {
                        if (i >= length) {
                            break;
                        }
                        try {
                            z = (Build.VERSION.SDK_INT >= 28 ? new PackageIdentityUt
                        } catch (PackageManager.NameNotFoundException | IOException
                            Log.e("PackageIdentity", "Could not check if package mat
                            z = false;
                        }
                        if (z) {
                            trustedWebActivityService.i = Binder.getCallingUid();
                            break;
                        }
                        i++;
                    }
                }
            }
            if (trustedWebActivityService.i != Binder.getCallingUid()) {
                throw new SecurityException("Caller is not verified as Trusted Web A
            }
        }

        @Override // android.support.customtabs.trusted.ITrustedWebActivityService
```

Finding 2



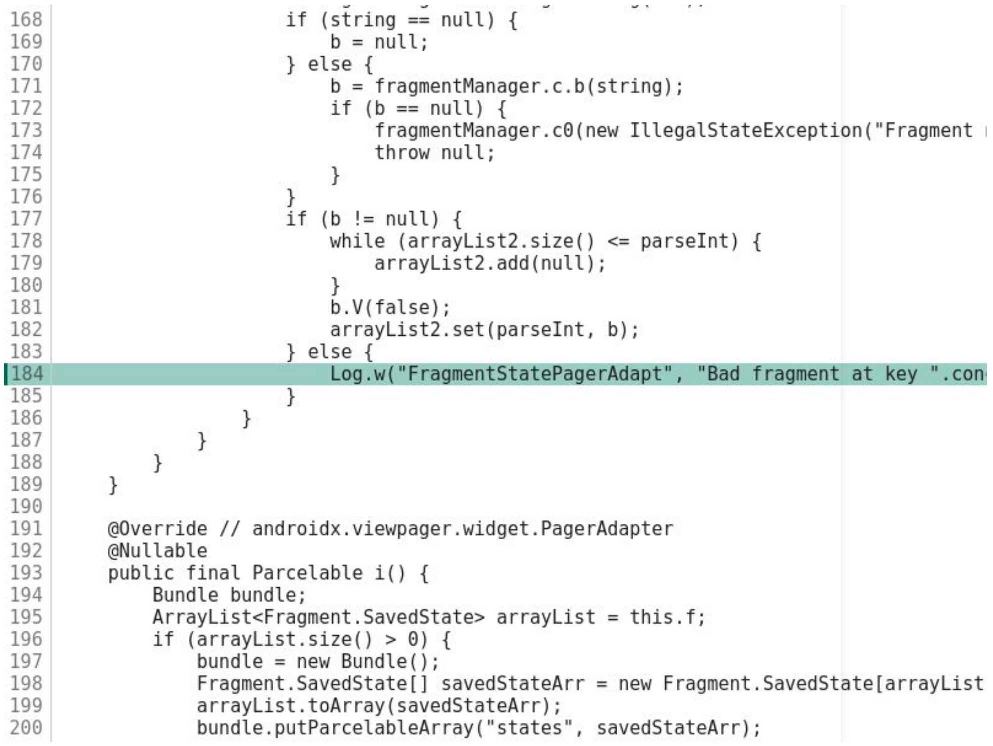
```
package androidx.constraintlayout.motion.widget;

import android.util.Log;
import java.util.HashMap;
/* loaded from: classes.dex */
public class KeyFrames {
    static {
        HashMap hashMap = new HashMap();
        try {
            hashMap.put("KeyAttribute", KeyAttributes.class.getConstructor(null));
            hashMap.put("KeyPosition", KeyPosition.class.getConstructor(null));
            hashMap.put("KeyCycle", KeyCycle.class.getConstructor(null));
            hashMap.put("KeyTimeCycle", KeyTimeCycle.class.getConstructor(null));
            hashMap.put("KeyTrigger", KeyTrigger.class.getConstructor(null));
        } catch (NoSuchMethodException e) {
            Log.e("KeyFrames", "unable to load", e);
        }
    }
}

```

Finding 3



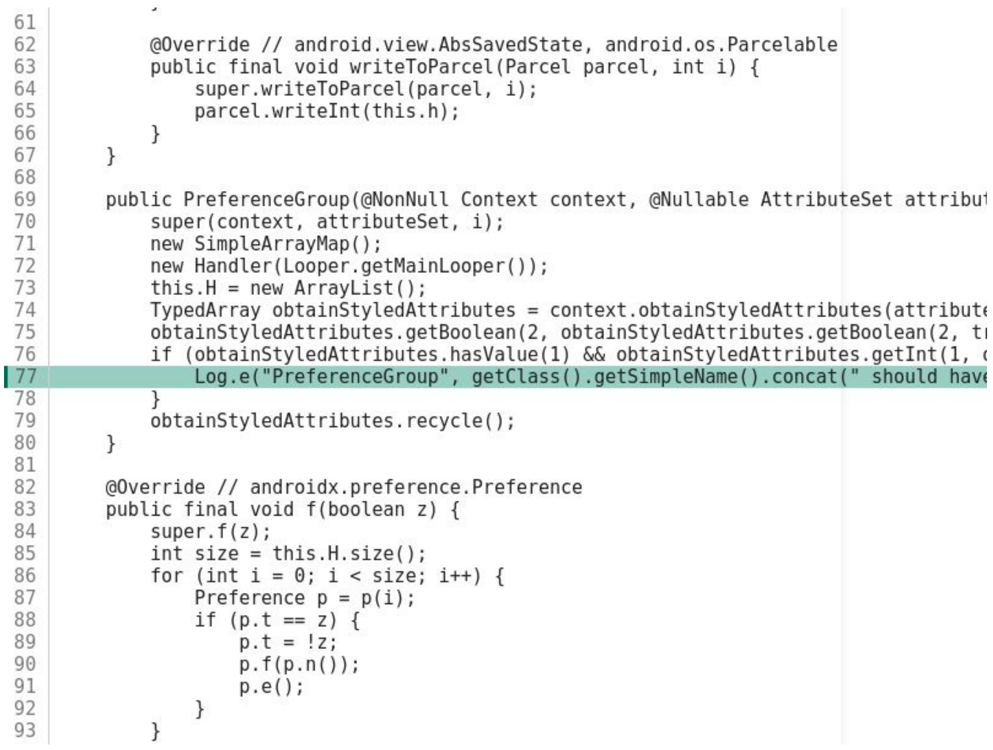
```
                    i4 = i > 0 ? Math.max(i4, iArr2[0]) : Math.min(i4, iArr2[0]);
                    i5 = i2 > 0 ? Math.max(i5, iArr2[1]) : Math.min(i5, iArr2[1]);
                    z = true;
                }
            }
        }
        iArr[0] = i4;
        iArr[1] = i5;
        if (z) {
            r(1);
        }
    }

    public final int l(int i) {
        int[] iArr = this.p;
        if (iArr == null) {
            Log.e("CoordinatorLayout", "No keylines defined for " + this + " - atte
            return 0;
        } else if (i < 0 || i >= iArr.length) {
            Log.e("CoordinatorLayout", "Keyline index " + i + " out of range for "
            return 0;
        } else {
            return iArr[i];
        }
    }

    @Override // androidx.core.view.NestedScrollingParent3
    public final void m(@NonNull View view, int i, int i2, int i3, int i4, int i5,
        Behavior behavior;
        int childCount = getChildCount();
        boolean z = false;
        int i6 = 0;
        int i7 = 0;
```

Finding 4



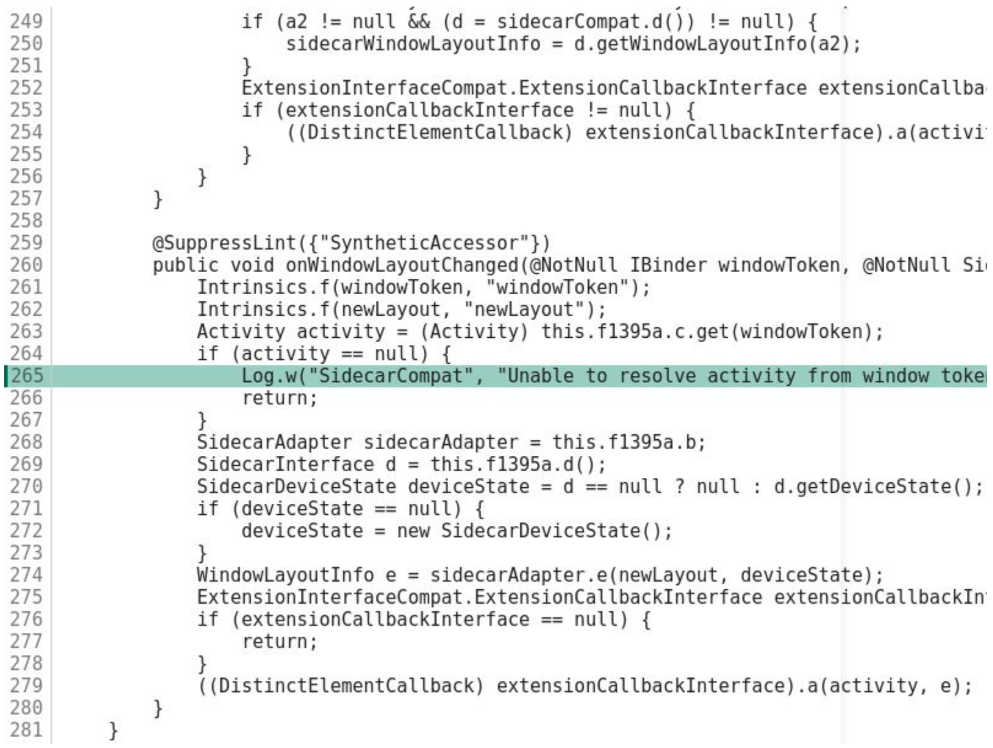
```
                    if (string == null) {
                        b = null;
                    } else {
                        b = fragmentManager.c.b(string);
                        if (b == null) {
                            fragmentManager.c0(new IllegalStateException("Fragment
                            throw null;
                        }
                    }
                    if (b != null) {
                        while (arrayList2.size() <= parseInt) {
                            arrayList2.add(null);
                        }
                        b.V(false);
                        arrayList2.set(parseInt, b);
                    } else {
                        Log.w("FragmentStatePagerAdapt", "Bad fragment at key ".con
                    }
                }
            }
        }
    }

    @Override // androidx.viewpager.widget.PagerAdapter
    @Nullable
    public final Parcelable i() {
        Bundle bundle;
        ArrayList<Fragment.SavedState> arrayList = this.f;
        if (arrayList.size() > 0) {
            bundle = new Bundle();
            Fragment.SavedState[] savedStateArr = new Fragment.SavedState[arrayList
            arrayList.toArray(savedStateArr);
            bundle.putParcelableArray("states", savedStateArr);
```

Finding 5



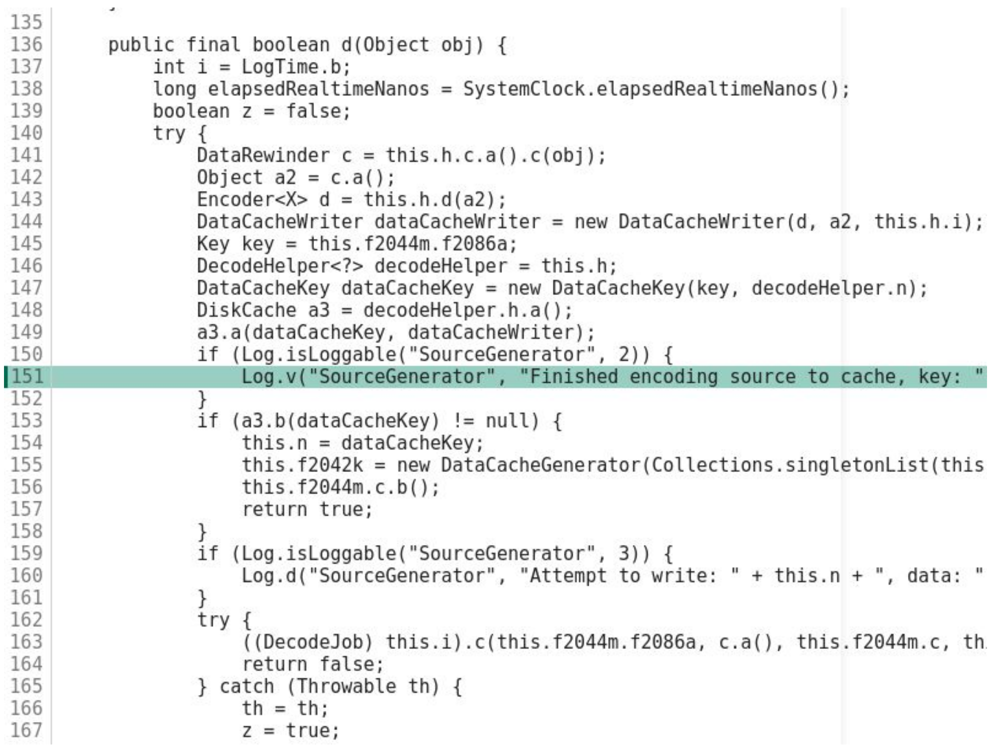
```
        @Override // android.view.AbsSavedState, android.os.Parcelable
        public final void writeToParcel(Parcel parcel, int i) {
            super.writeToParcel(parcel, i);
            parcel.writeInt(this.h);
        }
    }

    public PreferenceGroup(@NonNull Context context, @Nullable AttributeSet attribu
        super(context, attributeSet, i);
        new SimpleArrayMap();
        new Handler(Looper.getMainLooper());
        this.H = new ArrayList();
        TypedArray obtainStyledAttributes = context.obtainStyledAttributes(attribute
        obtainStyledAttributes.getBoolean(2, obtainStyledAttributes.getBoolean(2, t
        if (obtainStyledAttributes.hasValue(1) && obtainStyledAttributes.getInt(1,
            Log.e("PreferenceGroup", getClass().getSimpleName().concat(" should have
        }
        obtainStyledAttributes.recycle();
    }

    @Override // androidx.preference.Preference
    public final void f(boolean z) {
        super.f(z);
        int size = this.H.size();
        for (int i = 0; i < size; i++) {
            Preference p = p(i);
            if (p.t == z) {
                p.t = !z;
                p.f(p.n());
                p.e();
            }
        }
```

Finding 6



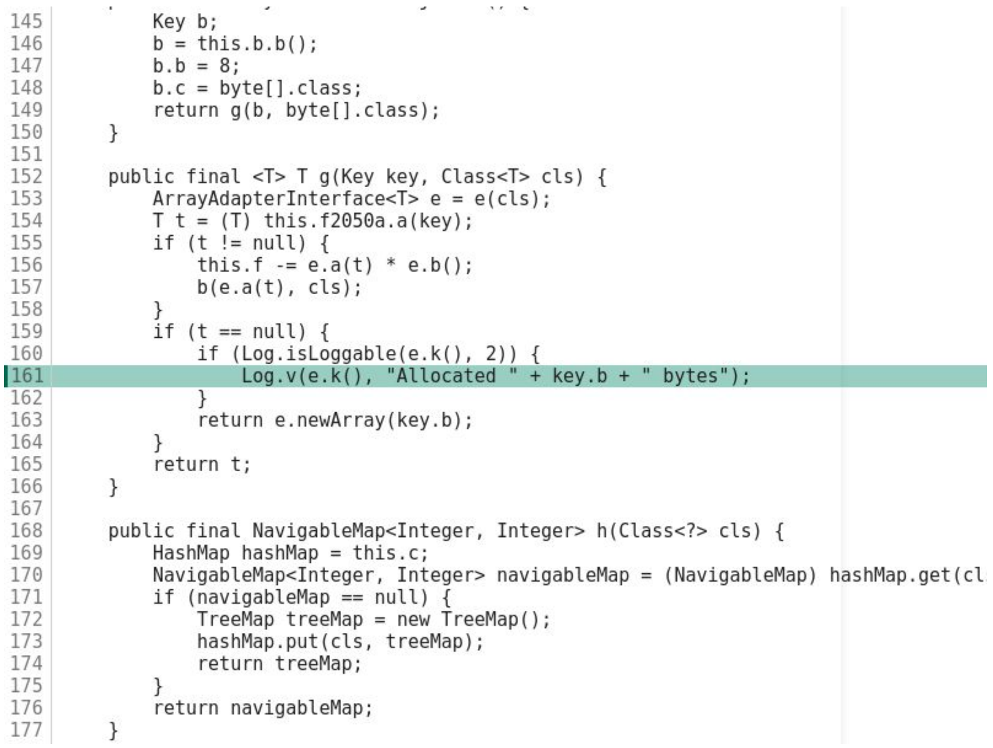
```
                if (a2 != null && (d = sidecarCompat.d()) != null) {
                    sidecarWindowLayoutInfo = d.getWindowLayoutInfo(a2);
                }
                ExtensionInterfaceCompat.ExtensionCallbackInterface extensionCallba
                if (extensionCallbackInterface != null) {
                    ((DistinctElementCallback) extensionCallbackInterface).a(activi
                }
            }
        }

        @SuppressLint({"SyntheticAccessor"})
        public void onWindowLayoutChanged(@NotNull IBinder windowToken, @NotNull Si
            Intrinsics.f(windowToken, "windowToken");
            Intrinsics.f(newLayout, "newLayout");
            Activity activity = (Activity) this.f1395a.c.get(windowToken);
            if (activity == null) {
                Log.w("SidecarCompat", "Unable to resolve activity from window toke
                return;
            }
            SidecarAdapter sidecarAdapter = this.f1395a.b;
            SidecarInterface d = this.f1395a.d();
            SidecarDeviceState deviceState = d == null ? null : d.getDeviceState();
            if (deviceState == null) {
                deviceState = new SidecarDeviceState();
            }
            WindowLayoutInfo e = sidecarAdapter.e(newLayout, deviceState);
            ExtensionInterfaceCompat.ExtensionCallbackInterface extensionCallbackIn
            if (extensionCallbackInterface == null) {
                return;
            }
            ((DistinctElementCallback) extensionCallbackInterface).a(activity, e);
        }
    }
```

Finding 7



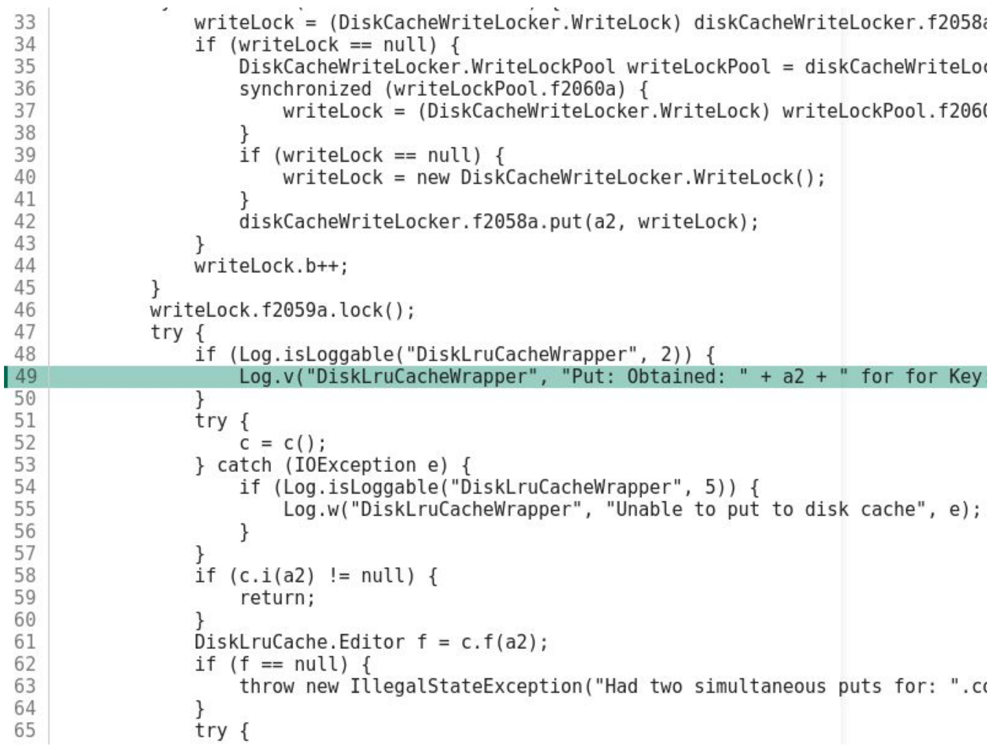
```
    public final boolean d(Object obj) {
        int i = LogTime.b;
        long elapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos();
        boolean z = false;
        try {
            DataRewinder c = this.h.c.a().c(obj);
            Object a2 = c.a();
            Encoder<X> d = this.h.d(a2);
            DataCacheWriter dataCacheWriter = new DataCacheWriter(d, a2, this.h.i);
            Key key = this.f2044m.f2086a;
            DecodeHelper<?> decodeHelper = this.h;
            DataCacheKey dataCacheKey = new DataCacheKey(key, decodeHelper.n);
            DiskCache a3 = decodeHelper.h.a();
            a3.a(dataCacheKey, dataCacheWriter);
            if (Log.isLoggable("SourceGenerator", 2)) {
                Log.v("SourceGenerator", "Finished encoding source to cache, key: "
            }
            if (a3.b(dataCacheKey) != null) {
                this.n = dataCacheKey;
                this.f2042k = new DataCacheGenerator(Collections.singletonList(this
                this.f2044m.c.b();
                return true;
            }
            if (Log.isLoggable("SourceGenerator", 3)) {
                Log.d("SourceGenerator", "Attempt to write: " + this.n + ", data: "
            }
            try {
                ((DecodeJob) this.i).c(this.f2044m.f2086a, c.a(), this.f2044m.c, th
                return false;
            } catch (Throwable th) {
                th = th;
                z = true;
```

Finding 8



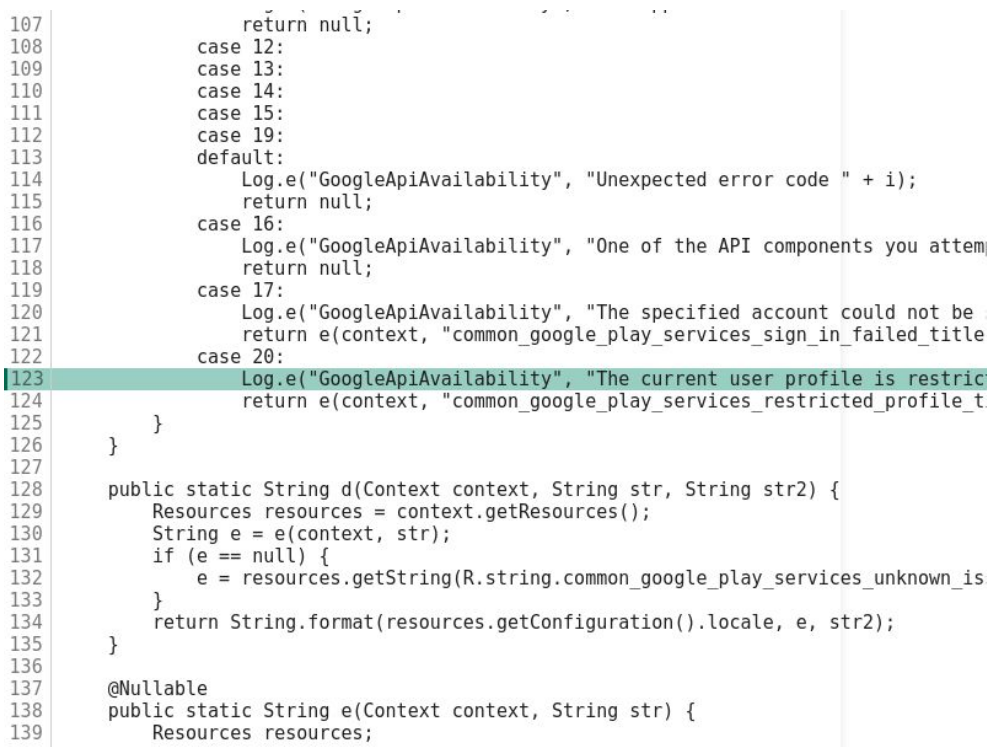
```
        Key b;
        b = this.b.b();
        b.b = 8;
        b.c = byte[].class;
        return g(b, byte[].class);
    }

    public final <T> T g(Key key, Class<T> cls) {
        ArrayAdapterInterface<T> e = e(cls);
        T t = (T) this.f2050a.a(key);
        if (t != null) {
            this.f -= e.a(t) * e.b();
            b(e.a(t), cls);
        }
        if (t == null) {
            if (Log.isLoggable(e.k(), 2)) {
                Log.v(e.k(), "Allocated " + key.b + " bytes");
            }
            return e.newArray(key.b);
        }
        return t;
    }

    public final NavigableMap<Integer, Integer> h(Class<?> cls) {
        HashMap hashMap = this.c;
        NavigableMap<Integer, Integer> navigableMap = (NavigableMap) hashMap.get(cl
        if (navigableMap == null) {
            TreeMap treeMap = new TreeMap();
            hashMap.put(cls, treeMap);
            return treeMap;
        }
        return navigableMap;
    }
```

Finding 9



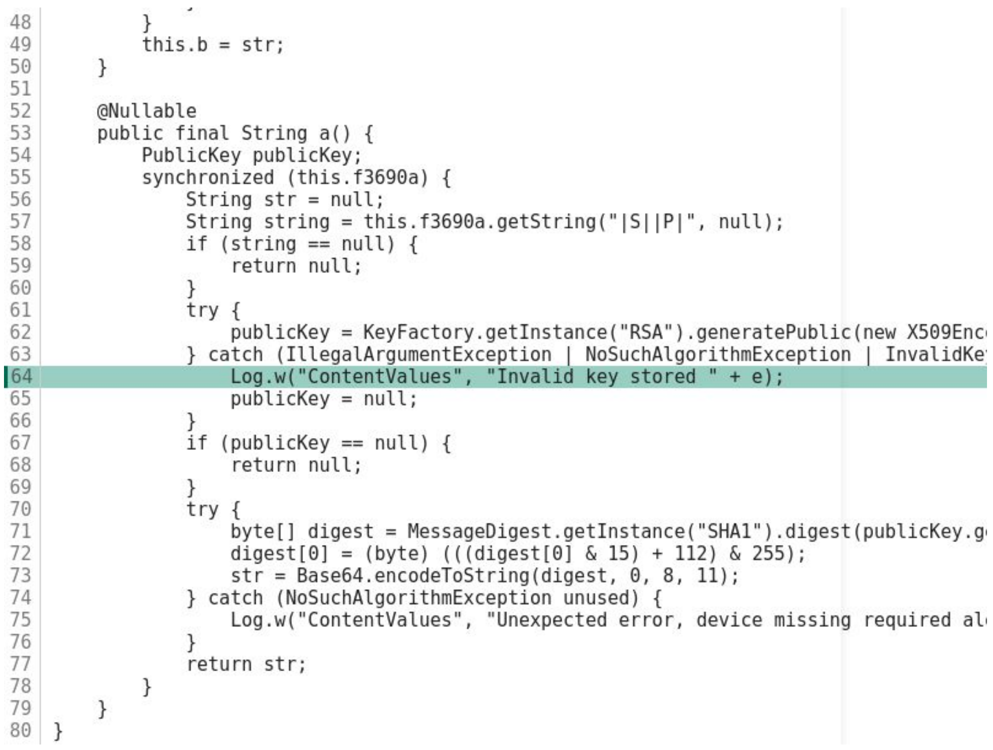
```
            writeLock = (DiskCacheWriteLocker.WriteLock) diskCacheWriteLocker.f2058a
            if (writeLock == null) {
                DiskCacheWriteLocker.WriteLockPool writeLockPool = diskCacheWriteLoc
                synchronized (writeLockPool.f2060a) {
                    writeLock = (DiskCacheWriteLocker.WriteLock) writeLockPool.f206
                }
                if (writeLock == null) {
                    writeLock = new DiskCacheWriteLocker.WriteLock();
                }
                diskCacheWriteLocker.f2058a.put(a2, writeLock);
            }
            writeLock.b++;
        }
        writeLock.f2059a.lock();
        try {
            if (Log.isLoggable("DiskLruCacheWrapper", 2)) {
                Log.v("DiskLruCacheWrapper", "Put: Obtained: " + a2 + " for for Key
            }
            try {
                c = c();
            } catch (IOException e) {
                if (Log.isLoggable("DiskLruCacheWrapper", 5)) {
                    Log.w("DiskLruCacheWrapper", "Unable to put to disk cache", e);
                }
            }
            if (c.i(a2) != null) {
                return;
            }
            DiskLruCache.Editor f = c.f(a2);
            if (f == null) {
                throw new IllegalStateException("Had two simultaneous puts for: ".co
            }
            try {
```

Finding 10



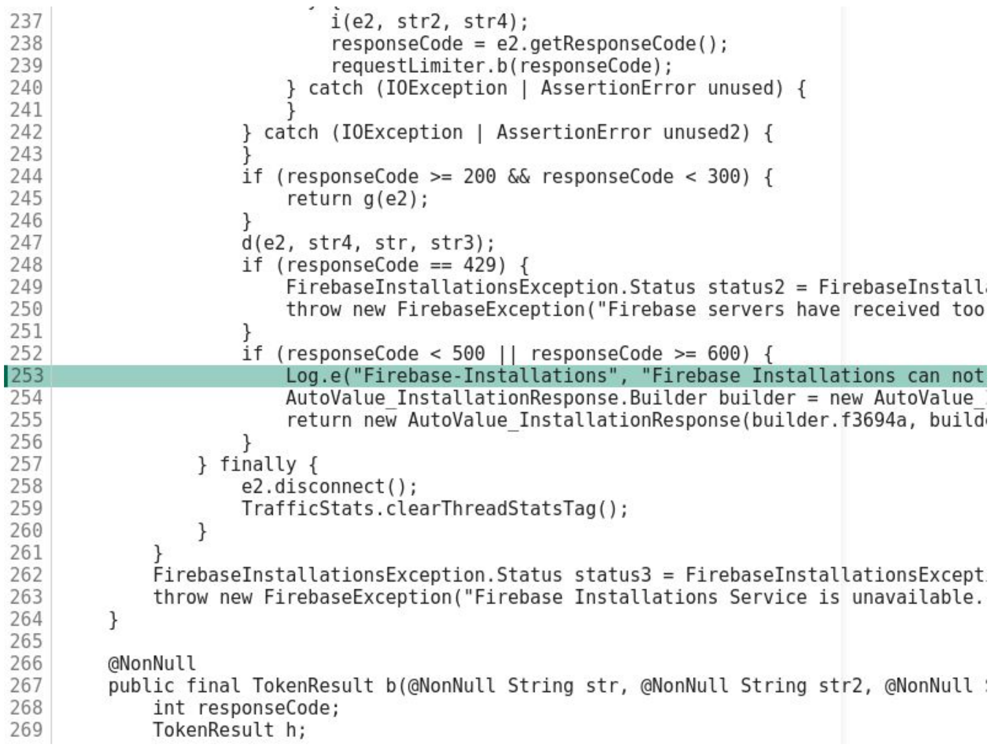
```
                return null;
            case 12:
            case 13:
            case 14:
            case 15:
            case 19:
            default:
                Log.e("GoogleApiAvailability", "Unexpected error code " + i);
                return null;
            case 16:
                Log.e("GoogleApiAvailability", "One of the API components you attem
                return null;
            case 17:
                Log.e("GoogleApiAvailability", "The specified account could not be
                return e(context, "common_google_play_services_sign_in_failed_title
            case 20:
                Log.e("GoogleApiAvailability", "The current user profile is restric
                return e(context, "common_google_play_services_restricted_profile_t
        }
    }

    public static String d(Context context, String str, String str2) {
        Resources resources = context.getResources();
        String e = e(context, str);
        if (e == null) {
            e = resources.getString(R.string.common_google_play_services_unknown_is
        }
        return String.format(resources.getConfiguration().locale, e, str2);
    }

    @Nullable
    public static String e(Context context, String str) {
        Resources resources;
```

Finding 11



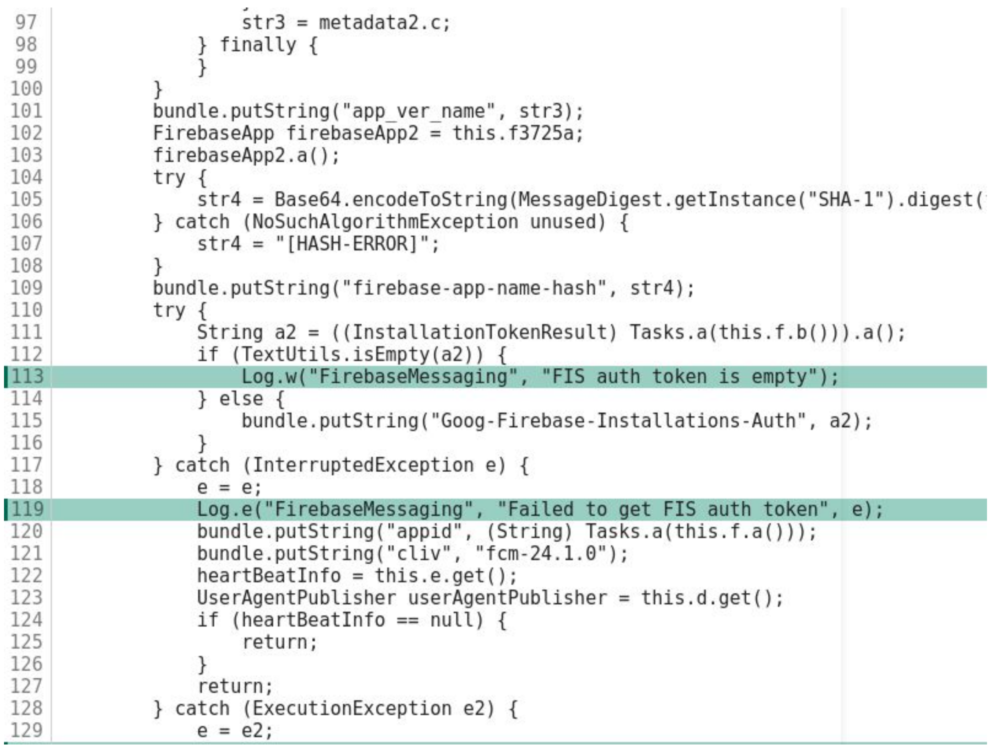
```
        }
        this.b = str;
    }

    @Nullable
    public final String a() {
        PublicKey publicKey;
        synchronized (this.f3690a) {
            String str = null;
            String string = this.f3690a.getString("|S||P|", null);
            if (string == null) {
                return null;
            }
            try {
                publicKey = KeyFactory.getInstance("RSA").generatePublic(new X509Enc
            } catch (IllegalArgumentException | NoSuchAlgorithmException | InvalidKe
                Log.w("ContentValues", "Invalid key stored " + e);
                publicKey = null;
            }
            if (publicKey == null) {
                return null;
            }
            try {
                byte[] digest = MessageDigest.getInstance("SHA1").digest(publicKey.g
                digest[0] = (byte) (((digest[0] & 15) + 112) & 255);
                str = Base64.encodeToString(digest, 0, 8, 11);
            } catch (NoSuchAlgorithmException unused) {
                Log.w("ContentValues", "Unexpected error, device missing required al
            }
            return str;
        }
    }
}
```

Finding 12



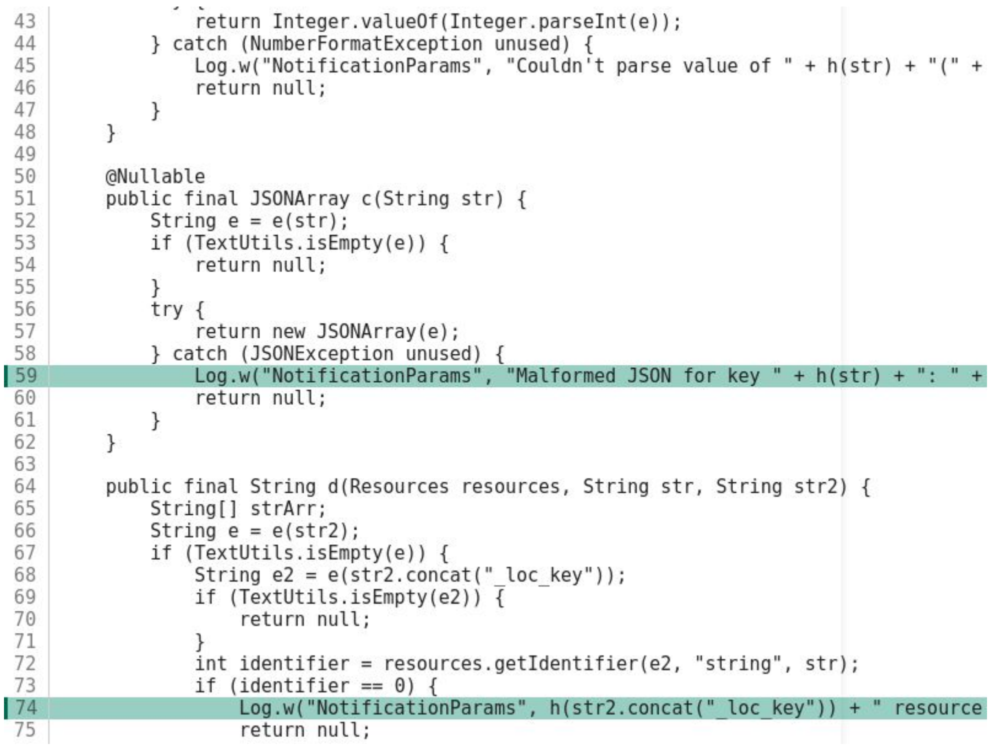
```
                        i(e2, str2, str4);
                        responseCode = e2.getResponseCode();
                        requestLimiter.b(responseCode);
                    } catch (IOException | AssertionError unused) {
                    }
                } catch (IOException | AssertionError unused2) {
                }
                if (responseCode >= 200 && responseCode < 300) {
                    return g(e2);
                }
                d(e2, str4, str, str3);
                if (responseCode == 429) {
                    FirebaseInstallationsException.Status status2 = FirebaseInstall
                    throw new FirebaseException("Firebase servers have received too
                }
                if (responseCode < 500 || responseCode >= 600) {
                    Log.e("Firebase-Installations", "Firebase Installations can not
                    AutoValue_InstallationResponse.Builder builder = new AutoValue_
                    return new AutoValue_InstallationResponse(builder.f3694a, build
                }
            } finally {
                e2.disconnect();
                TrafficStats.clearThreadStatsTag();
            }
        }
        FirebaseInstallationsException.Status status3 = FirebaseInstallationsExcept
        throw new FirebaseException("Firebase Installations Service is unavailable.
    }

    @NonNull
    public final TokenResult b(@NonNull String str, @NonNull String str2, @NonNull
        int responseCode;
        TokenResult h;
```

Finding 13



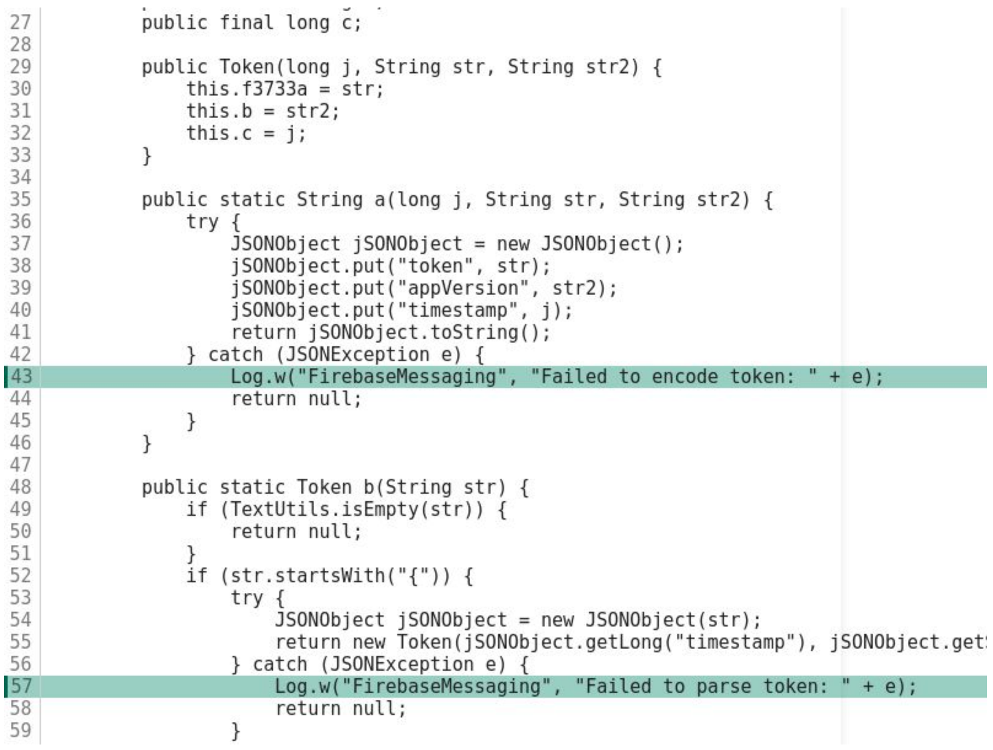
```
                str3 = metadata2.c;
            } finally {
            }
        }
        bundle.putString("app_ver_name", str3);
        FirebaseApp firebaseApp2 = this.f3725a;
        firebaseApp2.a();
        try {
            str4 = Base64.encodeToString(MessageDigest.getInstance("SHA-1").digest(
        } catch (NoSuchAlgorithmException unused) {
            str4 = "[HASH-ERROR]";
        }
        bundle.putString("firebase-app-name-hash", str4);
        try {
            String a2 = ((InstallationTokenResult) Tasks.a(this.f.b())).a();
            if (TextUtils.isEmpty(a2)) {
                Log.w("FirebaseMessaging", "FIS auth token is empty");
            } else {
                bundle.putString("Goog-Firebase-Installations-Auth", a2);
            }
        } catch (InterruptedException e) {
            e = e;
            Log.e("FirebaseMessaging", "Failed to get FIS auth token", e);
            bundle.putString("appid", (String) Tasks.a(this.f.a()));
            bundle.putString("cliv", "fcm-24.1.0");
            heartBeatInfo = this.e.get();
            UserAgentPublisher userAgentPublisher = this.d.get();
            if (heartBeatInfo == null) {
                return;
            }
            return;
        } catch (ExecutionException e2) {
            e = e2;
```

Finding 14



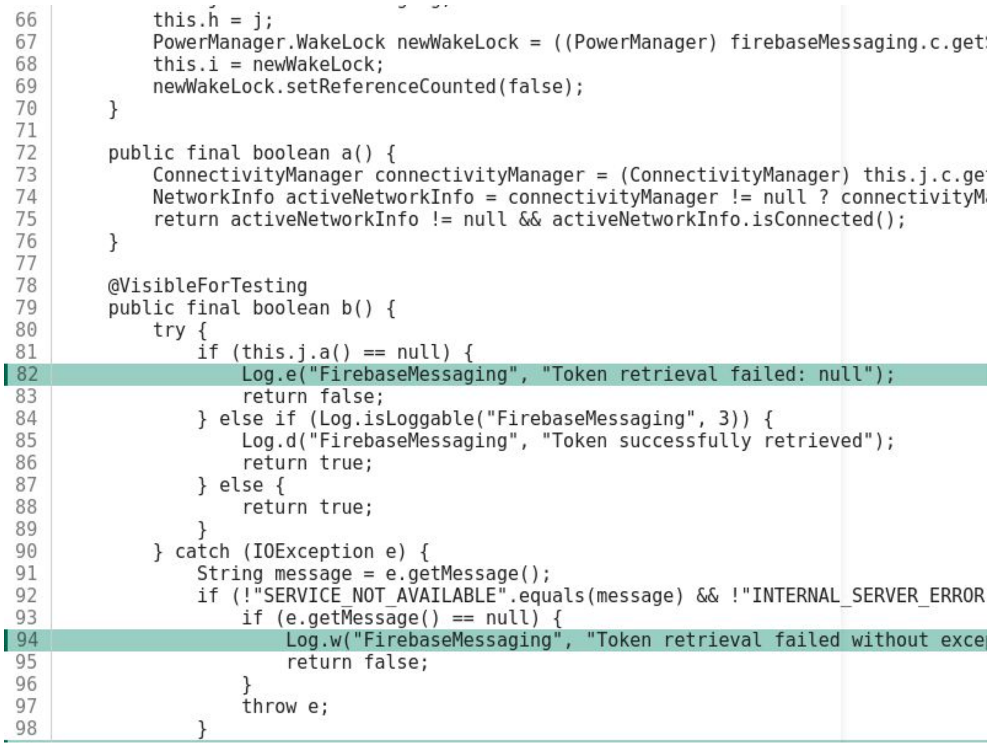
```
            return Integer.valueOf(Integer.parseInt(e));
        } catch (NumberFormatException unused) {
            Log.w("NotificationParams", "Couldn't parse value of " + h(str) + "(" +
            return null;
        }
    }

    @Nullable
    public final JSONArray c(String str) {
        String e = e(str);
        if (TextUtils.isEmpty(e)) {
            return null;
        }
        try {
            return new JSONArray(e);
        } catch (JSONException unused) {
            Log.w("NotificationParams", "Malformed JSON for key " + h(str) + ": " +
            return null;
        }
    }

    public final String d(Resources resources, String str, String str2) {
        String[] strArr;
        String e = e(str2);
        if (TextUtils.isEmpty(e)) {
            String e2 = e(str2.concat("_loc_key"));
            if (TextUtils.isEmpty(e2)) {
                return null;
            }
            int identifier = resources.getIdentifier(e2, "string", str);
            if (identifier == 0) {
                Log.w("NotificationParams", h(str2.concat("_loc_key")) + " resource
                return null;
```

Finding 15



```
public final long c;

public Token(long j, String str, String str2) {
    this.f3733a = str;
    this.b = str2;
    this.c = j;
}

public static String a(long j, String str, String str2) {
    try {
        JSONObject jSONObject = new JSONObject();
        jSONObject.put("token", str);
        jSONObject.put("appVersion", str2);
        jSONObject.put("timestamp", j);
        return jSONObject.toString();
    } catch (JSONException e) {
        Log.w("FirebaseMessaging", "Failed to encode token: " + e);
        return null;
    }
}

public static Token b(String str) {
    if (TextUtils.isEmpty(str)) {
        return null;
    }
    if (str.startsWith("{")) {
        try {
            JSONObject jSONObject = new JSONObject(str);
            return new Token(jSONObject.getLong("timestamp"), jSONObject.get
        } catch (JSONException e) {
            Log.w("FirebaseMessaging", "Failed to parse token: " + e);
            return null;
        }
```

Finding 16



```
    this.h = j;
    PowerManager.WakeLock newWakeLock = ((PowerManager) firebaseMessaging.c.get
    this.i = newWakeLock;
    newWakeLock.setReferenceCounted(false);
}

public final boolean a() {
    ConnectivityManager connectivityManager = (ConnectivityManager) this.j.c.ge
    NetworkInfo activeNetworkInfo = connectivityManager != null ? connectivityM
    return activeNetworkInfo != null && activeNetworkInfo.isConnected();
}

@VisibleForTesting
public final boolean b() {
    try {
        if (this.j.a() == null) {
            Log.e("FirebaseMessaging", "Token retrieval failed: null");
            return false;
        } else if (Log.isLoggable("FirebaseMessaging", 3)) {
            Log.d("FirebaseMessaging", "Token successfully retrieved");
            return true;
        } else {
            return true;
        }
    } catch (IOException e) {
        String message = e.getMessage();
        if (!"SERVICE_NOT_AVAILABLE".equals(message) && !"INTERNAL_SERVER_ERROR
            if (e.getMessage() == null) {
                Log.w("FirebaseMessaging", "Token retrieval failed without exce
                return false;
            }
            throw e;
        }
```

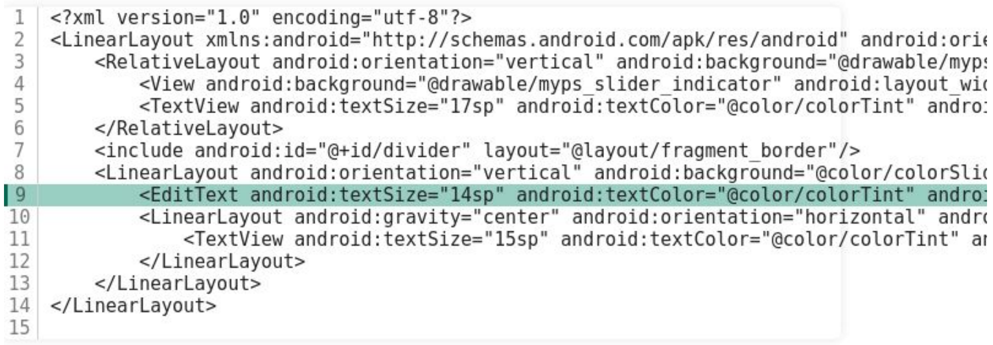
We would like to clarify the following:

1. **No Sensitive User Data Logged**
   - Our application does **not** record user credentials, personally identifiable information (PII), or license keys in any logs.
   - Any potentially sensitive input is handled internally and **never** written to device logs.
2. **Referenced Classes from Third-Party / Google Libraries**
   - The listed files (CoordinatorLayout.java, PreferenceGroup.java, various Glide and Firebase classes, etc.) are part of **standard Android libraries** or **open-source** components.
   - We have made **no modifications** that would cause these libraries to log private or sensitive data. For instance, Glide handles image loading and caching, which does **not** involve logging user-input text or confidential information.
3. **Compliance with MSTG-STORAGE-3**
   - We confirm that **no** user-sensitive information is exposed through logs.
   - Our production logs are strictly limited to operational or non-sensitive debugging statements and do **not** contain personal, license, or other confidential data.

MSTG-STORAGE-7

No sensitive data, such as passwords or pins, is exposed through the user interface.

Finding 1



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orie
    <RelativeLayout android:orientation="vertical" android:background="@drawable/myps
        <View android:background="@drawable/myps_slider_indicator" android:layout_wic
        <TextView android:textSize="17sp" android:textColor="@color/colorTint" andro:
    </RelativeLayout>
    <include android:id="@+id/divider" layout="@layout/fragment_border"/>
    <LinearLayout android:orientation="vertical" android:background="@color/colorSlic
        <EditText android:textSize="14sp" android:textColor="@color/colorTint" andro:
        <LinearLayout android:gravity="center" android:orientation="horizontal" andrc
            <TextView android:textSize="15sp" android:textColor="@color/colorTint" ar
        </LinearLayout>
    </LinearLayout>
</LinearLayout>
```

We would like to clarify the following:

1. **License Keys vs. Sensitive User Data**
   - The field in myps_activity_activate_slider.xml is used for entering a **license key** , **not** a password, PIN, or personally identifiable information (PII).
   - License keys do not grant access to user accounts or personal data, so they are **not** considered confidential in the same sense as passwords or PINs.
2. **No Exposure of Confidential Information**
   - We do not display user credentials, session tokens, or any data that could compromise user security.
   - The EditText in question is solely for product activation, and its contents do **not** reveal sensitive user details.
3. **Compliance with MSTG-STORAGE-7**
   - Since the license key is **not** treated as sensitive user data (e.g., a password or PIN), showing it in the UI does **not** violate MSTG-STORAGE-7.
   - No other UI elements in our application expose private or sensitive data.

Scan Verified

In this section you can find all the results based on the automatic evaluation, for the test cases in PASS it can be seen as informative, for the results marked as Fail you need to fix the issues or provide some feedback or justification in the respective area.

FAIL

PASS

MSTG-STORAGE-5

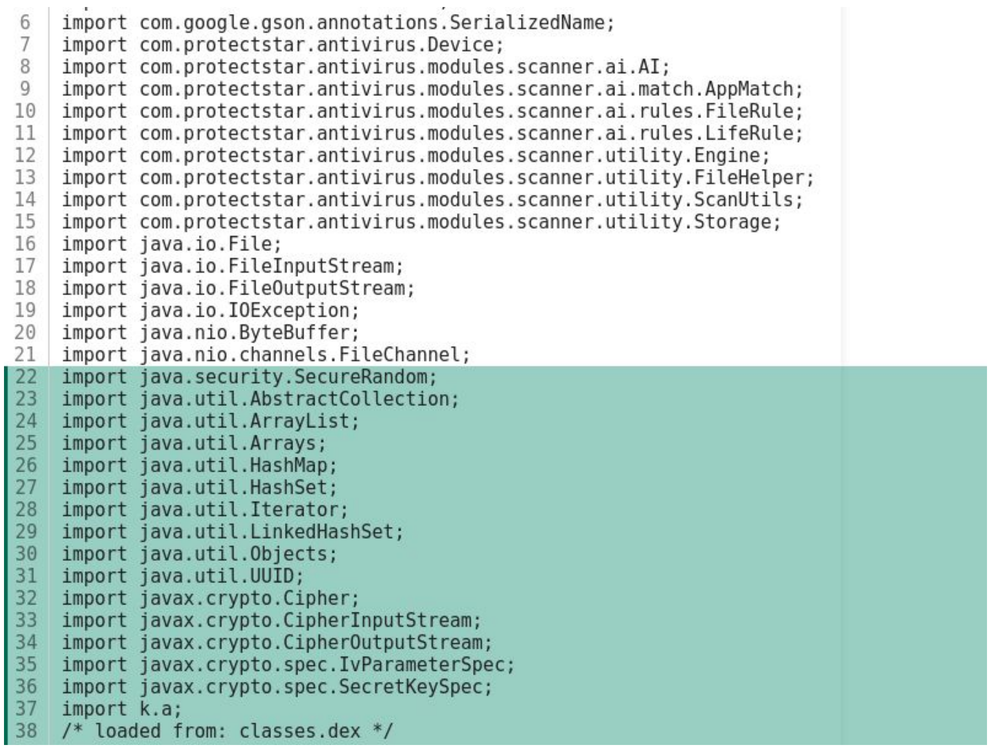
The keyboard cache is disabled on text inputs that process sensitive data.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-1

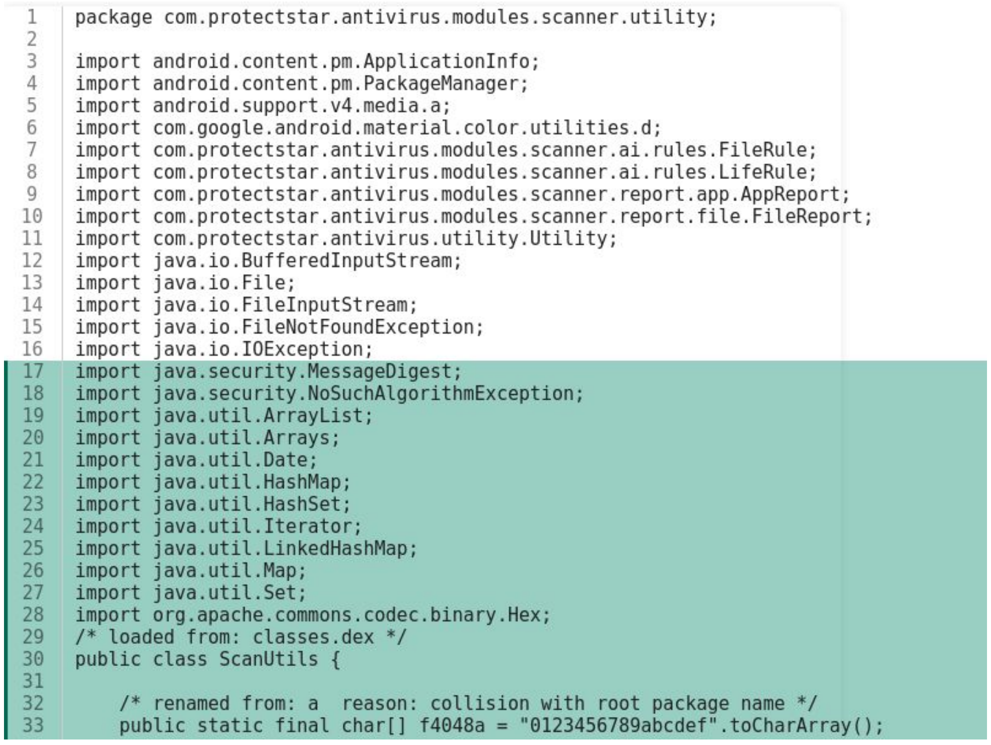
The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.

Finding 1



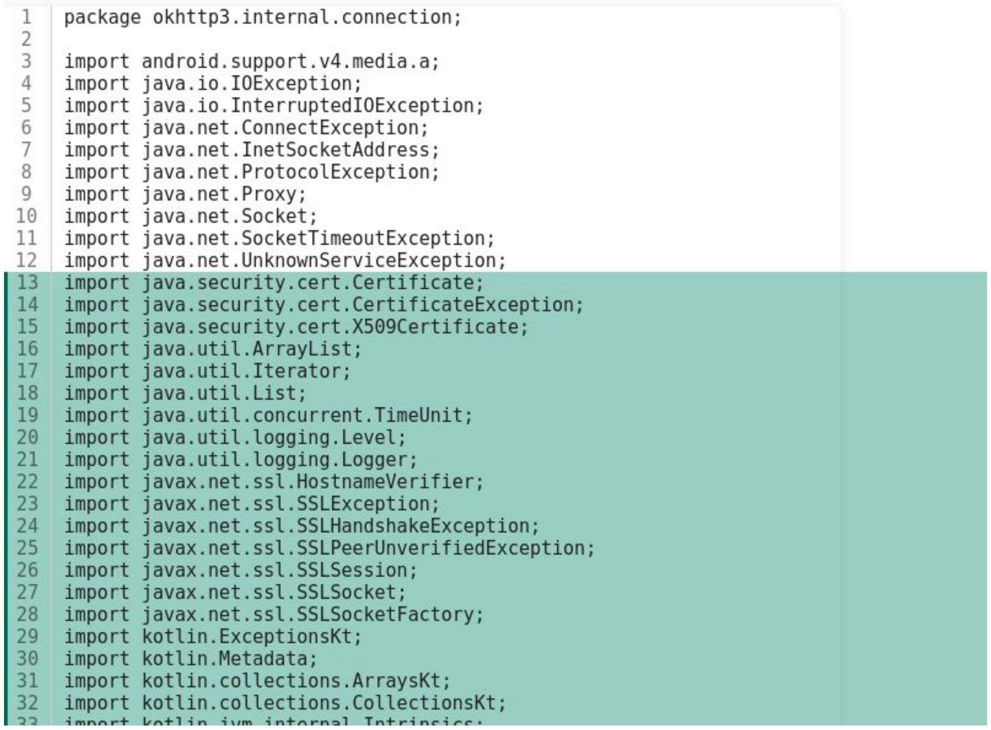
```
import com.google.gson.annotations.SerializedName;
import com.protectstar.antivirus.Device;
import com.protectstar.antivirus.modules.scanner.ai.AI;
import com.protectstar.antivirus.modules.scanner.ai.match.AppMatch;
import com.protectstar.antivirus.modules.scanner.ai.rules.FileRule;
import com.protectstar.antivirus.modules.scanner.ai.rules.LifeRule;
import com.protectstar.antivirus.modules.scanner.utility.Engine;
import com.protectstar.antivirus.modules.scanner.utility.FileHelper;
import com.protectstar.antivirus.modules.scanner.utility.ScanUtils;
import com.protectstar.antivirus.modules.scanner.utility.Storage;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.security.SecureRandom;
import java.util.AbstractCollection;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.Objects;
import java.util.UUID;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import k.a;
/* loaded from: classes.dex */
```

Finding 2



```
package com.protectstar.antivirus.modules.scanner.utility;

import android.content.pm.ApplicationInfo;
import android.content.pm.PackageManager;
import android.support.v4.media.a;
import com.google.android.material.color.utilities.d;
import com.protectstar.antivirus.modules.scanner.ai.rules.FileRule;
import com.protectstar.antivirus.modules.scanner.ai.rules.LifeRule;
import com.protectstar.antivirus.modules.scanner.report.app.AppReport;
import com.protectstar.antivirus.modules.scanner.report.file.FileReport;
import com.protectstar.antivirus.utility.Utility;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;
import org.apache.commons.codec.binary.Hex;
/* loaded from: classes.dex */
public class ScanUtils {

    /* renamed from: a  reason: collision with root package name */
    public static final char[] f4048a = "0123456789abcdef".toCharArray();
```

Finding 3



```
package okhttp3.internal.connection;

import android.support.v4.media.a;
import java.io.IOException;
import java.io.InterruptedIOException;
import java.net.ConnectException;
import java.net.InetSocketAddress;
import java.net.ProtocolException;
import java.net.Proxy;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.net.UnknownServiceException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLException;
import javax.net.ssl.SSLHandshakeException;
import javax.net.ssl.SSLPeerUnverifiedException;
import javax.net.ssl.SSLSession;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import kotlin.ExceptionsKt;
import kotlin.Metadata;
import kotlin.collections.ArraysKt;
import kotlin.collections.CollectionsKt;
import kotlin.jvm.internal.Intrinsics;
```

We would like to clarify the following details:

1.1.1.1. FileMatch.java (AES-256 Key & IV for Quarantine)

- **Purpose** :
  The **hardcoded AES key and IV** in this file are **not** intended for protecting sensitive user data. Instead, they encrypt **malicious or suspicious files** during quarantine, making them unusable while quarantined.
- **No Exposure of User Data** :
  These files are potentially harmful and do **not** belong to the user’s personal information or credentials. Thus, no user PII is at risk.
- **Reversibility** :
  If a user opts to restore a quarantined file, the same key can decrypt it. However, **no** confidential user data is exposed, since we do **not** store user secrets with this key.

1.1.2. No Sole Reliance on Hardcoded Keys

- **Non-User Data** :
  Because the quarantined files are **not** user-owned documents, the key does **not** safeguard personal data.
- **No Risk to User Credentials** :
  The encryption key for quarantine does **not** expose any user credentials, secrets, or PII.

1.1.3.3. ScanUtils.java (Helper Methods & SecureRandom)

- **Secure Deletion** :
  The flagged methods handle **secure overwriting** of files using random data from SecureRandom or a fixed pattern (e.g., 0xFF).
- **Message Digest** :
  These methods also leverage MessageDigest (SHA-1 & SHA-256) for generating file hashes. They are **not** used to encrypt sensitive user data with a hardcoded key.

1.1.4. RealConnection.java (OkHttp Library)

- **Network Connections** :
  RealConnection.java is part of the OkHttp library, which establishes secure **TLS** connections.
- **No Hardcoded Key** :
  This class does **not** rely on a hardcoded key for encrypting user data. OkHttp follows standard TLS practices with ephemeral keys and industry-best ciphers.

1.1.5. Compliance with MSTG-CRYPTO-1

- **No Hardcoded Keys for User Data** :
  We confirm that our app does **not** rely on a single, hardcoded symmetric key to protect sensitive or personal user information.
- **Quarantine-Specific Key** :
  The **hardcoded key** noted is strictly for **quarantine** to neutralize malicious files, not for protecting actual user secrets.
- **Proper Cryptographic Measures** :
  Should any user credentials or personal data need protection, we utilize recommended cryptographic APIs such as SecureRandom, ensuring compliance with industry standards.

We trust this explanation clarifies our encryption approach.
After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-2

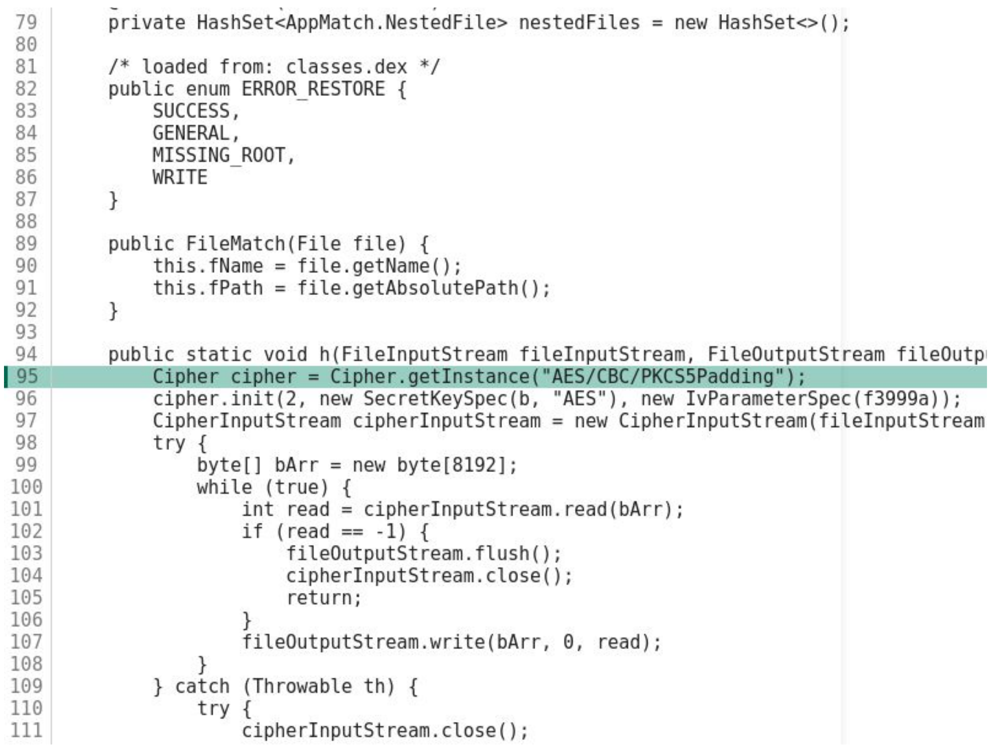
The app uses proven implementations of cryptographic primitives.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-3

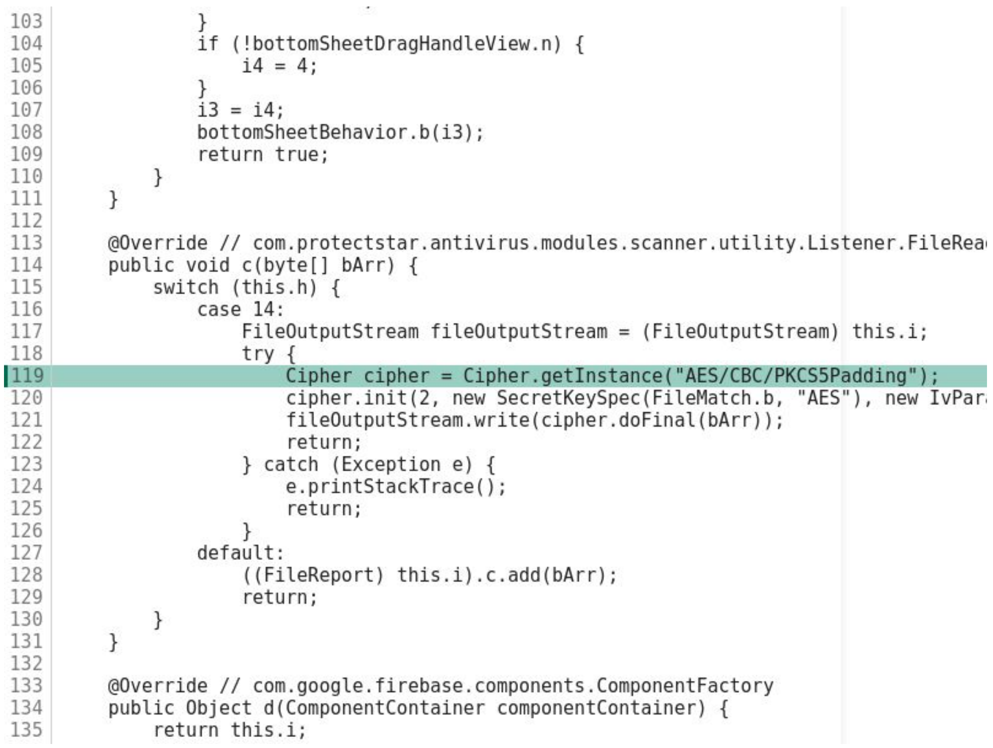
The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.

Finding 1



```
    private HashSet<AppMatch.NestedFile> nestedFiles = new HashSet<>();

    /* loaded from: classes.dex */
    public enum ERROR_RESTORE {
        SUCCESS,
        GENERAL,
        MISSING_ROOT,
        WRITE
    }

    public FileMatch(File file) {
        this.fName = file.getName();
        this.fPath = file.getAbsolutePath();
    }

    public static void h(FileInputStream fileInputStream, FileOutputStream fileOutp
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(2, new SecretKeySpec(b, "AES"), new IvParameterSpec(f3999a));
        CipherInputStream cipherInputStream = new CipherInputStream(fileInputStream
        try {
            byte[] bArr = new byte[8192];
            while (true) {
                int read = cipherInputStream.read(bArr);
                if (read == -1) {
                    fileOutputStream.flush();
                    cipherInputStream.close();
                    return;
                }
                fileOutputStream.write(bArr, 0, read);
            }
        } catch (Throwable th) {
            try {
                cipherInputStream.close();
```

Finding 2



```
            }
            if (!bottomSheetDragHandleView.n) {
                i4 = 4;
            }
            i3 = i4;
            bottomSheetBehavior.b(i3);
            return true;
        }
    }

    @Override // com.protectstar.antivirus.modules.scanner.utility.Listener.FileRea
    public void c(byte[] bArr) {
        switch (this.h) {
            case 14:
                FileOutputStream fileOutputStream = (FileOutputStream) this.i;
                try {
                    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                    cipher.init(2, new SecretKeySpec(FileMatch.b, "AES"), new IvPar
                    fileOutputStream.write(cipher.doFinal(bArr));
                    return;
                } catch (Exception e) {
                    e.printStackTrace();
                    return;
                }
            default:
                ((FileReport) this.i).c.add(bArr);
                return;
        }
    }

    @Override // com.google.firebase.components.ComponentFactory
    public Object d(ComponentContainer componentContainer) {
        return this.i;
```

We would like to clarify the following:

1.1.6. FileMatch.java (AES-256 for Quarantine)

- **Use-Case & Rationale**
 The **AES-256** encryption in FileMatch.java is used to quarantine **malicious or suspicious files** , preventing them from being executed while in quarantine. Since these files are *not* user-sensitive data (like personal documents or credentials), the core purpose is to **neutralize malware** , rather than protect PII.
- **Best-Practice Configuration**
 AES-256 is a **widely recognized modern cipher** , considered secure for this use-case. While the key is hardcoded, it is **not** meant for user data encryption, thus minimizing risk. If the user chooses to restore the quarantined file, the same key can decrypt it—again, *no* user credentials or personal data are involved.

1.1.7. k/a.java (Cryptographic Utility / Helper Methods)

- If the flagged code relates to **hashing** , **random data** , or other cryptographic utilities, we confirm:
  - **No Weak Ciphers** : We do not use deprecated algorithms like MD5 or SHA1 for cryptographic *security* .
  - **Secure Hashes** : Where applicable, we employ modern hashes (e.g., SHA-256).
  - **SecureRandom** : For any truly random needs in cryptographic contexts, we rely on SecureRandom, aligning with best-practice guidelines.

1.1.8. Adherence to MSTG-CRYPTO-3

- **Appropriate Ciphers & Modes**
 We use **AES-256** , a current industry standard, for neutralizing malicious files in quarantine. For any other cryptographic tasks (e.g., hashing or key generation), we adhere to **modern primitives** and libraries endorsed by the Android platform.
- **Non-Sensitive Data**
 The quarantined files (encrypted via AES-256) do **not** contain user credentials or personal info. This approach **isolates** and **inactivates** malicious data without risking user privacy.
- **No Outdated or Insecure Algorithms**
 We continuously evaluate our cryptographic approaches to ensure compliance with current recommendations (e.g., avoiding obsolete ciphers or insecure configurations).

Conclusion
 Our cryptographic usage matches MSTG-CRYPTO-3 for its specific purpose (malicious file quarantine and any utility hashing functions). We do not rely on outdated or improperly configured algorithms.
After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-4

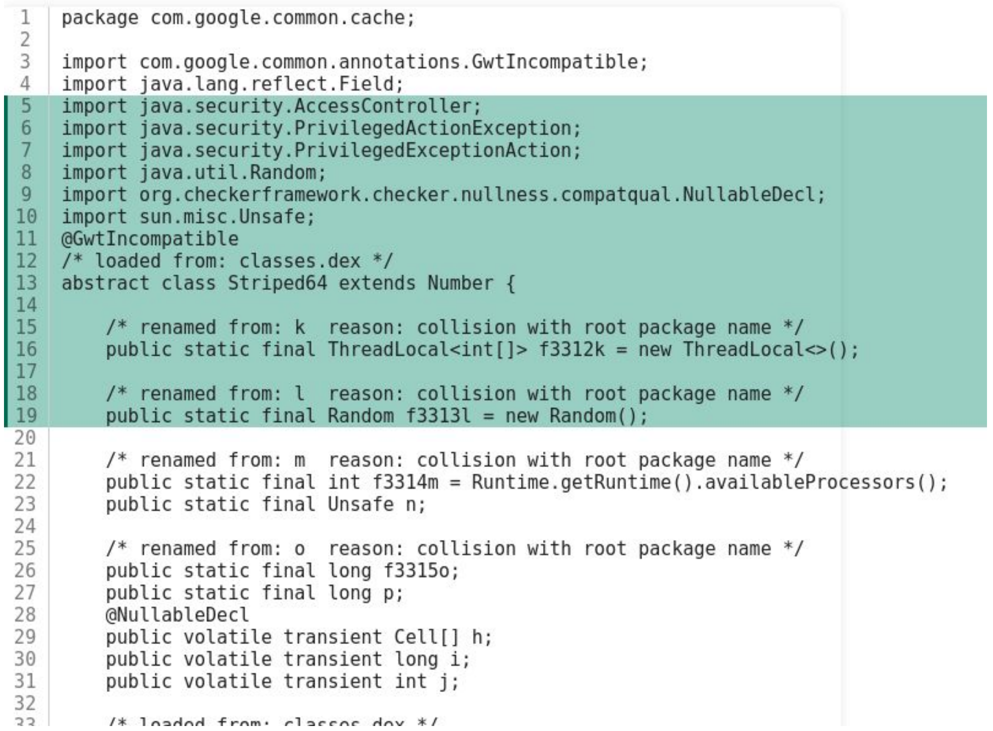
The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-6

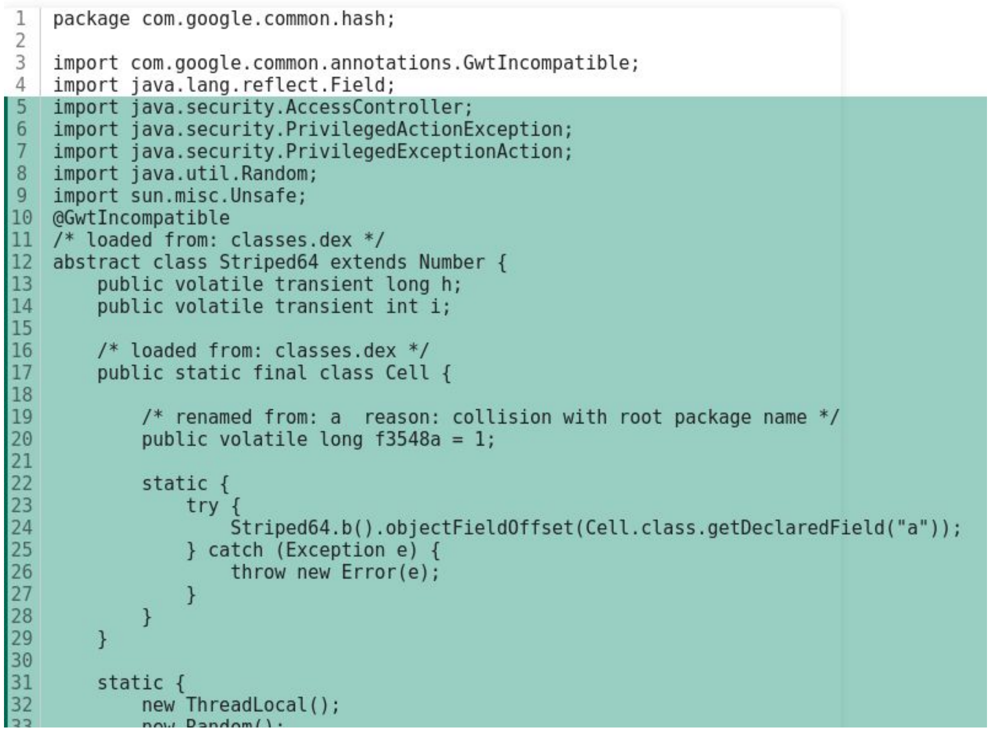
All random values are generated using a sufficiently secure random number generator.

Finding 1



```
package com.google.common.cache;

import com.google.common.annotations.GwtIncompatible;
import java.lang.reflect.Field;
import java.security.AccessController;
import java.security.PrivilegedActionException;
import java.security.PrivilegedExceptionAction;
import java.util.Random;
import org.checkerframework.checker.nullness.compatqual.NullableDecl;
import sun.misc.Unsafe;
@GwtIncompatible
/* loaded from: classes.dex */
abstract class Striped64 extends Number {

    /* renamed from: k  reason: collision with root package name */
    public static final ThreadLocal<int[]> f3312k = new ThreadLocal<>();

    /* renamed from: l  reason: collision with root package name */
    public static final Random f3313l = new Random();

    /* renamed from: m  reason: collision with root package name */
    public static final int f3314m = Runtime.getRuntime().availableProcessors();
    public static final Unsafe n;

    /* renamed from: o  reason: collision with root package name */
    public static final long f3315o;
    public static final long p;
    @NullableDecl
    public volatile transient Cell[] h;
    public volatile transient long i;
    public volatile transient int j;

```

Finding 2



```
package com.google.common.hash;

import com.google.common.annotations.GwtIncompatible;
import java.lang.reflect.Field;
import java.security.AccessController;
import java.security.PrivilegedActionException;
import java.security.PrivilegedExceptionAction;
import java.util.Random;
import sun.misc.Unsafe;
@GwtIncompatible
/* loaded from: classes.dex */
abstract class Striped64 extends Number {
    public volatile transient long h;
    public volatile transient int i;

    /* loaded from: classes.dex */
    public static final class Cell {

        /* renamed from: a  reason: collision with root package name */
        public volatile long f3548a = 1;

        static {
            try {
                Striped64.b().objectFieldOffset(Cell.class.getDeclaredField("a"));
            } catch (Exception e) {
                throw new Error(e);
            }
        }
    }

    static {
        new ThreadLocal();
        new Random();
```

We would like to clarify the following:

1. **Google-Provided Classes**
   The flagged files (Striped64.java) originate from a **Google** library. This code typically uses standard pseudo-random number generation (e.g., ThreadLocalRandom or Random) for **non-cryptographic** operations, such as internal hashing or concurrency controls.
2. **No Cryptographic Usage**
   In this context, the **random values** do not serve a **security-sensitive** purpose. They are not used for encrypting, signing, key derivation, or other operations requiring a secure PRNG like SecureRandom.
3. **No Violation of MSTG-CRYPTO-6**
   Since these random values are not employed in **cryptographic** scenarios, a fully cryptographic RNG is not required. For any actual cryptographic needs in our app, we rely on **SecureRandom** to ensure compliance with MSTG-CRYPTO-6.

We hope this clarifies why the use of a standard random generator here does not constitute a security risk.
After further analysis the application seems to comply successfully with the requirement.

MSTG-NETWORK-1

Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.

After further analysis the application seems to comply successfully with the requirement.

MSTG-NETWORK-2

The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.

After further analysis the application seems to comply successfully with the requirement.

MSTG-NETWORK-3

The app verifies the X.509 certificate of the remote endpoint when the secure channel is established.

After further analysis the application seems to comply successfully with the requirement.

MSTG-PLATFORM-1

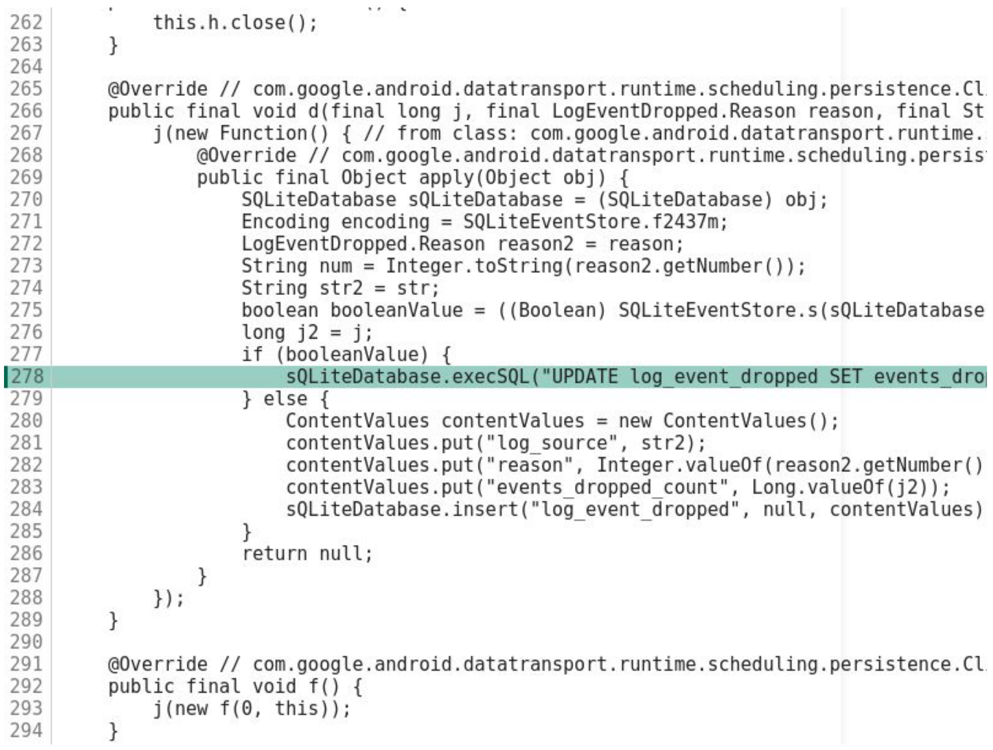
The app only requests the minimum set of permissions necessary.

After further analysis the application seems to comply successfully with the requirement.

MSTG-PLATFORM-2

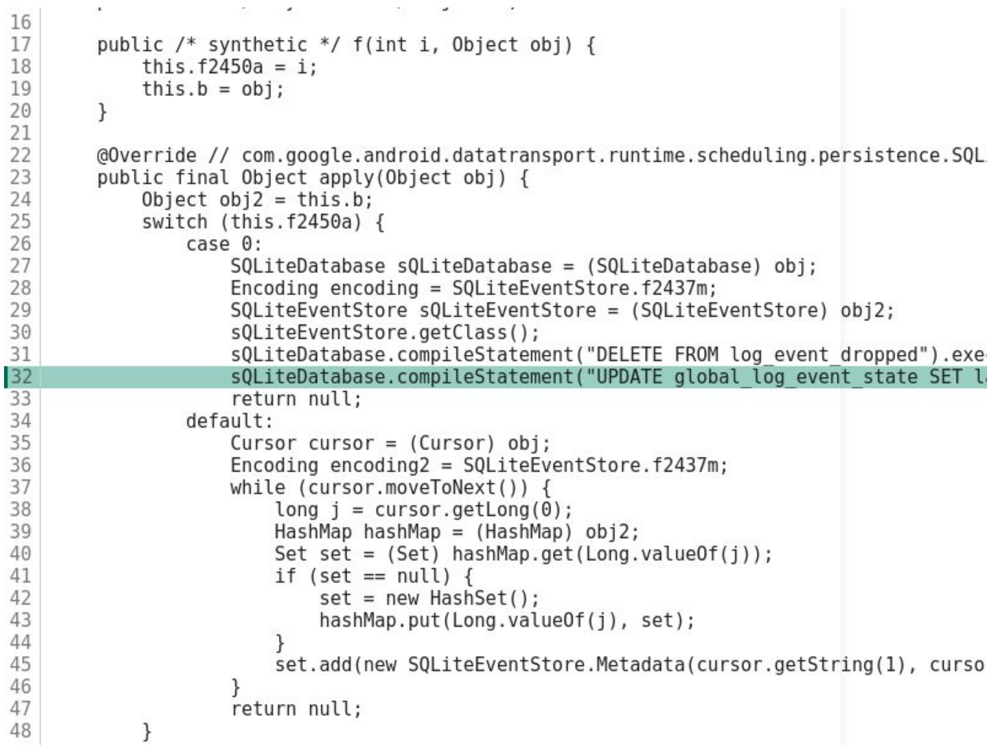
All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

Finding 1



```
    this.h.close();
}

@Override // com.google.android.datatransport.runtime.scheduling.persistence.Cl
public final void d(final long j, final LogEventDropped.Reason reason, final St
    j(new Function() { // from class: com.google.android.datatransport.runtime.
        @Override // com.google.android.datatransport.runtime.scheduling.persis
        public final Object apply(Object obj) {
            SQLiteDatabase sQLiteDatabase = (SQLiteDatabase) obj;
            Encoding encoding = SQLiteEventStore.f2437m;
            LogEventDropped.Reason reason2 = reason;
            String num = Integer.toString(reason2.getNumber());
            String str2 = str;
            boolean booleanValue = ((Boolean) SQLiteEventStore.s(sQLiteDatabase
            long j2 = j;
            if (booleanValue) {
                sQLiteDatabase.execSQL("UPDATE log_event_dropped SET events_dro
            } else {
                ContentValues contentValues = new ContentValues();
                contentValues.put("log_source", str2);
                contentValues.put("reason", Integer.valueOf(reason2.getNumber()
                contentValues.put("events_dropped_count", Long.valueOf(j2));
                sQLiteDatabase.insert("log_event_dropped", null, contentValues)
            }
            return null;
        }
    });
}

@Override // com.google.android.datatransport.runtime.scheduling.persistence.Cl
public final void f() {
    j(new f(0, this));
}
```

Finding 2



```
public /* synthetic */ f(int i, Object obj) {
    this.f2450a = i;
    this.b = obj;
}

@Override // com.google.android.datatransport.runtime.scheduling.persistence.SQL
public final Object apply(Object obj) {
    Object obj2 = this.b;
    switch (this.f2450a) {
        case 0:
            SQLiteDatabase sQLiteDatabase = (SQLiteDatabase) obj;
            Encoding encoding = SQLiteEventStore.f2437m;
            SQLiteEventStore sQLiteEventStore = (SQLiteEventStore) obj2;
            sQLiteEventStore.getClass();
            sQLiteDatabase.compileStatement("DELETE FROM log_event_dropped").exe
            sQLiteDatabase.compileStatement("UPDATE global_log_event_state SET l
            return null;
        default:
            Cursor cursor = (Cursor) obj;
            Encoding encoding2 = SQLiteEventStore.f2437m;
            while (cursor.moveToNext()) {
                long j = cursor.getLong(0);
                HashMap hashMap = (HashMap) obj2;
                Set set = (Set) hashMap.get(Long.valueOf(j));
                if (set == null) {
                    set = new HashSet();
                    hashMap.put(Long.valueOf(j), set);
                }
                set.add(new SQLiteEventStore.Metadata(cursor.getString(1), curso
            }
            return null;
    }
```

We would like to clarify the following:

1. **Google-Provided Library Code**
   The referenced files (SQLiteEventStore.java and f.java) belong to Google's open-source libraries. We have **not** modified or customized these library classes.
2. **Input Validation in Our Application**
   While these classes handle certain data operations internally, our own application logic ensures that any user-supplied or external data is properly **validated and sanitized** before being passed to these library methods. We do not allow **untrusted or potentially malicious inputs** to reach sensitive routines.
3. **Adherence to MSTG-PLATFORM-2**
   We adopt a **defensive approach** , validating all incoming data—whether from UI inputs, intents, custom URLs, or network sources—to reduce the risk of injection attacks or other security breaches. Our usage of these Google libraries does not override or bypass any existing security checks.
4. **Regular Updates and Security Monitoring**
   We consistently **update** our dependencies (including Google libraries) to ensure we receive the **latest security patches** and follow best practices.

After further analysis the application seems to comply successfully with the requirement.

MSTG-PLATFORM-3

The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-1

The app is signed and provisioned with a valid certificate, of which the private key is properly protected.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-2

The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).

After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-3

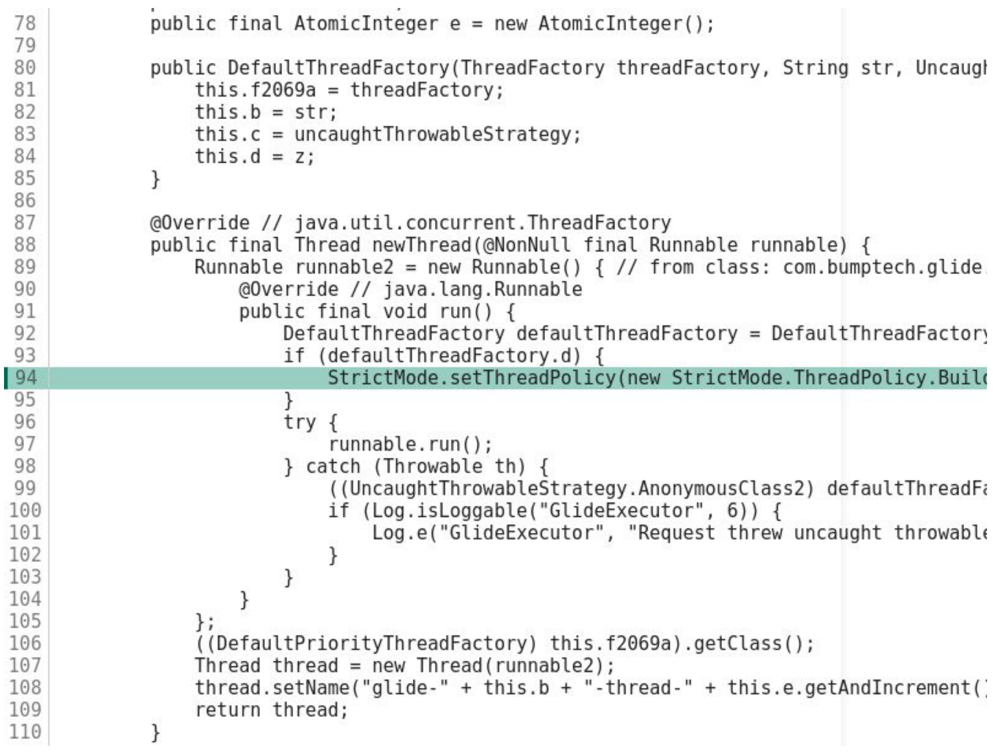
Debugging symbols have been removed from native binaries.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-4

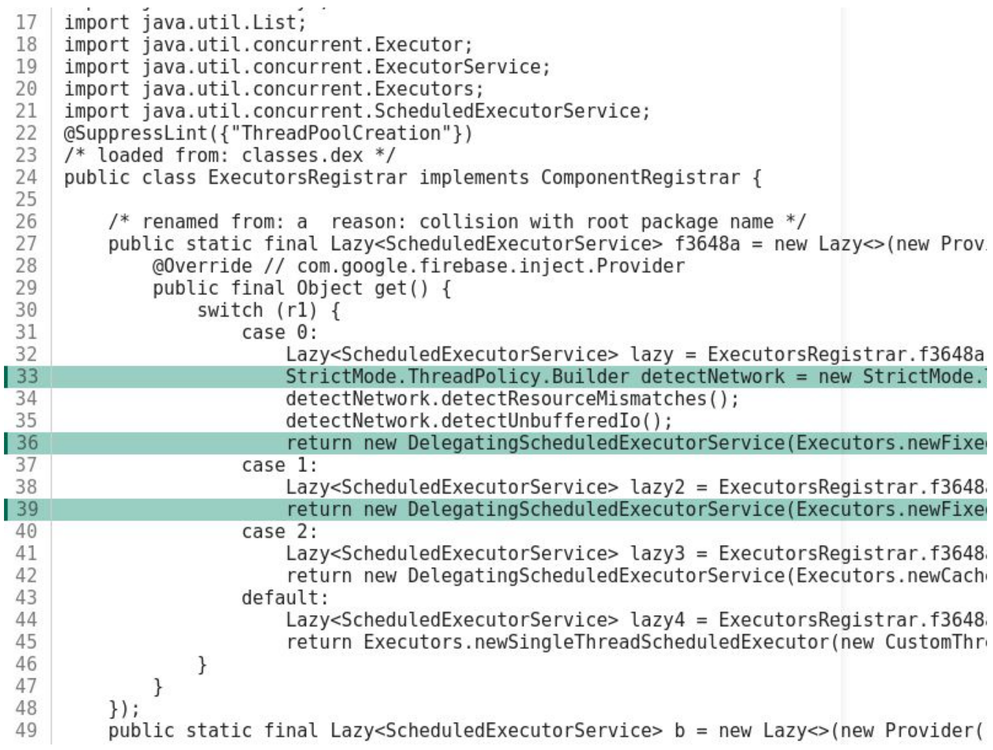
Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.

Finding 1



```
    public final AtomicInteger e = new AtomicInteger();

    public DefaultThreadFactory(ThreadFactory threadFactory, String str, Uncaugl
        this.f2069a = threadFactory;
        this.b = str;
        this.c = uncaughtThrowableStrategy;
        this.d = z;
    }

    @Override // java.util.concurrent.ThreadFactory
    public final Thread newThread(@NonNull final Runnable runnable) {
        Runnable runnable2 = new Runnable() { // from class: com.bumptech.glide
            @Override // java.lang.Runnable
            public final void run() {
                DefaultThreadFactory defaultThreadFactory = DefaultThreadFactory
                if (defaultThreadFactory.d) {
                    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Buil
                }
                try {
                    runnable.run();
                } catch (Throwable th) {
                    ((UncaughtThrowableStrategy.AnonymousClass2) defaultThreadFa
                    if (Log.isLoggable("GlideExecutor", 6)) {
                        Log.e("GlideExecutor", "Request threw uncaught throwable
                    }
                }
            }
        };
        ((DefaultPriorityThreadFactory) this.f2069a).getClass();
        Thread thread = new Thread(runnable2);
        thread.setName("glide-" + this.b + "-thread-" + this.e.getAndIncrement(
        return thread;
    }
```

Finding 2



```
import java.util.List;
import java.util.concurrent.Executor;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
@SuppressLint({"ThreadPoolCreation"})
/* loaded from: classes.dex */
public class ExecutorsRegistrar implements ComponentRegistrar {

    /* renamed from: a  reason: collision with root package name */
    public static final Lazy<ScheduledExecutorService> f3648a = new Lazy<>(new Prov
        @Override // com.google.firebase.inject.Provider
        public final Object get() {
            switch (r1) {
                case 0:
                    Lazy<ScheduledExecutorService> lazy = ExecutorsRegistrar.f3648a
                    StrictMode.ThreadPolicy.Builder detectNetwork = new StrictMode.
                    detectNetwork.detectResourceMismatches();
                    detectNetwork.detectUnbufferedIo();
                    return new DelegatingScheduledExecutorService(Executors.newFixe
                case 1:
                    Lazy<ScheduledExecutorService> lazy2 = ExecutorsRegistrar.f3648
                    return new DelegatingScheduledExecutorService(Executors.newFixe
                case 2:
                    Lazy<ScheduledExecutorService> lazy3 = ExecutorsRegistrar.f3648
                    return new DelegatingScheduledExecutorService(Executors.newCach
                default:
                    Lazy<ScheduledExecutorService> lazy4 = ExecutorsRegistrar.f3648
                    return Executors.newSingleThreadScheduledExecutor(new CustomThr
            }
        }
    });
    public static final Lazy<ScheduledExecutorService> b = new Lazy<>(new Provider(
```

We would like to clarify the following:

1. **GlideExecutor.java (Glide Library)**
    - This file is part of the open-source Glide library, which is used for **efficient image loading and caching** to ensure smooth scrolling and optimized performance in our application.
    - The referenced sections in GlideExecutor may contain **StrictMode** checks intended to detect unintended network operations on the main (UI) thread. This is a **standard Android best practice** to prevent performance bottlenecks (ANRs), not a debugging backdoor.
    - We do **not** include any verbose logs, hidden test code, or developer backdoors in our **production** version.
2. **ExecutorsRegistrar.java (Google Firebase Library)**
    - This class appears to belong to the **Google Firebase** framework, which provides background task management and other cloud-related services.
    - Similar to Glide, the flagged StrictMode usage helps identify inappropriate main-thread network calls. It is not a hidden or debug feature.
    - Our app does **not** introduce or utilize verbose logging, developer backdoors, or debug statements beyond what is standard in these libraries.

Conclusion

Our production build does **not** contain extra debugging code, test routines, or hidden settings. The StrictMode checks in these third-party libraries are purely for performance and best-practice compliance.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-9

Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.

After further analysis the application seems to comply successfully with the requirement.