Strings and the security analyst

The ability to work with strings is important in the cybersecurity profession. Previously, you were introduced to several ways to work with strings, including functions and methods. You also learned how to extract elements in strings using bracket notation and indices. This reading reviews these concepts and explains more about using the *.index()* method. It also highlights examples of string data you might encounter in a security setting.

String data in a security setting

As an analyst, string data is one of the most common data types you will encounter in Python. **String data** is data consisting of an ordered sequence of characters. It's used to store any type of information you don't need to manipulate mathematically (such as through division or subtraction). In a cybersecurity context, this includes IP addresses, usernames, URLs, and employee IDs.

You'll need to work with these strings in a variety of ways. For example, you might extract certain parts of an IP address, or you might verify whether usernames meet required criteria.

Working with indices in strings

Indices

An **index** is a number assigned to every element in a sequence that indicates its position. With strings, this means each character in the string has its own index.

Indices start at *0*. For example, you might be working with this string containing a device ID: "*h32rb17*". The following table indicates the index for each character in this string:

character	index
h	*0*
3	*1*
2	*2*
r	*3*
b	*4*
1	*5*

character	index
7	*6*

You can also use negative numbers as indices. This is based on their position relative to the last character in the string:

character	index
h	*-7*
3	*-6*
2	*-5*
r	*-4*
b	*-3*
1	*-2*
7	*-1*

Bracket notation

Bracket notation refers to the indices placed in square brackets. You can use bracket notation to extract a part of a string. For example, the first character of the device ID might represent a certain characteristic of the device. If you want to extract it, you can use bracket notation for this:

"*h32rb17*"*[0]*

This device ID might also be stored within a variable called *device_id*. You can apply the same bracket notation to the variable:

device_id = "*h32rb17*"

device_id[0]

In both cases, bracket notation outputs the character *h* when this bracket notation is placed inside a *print()* function. You can observe this by running the following code:

```
device_id = "h32rb17"
print("h32rb17"[0])
print(device_id[0])
```

```
h
h
```

You can also take a slice from a string. When you take a slice from a string, you extract more than one character from it. It's often done in cybersecurity contexts when you’re only interested in a specific part of a string. For example, this might be certain numbers in an IP address or certain parts of a URL.

In the device ID example, you might need the first three characters to determine a particular quality of the device. To do this, you can take a slice of the string using bracket notation. You can run this line of code to observe that it outputs "*h32*":

```
print("h32rb17"[0:3])
```

```
h32
```

Note: The slice starts at the *0* index, but the second index specified after the colon is excluded. This means the slice ends one position before index *3*, which is at index *2*.

String functions and methods

The *str()* and *len()* functions are useful for working with strings. You can also apply methods to strings, including the *.upper()*, *.lower()*, and *.index()* methods. A **method** is a function that belongs to a specific data type.

str() and len()

The *str()* function converts its input object into a string. As an analyst, you might use this in security logs when working with numerical IDs that aren't going to be used with mathematical processes. Converting an integer to a string gives you the ability to search through it and extract slices from it.

Consider the example of an employee ID *19329302* that you need to convert into a string. You can use the following line of code to convert it into a string and store it in a variable:

string_id = str(19329302)

The second function you learned for strings is the *len()* function, which returns the number of elements in an object.

As an example, if you want to verify that a certain device ID conforms to a standard of containing seven characters, you can use the *len()* function and a conditional. When you run the following

code, it will print a message if "*h32rb17*" has seven characters:

```
device_id_length = len("h32rb17")
if device_id_length == 7:
    print("The device ID has 7 characters.")
```

```
The device ID has 7 characters.
```

.upper() and .lower()

The *.upper()* method returns a copy of the string with all of its characters in uppercase. For example, you can change this department name to all uppercase by running the code "*Information Technology*"*.upper()*. It would return the string "*INFORMATION TECHNOLOGY*".

Meanwhile, the *.lower()* method returns a copy of the string in all lowercase characters. "*Information Technology*"*.lower()* would return the string "*information technology*".

.index()

The *.index()* method finds the first occurrence of the input in a string and returns its location. For example, this code uses the *.index()* method to find the first occurrence of the character "*r*" in the device ID "*h32rb17*":

```
print("h32rb17".index("r"))
```

```
3
```

The *.index()* method returns *3* because the first occurrence of the character "*r*" is at index *3*.

In other cases, the input may not be found. When this happens, Python returns an error. For instance, the code *print("h32rb17".index("a"))* returns an error because "*a*" is not in the string "*h32rb17*".

Also note that if a string contains more than one instance of a character, only the first one will be returned. For instance, the device ID "*r45rt46*" contains two instances of "*r*". You can run the following code to explore its output:

```
print("r45rt46".index("r"))
```

```
0
```

The output is *0* because *.index()* returns only the first instance of "*r*", which is at index *0*. The instance of "*r*" at index *3* is not returned.

Finding substrings with .index()

A **substring** is a continuous sequence of characters within a string. For example, "*llo*" is a substring of "*hello*".

The *.index()* method can also be used to find the index of the first occurrence of a substring. It returns the index of the first character in that substring. Consider this example that finds the first instance of the user "*tshah*" in a string:

```
tshah_index = "tsnow, tshah, bmoreno - updated".index("tshah")
print(tshah_index)
```

```
7
```

The *.index()* method returns the index *7*, which is where the substring "*tshah*" starts.

Note: When using the *.index()* method to search for substrings, you need to be careful. In the previous example, you want to locate the instance of "*tshah*". If you search for just "*ts*", Python will return *0* instead of *7* because "*ts*" is also a substring of "*tsnow*".

Key takeaways

As a security analyst, you will work with strings in a variety of ways. First, you might need to use bracket notation to work with string indices. Two functions you will likely use are *str()*, which converts an input into a string, and *len()*, which finds the length of a string. You can also use string methods, functions that only work on strings. These include *.upper()*, which converts all letters in a string into uppercase letters, *.lower()*, which converts all letters in a string into lowercase letters, and *.index()*, which returns the index of the first occurrence of its input within a string.

Revision #1
Created 24 December 2023 08:15:20 by naruzkurai
Updated 27 December 2023 11:35:24 by naruzkurai