Essential Python components for automation

Throughout this course, you explored coding in Python. You've focused on variables, conditional statements, iterative statements, functions, and a variety of ways to work with strings and lists. In this reading, you will explore why these are all essential components when automating tasks through Python, and you'll be introduced to another necessary component: working with files.

Automating tasks in Python

Automation is the use of technology to reduce human and manual effort to perform common and repetitive tasks. As a security analyst, you will primarily use Python to automate tasks.

You have encountered multiple examples of how to use Python for automation in this course, including investigating logins, managing access, and updating devices.

Automating cybersecurity-related tasks requires understanding the following Python components that you've worked with in this course:

Variables

A **variable** is a container that stores data. Variables are essential for automation. Without them, you would have to individually rewrite values for each action you took in Python.

Conditional statements

A **conditional statement** is a statement that evaluates code to determine if it meets a specified set of conditions. Conditional statements allow you to check for conditions before performing actions. This is much more efficient than manually evaluating whether to apply an action to each separate piece of data.

Iterative statements

An **iterative statement** is code that repeatedly executes a set of instructions. You explored two kinds of iterative statements: *for* loops and *while* loops. In both cases, they allow you to perform the same actions a certain number of times without the need to retype the same code each time. Using a *for* loop allows you to automate repetition of that code based on a sequence, and using a

while loop allows you to automate the repetition based on a condition.

Functions

A **function** is a section of code that can be reused in a program. Functions help you automate your tasks by reducing the need to incorporate the same code multiple places in a program. Instead, you can define the function once and call it wherever you need it.

You can develop your own functions based on your particular needs. You can also incorporate the built-in functions that exist directly in Python without needing to manually code them.

Techniques for working with strings

String data is one of the most common data types that you'll encounter when automating cybersecurity tasks through Python, and there are a lot of techniques that make working with strings efficient. You can use bracket notation to access characters in a string through their indices. You can also use a variety of functions and methods when working with strings, including *str()*, *len()*, and *.index()*.

Techniques for working with lists

List data is another common data type. Like with strings, you can use bracket notation to access a list element through its index. Several methods also help you with automation when working with lists. These include *.insert()*, *.remove()*, *.append()*, and *.index()*.

Example: Counting logins made by a flagged user

As one example, you may find that you need to investigate the logins of a specific user who has been flagged for unusual activity. Specifically, you are responsible for counting how many times this user has logged in for the day. If you are given a list identifying the username associated with each login attempt made that day, you can automate this investigation in Python.

To automate the investigation, you'll need to incorporate the following Python components:

- A *for* loop will allow you to iterate through all the usernames in the list.
- Within the *for* loop, you should incorporate a conditional statement to examine whether each username in the list matches the username of the flagged user.
- When the condition evaluates to *True*, you also need to increment a counter variable that keeps track of the number of times the flagged user appears in the list.

Additionally, if you want to reuse this code multiple times, you can incorporate it into a function. The function can include parameters that accept the username of the flagged user and the list to iterate through. (The list would contain the usernames associated with all login attempts made that day.) The function can use the counter variable to return the number of logins for that flagged user.

Working with files in Python

One additional component of automating cybersecurity-related tasks in Python is understanding how to work with files. Security-related data will often be initially found in log files. A **log** is a record of events that occur within an organization's systems. In logs, lines are often appended to the record as time progresses.

Two common file formats for security logs are *.txt* files and *.csv* files. Both *.txt* and *.csv* files are types of text files, meaning they contain only plain text. They do not contain images and do not specify graphical properties of the text, including font, color, or spacing. In a *.csv* file, or a "comma-separated values" file, the values are separated by commas. In a *.txt* file, there is not a specific format for separating values, and they may be separated in a variety of ways, including spaces.

You can easily extract data from *.txt* and *.csv* files. You can also convert both into other file formats.

Coming up, you'll learn how to import, read from, and write to files. You will also explore how to structure the information contained in files.

Key takeaways

It is important for security analysts to be able to automate tasks in Python. This requires knowledge of fundamental Python concepts, including variables, conditional statements, iterative statements, and techniques for working with strings and lists. In addition, the ability to work with files is also essential for automation in Python.

Revision #1
Created 2023-12-28 13:18:03 UTC by naruzkurai
Updated 2023-12-28 13:18:10 UTC by naruzkurai