Activity: Use regular expressions to find patterns

Benjamin Taylor - Google Cybersecurity

Introduction

Security analysts often analyze log files, including those that contain information about login attempts. For example as an analyst, you might flag IP addresses that relate to unusual attempts to log in to the system.

Another area of focus in cybersecurity is detecting devices that require updates. Software updates help prevent security issues due to vulnerabilities.

Using regular expressions in Python can help automate the processes involved in both of these areas of cybersecurity. Regular expression patterns and functions can be used to efficiently extract important information from strings and files.

In this lab, you'll write regular expressions to extract information such as device IDs or IP addresses.

Scenario

In this lab, you're working as a security analyst and your main tasks are as follows:

- extracting device IDs containing certain characters from a log; these characters correspond with a certain operating system that requires an update.
- extracting all IP addresses from a log and then comparing them to those that are flagged in a list.

Task 1

In order to work with regular expressions in Python, start by importing the `re` module. This module contains many functions that will help you work with regular expressions. By running the following code cell, the module will be available through the rest of the notebook.

```
# Import the `re` module in Python

import re
```

Task 2

Currently, you are looking for device IDs that begin with `"r15"`. These characters indicate that the device is running an operating system that must be updated.

You're given a log of device IDs, stored in a variable named `devices`. Your eventual goal is to extract the device IDs that start with the characters `"r15"`. For now, display the contents of the whole string to examine what it contains. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
# Assign `devices` to a string containing device IDs, each device ID
represented by alphanumeric characters

devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h
2j33krk 253be78 ac742a1 r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u
x3463ac i4l56nq g07h55q 081qc9t r159r1u"

# Display the contents of `devices`

print(devices)

r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk
253be78 ac742a1 r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u x3463ac
i4l56nq g07h55q 081qc9t r159r1u
```

Task 3

In this task, you'll write a pattern to find devices that start with the character combination of `"r15"`.

Use the regular expression symbols `\w` and `+` to create the pattern, and store it as a string in a variable named `target_pattern`.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell. Note that the code cell will contain only variable assignments, so running it will not produce an output.

```
# Assign `devices` to a string containing device IDs, each device ID
represented by alphanumeric characters

devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h
2j33krk 253be78 ac742a1 r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u
x3463ac i4l56nq g07h55q 081qc9t r159r1u"

# Assign `target_pattern` to a regular expression pattern for finding
device IDs that start with "r15"

target_pattern = "r15\w+"
```

Question 1

What regular expression pattern did you use? For each component of the pattern, what would happen if it were missing?

I used the "r15" followed by "\w+" to make sure the pattern starts with the "r15" and matches any string following.

Task 4

Use the `findall()` function from the `re` module to find the device IDs that the `target_pattern` matches with. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

Note: In order to use `re.findall()` in Tasks 4, 7, 8, 9 and 11, you must have previously run the code `import re` in Task 1.

```
# Assign `devices` to a string containing device IDs, each device ID
represented by alphanumeric characters

devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h
2j33krk 253be78 ac742a1 r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u
x3463ac i4l56nq g07h55q 081qc9t r159r1u"

# Assign `target_pattern` to a regular expression pattern for finding
device IDs that start with "r15"

target_pattern = "r15\w+"

# Use `re.findall()` to find the device IDs that start with "r15" and
display the results

print(re.findall(target_pattern, devices))
```

```
['r151dm4', 'r15xk9h', 'r15u9q5', 'r159r1u']
```

Task 5

Now, the next task you're responsible for is analyzing a network security log file and determining which IP addresses have been flagged for unusual activity.

You're given the log file as a string stored in a variable named `log_file`. There are some invalid IP addresses in the log file due to issues in data collection. Your eventual goal is to use regular expressions to extract the valid IP addresses from the string.

Start by displaying the contents of the `log_file` to examine the details inside. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
# Assign `log_file` to a string containing username, date, login time,
and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-
05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32
192.168.190.178 \narutley 2022-05-12 17:00:59 1923.1689.3.24 \nrjensen
2022-05-11 0:59:26 192.168.213.128 \naestrada 2022-05-09 19:28:12
```

```
1924.1680.27.57 \nasundara 2022-05-11 18:38:07 192.168.96.200 \ndkot
2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-12 23:38:46
19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153 \
njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08
12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01
192.168.148.115 \nyappiah 2022-05-12 10:37:22 192.168.103.10654 \
ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Display contents of `log_file`

print(log_file)
```

```
eraab 2022-05-10 6:03:41 192.168.152.148
iuduike 2022-05-09 6:46:40 192.168.22.115
smartell 2022-05-09 19:30:32 192.168.190.178
arutley 2022-05-12 17:00:59 1923.1689.3.24
rjensen 2022-05-11 0:59:26 192.168.213.128
aestrada 2022-05-09 19:28:12 1924.1680.27.57
asundara 2022-05-11 18:38:07 192.168.96.200
dkot 2022-05-12 10:52:00 1921.168.1283.75
abernard 2022-05-12 23:38:46 19245.168.2345.49
cjackson 2022-05-12 19:36:42 192.168.247.153
jclark 2022-05-10 10:48:02 192.168.174.117
alevitsk 2022-05-08 12:09:10 192.16874.1390.176
jrafael 2022-05-10 22:40:01 192.168.148.115
yappiah 2022-05-12 10:37:22 192.168.103.10654
daquino 2022-05-08 7:02:35 192.168.168.144
```

Task 6

In this task, you'll build a regular expression pattern that you can use later on to extract IP addresses that are in the form of xxx.xxx.xxx.xxx. In other words, you'll extract all IP addresses that contain four segments of three digits that are separated by periods.

Write a regular expression pattern that will match with these IP addresses and store it in a variable named `pattern`. Use the regular expression symbols `\d` and `\.` in your pattern. Note that the symbol `\d` matches with digits, in other words, any integer between 0 and 9. Be sure to replace the `### YOUR CODE HERE ###` with your own code. Since you'll just build the pattern here, there won't be any output when you run this cell.

```
# Assign `log_file` to a string containing username, date, login time,
and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-
05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32
192.168.190.178 \narutley 2022-05-12 17:00:59 1923.1689.3.24 \nrjensen
2022-05-11 0:59:26 192.168.213.128 \naestrada 2022-05-09 19:28:12
1924.1680.27.57 \nasundara 2022-05-11 18:38:07 192.168.96.200 \ndkot
2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-12 23:38:46
```

```
19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153 \
njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08
12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01
192.168.148.115 \nyappiah 2022-05-12 10:37:22 192.168.103.10654 \
ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Assign `pattern` to a regular expression pattern that will match
with IP addresses of the form xxx.xxx.xxx.xxx

pattern = "\d\d\d\.\d\d\d\.\d\d\d\.\d\d\d"
```

Task 7

In this task, you'll use the `re.findall()` function on the regular expression pattern stored in the `pattern` variable and the provided `log_file` to extract the corresponding IP addresses. Afterwards, run the cell and take note of what it outputs. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
# Assign `log_file` to a string containing username, date, login time,
and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-
05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32
192.168.190.178 \narutley 2022-05-12 17:00:59 1923.1689.3.24 \nrjensen
2022-05-11 0:59:26 192.168.213.128 \naestrada 2022-05-09 19:28:12
1924.1680.27.57 \nasundara 2022-05-11 18:38:07 192.168.96.200 \ndkot
2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-12 23:38:46
19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153 \
njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08
12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01
192.168.148.115 \nyappiah 2022-05-12 10:37:22 192.168.103.10654 \
ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Assign `pattern` to a regular expression pattern that will match
with IP addresses of the form xxx.xxx.xxx.xxx

pattern = "\d\d\d\.\d\d\d\.\d\d\d\.\d\d\d"

# Use the `re.findall()` function on `pattern` and `log_file` to
extract the IP addresses of the form xxx.xxx.xxx.xxx and display the
results

print(re.findall(pattern, log_file))
```

```
['192.168.152.148', '192.168.190.178', '192.168.213.128',
'192.168.247.153', '192.168.174.117', '192.168.148.115',
'192.168.103.106', '192.168.168.144']
```

Question 2

What are some examples of IP addresses that were extracted? What are some examples of IP addresses that were not extracted? Do any that were not extracted seem to be valid IP addresses?

Some examples are '192.168.152.148', or '192.168.190.178'. Some do seem to be valid even though not extracted.

Task 8

There are some valid IP addresses in the `log_file` that you haven't extracted yet. This is because each segment of digits in a valid IP address can have anywhere between one and three digits.

Adjust the regular expression in the `pattern` to allow for variation in the number of digits in each segment. You can do this by using the `+` symbol after the `\d` symbol. Afterwards, use the updated `pattern` to extract remaining IP addresses. Then, run the cell to analyze the results. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
# Assign `log_file` to a string containing username, date, login time,
and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-
05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32
192.168.190.178 \narutley 2022-05-12 17:00:59 1923.1689.3.24 \nrjensen
2022-05-11 0:59:26 192.168.213.128 \naestrada 2022-05-09 19:28:12
1924.1680.27.57 \nasundara 2022-05-11 18:38:07 192.168.96.200 \ndkot
2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-12 23:38:46
19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153 \
njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08
12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01
192.168.148.115 \nyappiah 2022-05-12 10:37:22 192.168.103.10654 \
ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Update `pattern` to a regular expression pattern that will match
with IP addresses with any variation in the number of digits per
segment

pattern = "192\.168\.\d+\.\d+"

# Use the `re.findall()` function on `pattern` and `log_file` to
extract the IP addresses of the updated form specifed above and
display the results

print(re.findall(pattern, log_file))
```

```
['192.168.152.148', '192.168.22.115', '192.168.190.178',
'192.168.213.128', '192.168.96.200', '192.168.247.153',
'192.168.174.117', '192.168.148.115', '192.168.103.10654',
'192.168.168.144']
```

Question 3

What gets extracted here? Do all extracted IP addresses have between one and three digits in every segment?

No, some have less than three digits.

Task 9

Note that all the IP addresses are now extracted but they also include invalid IP addresses with more than three digits per segment.

In this task, you'll update the `pattern` using curly brackets instead of the `+` symbol. In regular expressions, curly brackets can be used to represent an exact number of repetitions between two numbers. For example, `{2,4}` in a regular expression means between 2 and 4 occurrences of something. Applying this to an example, `\w{2,4}` would match with two, three, or four alphanumeric characters. Afterwards, you'll call the `re.findall()` function on the updated `pattern` and the `log_file` and store the output in a variable named `valid_ip_addresses`.

Then, display the contents of `valid_ip_addresses` and run the cell to analyze the results. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
# Assign `log_file` to a string containing username, date, login time,
and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-
05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32
192.168.190.178 \narutley 2022-05-12 17:00:59 1923.1689.3.24 \nrjensen
2022-05-11 0:59:26 192.168.213.128 \naestrada 2022-05-09 19:28:12
1924.1680.27.57 \nasundara 2022-05-11 18:38:07 192.168.96.200 \ndkot
2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-12 23:38:46
19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153 \
njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08
12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01
192.168.148.115 \nyappiah 2022-05-12 10:37:22 192.168.103.10654 \
ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Assign `pattern` to a regular expression that matches with all valid
IP addresses and only those

pattern = "192\.168\.\d{1,3}\.\d{1,3}"

# Use `re.findall()` on `pattern` and `log_file` and assign
`valid_ip_addresses` to the output

valid_ip_addresses = re.findall(pattern, log_file)
```

```
# Display the contents of `valid_ip_addresses`

print(valid_ip_addresses)

['192.168.152.148', '192.168.22.115', '192.168.190.178',
'192.168.213.128', '192.168.96.200', '192.168.247.153',
'192.168.174.117', '192.168.148.115', '192.168.103.106',
'192.168.168.144']
```

Question 4

What do you notice about the extracted IP addresses here compared to those extracted in the previous two tasks?

They are more accurate.

Task 10

Now, all of the valid IP addresses have been extracted. The next step is to identify flagged IP addresses.

You're given a list of IP addresses that have been previously flagged for unusual activity, stored in a variable named `flagged_addresses`. When these addresses are encountered, they should be investigated further. This list is just for educational purposes and contains examples of private IP addresses that are found only within internal networks.

Display this list and examine what it contains by running the cell. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
# Assign `flagged_addresses` to a list of IP addresses that have been
previously flagged for unusual activity

flagged_addresses = ["192.168.190.178", "192.168.96.200",
"192.168.174.117", "192.168.168.144"]

# Display the contents of `flagged_addresses`

print(flagged_addresses)

['192.168.190.178', '192.168.96.200', '192.168.174.117',
'192.168.168.144']
```

Task 11

Finally, you will write an iterative statement that loops through the `valid_ip_addresses` list and checks if each IP address is flagged. In the following code, the `address` will be the loop variable. Also, include a conditional that checks if the `address` belongs to the `flagged_addresses` list. If so, it should display `"The IP address ______ has been flagged for further analysis."` If not, it should display `"The IP address ______` `does not require further analysis."` Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

```python
# Assign `log_file` to a string containing username, date, login time, and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32 192.168.190.178 \narutley 2022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.128 \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:38:07 192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153 \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08 12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115 \nyappiah 2022-05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Assign `pattern` to a regular expression that matches with all valid IP addresses and only those

pattern = "\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"

# Use `re.findall()` on `pattern` and `log_file` and assign `valid_ip_addresses` to the output

valid_ip_addresses = re.findall(pattern, log_file)

# Assign `flagged_addresses` to a list of IP addresses that have been previously flagged for unusual activity

flagged_addresses = ["192.168.190.178", "192.168.96.200", "192.168.174.117", "192.168.168.144"]

# Iterative statement begins here
# Loop through `valid_ip_addresses` with `address` as the loop variable

for address in valid_ip_addresses:

    # Conditional begins here
    # If `address` belongs to `flagged_addresses`, display "The IP address ______ has been flagged for further analysis."

    if address in flagged_addresses:
        print("The IP address", address, "has been flagged for further analysis.")

    # Otherwise, display "The IP address ______ does not require further analysis."
    else:
        print("The IP address ", address, " does not require further
analysis.")
```

```
The IP address  192.168.152.148  does not require further analysis.
The IP address  192.168.22.115  does not require further analysis.
The IP address 192.168.190.178 has been flagged for further analysis.
The IP address  192.168.213.128  does not require further analysis.
The IP address 192.168.96.200 has been flagged for further analysis.
The IP address  192.168.247.153  does not require further analysis.
The IP address 192.168.174.117 has been flagged for further analysis.
The IP address  192.168.148.115  does not require further analysis.
The IP address  192.168.103.106  does not require further analysis.
The IP address 192.168.168.144 has been flagged for further analysis.
```

Conclusion

What are your key takeaways from this lab?

Regular expressions can help broaden a search but only find elements in a specific format.