Buffer Overflow in Python:

Problem Description:

A buffer overflow occurs when data is written beyond the boundaries of a fixed length buffer overwriting adjacent memory locations which may include other buffers, variables and program flow data. Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code.

Implementation:

Python allows various ways to create and manipulate arrays, if you use arrays of a predetermined size you may cause the program to throw an IndexError to avoid a buffer overflow:

```
buffer=[None]*10

for i in range(0,14):

    buffer[i]=7
```

In the code above, buffer has 10 elements but the loop attempts to writes through 15 elements, which results in an error.

Counter Measure:

Python makes an effort to avoid buffer overflow by checking the bounds of a buffer (like an array) and preventing any access beyond those bounds.

Make sure you have enough space: Before copying data to a fixed size block, make sure it is large enough to hold the data that you are going to copy. If it is not large enough, do not copy more data than your available space can hold. If your program is not able to continue properly after filling the available space, you may have to find some way to recover from the error.

Validate indices: If you have an integer variable, verify that it is within the proper bounds before you use it as an index to an array. This validation is particularly important for any values that might have been provided as user input or other untrusted input, such as information that might be read from a file or from a network connection.

Use alternative data structures that reduce the risk of overflows: When possible, use lists in Python without defining the initial size and use the .append method to add elements which can reduce your risk of buffer overflow vulnerabilities.