# Configuration Manual

MSc Research Project
Cyber Security

Vivek Singh Gusain
Student ID: 22212035

School of Computing
National College of Ireland

Supervisor: Raza Ul Mustafa

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | | | |
|---|---|---|---|
| **Student Name:** | Vivek Singh Gusain | | |
| **Student ID:** | 22212035 | | |
| **Programme:** | Masters in Cyber Security | **Year:** | 2023-2024 |
| **Module:** | MSc Research Practicum | | |
| **Lecturer:** | Raza Ul Mustafa | | |
| **Submission Due Date:** | 19th August 2024 | | |
| **Project Title:** | A Hybrid Approach to Generate Severity Scores for Prioritization of Vulnerabilities | | |
| **Word Count:** | 1102 | | |
| **Page Count:** | 10 | | |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Vivek Singh Gusain

**Date:** 19th August 2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project,** both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

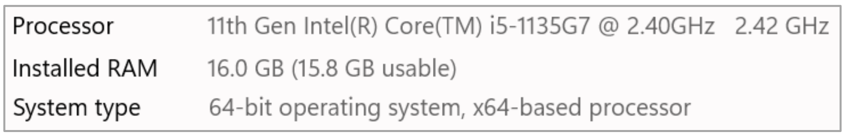
Vivek Singh Gusain
Student ID: 22212035

## 1 Introduction

This manual provides the details on the configurations required to implement the proposed methodology, VISERS. Section 2 mentions the device and software specifications used to run the python code implementing the algorithm. Section 3 describes the code written to find out all the possible values of exploitability with the value of Scope metrics as changed and unchanged. Section 4 describes the code written to implement the proposed algorithm along with the other algorithms used. Lastly, section 5 describes the output of the code.

## 2 System Configurations

### 2.1 Device Specification

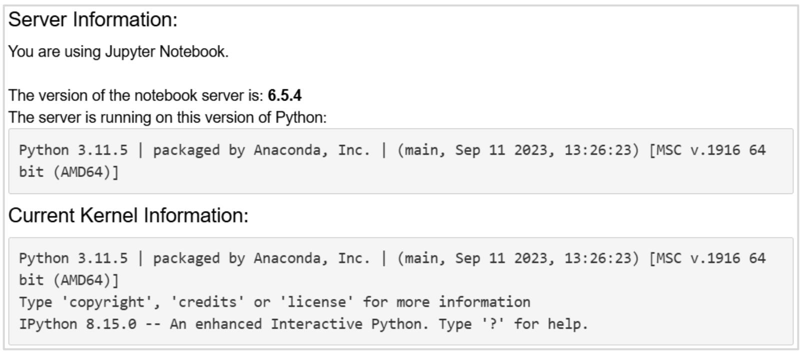


| | |
|---|---|
| Processor | 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz |
| Installed RAM | 16.0 GB (15.8 GB usable) |
| System type | 64-bit operating system, x64-based processor |

**Fig. 1 Device details**

### 2.2 Software Specification

Jupyter notebook was used to write the python code via Anaconda navigator. Fig. 1 shows the versions of the software utilized. (Anaconda, 2024)



Server Information:

You are using Jupyter Notebook.

The version of the notebook server is: **6.5.4**
The server is running on this version of Python:

```
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64
bit (AMD64)]
```

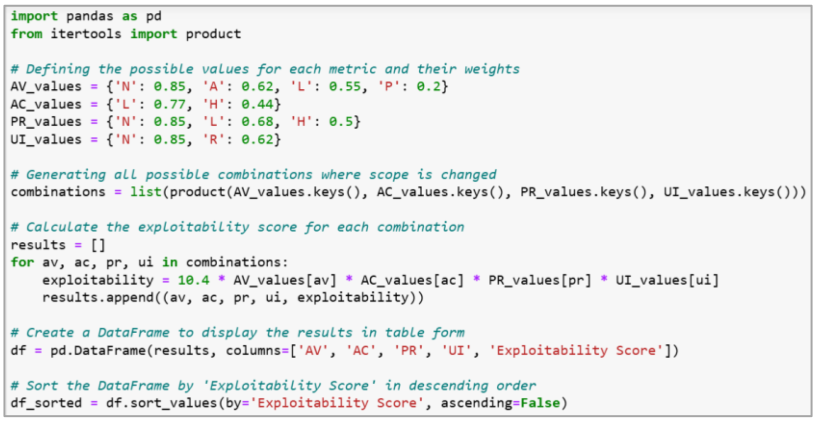
Current Kernel Information:

```
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64
bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.15.0 -- An enhanced Interactive Python. Type '?' for help.
```

**Fig. 2 Software information**

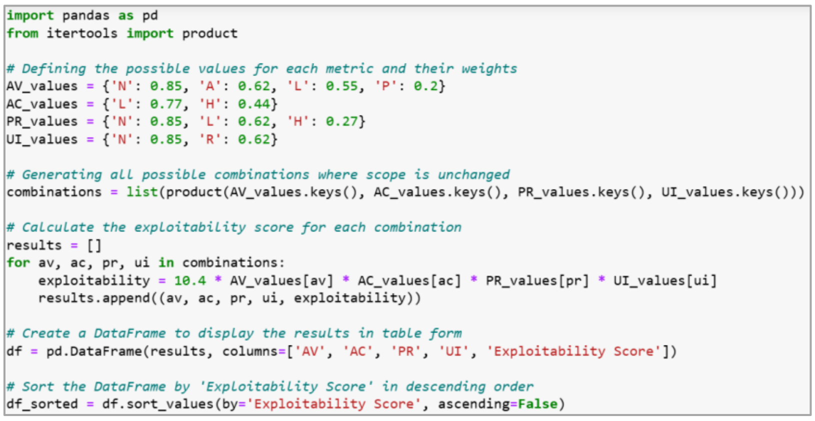
## 3 Calculating Possible Values of Exploitability

VISERS calculate the score using both qualitative and quantitative methods. For impact, defining the qualitative was easy as there are only three metrics involved in the configuration. But for exploitability, there are 5 metrics that play role in measuring exploitability. The metrics are access complexity, access vector, privileges required, user interaction, and scope. For privileges required metric, different weights are assigned with respect to the scope value. Scope value could be "changed" or "unchanged". This makes 48 possible combinations for each case to be calculated. To do the calculations, two sets of code were written. Fig. 3 shows the snippet of the code to calculate the values where the scope value is "changed" and fig. 4 shows the code where the scope is "unchanged". Next section will address the use of it. These files are saved with the names: *Possible values of Exploitability (S=C).ipynb* and *Possible values of Exploitability (S=U).ipynb*



```python
import pandas as pd
from itertools import product

# Defining the possible values for each metric and their weights
AV_values = {'N': 0.85, 'A': 0.62, 'L': 0.55, 'P': 0.2}
AC_values = {'L': 0.77, 'H': 0.44}
PR_values = {'N': 0.85, 'L': 0.68, 'H': 0.5}
UI_values = {'N': 0.85, 'R': 0.62}

# Generating all possible combinations where scope is changed
combinations = list(product(AV_values.keys(), AC_values.keys(), PR_values.keys(), UI_values.keys()))

# Calculate the exploitability score for each combination
results = []
for av, ac, pr, ui in combinations:
    exploitability = 10.4 * AV_values[av] * AC_values[ac] * PR_values[pr] * UI_values[ui]
    results.append((av, ac, pr, ui, exploitability))

# Create a DataFrame to display the results in table form
df = pd.DataFrame(results, columns=['AV', 'AC', 'PR', 'UI', 'Exploitability Score'])

# Sort the DataFrame by 'Exploitability Score' in descending order
df_sorted = df.sort_values(by='Exploitability Score', ascending=False)
```

**Fig. 3 Exploitability values when the scope is "Changed"**



```python
import pandas as pd
from itertools import product

# Defining the possible values for each metric and their weights
AV_values = {'N': 0.85, 'A': 0.62, 'L': 0.55, 'P': 0.2}
AC_values = {'L': 0.77, 'H': 0.44}
PR_values = {'N': 0.85, 'L': 0.62, 'H': 0.27}
UI_values = {'N': 0.85, 'R': 0.62}

# Generating all possible combinations where scope is unchanged
combinations = list(product(AV_values.keys(), AC_values.keys(), PR_values.keys(), UI_values.keys()))

# Calculate the exploitability score for each combination
results = []
for av, ac, pr, ui in combinations:
    exploitability = 10.4 * AV_values[av] * AC_values[ac] * PR_values[pr] * UI_values[ui]
    results.append((av, ac, pr, ui, exploitability))

# Create a DataFrame to display the results in table form
df = pd.DataFrame(results, columns=['AV', 'AC', 'PR', 'UI', 'Exploitability Score'])

# Sort the DataFrame by 'Exploitability Score' in descending order
df_sorted = df.sort_values(by='Exploitability Score', ascending=False)
```

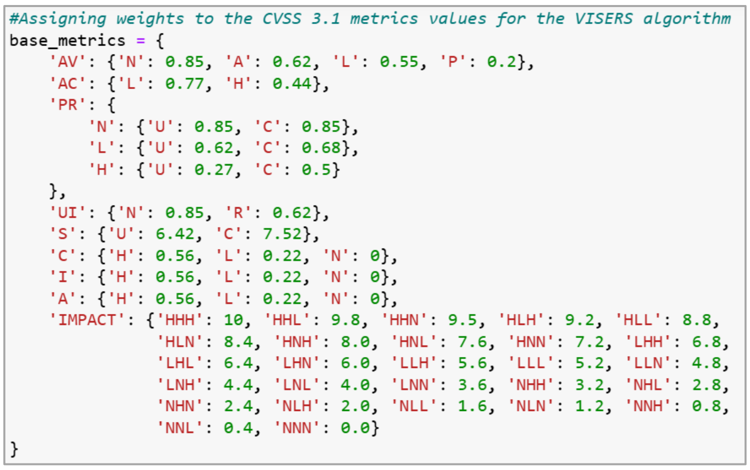
**Fig. 4 Exploitability values when the scope is "Unchanged"**

## 4 Implementation

As we have compared the proposed technique with some other prioritization techniques, the code written includes the calculation of all the techniques. The code file is saved with the name: *Final Project Code.ipynb*. Following sections explain the different parts of the code.

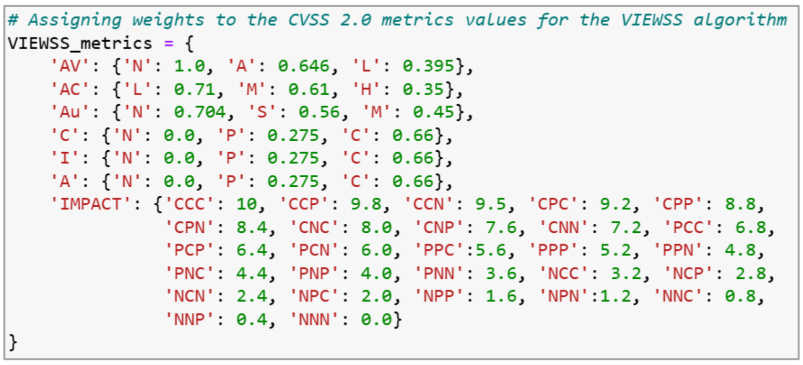
## I. Defining Metrics

Figures in this section portray the weights assigned to each metrics value. The IMPACT variable mentioned for VISERS, VIEWSS, and VRSS contains the values assigned to each of the combination for confidentiality, integrity, and availability.



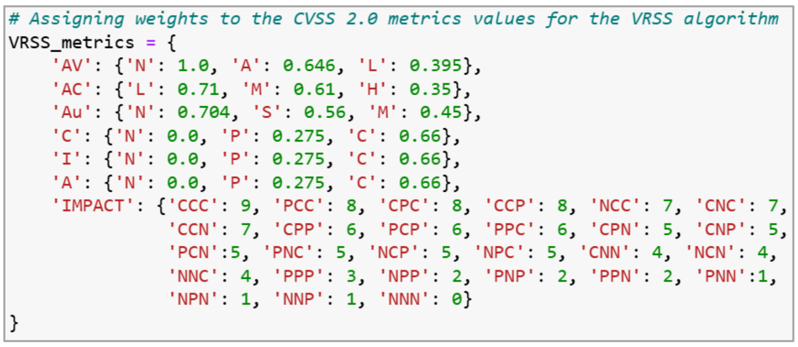
```
#Assigning weights to the CVSS 3.1 metrics values for the VISERS algorithm
base_metrics = {
    'AV': {'N': 0.85, 'A': 0.62, 'L': 0.55, 'P': 0.2},
    'AC': {'L': 0.77, 'H': 0.44},
    'PR': {
        'N': {'U': 0.85, 'C': 0.85},
        'L': {'U': 0.62, 'C': 0.68},
        'H': {'U': 0.27, 'C': 0.5}
    },
    'UI': {'N': 0.85, 'R': 0.62},
    'S': {'U': 6.42, 'C': 7.52},
    'C': {'H': 0.56, 'L': 0.22, 'N': 0},
    'I': {'H': 0.56, 'L': 0.22, 'N': 0},
    'A': {'H': 0.56, 'L': 0.22, 'N': 0},
    'IMPACT': {'HHH': 10, 'HHL': 9.8, 'HHN': 9.5, 'HLH': 9.2, 'HLL': 8.8,
               'HLN': 8.4, 'HNH': 8.0, 'HNL': 7.6, 'HNN': 7.2, 'LHH': 6.8,
               'LHL': 6.4, 'LHN': 6.0, 'LLH': 5.6, 'LLL': 5.2, 'LLN': 4.8,
               'LNH': 4.4, 'LNL': 4.0, 'LNN': 3.6, 'NHH': 3.2, 'NHL': 2.8,
               'NHN': 2.4, 'NLH': 2.0, 'NLL': 1.6, 'NLN': 1.2, 'NNH': 0.8,
               'NNL': 0.4, 'NNN': 0.0}
}
```

**Fig. 5 Metrics for VISERS**



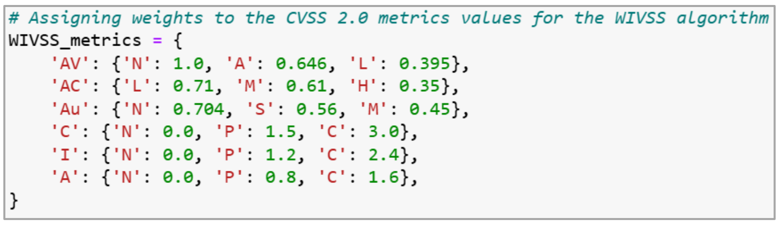
```
# Assigning weights to the CVSS 2.0 metrics values for the VIEWSS algorithm
VIEWSS_metrics = {
    'AV': {'N': 1.0, 'A': 0.646, 'L': 0.395},
    'AC': {'L': 0.71, 'M': 0.61, 'H': 0.35},
    'Au': {'N': 0.704, 'S': 0.56, 'M': 0.45},
    'C': {'N': 0.0, 'P': 0.275, 'C': 0.66},
    'I': {'N': 0.0, 'P': 0.275, 'C': 0.66},
    'A': {'N': 0.0, 'P': 0.275, 'C': 0.66},
    'IMPACT': {'CCC': 10, 'CCP': 9.8, 'CCN': 9.5, 'CPC': 9.2, 'CPP': 8.8,
               'CPN': 8.4, 'CNC': 8.0, 'CNP': 7.6, 'CNN': 7.2, 'PCC': 6.8,
               'PCP': 6.4, 'PCN': 6.0, 'PPC':5.6, 'PPP': 5.2, 'PPN': 4.8,
               'PNC': 4.4, 'PNP': 4.0, 'PNN': 3.6, 'NCC': 3.2, 'NCP': 2.8,
               'NCN': 2.4, 'NPC': 2.0, 'NPP': 1.6, 'NPN':1.2, 'NNC': 0.8,
               'NNP': 0.4, 'NNN': 0.0}
}
```

**Fig. 6 Metrics for VIEWSS**



```
# Assigning weights to the CVSS 2.0 metrics values for the VRSS algorithm
VRSS_metrics = {
    'AV': {'N': 1.0, 'A': 0.646, 'L': 0.395},
    'AC': {'L': 0.71, 'M': 0.61, 'H': 0.35},
    'Au': {'N': 0.704, 'S': 0.56, 'M': 0.45},
    'C': {'N': 0.0, 'P': 0.275, 'C': 0.66},
    'I': {'N': 0.0, 'P': 0.275, 'C': 0.66},
    'A': {'N': 0.0, 'P': 0.275, 'C': 0.66},
    'IMPACT': {'CCC': 9, 'PCC': 8, 'CPC': 8, 'CCP': 8, 'NCC': 7, 'CNC': 7,
               'CCN': 7, 'CPP': 6, 'PCP': 6, 'PPC': 6, 'CPN': 5, 'CNP': 5,
               'PCN':5, 'PNC': 5, 'NCP': 5, 'NPC': 5, 'CNN': 4, 'NCN': 4,
               'NNC': 4, 'PPP': 3, 'NPP': 2, 'PNP': 2, 'PPN': 2, 'PNN':1,
               'NPN': 1, 'NNP': 1, 'NNN': 0}
}
```

**Fig. 7 Metrics for VRSS**

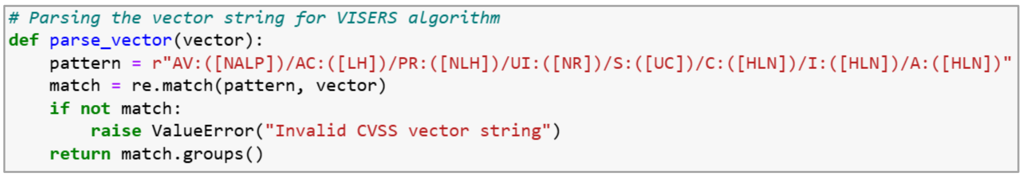


```
# Assigning weights to the CVSS 2.0 metrics values for the WIVSS algorithm
WIVSS_metrics = {
    'AV': {'N': 1.0, 'A': 0.646, 'L': 0.395},
    'AC': {'L': 0.71, 'M': 0.61, 'H': 0.35},
    'Au': {'N': 0.704, 'S': 0.56, 'M': 0.45},
    'C': {'N': 0.0, 'P': 1.5, 'C': 3.0},
    'I': {'N': 0.0, 'P': 1.2, 'C': 2.4},
    'A': {'N': 0.0, 'P': 0.8, 'C': 1.6},
}
```

**Fig. 8 Metrics for WIVSS**

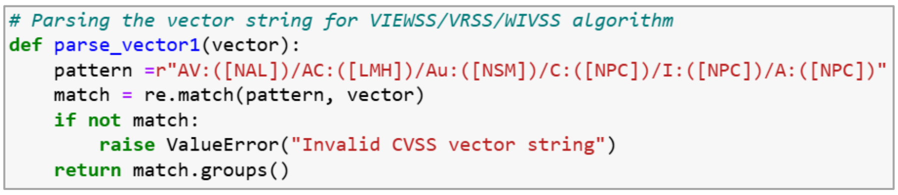
## II. Vector check

There are two functions defined to check the accuracy of the vector string received from the calculation functions which are discussed in the next section of this manual.



```
# Parsing the vector string for VISERS algorithm
def parse_vector(vector):
    pattern = r"AV:([NALP])/AC:([LH])/PR:([NLH])/UI:([NR])/S:([UC])/C:([HLN])/I:([HLN])/A:([HLN])"
    match = re.match(pattern, vector)
    if not match:
        raise ValueError("Invalid CVSS vector string")
    return match.groups()
```

**Fig. 9 Vector check for VISERS**



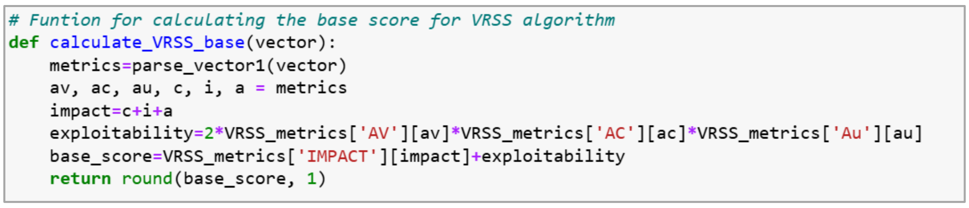
```
# Parsing the vector string for VIEWSS/VRSS/WIVSS algorithm
def parse_vector1(vector):
    pattern =r"AV:([NAL])/AC:([LMH])/Au:([NSM])/C:([NPC])/I:([NPC])/A:([NPC])"
    match = re.match(pattern, vector)
    if not match:
        raise ValueError("Invalid CVSS vector string")
    return match.groups()
```

**Fig. 10 Vector check for VIEWSS/VRSS/WIVSS**

## III. Calculation functions

There are 4 functions created to calculate the scores. VISERS code is explained in detail.

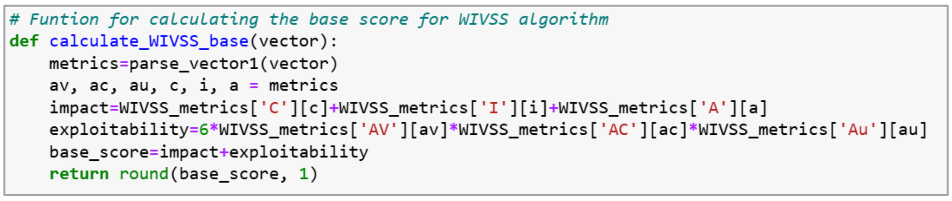
### *a. VRSS Calculation*



```
# Funtion for calculating the base score for VRSS algorithm
def calculate_VRSS_base(vector):
    metrics=parse_vector1(vector)
    av, ac, au, c, i, a = metrics
    impact=c+i+a
    exploitability=2*VRSS_metrics['AV'][av]*VRSS_metrics['AC'][ac]*VRSS_metrics['Au'][au]
    base_score=VRSS_metrics['IMPACT'][impact]+exploitability
    return round(base_score, 1)
```

**Fig. 11 Calculating base score using VRSS**

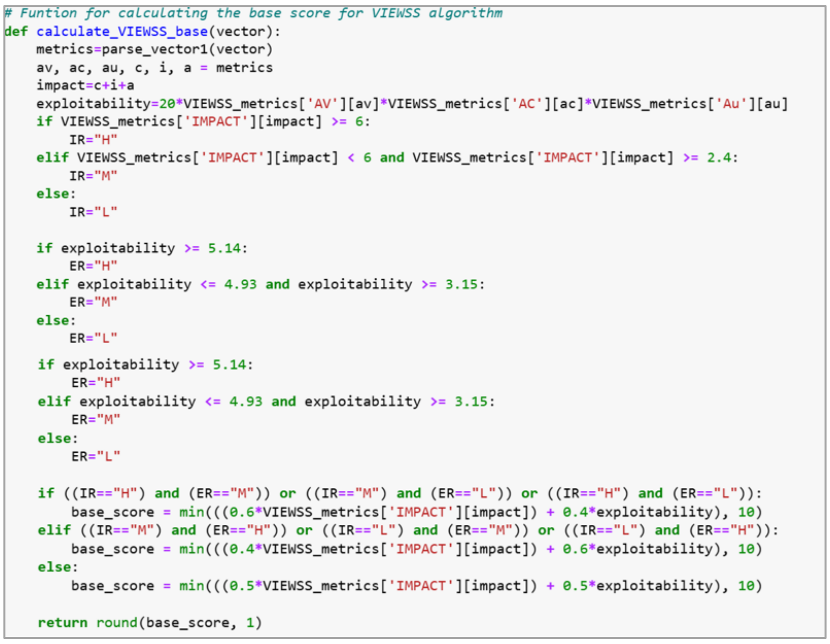
### *b. WIVSS Calculation*



```
# Funtion for calculating the base score for WIVSS algorithm
def calculate_WIVSS_base(vector):
    metrics=parse_vector1(vector)
    av, ac, au, c, i, a = metrics
    impact=WIVSS_metrics['C'][c]+WIVSS_metrics['I'][i]+WIVSS_metrics['A'][a]
    exploitability=6*WIVSS_metrics['AV'][av]*WIVSS_metrics['AC'][ac]*WIVSS_metrics['Au'][au]
    base_score=impact+exploitability
    return round(base_score, 1)
```

**Fig. 12 Calculating base score using WIVSS**

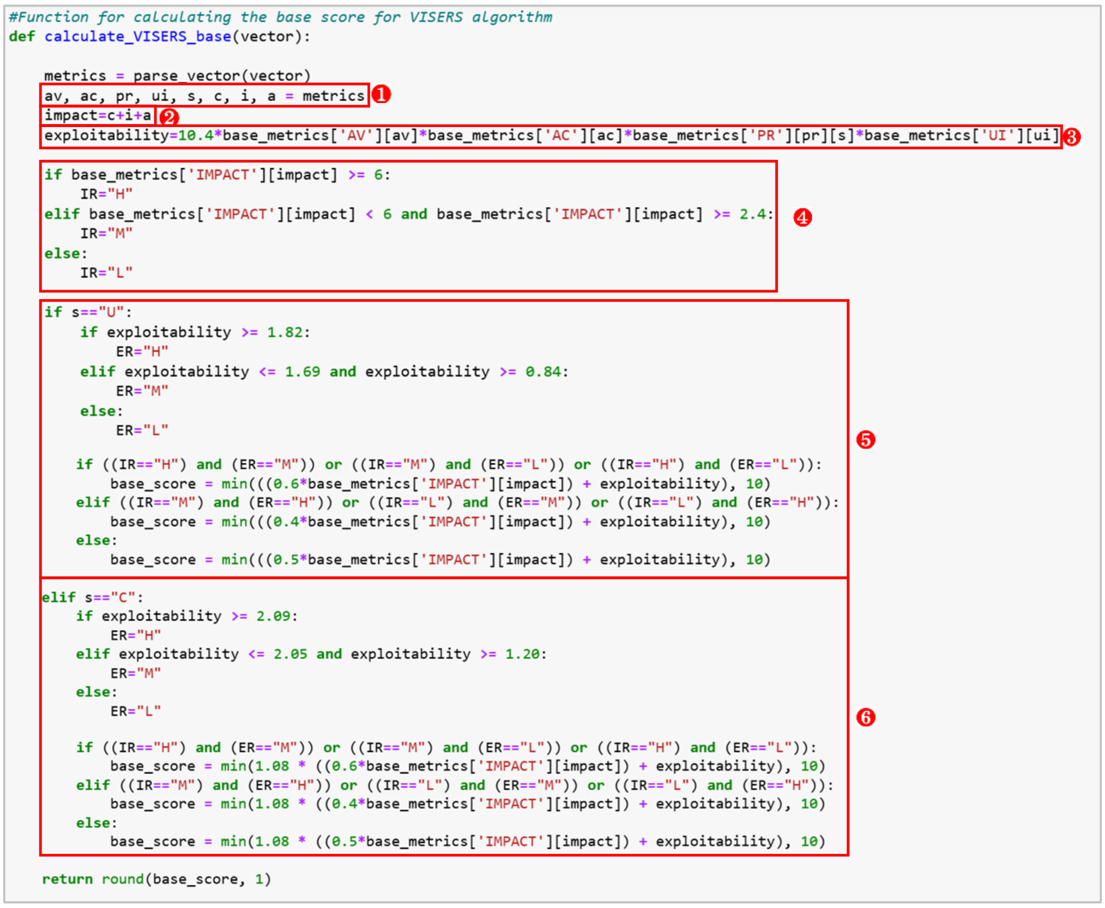
## *c. VIEWSS Calculation*



```python
# Funtion for calculating the base score for VIEWSS algorithm
def calculate_VIEWSS_base(vector):
    metrics=parse_vector1(vector)
    av, ac, au, c, i, a = metrics
    impact=c+i+a
    exploitability=20*VIEWSS_metrics['AV'][av]*VIEWSS_metrics['AC'][ac]*VIEWSS_metrics['Au'][au]
    if VIEWSS_metrics['IMPACT'][impact] >= 6:
        IR="H"
    elif VIEWSS_metrics['IMPACT'][impact] < 6 and VIEWSS_metrics['IMPACT'][impact] >= 2.4:
        IR="M"
    else:
        IR="L"

    if exploitability >= 5.14:
        ER="H"
    elif exploitability <= 4.93 and exploitability >= 3.15:
        ER="M"
    else:
        ER="L"

    if exploitability >= 5.14:
        ER="H"
    elif exploitability <= 4.93 and exploitability >= 3.15:
        ER="M"
    else:
        ER="L"

    if ((IR=="H") and (ER=="M")) or ((IR=="M") and (ER=="L")) or ((IR=="H") and (ER=="L")):
        base_score = min(((0.6*VIEWSS_metrics['IMPACT'][impact]) + 0.4*exploitability), 10)
    elif ((IR=="M") and (ER=="H")) or ((IR=="L") and (ER=="M")) or ((IR=="L") and (ER=="H")):
        base_score = min(((0.4*VIEWSS_metrics['IMPACT'][impact]) + 0.6*exploitability), 10)
    else:
        base_score = min(((0.5*VIEWSS_metrics['IMPACT'][impact]) + 0.5*exploitability), 10)

    return round(base_score, 1)
```

**Fig. 13 Calculating base score using VIEWSS**

## *d. VISERS Calculation*



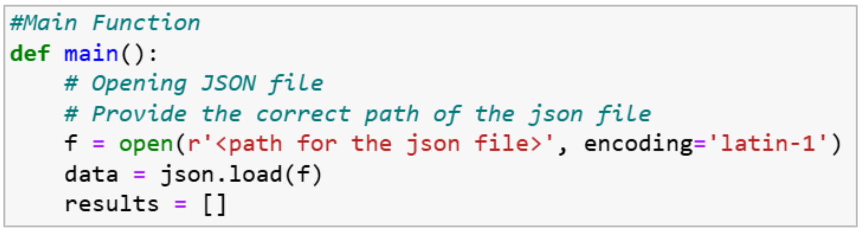
```python
#Function for calculating the base score for VISERS algorithm
def calculate_VISERS_base(vector):

    metrics = parse_vector(vector)
    av, ac, pr, ui, s, c, i, a = metrics ❶
    impact=c+i+a ❷
    exploitability=10.4*base_metrics['AV'][av]*base_metrics['AC'][ac]*base_metrics['PR'][pr][s]*base_metrics['UI'][ui] ❸

    if base_metrics['IMPACT'][impact] >= 6:
        IR="H"
    elif base_metrics['IMPACT'][impact] < 6 and base_metrics['IMPACT'][impact] >= 2.4:  ❹
        IR="M"
    else:
        IR="L"

    if s=="U":
        if exploitability >= 1.82:
            ER="H"
        elif exploitability <= 1.69 and exploitability >= 0.84:
            ER="M"
        else:
            ER="L"
                                                                                      ❺
        if ((IR=="H") and (ER=="M")) or ((IR=="M") and (ER=="L")) or ((IR=="H") and (ER=="L")):
            base_score = min(((0.6*base_metrics['IMPACT'][impact]) + exploitability), 10)
        elif ((IR=="M") and (ER=="H")) or ((IR=="L") and (ER=="M")) or ((IR=="L") and (ER=="H")):
            base_score = min(((0.4*base_metrics['IMPACT'][impact]) + exploitability), 10)
        else:
            base_score = min(((0.5*base_metrics['IMPACT'][impact]) + exploitability), 10)

    elif s=="C":
        if exploitability >= 2.09:
            ER="H"
        elif exploitability <= 2.05 and exploitability >= 1.20:
            ER="M"
        else:
            ER="L"
                                                                                      ❻
        if ((IR=="H") and (ER=="M")) or ((IR=="M") and (ER=="L")) or ((IR=="H") and (ER=="L")):
            base_score = min(1.08 * ((0.6*base_metrics['IMPACT'][impact]) + exploitability), 10)
        elif ((IR=="M") and (ER=="H")) or ((IR=="L") and (ER=="M")) or ((IR=="L") and (ER=="H")):
            base_score = min(1.08 * ((0.4*base_metrics['IMPACT'][impact]) + exploitability), 10)
        else:
            base_score = min(1.08 * ((0.5*base_metrics['IMPACT'][impact]) + exploitability), 10)

    return round(base_score, 1)
```

**Fig. 14 Calculating base score using VISERS**

1. parse_vector function returns a tuple. This can be used to assign values to the variables.
2. Values of confidentiality, integrity, and availability are combined to form a string and assigned to a variable.
3. Exploitability is calculated with the proposed formula
4. Using pre-defined values of impact variable, qualitative rating of impact is performed
5. For scope value as “unchanged”, qualitative rating of exploitability is performed. The range defined for exploitability is retrieved from the calculations mentioned in section 3 fig.4. The values are kept in descending order and equally divided among the ratings. Then on the basis of exploitability and impact rating final calculation of base score is done.
6. For scope value as “changed”, qualitative rating of exploitability is performed. The range defined for exploitability is retrieved from the calculations mentioned in section 3 fig. 3. The values are kept in descending order and equally divided among the ratings. Then on the basis of exploitability and impact rating final calculation of base score is done. The only difference is that a factor of 1.08 is multiplied to generate the base score.

## IV. Main Function

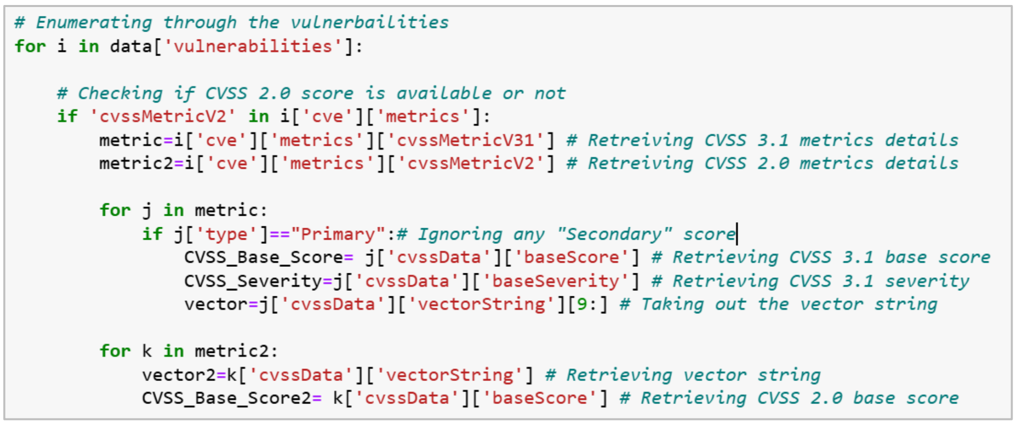
The main function starts with accessing the dataset file, *Dataset.json*. The data is loaded from the file into a variable and an empty list is created.



```
#Main Function
def main():
    # Opening JSON file
    # Provide the correct path of the json file
    f = open(r'<path for the json file>', encoding='latin-1')
    data = json.load(f)
    results = []
```

**Fig. 15 Opening the JSON formatted dataset file**

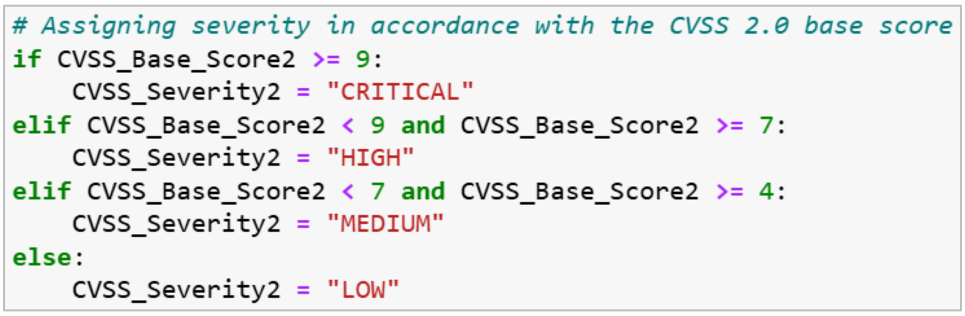
The vulnerabilities are enumerated using a “for” loop. Base scores, severity, and vector string is fetched for CVSS 3.1 and assigned to variables. For CVSS 2.0, only base score and string vector is fetched.



```
# Enumerating through the vulnerbailities
for i in data['vulnerabilities']:

    # Checking if CVSS 2.0 score is available or not
    if 'cvssMetricV2' in i['cve']['metrics']:
        metric=i['cve']['metrics']['cvssMetricV31'] # Retreiving CVSS 3.1 metrics details
        metric2=i['cve']['metrics']['cvssMetricV2'] # Retreiving CVSS 2.0 metrics details

        for j in metric:
            if j['type']=="Primary":# Ignoring any "Secondary" score
                CVSS_Base_Score= j['cvssData']['baseScore'] # Retrieving CVSS 3.1 base score
                CVSS_Severity=j['cvssData']['baseSeverity'] # Retrieving CVSS 3.1 severity
                vector=j['cvssData']['vectorString'][9:] # Taking out the vector string

        for k in metric2:
            vector2=k['cvssData']['vectorString'] # Retrieving vector string
            CVSS_Base_Score2= k['cvssData']['baseScore'] # Retrieving CVSS 2.0 base score
```

**Fig. 16 Retrieving CVSS 3.1 and CVSS 2.0 data**

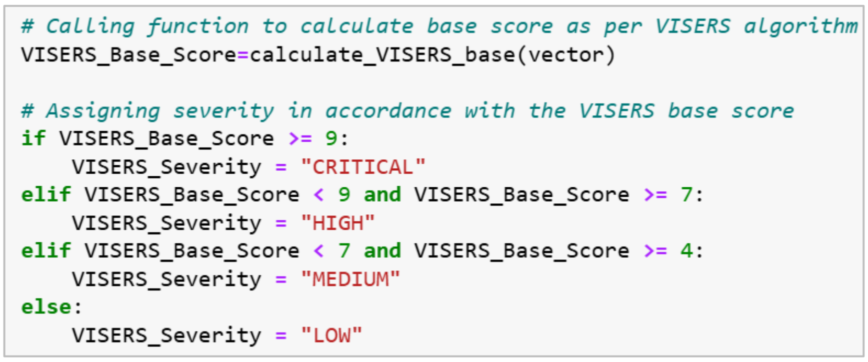
Severity for CVSS 3.1 base scores can be retrieved directly from the data but for all other techniques the severity rating is assigned as per the CVSS 3.1 severity levels. (*NVD - Vulnerability Metrics*, 2024)



```python
# Assigning severity in accordance with the CVSS 2.0 base score
if CVSS_Base_Score2 >= 9:
    CVSS_Severity2 = "CRITICAL"
elif CVSS_Base_Score2 < 9 and CVSS_Base_Score2 >= 7:
    CVSS_Severity2 = "HIGH"
elif CVSS_Base_Score2 < 7 and CVSS_Base_Score2 >= 4:
    CVSS_Severity2 = "MEDIUM"
else:
    CVSS_Severity2 = "LOW"
```

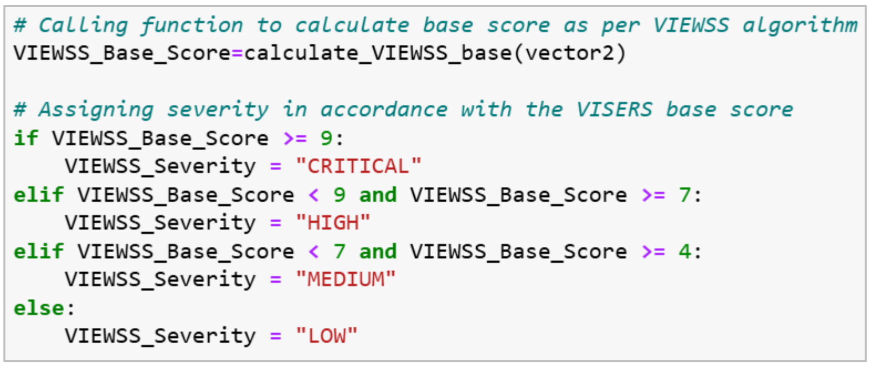
**Fig. 17 Assigning severity rating to CVSS 2.0 base scores**

For all other algorithms, base score calculation function is called from the main function and severity ratings are assigned.



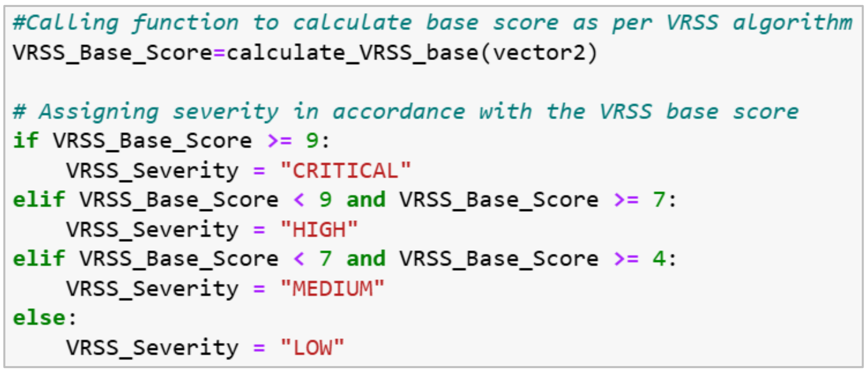
```python
# Calling function to calculate base score as per VISERS algorithm
VISERS_Base_Score=calculate_VISERS_base(vector)

# Assigning severity in accordance with the VISERS base score
if VISERS_Base_Score >= 9:
    VISERS_Severity = "CRITICAL"
elif VISERS_Base_Score < 9 and VISERS_Base_Score >= 7:
    VISERS_Severity = "HIGH"
elif VISERS_Base_Score < 7 and VISERS_Base_Score >= 4:
    VISERS_Severity = "MEDIUM"
else:
    VISERS_Severity = "LOW"
```

**Fig. 18 Assigning severity ratings to VISERS base scores**



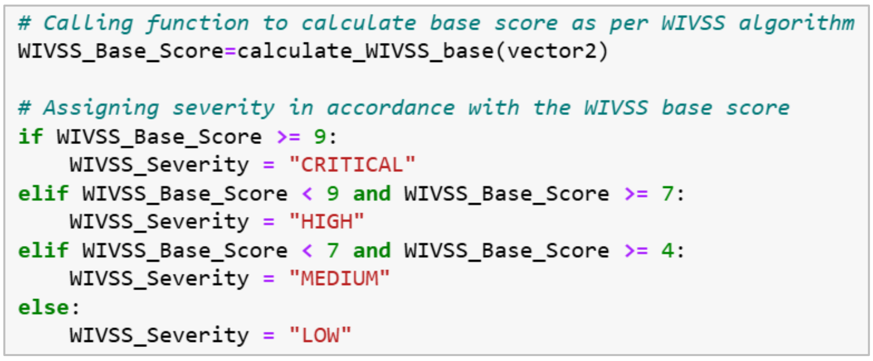
```python
# Calling function to calculate base score as per VIEWSS algorithm
VIEWSS_Base_Score=calculate_VIEWSS_base(vector2)

# Assigning severity in accordance with the VISERS base score
if VIEWSS_Base_Score >= 9:
    VIEWSS_Severity = "CRITICAL"
elif VIEWSS_Base_Score < 9 and VIEWSS_Base_Score >= 7:
    VIEWSS_Severity = "HIGH"
elif VIEWSS_Base_Score < 7 and VIEWSS_Base_Score >= 4:
    VIEWSS_Severity = "MEDIUM"
else:
    VIEWSS_Severity = "LOW"
```

**Fig. 19 Assigning severity ratings to VIEWSS base scores**



```python
#Calling function to calculate base score as per VRSS algorithm
VRSS_Base_Score=calculate_VRSS_base(vector2)

# Assigning severity in accordance with the VRSS base score
if VRSS_Base_Score >= 9:
    VRSS_Severity = "CRITICAL"
elif VRSS_Base_Score < 9 and VRSS_Base_Score >= 7:
    VRSS_Severity = "HIGH"
elif VRSS_Base_Score < 7 and VRSS_Base_Score >= 4:
    VRSS_Severity = "MEDIUM"
else:
    VRSS_Severity = "LOW"
```

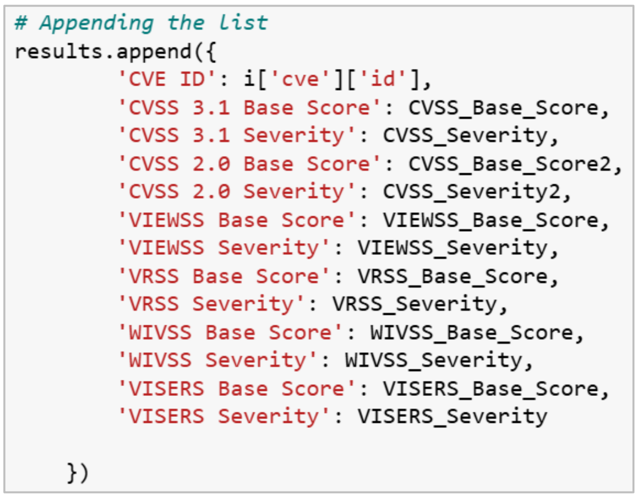
**Fig. 20 Assigning severity ratings to VRSS base scores**



```python
# Calling function to calculate base score as per WIVSS algorithm
WIVSS_Base_Score=calculate_WIVSS_base(vector2)

# Assigning severity in accordance with the WIVSS base score
if WIVSS_Base_Score >= 9:
    WIVSS_Severity = "CRITICAL"
elif WIVSS_Base_Score < 9 and WIVSS_Base_Score >= 7:
    WIVSS_Severity = "HIGH"
elif WIVSS_Base_Score < 7 and WIVSS_Base_Score >= 4:
    WIVSS_Severity = "MEDIUM"
else:
    WIVSS_Severity = "LOW"
```

**Fig. 21 Assigning severity ratings to WIVSS base scores**

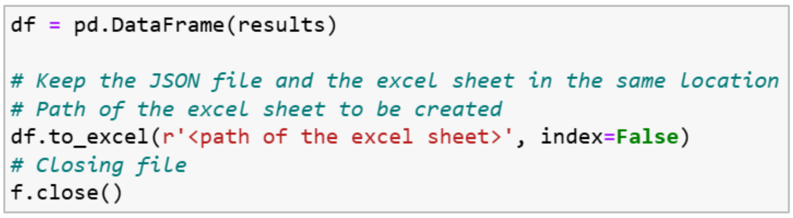
The required data is appended into the list created at the beginning of the main function.



```python
# Appending the list
results.append({
        'CVE ID': i['cve']['id'],
        'CVSS 3.1 Base Score': CVSS_Base_Score,
        'CVSS 3.1 Severity': CVSS_Severity,
        'CVSS 2.0 Base Score': CVSS_Base_Score2,
        'CVSS 2.0 Severity': CVSS_Severity2,
        'VIEWSS Base Score': VIEWSS_Base_Score,
        'VIEWSS Severity': VIEWSS_Severity,
        'VRSS Base Score': VRSS_Base_Score,
        'VRSS Severity': VRSS_Severity,
        'WIVSS Base Score': WIVSS_Base_Score,
        'WIVSS Severity': WIVSS_Severity,
        'VISERS Base Score': VISERS_Base_Score,
        'VISERS Severity': VISERS_Severity

    })
```

**Fig. 22 Appending the required data into the list**

The code imports Pandas, an open-source data analysis and manipulation tool for python, to export the data into an excel sheet.

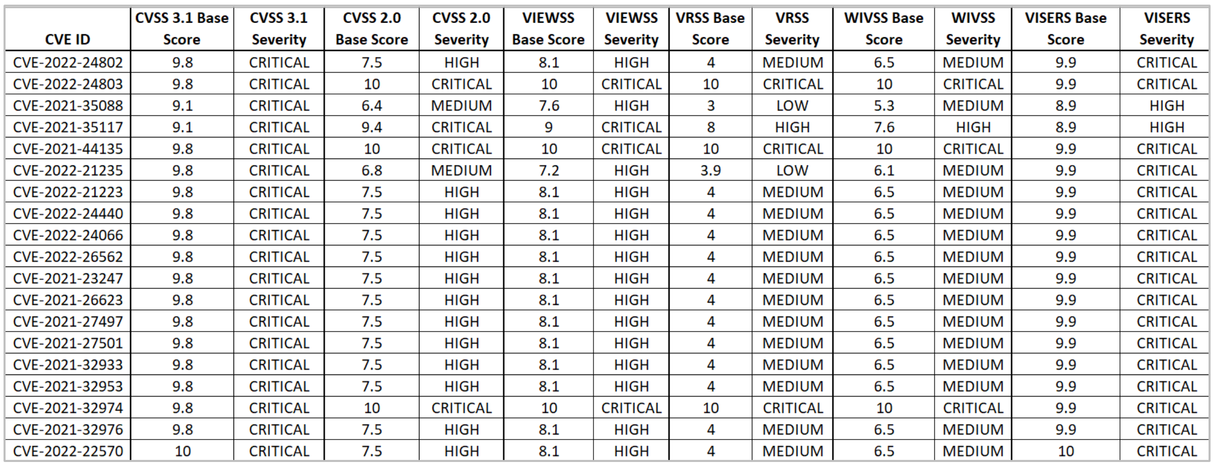


```
df = pd.DataFrame(results)

# Keep the JSON file and the excel sheet in the same location
# Path of the excel sheet to be created
df.to_excel(r'<path of the excel sheet>', index=False)
# Closing file
f.close()
```

**Fig. 23 Creating an excel sheet with the required data**

## 5 Result

The final output of the code is an excel sheet containing CVE IDs along with the base score and severity rating of the algorithms; CVSS 3.1, CVSS 2.0, VRSS, WIVSS, VIEWSS, and VISERS. The excel sheet is saved with the name: *Code_Result.xlsx*. Fig. 24 is the snippet of the excel file.



| CVE ID | CVSS 3.1 Base Score | CVSS 3.1 Severity | CVSS 2.0 Base Score | CVSS 2.0 Severity | VIEWSS Base Score | VIEWSS Severity | VRSS Base Score | VRSS Severity | WIVSS Base Score | WIVSS Severity | VISERS Base Score | VISERS Severity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CVE-2022-24802 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2022-24803 | 9.8 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 9.9 | CRITICAL |
| CVE-2021-35088 | 9.1 | CRITICAL | 6.4 | MEDIUM | 7.6 | HIGH | 3 | LOW | 5.3 | MEDIUM | 8.9 | HIGH |
| CVE-2021-35117 | 9.1 | CRITICAL | 9.4 | CRITICAL | 9 | CRITICAL | 8 | HIGH | 7.6 | HIGH | 8.9 | HIGH |
| CVE-2021-44135 | 9.8 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 9.9 | CRITICAL |
| CVE-2022-21235 | 9.8 | CRITICAL | 6.8 | MEDIUM | 7.2 | HIGH | 3.9 | LOW | 6.1 | MEDIUM | 9.9 | CRITICAL |
| CVE-2022-21223 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2022-24440 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2022-24066 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2022-26562 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2021-23247 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2021-26623 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2021-27497 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2021-27501 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2021-32933 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2021-32953 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2021-32974 | 9.8 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 10 | CRITICAL | 9.9 | CRITICAL |
| CVE-2021-32976 | 9.8 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 9.9 | CRITICAL |
| CVE-2022-22570 | 10 | CRITICAL | 7.5 | HIGH | 8.1 | HIGH | 4 | MEDIUM | 6.5 | MEDIUM | 10 | CRITICAL |

**Fig. 24 Snippet of the excel produced as Output**

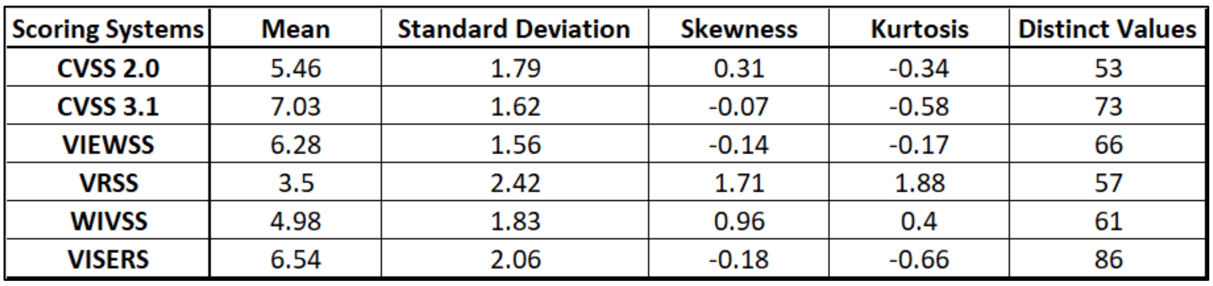
## 6 Analysis

The data is analysed utilizing the statistical measures: mean, standard deviation, skewness, kurtosis, and distinct values. MS Excel provides functions to calculate these measures.

**Table 1: MS Excel functions**

| **Function Name** | **Usage** |
|---|---|
| **AVERAGE** | Calculates the mean |
| **STDEV.S** | Calculates the standard deviation |
| **SKEW** | Calculates the skewness |
| **KURT** | Calculates the kurtosis |

The distinct values can be obtained by removing the duplicate values of the base scores and counting the unique ones. The values obtained from these measures are presented in an excel sheet named *Analysis.xlsx*. Following are the snippets from that excel file.



| Scoring Systems | Mean | Standard Deviation | Skewness | Kurtosis | Distinct Values |
|---|---|---|---|---|---|
| **CVSS 2.0** | 5.46 | 1.79 | 0.31 | -0.34 | 53 |
| **CVSS 3.1** | 7.03 | 1.62 | -0.07 | -0.58 | 73 |
| **VIEWSS** | 6.28 | 1.56 | -0.14 | -0.17 | 66 |
| **VRSS** | 3.5 | 2.42 | 1.71 | 1.88 | 57 |
| **WIVSS** | 4.98 | 1.83 | 0.96 | 0.4 | 61 |
| **VISERS** | 6.54 | 2.06 | -0.18 | -0.66 | 86 |

**Fig. 25 Retrieved statistical values**

A visual representation of qualitative distribution of vulnerabilities for all the techniques is also presented in this excel sheet.

## References

Anaconda. (2024). *Anaconda installer file hashes*. Anaconda Distribution.

*NVD - Vulnerability Metrics*. (2024). Retrieved August 11, 2024, from https://nvd.nist.gov/vuln-metrics/cvss