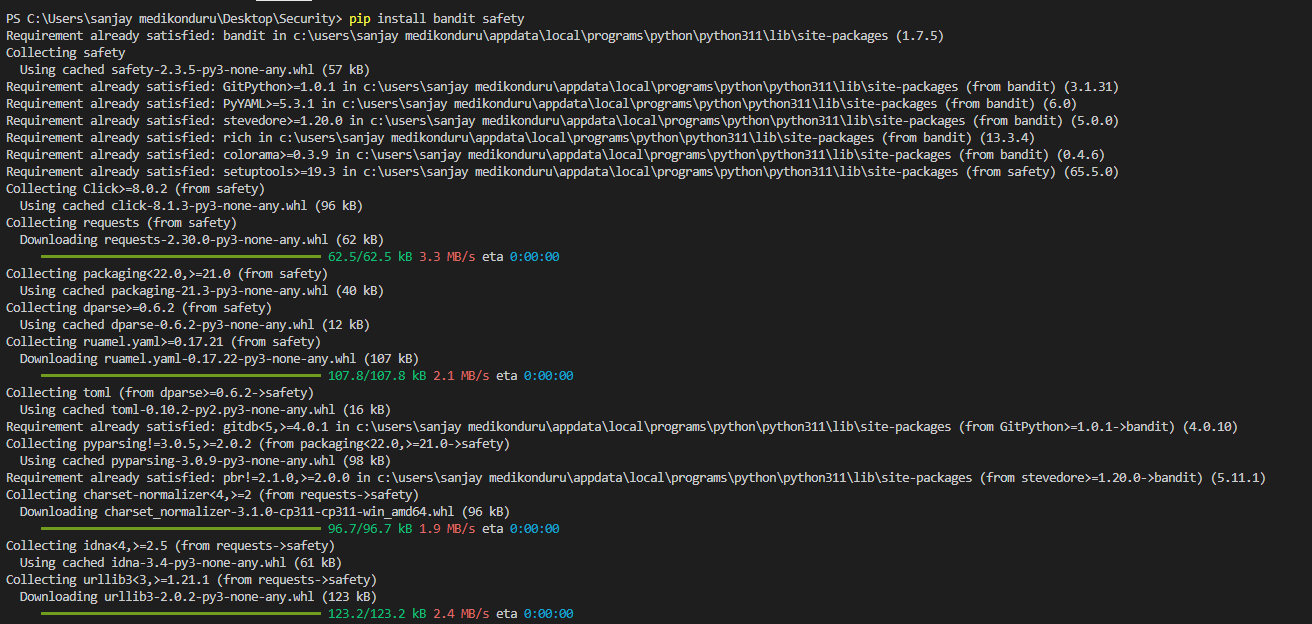
**User Guide for Bandit:**

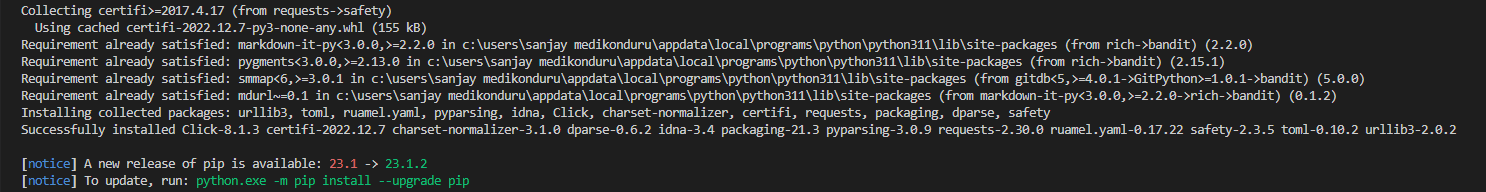
**Introduction:**

Bandit is an open-source vulnerability scanning tool that is designed to find security issues in python. The way the tool does this is by processing each file, building an AST (Application Security Testing), and then runs the plugins against those AST nodes. Bandit then provides the user with a vulnerability report against the source code after scanning through all those files.

**How to Install Bandit:**

Bandit is a pip-download. To download the tool, open up terminal and type the command:

**Command: pip install bandit safety**

****

**How to use Bandit:**

To use the tool, firstly a python script must be written and saved for scanning. Then once the python file is saved, and ready to be scanned, go to the terminal, and use the bandit command:

**Command: bandit “python file”**

****

Bandit then generates a report with any possible security threats.

For example:

**Example One)**

This is a sample python code that takes a number ‘n’ as an input and returns the factorial of that input using recursion

A picture containing text, screenshot, display, software

Description automatically generated

Bandit is then used to generate a vulnerability report for the code that’s provided. We can see that Bandit has seen no vulnerabilities within the code, and no security issues were found.

A screenshot of a computer program

Description automatically generated with medium confidence

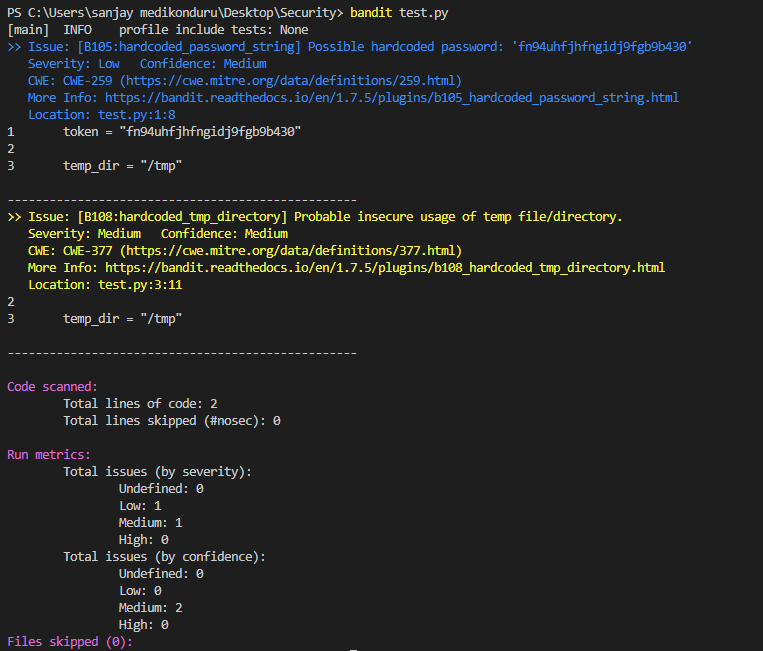
**Example Two)**

Here is an example of a Python code, that is random and consists of random inputs

A screenshot of a computer program

Description automatically generated with medium confidence

When Bandit is used again for this code, it generates a report with a list of possible vulnerabilities within the code.



Bandit here has reported the possible vulnerabilities that may exist within the source code. The two main problems it identified are:

Example One)

fn94uhfjhfngidj9fgb9b430

Bandit classifies this line as a possible hard codded password. A hard codded password can create a significant pathway that allows the attacker to bypass any authentication.

Example Two)

temp\_dir = "/tmp"

Bandit has classified this line as a possible hardcoded directory, and there maybe a misuse of the temp file and directory.

**How does Bandit classify how dangerous a vulnerability is:**

Bandit classifies the security level into two categories, one being the severity of the possible vulnerability, and then the confidence (accuracy of the detection) of the possible vulnerability.

A screenshot of a computer screen

Description automatically generated with medium confidence

In this bandit report from before we can see that the level severity of the attacks. With one vulnerability having a low-level severity, and the other being a mid-level severity.

We can also see the confidence of the vulnerability which is referring to the accuracy of the vulnerability detection, with both possible vulnerabilities being mid-level.

**How to filter the severity and confidence of the possible vulnerabilities in Bandit?**

Bandit has specific commands that can generate reports according to the level of severity and confidence.

**Filtering Severity:**

Command: Bandit “python file” -lll | **Reporting high-level severity vulnerabilities**

Command: Bandit “python file” -ll | **Reporting mid-level severity vulnerabilities**

Command: Bandit “python file” -l | **Reporting low-level severity vulnerabilities**

**Example)**

**A screenshot of a computer program

Description automatically generated with medium confidence**

The above report is for the same source code used in previous examples however we can see that when used the filter -lll, it only reports any high-level possible vulnerabilities, and we can clearly see there are none.

**Filtering Confidence:**

Command: Bandit “python file” -iii| **Reporting high-level Confidence vulnerabilities**

Command: Bandit “python file” -ii| **Reporting mid-level Confidence vulnerabilities**

Command: Bandit “python file” -i| **Reporting low-level Confidence vulnerabilities**

A picture containing text, screenshot

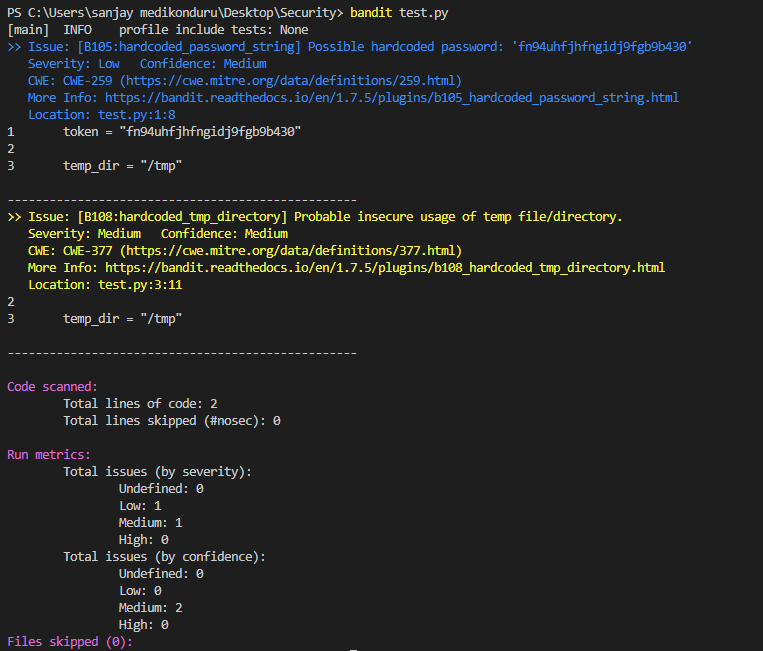
Description automatically generated**Example Two)**

Looking at the example here we can see that Bandit reports no vulnerabilities with high confidence.

**Bandit command to ignore a specific vulnerability:**

If there is a specific part or line of the code where programmers know is not a possible vulnerability, Bandit has a command to ignore that part from the final report:

Example:



Looking at the above image if we know that the hard codded password is not a possible vulnerability, we can use the nosec command that lets bandit ignore that part of the code. Bandit classifies the hardcoded password string as B105, so in order to ignore it we used the command:



# nosec : B105

This command would give us the result:

**A screenshot of a computer

Description automatically generated**

The example here doesn’t report to the hard codded password string vulnerability from before as we have advised bandit to ignore the vulnerability.

**Then End**