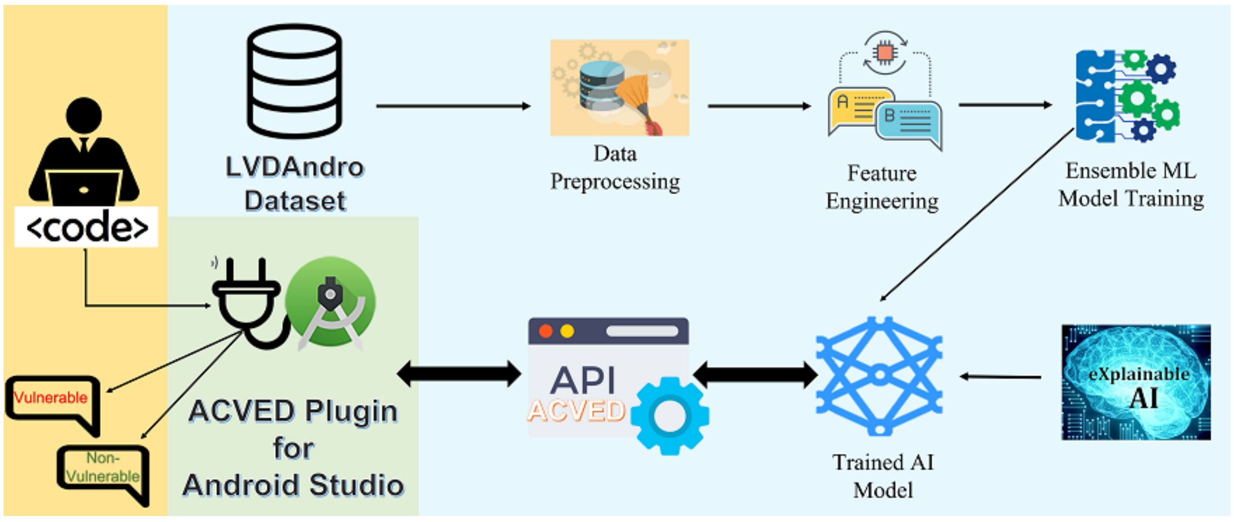
Android Code Vulnerability Early Detection (ACVED) Plugin- User Guide

Due to the constant and growing demands of users, as well as their ever-changing needs, Android applications are being rapidly released and updated. However, in the rush to develop these apps, the focus may be more on functionality rather than security and identifying vulnerabilities in the source code. This is partly because there are not enough automated mechanisms available to assist app developers in mitigating these vulnerabilities.

To address this issue, an AI-powered plugin called Android Code Vulnerability Early Detection (ACVED) can be integrated with Android Studio to provide real-time support for mitigating source code vulnerabilities. As you work on a specific source code line, the plugin can provide the vulnerability status for that line.

ACVED has a highly accurate and efficient ensemble learning model running in the backend, which can detect source code vulnerabilities and their CWE categories with a 95% accuracy rate. Additionally, explainable AI techniques are employed to provide source code vulnerability prediction probabilities for each word.

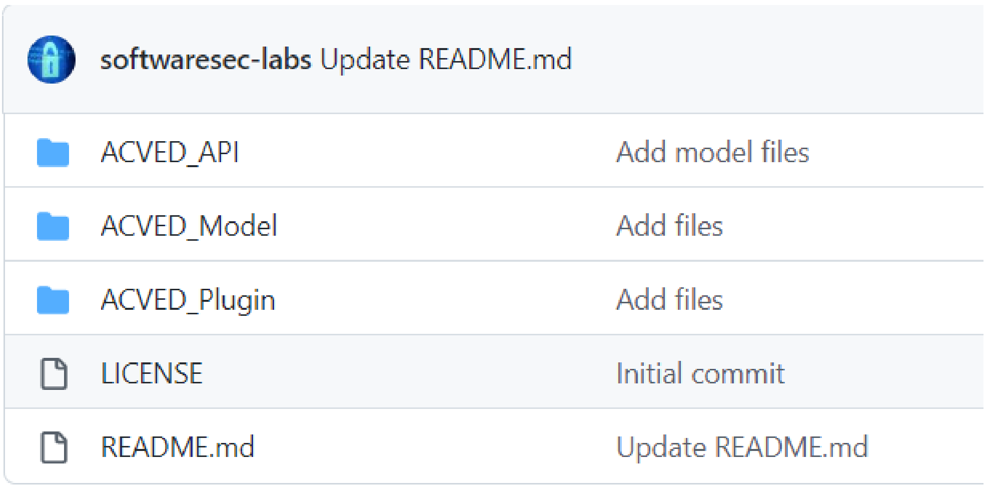
The model is regularly updated with new training data from the LVDAndro dataset, which allows for the detection of novel vulnerabilities using the ACVED plugin.



Installation Instructions

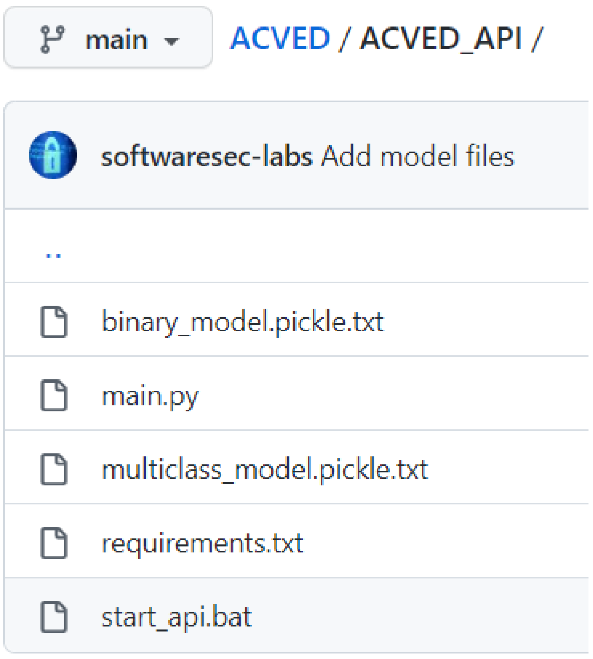
Downloading

The latest version of the ACVED plugin and the API can be downloaded from https://github.com/softwaresec-labs/ACVED



softwaresec-labs Update README.md	
ACVED_API	Add model files
ACVED_Model	Add files
ACVED_Plugin	Add files
LICENSE	Initial commit
README.md	Update README.md

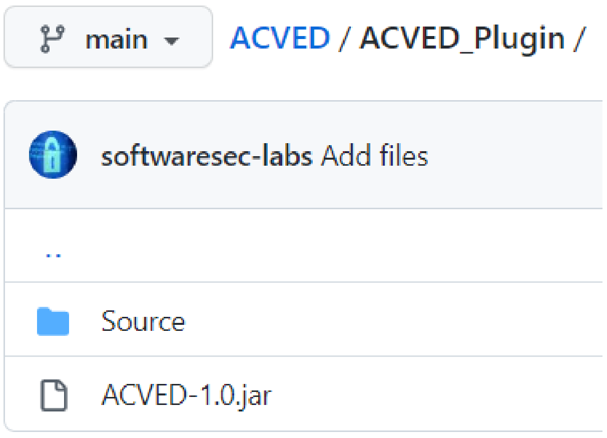
You have to use the files in the ACVED API and the ACVED Plugin directories.



main ACVED / ACVED_API /

softwaresec-labs Add model files

- ..
- binary_model.pickle.txt
- main.py
- multiclass_model.pickle.txt
- requirements.txt
- start_api.bat



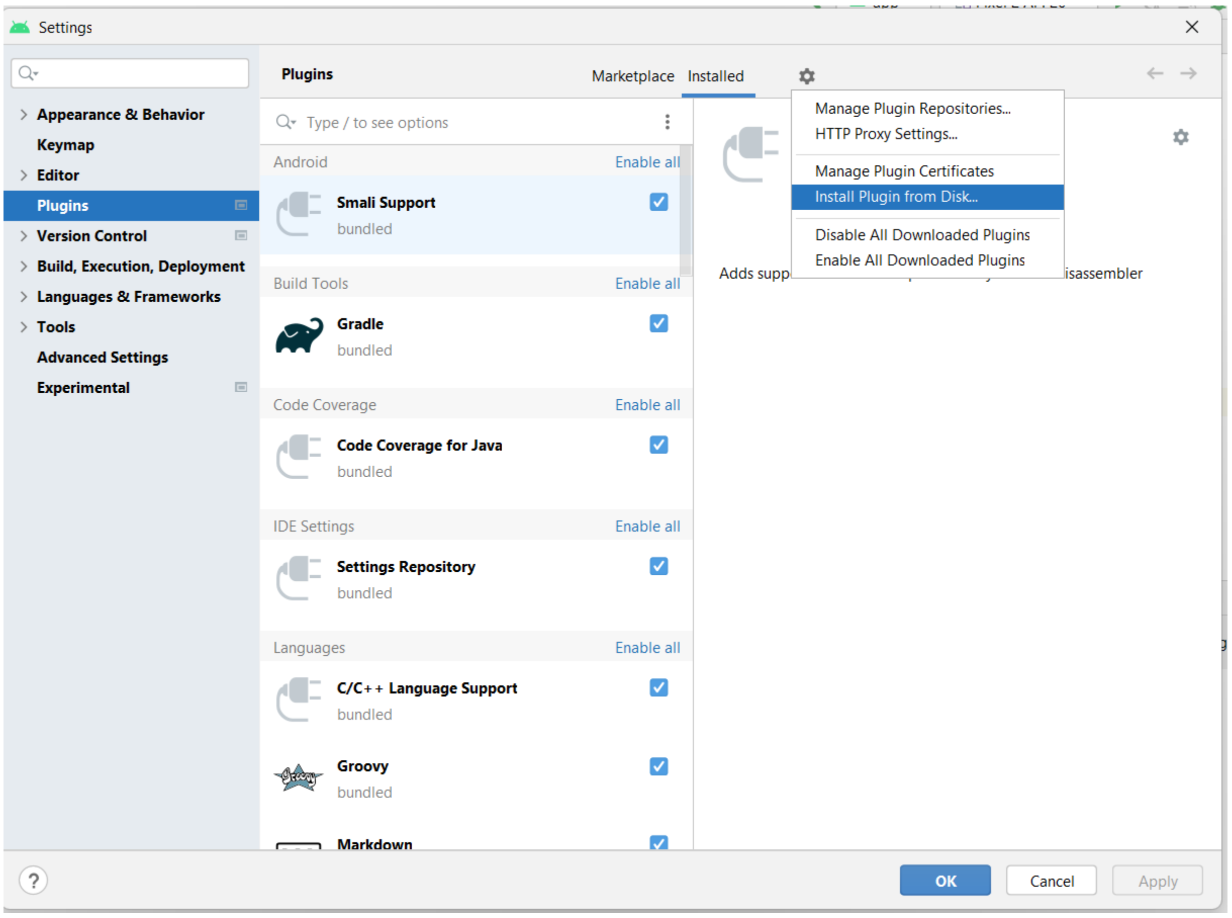
main ACVED / ACVED_Plugin /

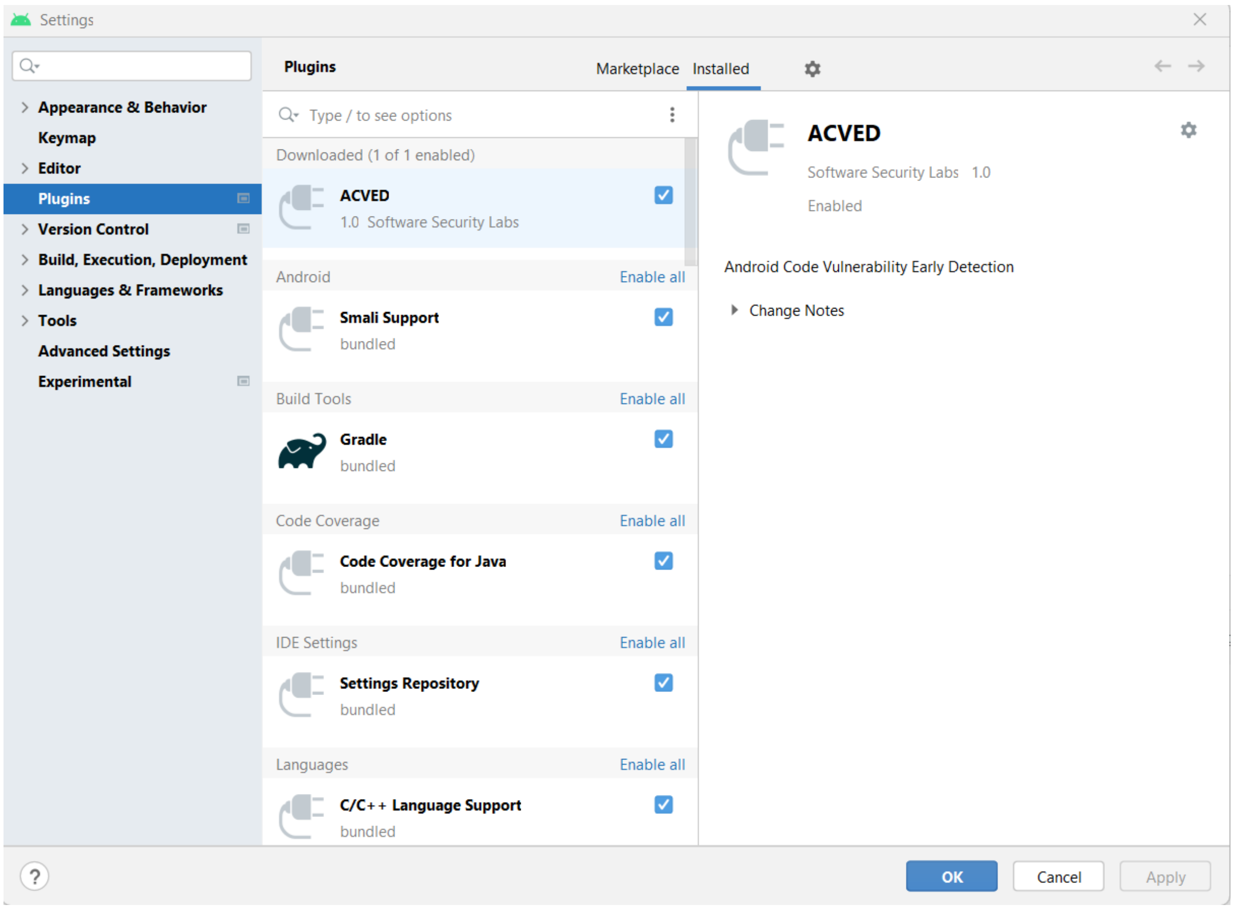
softwaresec-labs Add files

- ..
- Source
- ACVED-1.0.jar

Installing Plugin

The plugin file is available in ACVED_Plugin directory named as ACVED-1.0.jar. You can install the plugin to the Android studio latest version (tested on 2021.2.1) by going to File → Settings and go to the preference section. Then select the jar file by going to the Install Plugin from Disk option. This is shown in the below figure. After that you need to restart the Android Studio to finalise the plugin installation.

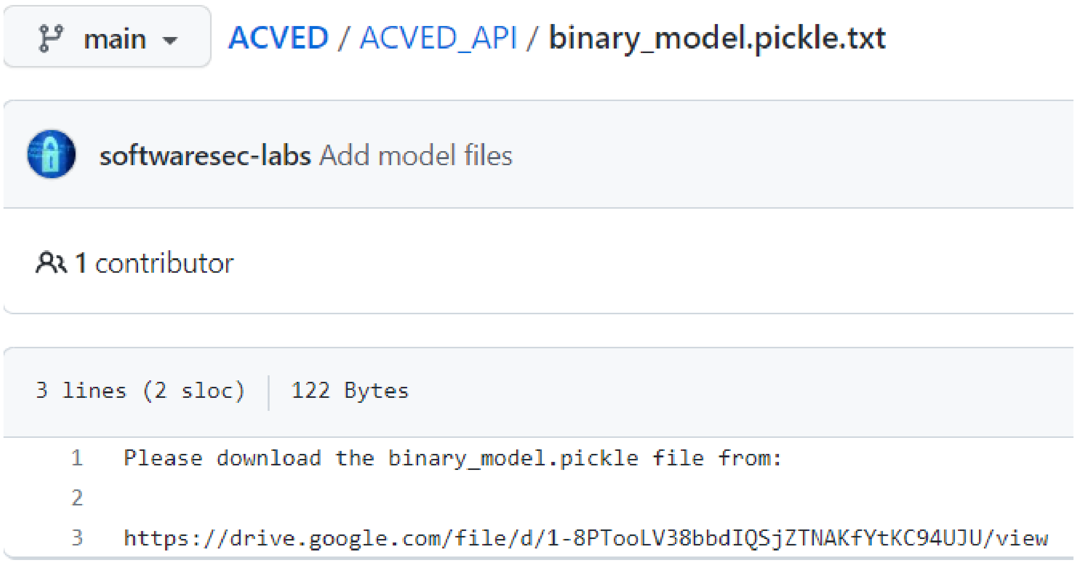




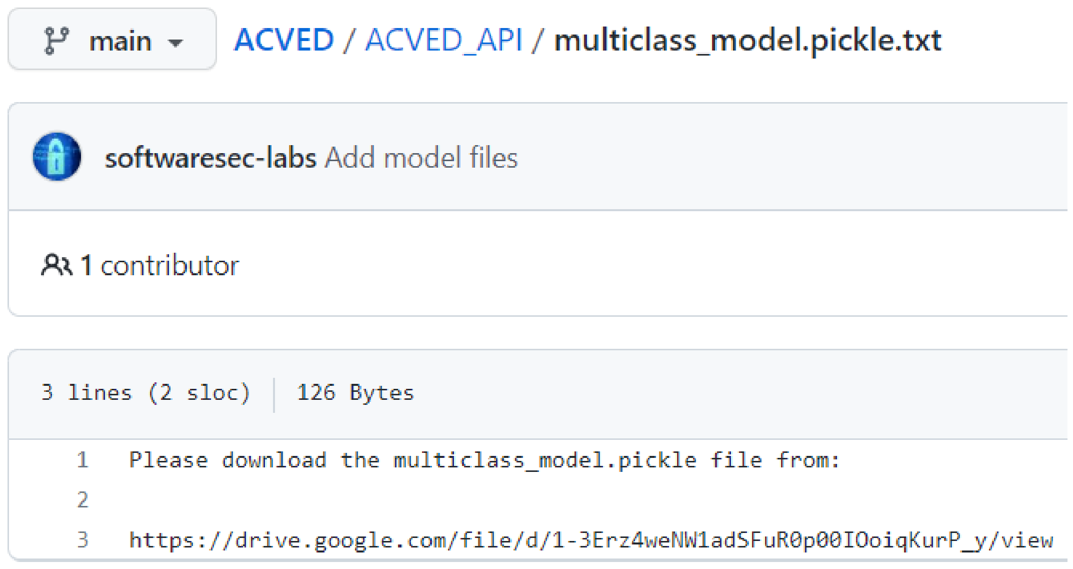
Using the ACVED Plugin

Starting ACVED API

You can find the ACVED API in the ACVED_API directory. Please note that you have to download the model files from the links provided in the binary_model.pickle.txt file and the multiclass_model.picke.txt file and copy them to your local folder where ACVED_API located.


main
ACVED / ACVED_API / binary_model.pickle.txt
softwaresec-labs Add model files
1 contributor
3 lines (2 sloc) | 122 Bytes

```
Please download the binary_model.pickle file from:

https://drive.google.com/file/d/1-8PTooLV38bbdIQSjZTNAKfYtKC94UJU/view
```


main

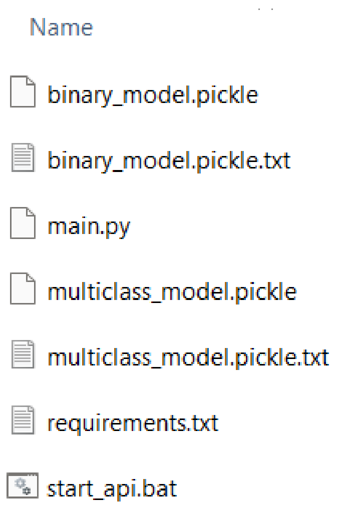
ACVED / ACVED_API / multiclass_model.pickle.txt

softwaresec-labs Add model files

1 contributor

```
3 lines (2 sloc) | 126 Bytes

Please download the multiclass_model.pickle file from:

https://drive.google.com/file/d/1-3Erz4weNW1adSFuR0p00IOoiqKurP_y/view
```

Once you have copied those model files, your ACVED_API folder should look like below.


Name

- binary_model.pickle
- binary_model.pickle.txt
- main.py
- multiclass_model.pickle
- multiclass_model.pickle.txt
- requirements.txt
- start_api.bat

ACVED API is a python-based web API. However, you do not need to have an understanding of python in order to execute it. However you need to have python3 installed in your computer. API can be started by executing the start_api.bat file available in ACVED_API directory.

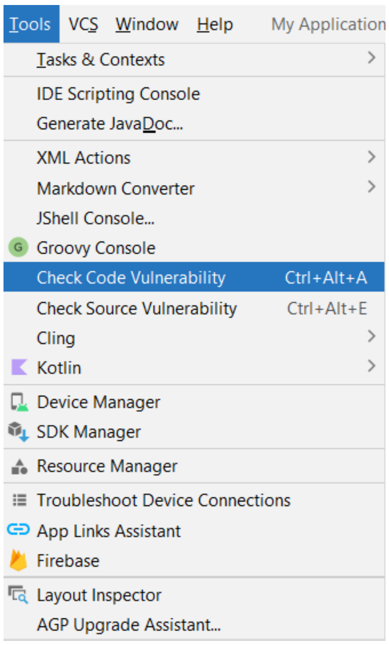
Detecting the Vulnerable Code Lines

The ACVED plugin provides two options to detect vulnerabilities.

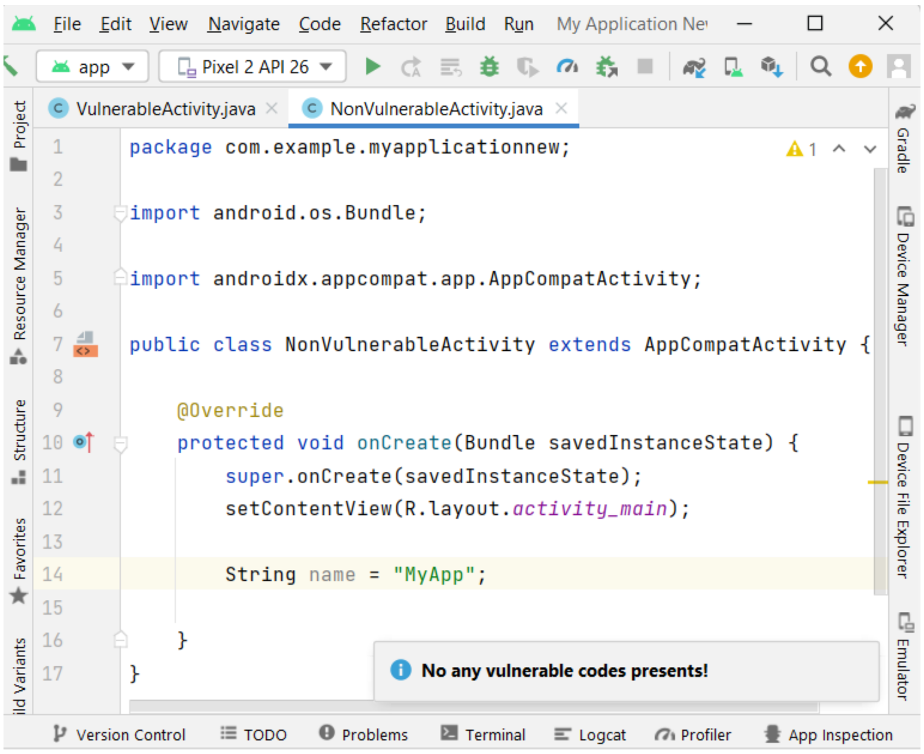
- Quick Check
  - Scan the whole source code file to detect the presence of vulnerable source code.
- Detailed Check
  - Detect if any vulnerability is associated with a particular code line.

The quick check option can be activated by following Tools → Check Source Vulnerability or by pressing CTRL+ALT+E in the Android Studio. Once this option is used, the developer will be notified of the vulnerable code lines and their CWE-IDs. It is a quick search to detect vulnerabilities at once.

The detailed check option can be activated by following Tools → Check Code Vulnerability or by pressing CTRL+ALT+A while the cursor has focused on a particular code line. These options are shown in below figure.



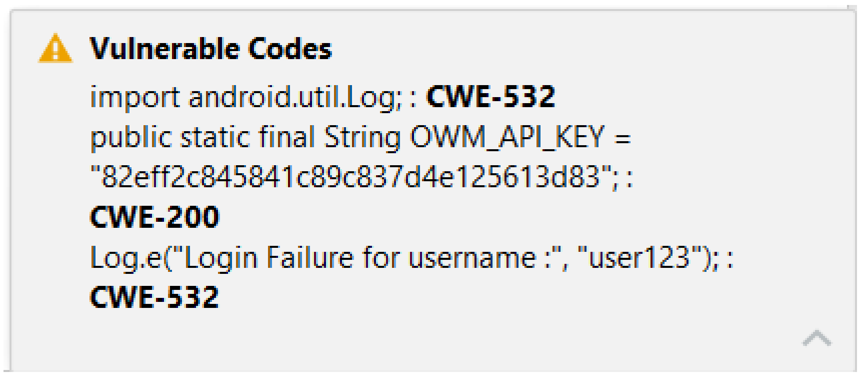
The balloon notification from the ACVED plugin indicates the results received from the API. Once a quick search has been performed, the developer gets a balloon notification with a status of vulnerable codes presence in the source file. A notification will be received as in below Figure if there are no vulnerable codes.



```
package com.example.myapplicationnew;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

public class NonVulnerableActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        String name = "MyApp";

    }
}
```

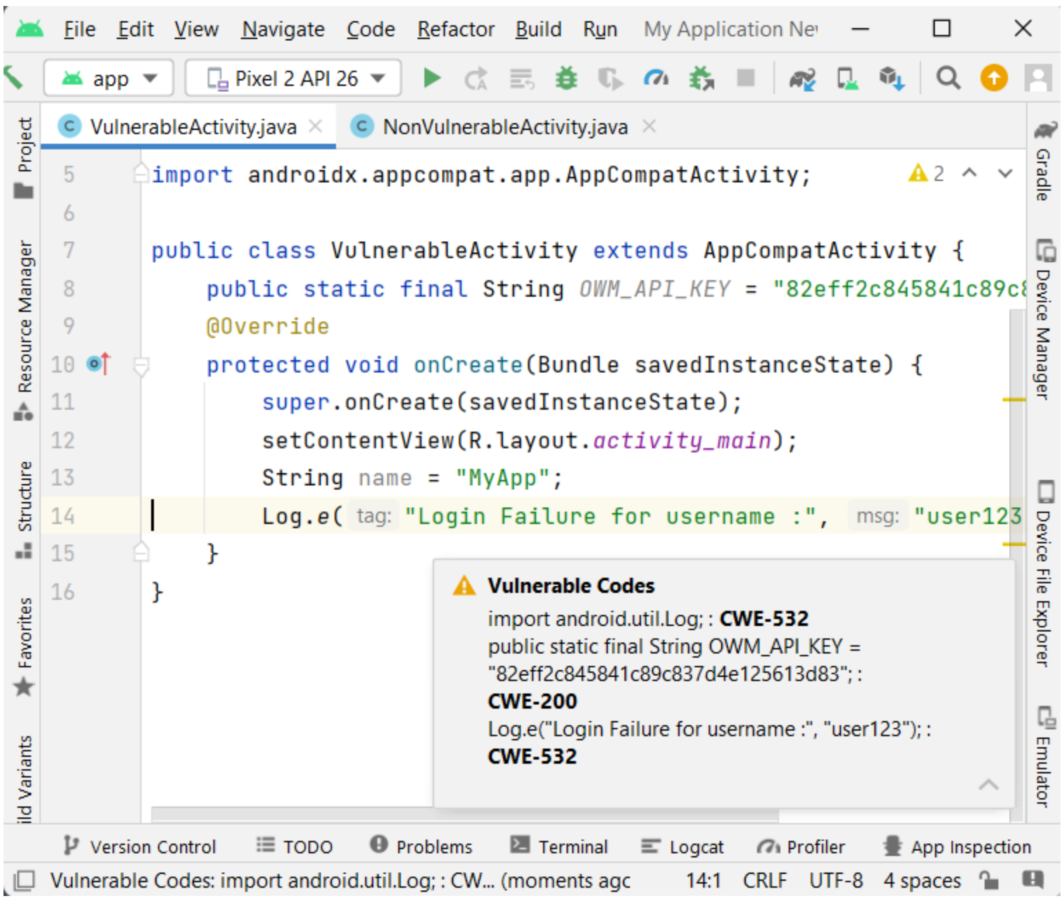
No any vulnerable codes presents!

If vulnerable codes exist, a notification similar to below figures will be received by indicating vulnerable code lines and their CWE-IDs.



Vulnerable Codes

import android.util.Log; : **CWE-532**
public static final String OWM_API_KEY = "82eff2c845841c89c837d4e125613d83"; : **CWE-200**
Log.e("Login Failure for username :", "user123"); : **CWE-532**

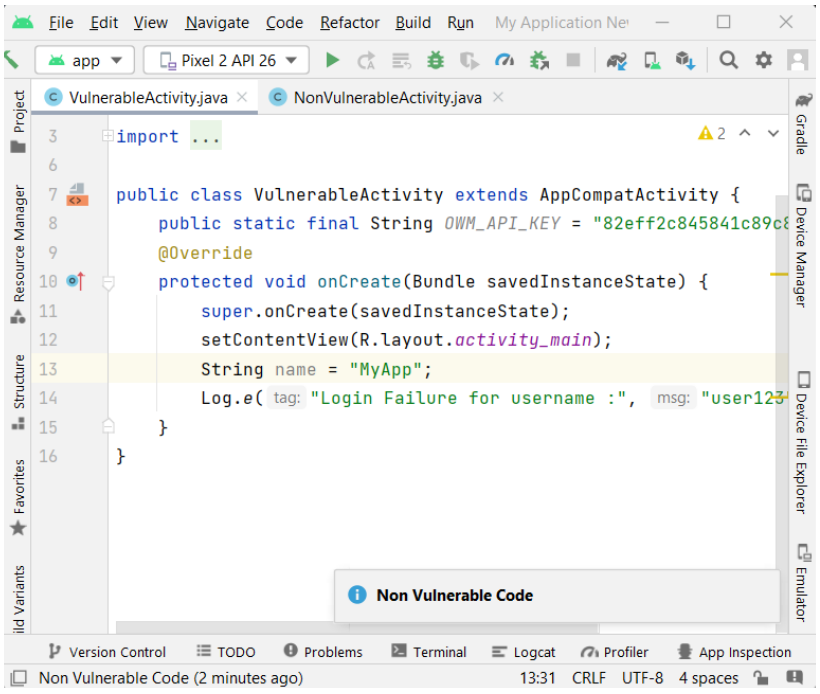


```
import androidx.appcompat.app.AppCompatActivity;

public class VulnerableActivity extends AppCompatActivity {
    public static final String OWM_API_KEY = "82eff2c845841c89c8
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String name = "MyApp";
        Log.e( tag: "Login Failure for username :",  msg: "user123
    }
}
```

Vulnerable Codes
import android.util.Log; : **CWE-532**
public static final String OWM_API_KEY = "82eff2c845841c89c837d4e125613d83"; : **CWE-200**
Log.e("Login Failure for username :", "user123"); : **CWE-532**

Vulnerable Codes: import android.util.Log; : CW... (moments agc   14:1 CRLF UTF-8 4 spaces

When performing a detailed check, a notification will be received with the source code's vulnerability status. A notification, as in below figure, will be received if the code is not vulnerable. The focus is for the String name = "MyApp" statement.



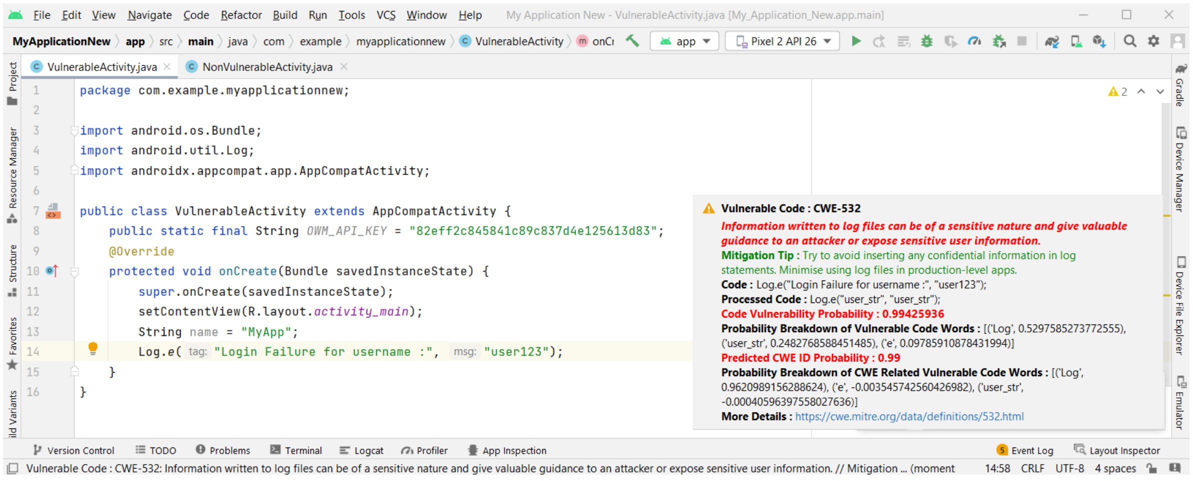
```
import ...

public class VulnerableActivity extends AppCompatActivity {
    public static final String OWM_API_KEY = "82eff2c845841c89c8
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String name = "MyApp";
        Log.e( tag: "Login Failure for username :",  msg: "user123
    }
}
```

Non Vulnerable Code

Non Vulnerable Code (2 minutes ago)   13:31 CRLF UTF-8 4 spaces

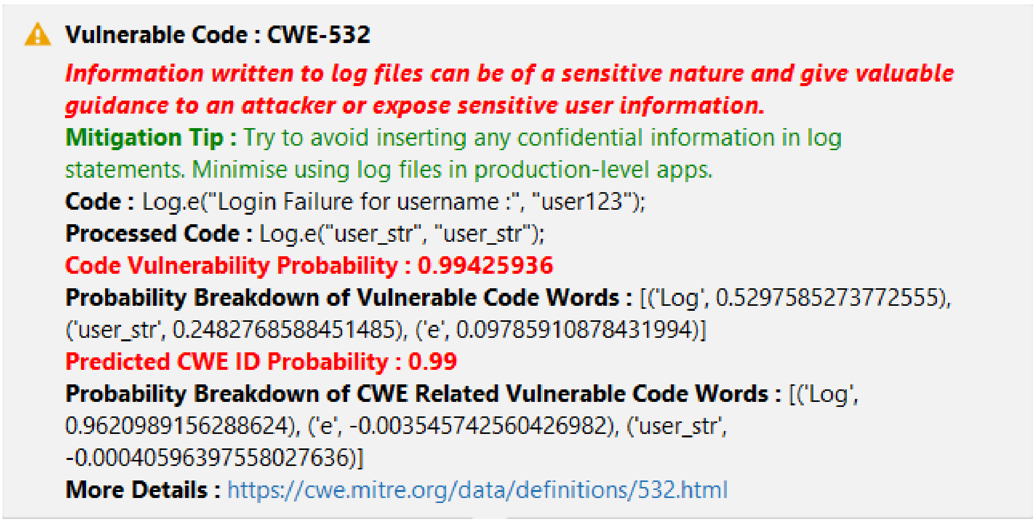
If the cursor-focused code is vulnerable, a description of the vulnerability and a tip for mitigating it will be provided as a balloon notification along with the source code and its processed version. The probability of vulnerability status (binary class) prediction, the CWE-ID, and the prediction probability of the CWE category (multi-class classification prediction) are also included in the notification.

Moreover, how each word contributes to the probability in both binary and multi-class classification approaches is also indicated. The notification type will also vary (information or warning) according to the severity of the vulnerability. Additionally, to provide more details of the vulnerability, ACVED will suggest how to overcome it by referring to the CWE repository. An example execution of a detailed check on the vulnerable code line is shown in below figures. The cursor has the focus on Log.e("Login Failure for username :","user123"); statement in this example.



```java
package com.example.myapplicationnew;

import android.os.Bundle;
import android.util.Log;
import androidx.appcompat.app.AppCompatActivity;

public class VulnerableActivity extends AppCompatActivity {
    public static final String OWM_API_KEY = "82eff2c845841c89c837d4e125613d83";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String name = "MyApp";
        Log.e( tag: "Login Failure for username :",  msg: "user123");
    }
}
```

Vulnerable Code : CWE-532: Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information. // Mitigation ... (moment



Vulnerable Code : CWE-532

Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information.

Mitigation Tip : Try to avoid inserting any confidential information in log statements. Minimise using log files in production-level apps.

Code : Log.e("Login Failure for username :", "user123");

Processed Code : Log.e("user_str", "user_str");

Code Vulnerability Probability : 0.99425936

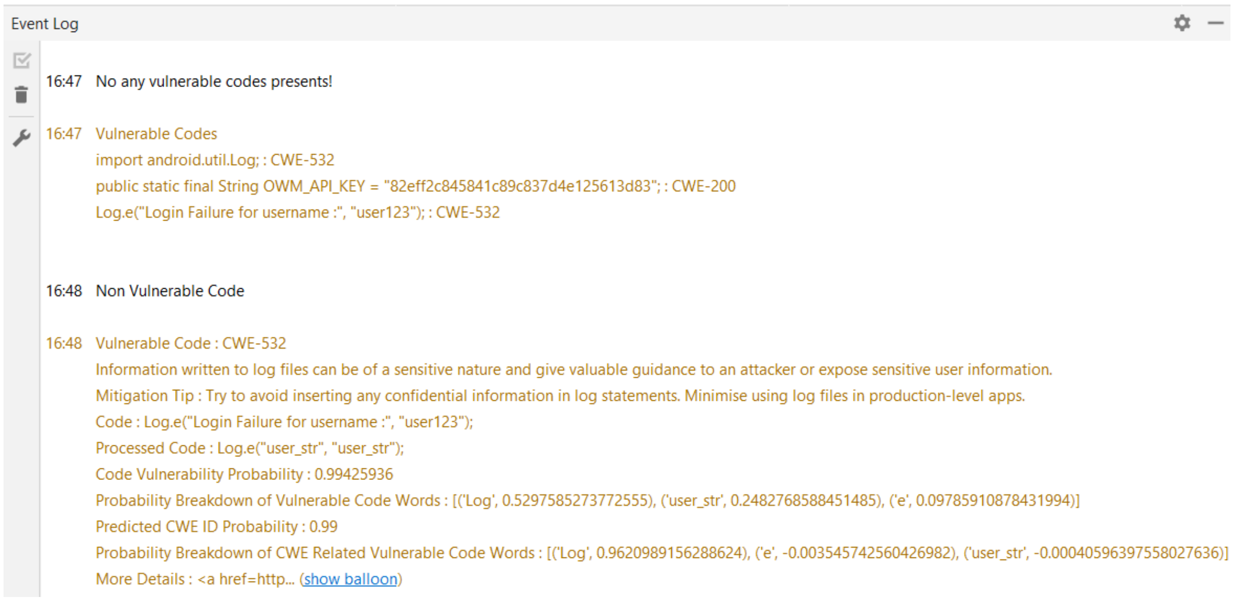
Probability Breakdown of Vulnerable Code Words : [('Log', 0.5297585273772555), ('user_str', 0.2482768588451485), ('e', 0.097859108784319994)]

Predicted CWE ID Probability : 0.99

Probability Breakdown of CWE Related Vulnerable Code Words : [('Log', 0.9620989156288624), ('e', -0.003545742560426982), ('user_str', -0.00040596397558027636)]

More Details : https://cwe.mitre.org/data/definitions/532.html

The Android Studio Event Log is also updated when these executions happen, as shown in Figure.



Event Log

```
16:47   No any vulnerable codes presents!

16:47   Vulnerable Codes
        import android.util.Log; : CWE-532
        public static final String OWM_API_KEY = "82eff2c845841c89c837d4e125613d83"; : CWE-200
        Log.e("Login Failure for username :", "user123"); : CWE-532


16:48   Non Vulnerable Code

16:48   Vulnerable Code : CWE-532
        Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information.
        Mitigation Tip : Try to avoid inserting any confidential information in log statements. Minimise using log files in production-level apps.
        Code : Log.e("Login Failure for username :", "user123");
        Processed Code : Log.e("user_str", "user_str");
        Code Vulnerability Probability : 0.99425936
        Probability Breakdown of Vulnerable Code Words : [('Log', 0.5297585273772555), ('user_str', 0.2482768588451485), ('e', 0.097859108784319 94)]
        Predicted CWE ID Probability : 0.99
        Probability Breakdown of CWE Related Vulnerable Code Words : [('Log', 0.9620989156288624), ('e', -0.003545742560426982), ('user_str', -0.00040596397558027636)]
        More Details : <a href=http... (show balloon)
```