Overview

PyTTPKeywords is a set of Python scripts used to recommend a set of TTPs based on a provided reference article. The contents of the reference article are scanned for keywords, and a set of TTPs is recommended based on the found keywords.

This documentation is organized into sections explaining how to use the scripts, as well as the different components necessary to run them. The general sections are as follows:

1. General Script Operation – How to use the scripts and output examples
2. Common Questions – How to perform some common tasks related to the scripts
3. Necessary Dependencies – External modules and tools that the script uses
4. Main Components – A general overview of the Python scripts and the JSON files
5. Script Breakdowns – A detailed explanation for each of the Python scripts
6. JSON File Breakdowns – A detailed explanation for each of the JSON files

Each section will provide a detailed explanation of the components necessary to make the script run. This documentation is meant to make the different components easier to understand and modify for specific implementations.

The different sections are listed in the Table of Contents on the following page.

This document is long, as it covers the different file interactions, the scripts, the JSON files, and the functions needed to run them. For the sake of keeping things simple, the first few pages contain a general overview, as well as some common tasks that might be performed. The rest of the document is used to give a comprehensive understanding of how the scripts work, and how everything interacts with each other.

Table of Contents

General Script Operation

The two main operations for the set of scripts are *keyword generation*, and *keyword recommendation*. *Keyword generation* refers to the creation of the set of {TTP : Keyword} associations, while *keyword recommendation* refers to scraping a webpage for generated keywords.

Both operations are done as follows:

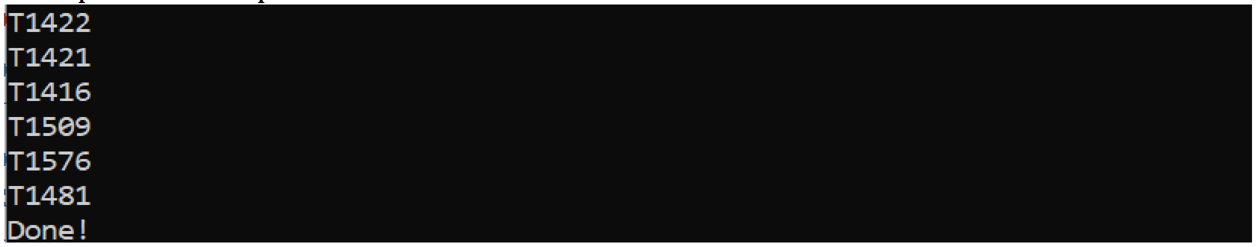
Keyword Generation

Keyword generation is done using the generate_keywords.py script. An important thing to note is that generate_keywords.py needs Python 3 to properly run. There are two options to run the script:

```
python generate_keywords.py

python generate_keywords.py [mobile|enterprise]
```

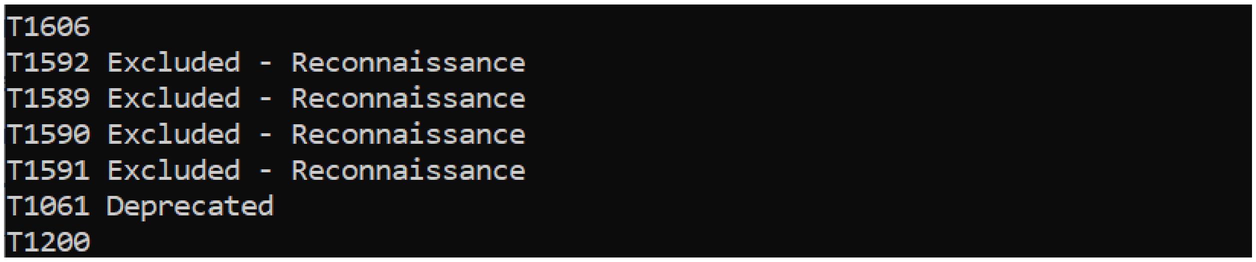
If the script is run with no parameters, it will generate keywords for both the mobile and enterprise matrices. If either mobile or enterprise is specified, the script will only generate keywords for the selected matrix. When running the script, it will print out TTP IDs as it generates their keywords. After all keywords have been generated, it will print "Done!". An example of the output:



```
T1422
T1421
T1416
T1509
T1576
T1481
Done!
```

Screenshot 1: Example output for the generate_keywords.py script

In some cases, the TTP might have been deprecated, or it might be a part of the excluded tactics. In this case, the output for the TTP would look like this:



```
T1606
T1592 Excluded - Reconnaissance
T1589 Excluded - Reconnaissance
T1590 Excluded - Reconnaissance
T1591 Excluded - Reconnaissance
T1061 Deprecated
T1200
```

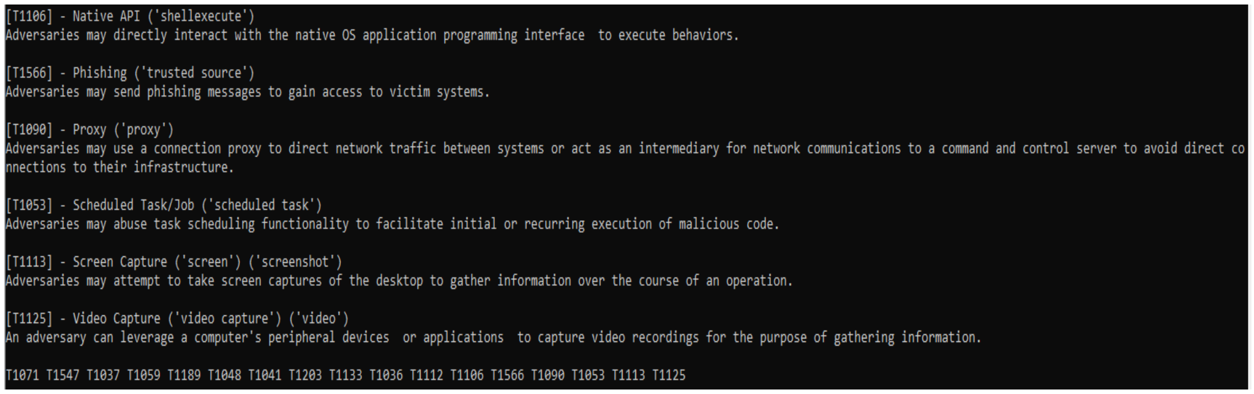
Screenshot 2: Example output for excluded/deprecated TTPs when generating keywords

Keyword Recommendation

Keyword recommendation is done using the recommend_ttps.py script. Like generate_keywords.py, recommend_keywords.py needs Python 3 to properly run. The script is run using the following parameters:

```
python recommend_ttps.py [mobile|enterprise] [url]
```

The first parameter is the matrix to recommend TTPs for. Since Enterprise and Mobile TTPs have different sets of keywords, it's important to distinguish them. The second parameter is the URL to the research article to scrape for TTPs. The output would then look like the following:



```
[T1106] - Native API ('shellexecute')
Adversaries may directly interact with the native OS application programming interface  to execute behaviors.

[T1566] - Phishing ('trusted source')
Adversaries may send phishing messages to gain access to victim systems.

[T1090] - Proxy ('proxy')
Adversaries may use a connection proxy to direct network traffic between systems or act as an intermediary for network communications to a command and control server to avoid direct co
nnections to their infrastructure.

[T1053] - Scheduled Task/Job ('scheduled task')
Adversaries may abuse task scheduling functionality to facilitate initial or recurring execution of malicious code.

[T1113] - Screen Capture ('screen') ('screenshot')
Adversaries may attempt to take screen captures of the desktop to gather information over the course of an operation.

[T1125] - Video Capture ('video capture') ('video')
An adversary can leverage a computer's peripheral devices  or applications  to capture video recordings for the purpose of gathering information.

T1071 T1547 T1037 T1059 T1189 T1048 T1041 T1203 T1133 T1036 T1112 T1106 T1566 T1090 T1053 T1113 T1125
```

Screenshot 3: Example output when running the recommend_ttps.py script

For every TTP that is recommended, the detected keywords are listed, and a short description of the TTP is printed to the screen to provide some context. It can be difficult to determine whether a TTP is applicable using only the ID and the name, so a set of short descriptions is generated whenever keywords are generated.

Unfortunately, some websites use various methods to prevent scripts from automatically scraping their contents. This script outputs an error message for Cloudflare's IUAM mode, and 403 Forbidden errors, but it might miss other techniques.

Common Questions

Q: How do you remove user-specified keyword additions?

- The user-specified keyword additions can be removed from the *user_enterprise_keywords.json* and *user_mobile_keywords.json* files
- Run the generate_keywords.py script afterwards to remove the supplementary keywords from the final set of keywords

Q: How do you remove user-specified keyword exclusions?

- The user-specified keyword exclusions can be removed from the 'match_ignore' key in the *excluded_enterprise_keywords.json* and *excluded_mobile_keywords.json* files
- Run the generate_keywods.py script afterwards to add in the keywords that were being excluded to the final set of keywords

Q: How do you generate a fresh set of keywords? (i.e., on MITRE FRAMEWORK update)

- Run the generate_keywords.py script, using "python generate_keywords.py" for a full set of keywords, or "python generate_keywords.py [mobile|enterprise]" to generate for a specific matrix
- The command needs to be run as a user with the appropriate permissions, and the necessary dependencies installed

Q: How do you ignore keyword suggestions in bulk?

- The necessary files to do this are *excluded_enterprise_keywords.json* and *excluded_mobile_keywords.json*
- Add the term to ignore under the 'ignore' or the 'ignore_if_len_lt_3' keys in the file
- If the term is to be removed from keywords, but not ignored, use the 'replace' key
- Run generate_keywords.py once any additions are done

Q: How do you add keyword suggestions in bulk?

- The format_additional_keywords.py script is used to do this, using similar terms in TTP names to add keywords in bulk
- Run the format_additional_keywords.py script when the additions are made, then the generate_keywords.py script to update the set of keywords

Necessary Dependencies

The scripts make use of multiple Python modules to perform their tasks. These external dependencies need to be installed on the system meant to run the scripts. The external dependencies are the following:

pyattck

pyattck is a Python framework for MITRE ATT&CK. It provides a good way to obtain MITRE information from Python scripts and is the backbone of the generate_keywords.py script.

pyattck objects provide generate_keywords.py with a large amount of information for keyword generation. It can provide a list of Mobile or Enterprise techniques, as well as information about the techniques. pyattck provides multiple sets of information, such as threat actors, malware families, techniques, known tools, and associated commands. Multiple of these components are used when generating the keywords automatically.

GitHub: https://github.com/swimlane/pyattck

Installation:

```
pip install pyattck
```

rake_nltk

rake_nltk is a Python module that is used to extract keywords form a given text. The generate_keywords.py script makes use of rake_nltk to generate a list of keywords using the TTP descriptions provided by pyattck. It can be run to prioritise the more “important” words, the more frequent words, or a balance between both.

Pypi: https://pypi.org/project/rake-nltk/

Installation:

```
pip install rake-nltk
```

LemmInflect

LemmInflect is a module that is used to find the lemma and the inflections for words. The module is used as a part of generate_keywords.py to improve the quality of the keywords. As an example: rake_nltk might generate they key term “performs lateral movement”. LemmInflect would be used to add “perform lateral movement”, “performed lateral movement”, and “performing lateral movement” as key terms. This is just to increase the coverage gained from the automatically generated keywords.

GitHub: https://github.com/bjascob/LemmInflect

Installation:

```
pip install lemminflect
```

spaCy

spaCy is a Python module for natural language processing. The generate_keywords.py script makes use of spaCy to detect whether words are verbs or not. If a word in a key term is found to be a verb, its inflections are then found using LemmInflect.

Website: https://spacy.io/

Installation:

```
pip install -U pip setuptools wheel
pip install -U spacy
python -m spacy download en_core_web_sm
```

BeautifulSoup

BeautifulSoup is a library that is used for parsing and scraping web pages. The recommend_ttps.py script makes use of BeautifulSoup to parse through the HTML for the provided URL’s web page. The script makes use of it to remove any HTML tags, as well as sidebars and text that might not be a part of the article itself.

Pypi: https://pypi.org/project/beautifulsoup4/

Installation:

```
pip install beautifulsoup4
```

Main Components

This section is a general overview of the different components, and what they are used for. The different components are covered in greater depth in the "Script Breakdowns" and "JSON File Breakdowns" sections of this documentation.

The main components of the tool can be separated into 2 categories: The Python scripts, and the JSON files.

Python Scripts

There are 6 Python scripts that are used for this tool:

- recommend_ttps.py
- generate_keywords.py
- format_additional_keywords.py
- add_exclusions.py
- add_user_keywords.py
- add_user_keyword_changes.py

While there are 6 total Python scripts, the ones doing the bulk of the work are recommend_ttps.py and generate_keywords.py. The format_additional_keywords.py script is only used whenever custom keywords are added (as opposed to automatically generated keywords). The add_exclusions.py, add_user_keywords.py and add_user_keyword_changes.py scripts are meant to allow users to add changes themselves, should these scripts be implemented as a part of a website. These were added so that the users could help improve the keyword quality, as there is a limit to what automatic keyword generation can do.

JSON Files

There is a total of 10 JSON files that are used for this tool:

- final_enterprise_keywords.json
- final_mobile_keywords.json
- excluded_enterprise_keywords.json
- excluded_mobile_keywords.json
- additional_enterprise_keywords.json
- additional_mobile_keywords.json
- enterprise_ttp_descriptions.json
- mobile_ttp_descriptions.json
- user_enterprise_keywords.json
- user_mobile_keywords.json

The large number of files are mostly for flexibility. The Enterprise and Mobile files are kept separate so that the recommend_ttps.py and generate_keywords.py scripts can be run against a specific matrix.

Script Breakdowns

recommend_ttps.py

Usage:

```
python recommend_ttps.py [mobile|enterprise] [url]
```

Script Caller:

Manually called from the CLI, or can be called from other scripts in the case of a web server implementation

JSON files needed:

- *final_enterprise_keywords.json* or *final_mobile_keywords.json*
- *enterprise_ttp_descriptions.json* or *mobile_ttp_descriptions.json*

The text below is an example of the output generated by the script:

```
[user@server path]$ python recommend_ttps.py enterprise
https://www.trendmicro.com/en_ca/research/21/g/biopass-rat-new-malware-sniffs-
victims-via-live-streaming.html

[T1053] - Scheduled Task/Job ('scheduled task')
Adversaries may abuse task scheduling functionality to facilitate initial or recurring
execution of malicious code.

[T1113] - Screen Capture ('screen') ('screenshot')
Adversaries may attempt to take screen captures of the desktop to gather information over
the course of an operation.

[T1125] - Video Capture ('video capture') ('video')
An adversary can leverage a computer's peripheral devices  or applications  to capture
video recordings for the purpose of gathering information.

T1053 T1113 T1125
```

The output can be modified as needed, with symbols being added to parse it into its different components. The output shown above is for the script version meant to run from the CLI. As an example, one implementation was formatted as follows:

```
T1125 - Video Capture | ('video capture') ('video') |
An adversary can leverage a computer's peripheral devices or applications  to capture
video recordings for the purpose of gathering information.

T1125
```

generate_keywords.py

Usage:

```
python generate_keywords.py

python generate_keywords.py [mobile|enterprise]
```

Script Caller:

```
No specific caller – normally called manually, but could also be called using a cron job
```

JSON files needed:

- *final_enterprise_keywords.json* and *final_mobile_keywords.json*
- *enterprise_ttp_descriptions.json* and *mobile_ttp_descriptions.json*
- *excluded_enterprise_keywords.json* and *excluded_mobile_keywords.json*
- *additional_enterprise_keywords.json* and *additional_mobile_keywords.json*

The generate_keywords.py script uses the pyattck, rake_nltk, LemmInflect and spaCy libraries to generate a set pf keywords. The relationship between the script and the necessary JSON files are as follows:

1. The script generates a set of keywords, and stores them in a list
2. Either the *excluded_enterprise_keywords.json* or *excluded_mobile_keywords.json* file is opened to retrieve the list of terms to remove or ignore from the generated key terms
3. Once the keywords are refined, either the *additional_enterprise_keywords.json* or the *additional_mobile_keywords.json* file is opened, and the additional keywords are added to the current list of refined keywords
4. The refined keyword list is set as a value in a dictionary, with the TTP ID acting as the key
5. The final keyword dictionary is stored in either the *final_enterprise_keywords.json* or the *final_mobile_keywords.json* file
6. The script generates a short description for each of the TTPs, and stores them all into a Python dictionary with the format {“TTP ID” : [“TTP Name”, “Short TTP description”]}
7. The dictionary is then saved to either the *enterprise_ttp_descriptions.json* or the *mobile_ttp_descriptions.json* files

The files depend on the options specified when running the script.

The functions in generate_keywords.py are as follows (In the order they are called):

generate_keywords(keyword_list, techniques, additional_keywords_file, excluded_words_file, final_keywords_file, ttp_description_file, tool_list, user_file)

Description: This is the function that is called to generate a set of keywords, for either the Enterprise or Mobile matrices. One call to this function generates a full set of keywords, calling other functions as it needs them. This function is mostly used to manage the other function calls.

Caller: The script itself (not called from within a function)

Input:

- keyword_list
    - A list used to temporarily store the generated keywords
- techniques
    - A list of pyattck technique objects
- additional_keywords_file
    - The filename of the file containing the extra keywords to be added to the list
    - This is either *additional_enterprise_keywords.json* or *additional_mobile_keywords.json*
- excluded_keywords_file
    - The filename of the file containing terms used to clean up the keywords
    - This is either *excluded_enterprise_keywords.json* or *excluded_mobile_keywords.json*
- final_keywords_file
    - The filename of the file used to store the final set of keywords
    - This is either *final_enterprise_keywords.json* or *final_mobile_keywords.json*
- ttp_description_file
    - The filename for the file used to store the TTP IDs, names, and descriptions
    - This is either *enterprise_ttp_descriptions.json* or *mobile_ttp_descriptions.json*
- tool_list
    - A list of pyattck tool objects that are associated to a given matrix
    - This is either attack.enterprise.tools or attack.mobile.tools, where attack is the generated pyattck object
- user_file
    - Filename for the file containing the user-specified keywords

Output: The function has no specific output. At the end of the function, the generated keywords are stored into their designated JSON file.

check_tactic(technique)

Description: This function is used to filter out techniques that are a part of excluded tactics. As an example, the Reconnaissance and the Resource Development tactics take place before an attack, so they aren't applicable to a malware's behavior. To avoid detections, keywords aren't generated for the excluded tactics.

Caller: generate_keywords()

Input:

- technique
    - A pyattck technique object
    - This object is obtained by looping through the attack.enterprise.techniques or the attack.mobile.techniques object lists

Output: True if the technique is to be ignored, False if the technique is to be kept

extract_keywords(text)

Description: This function uses rake_nltk to generate a list of keywords using a given technique's description. The technique descriptions are obtained using pyattck.

Caller: generate_keywords()

Input:

- text
    - A string containing the description for a MITRE technique
    - This object is obtained using pyattck's technique object's description member, which is technique.description

Output: A list of generated keywords.

clean_up_keywords(raw_list, clean_list, excluded_keywords)

Description: This function is used to clean up the automatically generated keywords. Some of the keywords generated by rake_nltk are general and aren't unique enough to be considered keywords. There are 3 possible options when cleaning up the keywords. The action is performed based on the detected word's key in the exclusion file. The actions are as follows:

- For the "replace" key, any detected "replace" terms are removed from the generated keywords, with the remainder acting as the refined version of the generated keywords
    - For example: if "adversaries may" is one of the "replace" terms, "adversaries may take screenshots" would be refined to "take screenshots"
- For the "ignore" key, any generated keywords containing an "ignore" term aren't added to the set of refined keywords
    - For example: if "malware" is one of the "ignore" terms, then any generated keywords containing "malware" won't be copied over to the refined terms
- For the "ignore_if_len_lt_3" key, any generated keywords of 2 words or less are ignored if they contain an "ignore_if_len_lt_3" term

Caller: generate_keywords()

Input:

- raw_list
    - List containing the unrefined list of generated keywords
- clean_list
    - List used to store the refined list of generated keywords
- excluded_keywords
    - Python dictionary containing the terms associated to the "replace", "ignore", and "ignore_if_len_lt_3" keys
    - Either obtained from *excluded_enterprise_keywords.json* or *excluded_mobile_keywords.json*

Output: This function does not have a specific return value. The refined keywords are added to clean_list.

add_supplementary_keywords(keywords, name, current_keywords)

Description: This function is used to add additional keywords to the automatically generated ones. As an example, Microsoft Office file extensions such as ".docx" or ".xlsb" might be added as keywords to any phishing TTPs. These keywords are manually created and formatted using the format_additional_keywords.py script, which is the next script covered in this section.

Caller: generate_keywords()

Input:

- keywords
    - Dictionary containing the keywords to add, formatted as {"Part in technique name" : ["list", "of", "keywords"]}
    - The additional keywords are read from either the *additional_enterprise_keywords.json* or the *additional_mobile_keywords.json* files
- name
    - A string containing the name for the current technique
        - This is obtained using pyattck's technique object's name member, which is obtained using technique.name
    - The name is what is being checked to decide whether a set of keywords are to be added to a technique
    - An example:
        - For this example, we have {"phish" : [".docx", ".xlsb"]} as the value of keywords
        - Since the key "phish" is found in the technique name "Phishing", ".docx" and ".xlsb" would then be added as keywords associated to the key "T1566", which is the Enterprise TTP ID for "Phishing"
- current_keywords
    - The list containing the technique's current keywords

Output: This function does not have a specific return value. The additional keywords are added to current_keywords.

generate_inflections(keywords_list, current_keywords)

Description: This function is used to increase the coverage of the generated keywords once they are refined. Since the generated keywords are only a single inflection, their coverage isn't ideal. LemmInflect and spaCy are used here to do this.

- The inflections in the automatically generated keywords can be inconsistent, so all their variations are added as keywords
  - As an example: if "perform lateral movement" was in the list of refined keywords, "performs lateral movement", "performing lateral movement", and "performed lateral movement" would be added as well

Caller: generate_keywords()

Input:

- keywords_list
  - This is a copy of the current set of keywords (a copy is used to avoid infinite loops)
  - This is the list of keywords that is used as a reference when generating the inflections
- current_keywords
  - This is the original list containing the keywords
  - As inflections are generated from keywords_list, they are added to current_keywords

Output: This function does not have a specific return value. The inflected keywords are added to the current_keywords list.

add_associated_commands(attack_technique, current_keywords)

Description: This function makes use of pyattck's associated commands member, where techniques have a set of commands associated to them, indicated in the technique.command_list member. These commands are added as keywords.

Caller: generate_keywords()

Input:

- attack_technique
  - A pyattck technique object
- current_keywords
  - The list containing the current set of keywords for the technique

Output: This function does not have a specific return value. The associated commands are added to the current_keywords list.

remove_user_specified_keywords(excluded_keywords)

Description: This function was added later in the project, after an additional script was added allowing users to specify keywords to remove from a possible web page implementation. The script add_exclusions.py was added to allow users to provide a list of keywords that aren't applicable to the TTP being recommended. The add_exclusions.py script adds the user-provided keywords to either the *excluded_enterprise_keywords.json* or the *excluded_mobile_keywords.json* files.

Caller: generate_keywords()

Input:

- excluded_keywords
    - A Python dictionary containing the keywords, as well as their different operations
    - This is the dictionary covered in clean_up_keywords() on page 12
    - For this specific function, a new key 'match_ignore' is used
        - 'match_ignore' is for keywords specified by the users using the add_exclusions.py script
        - If any of the current keywords are an exact match for the 'match_ignore' keywords, they are removed

Output:

- This function does not have a specific return value
- If the function finds a keyword in the keyword_dict dictionary values matching one of the terms in excluded_keywords['match_ignore'], the detected term is removed from keyword_dict

add_associated_tools(tool_list)

Description: The pyattck enterprise and mobile classes each have a list of tool objects. These tool objects have a member called techniques to list all techniques the tools can be associated with. The tool class also has a member called additional_names, which are aliases for the tool. The tool names and any additional names are added as keywords for the techniques the tools are associated to.

Caller: generate_keywords()

Input:

- tool_list
    - A list of pyattck tool objects, either from attack.enterprise.tools or attack.mobile.tools

Output: This does not have a specific return value. The associated tool names are added to the keyword_dict dictionary.

add_user_keywords(user_file)

Description: This function is used to add user-specified keywords to the final dictionary of keywords, before it is dumped to the final keywords file

Caller: generate_keywords()

Input:

- User_file
    - Name of the JSON file being used to store the set of user-specified keyword additions
    - This is either *user_enterprise_keywords.json* or *user_mobile_keywords.json*

Output: This function does not have a specific return value. The keywords are added to their respective keys in the dictionary containing the set of generated keywords.

generate_ttp_descriptions(output_file, techniques)

Description: This function is used to help provide some context for the TTP recommendations, so that the users can more easily determine whether a TTP recommendation is applicable to the article being scraped. This function formats technique names and descriptions to print out to the user, to provide them with more information than just a TTP ID.

Caller: The script itself (not called from within a function)

Input:

- output_file
    - Name of the JSON file being used to store the final set of keywords
    - This is either *final_enterprise_keywords.json* or *final_mobile_keywords.json*
- techniques
    - A list of pyattck technique objects

Output: This function does not have a specific return value. The dictionary containing the technique IDs, names and descriptions is saved to the output_file.

format_additional_keywords.py

Usage:

```
python format_additional_keywords.py
```

Script Caller:

No specific caller – called manually after the keywords inside of the script are changed

JSON files needed:

- *additional_enterprise_keywords.json* and *additional_mobile_keywords.json*

The format_additional_keywords.py script is used to format files containing custom keywords. While automatically generating keywords is better for long-term operation, the generated keywords are generally limited. There are a lot of keywords that wouldn't be picked up by rake_nltk when given a technique description. To use the phishing example again, the description for the Enterprise Phishing technique does not list the different Microsoft Office file extensions, which are commonly used in phishing campaigns. The format_additional_keywords.py script is used to compensate for this issue.

To return to the phishing example, the format_additional_keywords.py script is essentially a bunch of lines such as the following:

```
raw_terms = ['social engineer', 'spearphish', 'phish', 'pdf', 'doc ', 'xls', 'ppt', 'docx', 'docm']

append_keywords('phishing', raw_terms)
```

the append_keywords() function just adds the list of additional keywords to a dictionary, which is then saved to either *additional_enterprise_keywords.json* or *additional_mobile_keywords.json*.

The manually created keywords are then used by generate_keywords.py whenever it runs.

add_exclusions.py

Usage:

```
python add_exclusions.py [mobile|enterprise] [“string,of,comma,separated,keywords”]
```

Script Caller:

Can be manually called from the CLI, or be implemented into a web server

JSON files needed:

- *excluded_enterprise_keywords.json* and *excluded_mobile_keywords.json*

The add_exclusions.py script is used to allow users to help refine keywords by adding in specific TTP:keywords associations. The script takes in 2 parameters; the first is the matrix that the refining is targeted towards, and the second is a string of comma-separated keywords to remove (enclosed in double quotes). The string is split using the commas as the delimiter, and the keywords are added as values under the ‘match_ignore’ key in the *excluded_enterprise_keywords.json* and *excluded_mobile_keywords.json* files.

add_user_keywords.py

Usage:

```
python add_user_keywords [mobile|enterprise] [“Txxx:multiple,keywords:Txxx:etc.”]
```

Script Caller:

Can be manually called from the CLI, or be implemented into a web server

JSON files needed:

- *user_enterprise_keywords.json* and *user_mobile_keywords.json*

The add_user_keywords.py script is used to allow users to add specific TTP:keyword associations to increase the keyword coverage. The idea behind this script is that the user might notice that the information in a reference article might indicate a specific TTP, while the TTP itself isn’t being detected by the current set of keywords. This script allows the user to add keywords they believe are missing, improving the quality of the keywords in the process.

add_user_keyword_changes.py

Usage:

```
python add_user_keyword_changes.py [mobile|enterprise]
```

Script Caller:

```
Can be manually called from the CLI, or be implemented into a web server
```

JSON files needed:

- *user_enterprise_keywords.json* and *user_mobile_keywords.json*
- *excluded_enterprise_keywords.json* and *excluded_mobile_keywords.json*
- *final_enterprise_keywords.json* and *final_mobile_keywords.json*

The add_user_keyword_changes.py script is used to update the file contents in the *final_enterprise_keywords.json* and *final_mobile_keywords.json* files, instead of having to run generate_keywords.py, as it is more time consuming. This script was created to update the keywords files quickly once a user submits changes, to minimize the time from the input to the updated set of recommendations.

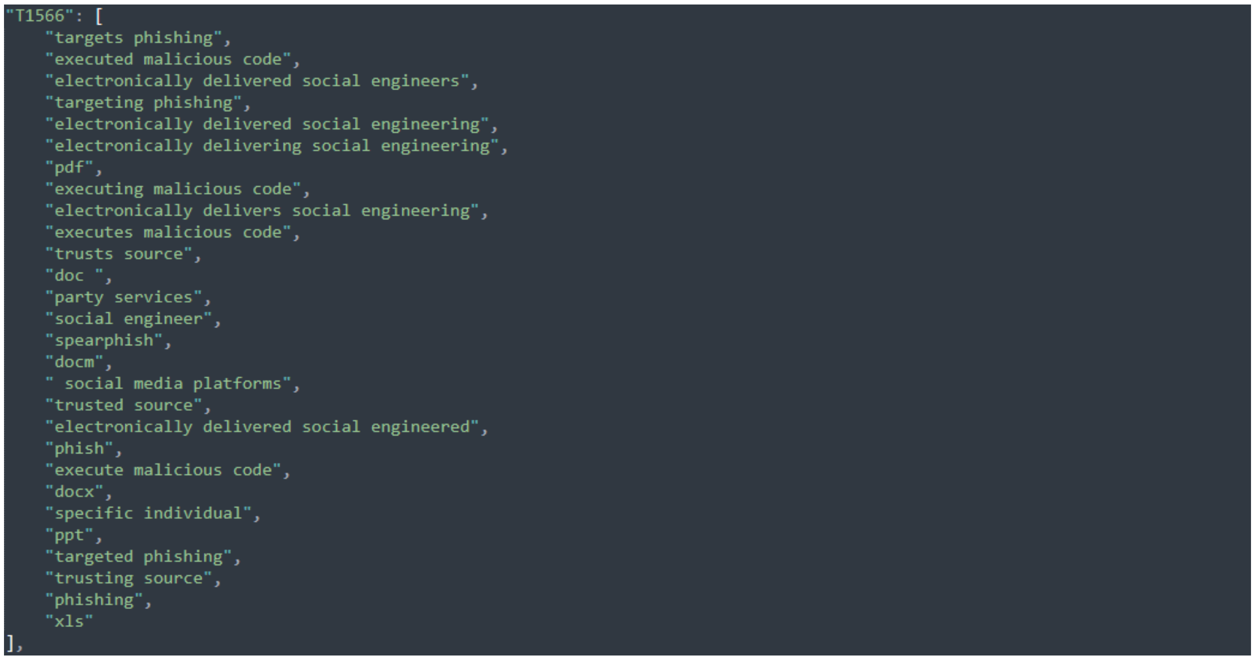
JSON File Breakdowns

All the JSON files are meant to be stored in a relative path, from the main Python scripts:

```
./ttpkeywords/
```

final_enterprise_keywords.json and **final_mobile_keywords.json**

Description: These are the files that are used to store the final set of generated keywords. Both files are formatted the same, with the information stored under the format {"TTP ID" : ["list", "of", "keywords", "here"]}. An example screenshot of the formatting:



```
"T1566": [
    "targets phishing",
    "executed malicious code",
    "electronically delivered social engineers",
    "targeting phishing",
    "electronically delivered social engineering",
    "electronically delivering social engineering",
    "pdf",
    "executing malicious code",
    "electronically delivers social engineering",
    "executes malicious code",
    "trusts source",
    "doc ",
    "party services",
    "social engineer",
    "spearphish",
    "docm",
    " social media platforms",
    "trusted source",
    "electronically delivered social engineered",
    "phish",
    "execute malicious code",
    "docx",
    "specific individual",
    "ppt",
    "targeted phishing",
    "trusting source",
    "phishing",
    "xls"
],
```

Screenshot 4: Keyword formatting example in the *final_enterprise_keywords.json* file

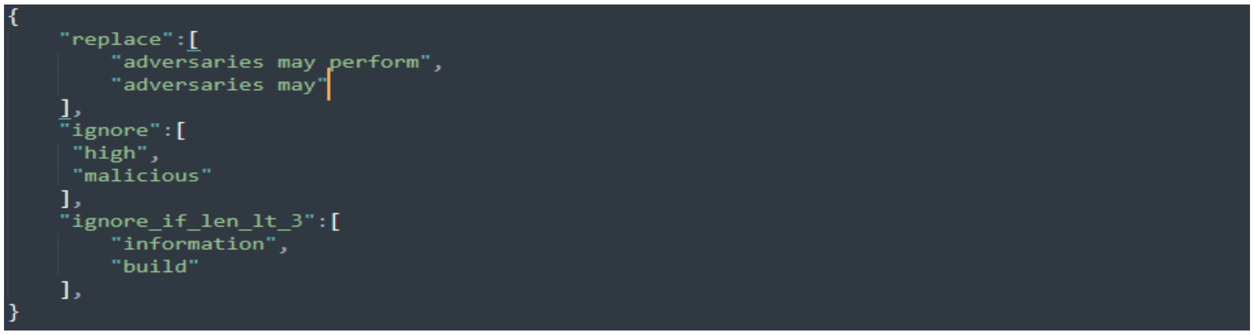
Associated Scripts: recommend_ttps.py, generate_keywords.py and add_user_keyword_changes.py

excluded_enterprise_keywords.json and excluded_mobile_keywords.json

Description: These are the files that store the words used to refine the generated keywords. There are 3 keys being used in the file, with each key representing a different action when refining the keywords:

- "replace"
  - If the generated keywords contain any of the terms under the "replace" key, then the discovered terms are removed from the keywords, with the remainder acting as the refined keywords
  - Example: with the file contents being {"replace" : ["adversaries may"]}, the generated keyword "adversaries may take screenshots" would be refined to "take screenshots"
- "ignore"
  - If any of the terms under "ignore" are found in a set of keywords, that set of keywords is excluded from the final refined version
  - Example: with the file contents being {"ignore" : ["malware"]}, the generated keywords "download malware" would be excluded from the list of refined keywords
- "ignore_if_len_lt_3"
  - This does the same thing as the "ignore" key, but the keywords are only ignored if the length of a set of keywords is 2 words or less
  - This is for some more specific use cases, where the work only provides value with some context, as opposed to on its own
  - For example: with the file contents being {"ignore_if_len_lt_3 : ["command execution"]}, the keywords "command execution" would be ignored, while the keywords "remote command execution" would be kept

An example screenshot of the formatting in the files:



```
{
    "replace":[
        "adversaries may perform",
        "adversaries may"
    ],
    "ignore":[
      "high",
      "malicious"
    ],
    "ignore_if_len_lt_3":[
        "information",
        "build"
    ],
}
```

Screenshot 5: Keyword formatting example in the excluded keywords files

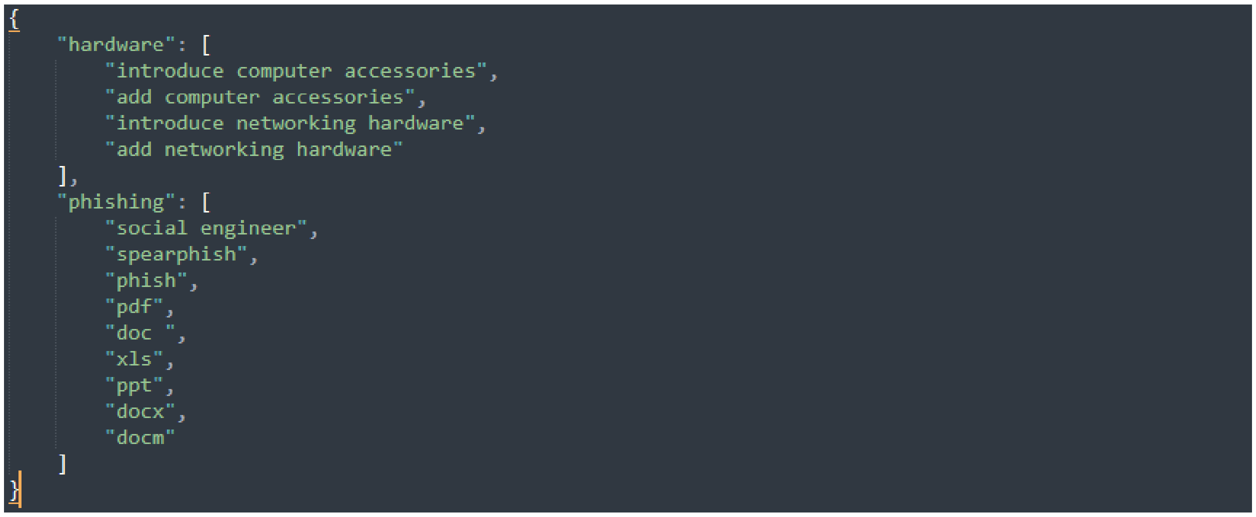
Later in the project, the 'match_ignore' key was implemented for user-specified keyword exclusions using the add_exclusions.py script.

Associated Scripts: generate_keywords.py, add_exclusions.py and add_user_keyword_changes.py

additional_enterprise_keywords.json and **additional_mobile_keywords.json**

Description: These are the files that store the custom made keywords. The words themselves are declared in the format_additional_keywords.py script, and are saved into the JSON file corresponding to the keywords' matrices.

An example screenshot of the formatting in the files:



```
{
    "hardware": [
        "introduce computer accessories",
        "add computer accessories",
        "introduce networking hardware",
        "add networking hardware"
    ],
    "phishing": [
        "social engineer",
        "spearphish",
        "phish",
        "pdf",
        "doc ",
        "xls",
        "ppt",
        "docx",
        "docm"
    ]
}
```

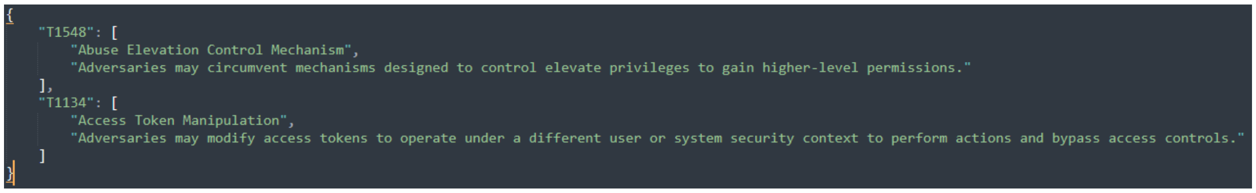
Screenshot 6: Keyword formatting example in *additional_enterprise_keywords.json*

Associated Scripts: format_additional_keywords.py and generate_keywords.py

enterprise_ttp_descriptions.json and mobile_ttp_descriptions.json

Description: These are the files used to store the technique IDs, their names, and a short description.

An example screenshot of the formatting in the files:



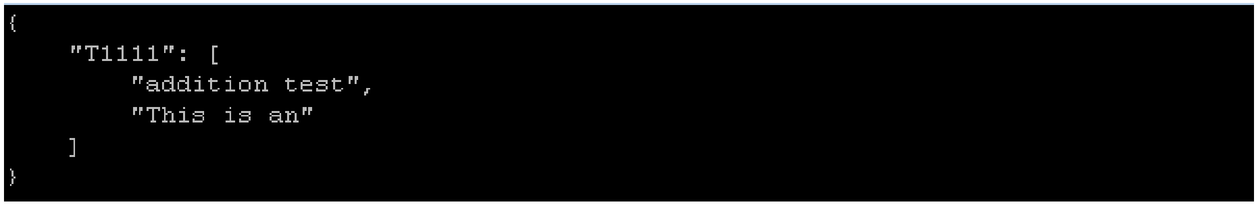
```
{
    "T1548": [
        "Abuse Elevation Control Mechanism",
        "Adversaries may circumvent mechanisms designed to control elevate privileges to gain higher-level permissions."
    ],
    "T1134": [
        "Access Token Manipulation",
        "Adversaries may modify access tokens to operate under a different user or system security context to perform actions and bypass access controls."
    ]
}
```

Screenshot 7: Technique description formatting in *enterprise_ttp_descriptions.json*

Associated Scripts: recommend_ttps.py and generate_keywords.py user_enterprise_keywords.json and user_mobile_keywords.json

Description: These are the files that are used to store the user-specified supplementary keywords. The automatic keyword generation is limited in what it can do, so these files are used to store the keywords that are manually associated to a TTP from the users' input.

An example screenshot of the formatting in the files:



```
{
    "T1111": [
        "addition test",
        "This is an"
    ]
}
```

Screenshot 8: Supplementary TTP keyword formatting in *user_enterprise_keywords.json*

Associated Scripts: add_user_keywords.py and add_user_keyword_changes.py