pyattck Documentation

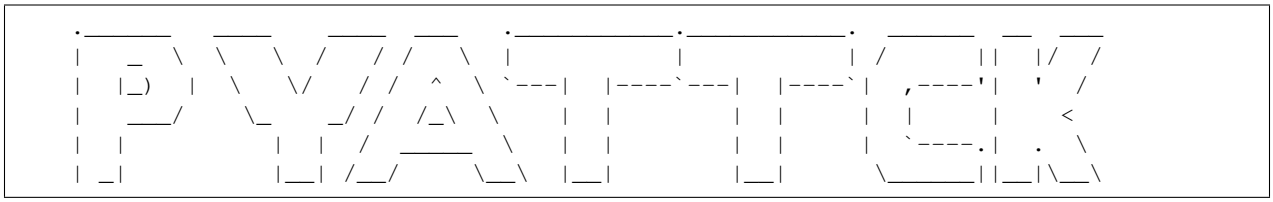
Release 2.0.0

Swimlane, Josh Rickard (MSAdministrator)

Oct 22, 2021

Contents

pyattck



```
A Python package to interact with MITRE ATT&CK Frameworks
```

Current Version is 5.0.0

pyattck is a light-weight framework for MITRE ATT&CK Frameworks. This package extracts details from the MITRE Enterprise, PRE-ATT&CK, Mobile, and ICS Frameworks.

CHAPTER 1

Why?

`pyattck` assist organizations and individuals with accessing MITRE ATT&CK Framework(s) in a programmatic way. Meaning, you can access all defined actors, malwares, mitigations, tactics, techniques, and tools defined by the Enterprise, Mobile, Pre-Attck, and ICS frameworks via a command-line utility or embedding into your own code base.

There are many reasons why you would want to access this data in an automated (scripted/coded) way but a few examples are:

- Generate reports with additional details about a technique (or any object defined in the framework)
- A build pipeline of detection rules with additional MITRE ATT&CK details for categorization
- Quickly searching for specific details about a technique without navigating a web page

There are other benefits that `pyattck` provide as well which includes the ability to provide additional contextual data. You can find more information about this data here but the basics are that `pyattck` utilizes multiple open-source repositorties to gather additional contextual data like commands used to execute a technique, country and other details about a malicious actor, other variants of malware similar to a defined tool/malware, etc.

This additional context is what makes `pyattck` truly powerful and enables people to build more robust testing and validation of their detection rules, validates testing assumptions, etc. Truly there are countless ways that `pyattck` could be utilized to help blue, red, and purple teams defend organizations (and themselves).

CHAPTER 2

Features

The **pyattck** package retrieves all Tactics, Techniques, Actors, Malware, Tools, and Mitigations from the MITRE ATT&CK Frameworks as well as any defined relationships within the MITRE ATT&CK dataset (including subtechniques).

In addition, Techniques, Actors, and Tools (if applicable) now have collected data from third-party resources that are accessible via properties on a technique. For more detailed information about these features, see External Datasets.

The **pyattck** package allows you to:

- Specify a URL or local file path for the MITRE ATT&CK Enterprise Framework json, generated dataset, and/or a config.yml file.
- Retrieve an image_logo of an actor (when available). If an image_logo isn't available, it generates an ascii_logo.
- Search the external dataset for external commands that are similar using `search_commands`.
- Access data from the MITRE PRE-ATT&CK Framework
- Access data from the MITRE Mobile ATT&CK Framework
- Access data from the MITRE ICS ATT&CK Framework
- Access subtechniques as nested objects or you can turn it off and access as normal technique
- Access compliance controls (currently NIST 800-53 v5) related to a MITRE ATT&CK Technique

CHAPTER 3

Table of Contents

3.1 Installation

You can install **pyattack** on OS X, Linux, or Windows. You can also install it directly from the source. To install, see the commands under the relevant operating system heading, below.

3.1.1 Prerequisites

The following libraries are required and installed by pyattck:

```
requests
pendulum>=1.2.3,<1.3
pyfiglet==0.8.post1
PyYaml>=5.4.1
Pillow==8.2.0
fire==0.3.1
```

3.1.2 macOS, Linux and Windows:

```
pip install pyattck
```

3.1.3 Installing from source

```
git clone https://github.com/swimlane/pyattck.git
cd pyattck
python setup.py install
```

3.2 Usage example

To use **pyattck** you must instantiate an **Attck** object. Although you can interact directly with each class, the intended use is through a **Attck** object:

```
from pyattck import Attck

attack = Attck()
```

By default, `subtechniques` are accessible under each technique object. You can turn this behavior off by passing `nested_subtechniques=False` when creating your `Attck` object.

As an example, the default behavior looks like the following example:

```
from pyattck import Attck

attack = Attck()

for technique in attack.enterprise.techniques:
    print(technique.id)
    print(technique.name)
    for subtechnique in technique.subtechniques:
        print(subtechnique.id)
        print(subtechnique.name)
```

You can access the following `main` properties on your **Attck** object:

- enterprise
- preattack
- mobile
- ics

Once you specify the MITRE ATT&CK Framework, you can access additional properties.

Here are the accessible objects under the *Enterprise* property:

- *actors*
- *controls*
- *malwares*
- *mitigations*
- *tactics*
- *techniques*
- *tools*

For more information on object types under the `enterprise` property, see *Enterprise*.

Here are the accessible objects under the *PreAttck* property:

- *actors*
- *tactics*
- *techniques*

For more information on object types under the `preattck` property, see *PreAttck*.

Here are the accessible objects under the *Mobile* property:

- *actors*
- *malwares*
- *mitigations*
- *tactics*
- *techniques*
- *tools*

For more information on object types under the `mobile` property, see *Mobile*.

Here are the accessible objects under the *ICS* property:

- *controls*
- *malwares*
- *mitigations*
- *tactics*
- *techniques*

For more information on object types under the `ics` property, see *ICS*.

3.3 Configuration

`pyattck` allows you to configure if you store external data and where it is stored.

```
from pyattck import Attck

attck = Attck(
    nested_subtechniques=True,
    use_config=False,
    save_config=False,
    config_file_path='~/pyattck/config.yml',
    data_path='~/pyattck/data',
    enterprise_attck_json="https://raw.githubusercontent.com/mitre/cti/master/
↪enterprise-attack/enterprise-attack.json",
    pre_attck_json="https://raw.githubusercontent.com/mitre/cti/master/pre-attack/pre-
↪attack.json",
    mobile_attck_json="https://raw.githubusercontent.com/mitre/cti/master/mobile-
↪attack/mobile-attack.json",
    ics_attck_json="https://raw.githubusercontent.com/mitre/cti/master/ics-attack/ics-
↪attack.json",
    nist_controls_json="https://raw.githubusercontent.com/center-for-threat-informed-
↪defense/attack-control-framework-mappings/master/frameworks/ATT%26CK-v9.0/nist800-
↪53-r5/stix/nist800-53-r5-controls.json",
```

(continues on next page)

(continued from previous page)

```
    generated_attck_json="https://swimlane-pyattck.s3.us-west-2.amazonaws.com/
↪generated_attck_data.json",
    generated_nist_json="https://swimlane-pyattck.s3.us-west-2.amazonaws.com/attck_to_
↪nist_controls.json",
    **kwargs
)
```

By default, `pyattck` will (now) pull the latest external data from their respective locations using HTTP GET requests. `pyattck` currently pulls from the following locations:

- enterprise_attck_json="https://raw.githubusercontent.com/mitre/cti/master/enterprise-attack/enterprise-attack.json"
- pre_attck_json="https://raw.githubusercontent.com/mitre/cti/master/pre-attack/pre-attack.json"
- mobile_attck_json="https://raw.githubusercontent.com/mitre/cti/master/mobile-attack/mobile-attack.json"
- ics_attck_json="https://raw.githubusercontent.com/mitre/cti/master/ics-attack/ics-attack.json"
- nist_controls_json="https://raw.githubusercontent.com/center-for-threat-informed-defense/attack-control-framework-mappings/master/frameworks/ATT%26CK-v9.0/nist800-53-r5/stix/nist800-53-r5-controls.json"
- generated_attck_json="https://swimlane-pyattck.s3.us-west-2.amazonaws.com/generated_attck_data.json"
- generated_nist_json="https://swimlane-pyattck.s3.us-west-2.amazonaws.com/attck_to_nist_controls.json"

You have several options when instantiating the `Attck` object. As of `4.0.0` you can now specify any of the following options:

- use_config - When you specify this argument as `True` pyattck will attempt to retrieve the configuration specified in the `config_file_path` location. If this file is corrupted or cannot be found, we will default to retrieving data from the specified `*_attck_json` locations.
- save_config - When you specify this argument as `True` pyattck will save the configuration file to the specified location set by `config_file_path`. Additionally, we will save all downloaded files to the `data_path` location specified. If you have specified a local path location instead of a download URL for any of the `*_attck_json` parameters we will save this location in our configuration and reference this location going forward.
- config_file_path - The path to store a configuration file. Default is `~/pyattck/config.yml`
- data_path - The path to store any data files downloaded to the local system. Default is `~/pyattck/data`

3.3.1 JSON Locations

Additionally, you can specify the location for each individual `*_attck_json` files by passing in either a URI or a local file path. If you have passed in a local file path, we will simply read from this file.

If you have used the default values or specified an alternative URI location to retrieve these JSON files from, you can additionally pass in `**kwargs` that will be passed along to the `Requests` python package when performing any HTTP requests.

3.4 Note

We understand that there are many different open-source projects being released, even on a daily basis, but we wanted to provide a straightforward Python package that allowed the user to identify known relationships between all verticals of the MITRE ATT&CK Framework.

If you are unfamiliar with the MITRE ATT&CK Framework, there are a few key components to ensure you have a firm grasp around. The first is Tactics & Techniques. When looking at the MITRE ATT&CK Framework, the Tactics are the columns and represent the different phases of an attack.

> The MITRE ATT&CK Framework is NOT an all encompassing/defacto security coverage map - it is rather a FRAMEWORK and additional avenues should also be considered when assessing your security posture.

Techniques are the rows of the framework and are categorized underneath specific Tactics (columns). They are data points within the framework that provides guidance when assessing your security gaps. Additionally, (most) Techniques contain mitigation guidance in addition to information about their relationship to tools, malware, and even actors/groups that have used this technique during recorded attacks.

This means, if your organization is focused on TTPs (Tactics Techniques and Procedures) used by certain actors/groups then MITRE ATT&CK Framework is perfect for you. If you are not at this security maturing within your organization, no worries! The ATT&CK Framework still provides really good guidance in a simple and straightforward layout, but programmatically it is not straightforward–especially if you wanted to measure (or map) your security controls using the framework.

3.4.1 Developing and Testing

You can add features or bugs or run the code in a development environement.

1. To get a development and testing environment up and running, use this Dockerfile.
2. To use the `Dockerfile` run, cd to this repository directory and run:

```
docker build --force-rm -t pyattck .
```

1. Next, run the docker container:

```
docker run pyattck
```

Running this calls the test python file in bin/test.py.

1. Modify the test python file for additional testing and development.

3.5 Running the tests

Tests within this project should cover all available properties and methods. As this project grows the tests will become more robust but for now we are testing that they exist and return outputs.

3.6 Contributing

Please read CONTRIBUTING.md for details on our code of conduct, and the process for submitting pull requests to us.

3.7 Versioning

We use SemVer for versioning.

3.8 Change Log

For details on features for a specific version of `pyattck`, see the CHANGELOG.md.

3.9 Authors

- Josh Rickard - *Initial work* - MSAdministrator

See also the list of contributors.

3.10 License

This project is licensed under the MIT License.

3.11 Acknowledgments

First of all, I would like to thank everyone who contributes to open-source projects, especially the maintainers and creators of these projects. Without them, this capability would not be possible.

This data set is generated from many different sources. As we continue to add more sources, we will continue to add them here. Again thank you to all of these projects. In no particular order, `pyattck` utilizes data from the following projects:

- Mitre ATT&CK APT3 Adversary Emulation Field Manual
- Atomic Red Team (by Red Canary)
- Atomic Threat Coverage
- attck_empire (by dstepanic)
- sentinel-attack (by BlueTeamLabs)
- Litmus_test (by Kirtar22)
- nsm-attack (by oxtf)
- osquery-attck (by teoseller)
- Mitre Stockpile
- SysmonHunter (by baronpan)
- ThreatHunting-Book (by 12306Bro)
- threat_hunting_tables (by dwestgard)
- APT Groups & Operations
- C2Matrix (by @jorgeorchilles, @brysonbort, @adam_mashinchi)
- Elemental
- MalwareArchaeology - ATTACK
- Attack-Technique-Dataset

3.11.1 Configuration

pyattck allows you to configure if you store external data and as well as where it is stored. Below shows all available parameters when instantiating the Attck object.

```
from pyattck import Attck

attck = Attck(
    nested_subtechniques=True,
    use_config=False,
    save_config=False,
    config_file_path='~/pyattck/config.yml',
    data_path='~/pyattck/data',
    enterprise_attck_json="https://raw.githubusercontent.com/mitre/cti/master/
↪enterprise-attack/enterprise-attack.json",
    pre_attck_json="https://raw.githubusercontent.com/mitre/cti/master/pre-attack/pre-
↪attack.json",
    mobile_attck_json="https://raw.githubusercontent.com/mitre/cti/master/mobile-
↪attack/mobile-attack.json",
    ics_attck_json="https://raw.githubusercontent.com/mitre/cti/master/ics-attack/ics-
↪attack.json",
    nist_controls_json="https://raw.githubusercontent.com/center-for-threat-informed-
↪defense/attack-control-framework-mappings/master/frameworks/ATT%26CK-v9.0/nist800-
↪53-r5/stix/nist800-53-r5-controls.json",
    generated_attck_json="https://swimlane-pyattck.s3.us-west-2.amazonaws.com/
↪generated_attck_data.json",
    generated_nist_json="https://swimlane-pyattck.s3.us-west-2.amazonaws.com/attck_to_
↪nist_controls.json",
    **kwargs
)
```

By default, pyattck will pull the latest external data from their respective locations using HTTP GET requests. pyattck currently pulls from the following locations:

- enterprise_attck_json="https://raw.githubusercontent.com/mitre/cti/master/enterprise-attack/enterprise-attack.json"
- pre_attck_json="https://raw.githubusercontent.com/mitre/cti/master/pre-attack/pre-attack.json"
- mobile_attck_json="https://raw.githubusercontent.com/mitre/cti/master/mobile-attack/mobile-attack.json"
- ics_attck_json="https://raw.githubusercontent.com/mitre/cti/master/ics-attack/ics-attack.json"
- nist_controls_json="https://raw.githubusercontent.com/center-for-threat-informed-defense/attack-control-framework-mappings/master/frameworks/ATT%26CK-v9.0/nist800-53-r5/stix/nist800-53-r5-controls.json"
- generated_attck_json="https://swimlane-pyattck.s3.us-west-2.amazonaws.com/generated_attck_data.json"
- generated_nist_json="https://swimlane-pyattck.s3.us-west-2.amazonaws.com/attck_to_nist_controls.json"

You have several options when instantiating the Attck object. As of 4.0.0 you can now specify any of the following options:

- use_config - When you specify this argument as True pyattck will attempt to retrieve the configuration specified in the config_file_path location. If this file is corrupted or cannot be found, we will default to retrieving data from the specified *_attck_json locations.
- save_config - When you specify this argument as True pyattck will save the configuration file to the specified location set by config_file_path. Additionally, we will save all downloaded files to the data_path location specified. If you have specified a local path location instead of a download URL for any of the

*_attck_json parameters we will save this location in our configuration and reference this location going forward.

- config_file_path - The path to store a configuration file. Default is ~/pyattck/config.yml
- data_path - The path to store any data files downloaded to the local system. Default is ~/pyattck/data

JSON Locations

Additionally, you can specify the location for each individual *_attck_json files by passing in either a URI or a local file path. If you have passed in a local file path, we will simply read from this file.

If you have used the default values or specified an alternative URI location to retrieve these JSON files from, you can additionally pass in **kwargs that will be passed along to the Requests python package when performing any HTTP requests.

Configuration Class

class pyattck.configuration.**Configuration**
Bases: object

3.11.2 Attck

This documentation provides details about the main entry point called Attck within the pyattck package.

This class provides access to the MITRE Enterprise, PRE-ATT&CK, Mobile, and ICS Frameworks.

- MITRE Enterprise ATT&CK Framework
- MITRE PRE-ATT&CK Framework
- MITRE Mobile ATT&CK Framework
- MITRE ICS ATT&CK Framework

By default, subtechniques are accessible under each technique object.

As an example, the default behavior looks like the following example:

```
from pyattck import Attck

attack = Attck()

for technique in attack.enterprise.techniques:
    print(technique.id)
    print(technique.name)
    for subtechnique in technique.subtechniques:
        print(subtechnique.id)
        print(subtechnique.name)
```

You can turn this behavior off by passing nested_subtechniques=False when creating your Attck object. When turning this feature off you can access subtechniques on the same level as all other techniques. Here's an example:

```
from pyattck import Attck

attack = Attck()
```

(continues on next page)

(continued from previous page)

```
for technique in attack.enterprise.techniques:
    print(technique.id)
    print(technique.name)
    print(f"checking if technique is subtechnique: {technique.subtechnique}")
```

Attck Class

class pyattck.attck.**Attck**(*nested_subtechniques=True*, *use_config=False*, *save_config=False*, *config_file_path='~/pyattck/config.yml'*, *data_path='~/pyattck/data'*, *enterprise_attck_json='https://raw.githubusercontent.com/mitre/cti/master/enterprise-attack/enterprise-attack.json'*, *pre_attck_json='https://raw.githubusercontent.com/mitre/cti/mast attack/pre-attack.json'*, *mobile_attck_json='https://raw.githubusercontent.com/mitre/cti/master/ attack/mobile-attack.json'*, *ics_attck_json='https://raw.githubusercontent.com/mitre/cti/master/ic attack/ics-attack.json'*, *nist_controls_json='https://raw.githubusercontent.com/center-for-threat-informed-defense/attack-control-framework-mappings/master/frameworks/ATT%26CK-v9.0/nist800-53-r5/stix/nist800-53-r5-controls.json'*, *generated_attck_json='https://swimlane-pyattck.s3.us-west-2.amazonaws.com/generated_attck_data.json'*, *generated_nist_json='https://swimlane-pyattck.s3.us-west-2.amazonaws.com/attck_to_nist_controls.json'*, ***kwargs*)

Bases: object

Interface to all MITRE ATT&CK frameworks.

Currently, this class enables access to the Enterprise & PRE-ATT&CK frameworks with others coming soon. To acccess each framework, use the following properties

- enterprise
- preattack

This interface enables you to retrieve all properties within each item in the MITRE ATT&CK Enterprise Framework.

The following categorical items can be accessed using this class:

1. Tactics (Tactics are the phases defined by MITRE ATT&CK)
2. **Techniques (Techniques are the individual actions which can** accomplish a tactic)
3. **Mitigations (Mitigations are recommendations to prevent or** protect against a technique)
4. **Actors (Actors or Groups are identified malicious** actors/groups which have been identified and documented by MITRE & third-parties)
5. Tools (Tools are software used to perform techniques)
6. **Malwares (Malwares are specific pieces of malware used by** actors (or in general) to accomplish a technique)

You can also search the external dataset for external commands that are similar using the *search_commands* method.

```
from pyattck import Attck

attck = Attck()
```

(continues on next page)

(continued from previous page)

```
for search in attck.enterprise.search_commands('powershell'):
    print(search['technique'])
    print(search['reason_for_match'])
```

You can access additional datasets related to a technique. These datasets are [documented here](https://github.com/swimlane/pyattck-data).

Example: Once an Attck object is instantiated, you can access each object type as a list of objects (e.g. techniques, tactics, actors, etc.)

You can iterate over each object list and access specific properties and relationship properties of each.

The following relationship properties are accessible:

1. **Actors**
    1. Tools used by the Actor or Group
    2. Malware used by the Actor or Group
    3. Techniques this Actor or Group uses
2. **Malwares**
    1. Actor or Group(s) using this malware
    2. Techniques this malware is used with
3. **Mitigations**
    1. Techniques related to a specific set of mitigation suggestions
4. **Tactics**
    1. Techniques found in a specific Tactic (phase)
5. **Techniques**
    1. Tactics a technique is found in
    2. Mitigation suggestions for a given technique
    3. Actor or Group(s) identified as using this technique
6. **Tools**
    1. Techniques that the specified tool is used within
    2. Actor or Group(s) using a specified tool

1. To iterate over a list, do the following:

```
from pyattck import Attck

attck = Attck()

for technique in attck.enterprise.techniques:
    print(technique.id)
    print(technique.name)
    print(technique.description)
    # etc.
for mitigation in attck.enterprise.mitigations:
```

(continues on next page)

(continued from previous page)

```
    print(mitigation.id)
    print(mitigation.name)
    print(mitigation.description)
    # etc.
```

2. To access relationship properties, do the following:

```
from pyattck import Attck

attck = Attck()

for technique in attck.enterprise.techniques:
    print(technique.id)
    print(technique.name)
    print(technique.description)
    # etc.

    for actor in technique.enterprise.actors:
        print(actor.id)
        print(actor.name)
        print(actor.description)
        # etc.

for mitigation in attck.enterprise.mitigations:
    print(mitigation.id)
    print(mitigation.name)
    print(mitigation.description)
    # etc.

    for technique in mitigation.enterprise.techniques:
        print(technique.name)
        print(technique.description)
        # etc.
```

Arguments: nested_subtechniques (bool, optional): Whether not to iterate over nested subtechniques. Defaults to True. use_config (bool, optional): Specifies if a configuration file should be used or not. Defaults to False. save_config (bool, optional): Specifies if pyattck should save a configuration file based on the provided

values. Defaults to False.

config_file_path (str, optional): Path to a yaml configuration file which contains two key value pairs. Defaults to '~/pyattck/config.yml'.

data_path (str, optional): Path to store the external data locally on your system. Defaults to '~/pyattck/data'. enterprise_attck_json (str, optional): A URL or local file path to the MITRE ATT&CK Json file.

Defaults to https://raw.githubusercontent.com/mitre/cti/master/enterprise-attack/enterprise-attack.json.

pre_attck_json (str, optional): A URL or local file path to the MITRE Pre-ATT&CK Json file. Defaults to https://raw.githubusercontent.com/mitre/cti/master/pre-attack/pre-attack.json.

mobile_attck_json (str, optional): A URL or local file path to the MITRE Mobile ATT&CK Json file. Defaults to https://raw.githubusercontent.com/mitre/cti/master/mobile-attack/mobile-attack.json.

ics_attck_json (str, optional): A URL or local file path to the MITRE ICS ATT&CK JSON file.
Defaults to https://raw.githubusercontent.com/mitre/cti/master/ics-attack/ics-attack.json.

nist_controls_json (str, optional): A URL or local file path to the NIST Controls Json file.
Defaults to https://raw.githubusercontent.com/center-for-threat-informed-defense/attack-control-framework-mappings/master/frameworks/ATT%26CK-v9.0/nist800-53-r5/stix/nist800-53-r5-controls.json.

generated_attck_json (str, optional): A URL or local file path to the Generated MITRE ATT&CK Json file.
Defaults to https://swimlane-pyattck.s3.us-west-2.amazonaws.com/generated_attck_data.json.

generated_nist_json (str, optional): A URL or local file path to the Generated NIST Controls Mapping Json file.
Defaults to https://swimlane-pyattck.s3.us-west-2.amazonaws.com/attck_to_nist_controls.json.

kwargs (dict, optional): Provided kwargs will be passed to any HTTP requests using the Requests library.
Defaults to None.

Returns: [Attck]: Returns a Attck object that contains all data from MITRE ATT&CK Frameworks

enterprise
Retrieve objects from the Enterprise MITRE ATT&CK Framework and additional generated data which provides additional context

Returns: Enterprise: Returns an Enterprise object

ics
Retrieve objects from the MITRE ICS ATT&CK Framework

Returns: PreAttack: Returns an ICSAttck object

mobile
Retrieve objects from the MITRE Mobile ATT&CK Framework

Returns: PreAttack: Returns an MobileAttack object

preattack
Retrieve objects from the MITRE PRE-ATT&CK Framework

Returns: PreAttack: Returns an PreAttack object

update() → bool

3.11.3 Enterprise

This documentation provides details about the Enterprise class within the `pyattck` package.

The Enterprise class provides detailed information about data within the Enterprise MITRE ATT&CK framework

You can also search the external dataset for external commands that are similar using the `search_commands` method.

```
from pyattck import Attck

attack = Attck()

for search in attack.enterprise.search_commands('powershell'):
    print(search['technique'])
    print(search['reason_for_match'])
```

Each of the `main` properties (above) can return a json object of the entire object or you can access each property individually. An example of this is here:

```
from pyattck import Attck

attack = Attck()

# accessing techniques and their properties
for technique in attack.enterprise.techniques:
        # if you want to return individual properties of this object you call them␣
↪directly
        print(technique.id)
        print(technique.name)
        print(technique.alias)
        print(technique.description)
        print(technique.stix)
        print(technique.platforms)
        print(technique.permissions)
        print(technique.wiki)
        .....
```

The following is only a small sample of the available properties on each object and each object type (actors, malware, mitigations, tactics, techniques, and tools) will have different properties that you can access.

- Every data point has exposed properties that allow the user to retrieve additional data based on relationships:
  - Actor
    - Relationship Objects
      - Tools used by the Actor or Group
      - Malware used by the Actor or Group
      - Techniques this Actor or Group uses
    - External Data
      - Retrieve a logo for an actor using either image_logo or ascii_logo properties
      - country which this actor or group may be associated with (attribution is hard)
      - operations
      - attribution_links
      - known_tools
      - targets
      - additional_comments
      - external_description
  - Malware
    - Actor or Group(s) using this malware
    - Techniques this malware is used with
  - Mitigation
    - Techniques related to a specific set of mitigation suggestions
  - Tactic
    - Techniques found in a specific Tactic (phase)
  - Technique

        * Relationship Objects
            - Tactics a technique is found in
            - Mitigation suggestions for a given technique
            - Actor or Group(s) identified as using this technique
        * External Data
            - command_list - A list of commands from multiple open-source tools and repositories that contain potential commands used by a technique
            - commands - A list of property objects that contain the `Name`, `Source`, `and`Command‘ dataset
            - queries - A list of potential queries for different products to identify threats within your environment by technique
            - datasets - A list of the datasets as it relates to a technique
            - possible_detections - A list of potential detections for different products (e.g. NSM rules) as it relates to a technique
            - For more detailed information about these features, please view the following External Datasets
    - Tools
        * Relationship Objects
            - Techniques that the specified tool is used within
            - Actor or Group(s) using a specified tool
        * External Data
            - additional_names for the specified tool
            - attribution_links associated with the specified tool
            - additional_comments about the specified tool
            - family of the specified tool

Below shows you how you can access each of object types and their properties. Additionally, you can access related object types associated with this selected object type:

```
from pyattck import Attck

attack = Attck()

for actor in attack.enterprise.actors:
    print(actor.id)
    print(actor.name)

    # accessing malware used by an actor or group
    for malware in actor.malwares:
        print(malware.id)
        print(malware.name)

    # accessing tools used by an actor or group
    for tool in actor.tools:
        print(tool.id)
        print(tool.name)

    # accessing techniques used by an actor or group
```

(continues on next page)

(continued from previous page)

```
    for technique in actor.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

# accessing malware
for malware in attack.enterprise.malwares:
    print(malware.id)
    print(malware.name)

    # accessing actor or groups using this malware
    for actor in malware.actors:
        print(actor.id)
        print(actor.name)

    # accessing techniques that this malware is used in
    for technique in malware.techniques:
        print(technique.id)
        print(technique.name)

# accessing mitigation
for mitigation in attack.enterprise.mitigations:
    print(mitigation.id)
    print(mitigation.name)

    # accessing techniques related to mitigation recommendations
    for technique in mitigation.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

# accessing tactics
for tactic in attack.enterprise.tactics:
    print(tactic.id)
    print(tactic.name)

    # accessing techniques related to this tactic
    for technique in tactic.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)
```

(continues on next page)

(continued from previous page)

```
# accessing techniques
for technique in attack.enterprise.techniques:
    print(technique.id)
    print(technique.name)
    # you can also access generated data sets on aa technique
    print(technique.command_list)
    print(technique.commands)
    print(technique.queries)
    print(technique.datasets)
    print(technique.possible_detections)

    # accessing tactics that this technique belongs to
    for tactic in technique.tactics:
        print(tactic.id)
        print(tactic.name)

    # accessing mitigation recommendations for this technique
    for mitigation in technique.mitigations:
        print(mitigation.id)
        print(mitigation.name)

    # accessing actors using this technique
    for actor in technique.actors:
        print(actor.id)
        print(actor.name)

# accessing tools
for tool in attack.enterprise.tools:
    print(tool.id)
    print(tool.name)

    # accessing techniques this tool is used in
    for technique in tool.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

    # accessing actor or groups using this tool
    for actor in tool.actors:
        print(actor.id)
        print(actor.name)
```

Enterprise Class

Actor

This documentation provides details about Actor class within the `pyattck` package.

The `AttckActor` class provides detailed information about identified actors & groups within the MITRE ATT&CK Framework. Additionally, an `AttckActor` object allows the user to access additional relationships within the

MITRE ATT&CK Framework:

- Tools used by the Actor or Group
- Malware used by the Actor or Group
- Techniques this Actor or Group uses

You can also access external data properties. The following properties are generated using external data:

- country
- operations
- attribution_links
- known_tools
- targets
- additional_comments
- external_description

You can retrieve the entire dataset using the `external_dataset` property.

An additional fun feature is that you can now retrieve a logo for a actor. Currently, a limited set of logos are provided. If a logo is not provided, then `pyattck` will generate one using ascii art.

You can access these logos using the following properties:

- ascii_logo
- image_logo

AttckActor Class

Control

This documentation provides details about Control class within the `pyattck` package.

The `AttckControl` class provides detailed information about compliance controls related to a techniques within the MITRE ATT&CK Framework.

> Currently the `AttckControl` class only supports data for NIST 800-53

AttckActor Class

Malware

This documentation provides details about the `AttckMalware` class within the `pyattck` package.

This class provides detailed information about identified malware used by actors or impact specific techniques within the MITRE Enterprise ATT&CK Framework. Additionally, a `AttckMalware` object allows the user to access additional relationships within the MITRE ATT&CK Framework:

- Actor or Group(s) using this malware
- Techniques this malware is used with

AttckMalware Class

Mitigation

This documentation provides details about the `AttckMitigation` class within the `pyattck` package.

This class provides detailed information provided by MITRE to help mitigate specific techniques within the MITRE Enterprise ATT&CK Framework. Additionally, a `AttckMitigation` object allows the user to access additional relationships within the MITRE Enterprise ATT&CK Framework:

- Techniques related to a specific set of mitigation suggestions

AttckMitigation Class

Tactic

This documentation provides details about the `AttckTactic` class within the `pyattck` package.

This class provides information about the tactics (columns) within the MITRE Enterprise ATT&CK Framework. Additionally, a `AttckTactic` object allows the user to access additional relationships within the MITRE Enterprise ATT&CK Framework:

- Techniques found in a specific Tactic (phase)

AttckTactic Class

Technique

This documentation provides details about the `AttckTechnique` class within the `pyattck` package.

This class provides information about the techniques found under each tactic (columns) within the MITRE Enterprise ATT&CK Framework. Additionally, a `AttckTechnique` object allows the user to access additional relationships within the MITRE Enterprise ATT&CK Framework:

- Tactics a technique is found in
- Mitigation suggestions for a given technique
- Actor or Group(s) identified as using this technique

Each technique enables you to access the following properties on the object:

- command_list - A list of commands associated with a technique
- commands = A list of dictionary objects containing source, command, and provided name associated with a technique
- queries = A list of dictionary objects containing product, query, and name associated with a technique
- datasets = A list of raw datasets associated with a technique
- possible_detections = A list of raw datasets containing possible detection methods for a technique
- data_sources = A list of raw datasets containing data sources listed for the technique

AttckTechnique Class

Tools

This documentation provides details about the `AttckTools` class within the `pyattck` package.

You can also access external data properties. The following properties are generated using external data:

1. additional_names
2. attribution_links
3. additional_comments
4. family

You can retrieve the entire dataset using the `external_dataset` property.

You can also access external data properties from the C2 Matrix project. The following properties are generated using C2 Matrix external data:

- HTTP
- Implementation
- Custom Profile
- DomainFront
- Multi-User
- SMB
- Kill Date
- macOS
- GitHub
- Key Exchange
- Chaining
- Price
- TCP
- Proxy Aware
- HTTP3
- HTTP2
- Date
- Evaluator
- Working Hours
- Slack
- FTP
- Version Reviewed
- Logging
- Name

- License
- Windows
- Stego
- Notes
- Server
- Actively Maint.
- Dashboard
- DNS
- Popular Site
- ICMP
- IMAP
- DoH
- Jitter
- How-To
- ATT&CK Mapping
- Kali
- Twitter
- MAPI
- Site
- Agent
- API
- UI
- Linux

You can retrieve the entire dataset using the `c2_data` property.

This class provides information about the tools used by actors or groups within the MITRE Enterprise ATT&CK Framework. Additionally, a `AttckTools` object allows the user to access additional relationships within the MITRE Enterprise ATT&CK Framework:

- Techniques that the specified tool is used within
- Actor or Group(s) using a specified tool

AttckTools Class

3.11.4 PreAttck

This documentation provides details about the PreAttck class within the `pyattck` package.

The PreAttck class provides detailed information about data within the MITRE PRE-ATT&CK framework

Each of the `main` properties (above) can return a json object of the entire object or you can access each property individually. An example of this is here:

```
from pyattck import Attck

attack = Attck()

# accessing techniques and their properties
for technique in attack.preattack.techniques:
        # if you want to return individual properties of this object you call them␣
↪directly
        print(technique.id)
        print(technique.name)
        print(technique.alias)
        print(technique.description)
        print(technique.stix)
        print(technique.platforms)
        print(technique.permissions)
        print(technique.wiki)
        .....
```

The following is only a small sample of the available properties on each object and each object type (actors, tactics, and techniques) will have different properties that you can access.

- Every data point has exposed properties that allow the user to retrieve additional data based on relationships:
  - Actor
    - Relationship Objects
      - Techniques this Actor or Group uses
    - External Data
      - Retrieve a logo for an actor using either image_logo or ascii_logo properties
      - country which this actor or group may be associated with (attribution is hard)
      - operations
      - attribution_links
      - known_tools
      - targets
      - additional_comments
      - external_description
  - Tactic
    - Techniques found in a specific Tactic (phase)
  - Technique
    - Relationship Objects
      - Tactics a technique is found in
      - Actor or Group(s) identified as using this technique

Below shows you how you can access each of object types and their properties. Additionally, you can access related object types associated with this selected object type:

```
from pyattck import Attck

attack = Attck()

for actor in attack.preattack.actors:
    print(actor.id)
    print(actor.name)

    # accessing techniques used by an actor or group
    for technique in actor.techniques:
        print(technique.id)
        print(technique.name)

# accessing tactics
for tactic in attack.preattack.tactics:
    print(tactic.id)
    print(tactic.name)

    # accessing techniques related to this tactic
    for technique in tactic.techniques:
        print(technique.id)
        print(technique.name)

# accessing techniques
for technique in attack.preattack.techniques:
    print(technique.id)
    print(technique.name)

    # accessing tactics that this technique belongs to
    for tactic in technique.tactics:
        print(tactic.id)
        print(tactic.name)

    # accessing actors using this technique
    for actor in technique.actors:
        print(actor.id)
        print(actor.name)
```

PreAttck Class

PreAttckActor

This documentation provides details about PreAttckActor class within the `pyattck` package.

The `PreAttckActor` class provides detailed information about identified actors & groups within the MITRE PRE-ATT&CK Framework. Additionally, an `PreAttckActor` object allows the user to access additional relationships within the MITRE PRE-ATT&CK Framework:

- Techniques this Actor or Group uses

You can also access external data properties. The following properties are generated using external data:

- country
- operations
- attribution_links

- known_tools
- targets
- additional_comments
- external_description

You can retrieve the entire dataset using the `external_dataset` property.

An additional fun feature is that you can now retrieve a logo for a actor. Currently, a limited set of logos are provided. If a logo is not provided, then `pyattck` will generate one using ascii art.

You can access these logos using the following properties:

- ascii_logo

PreAttckActor Class

PreAttckTactic

This documentation provides details about the `PreAttckTactic` class within the `pyattck` package.

This class provides information about the tactics (columns) within the MITRE PRE-ATT&CK Framework. Additionally, a `PreAttckTactic` object allows the user to access additional relationships within the MITRE PRE-ATT&CK Framework:

- Techniques found in a specific Tactic (phase)

PreAttckTactic Class

PreAttckTechnique

This documentation provides details about the `PreAttckTechnique` class within the `pyattck` package.

This class provides information about the techniques found under each tactic (columns) within the MITRE PRE-ATT&CK Framework. Additionally, a `PreAttckTechnique` object allows the user to access additional relationships within the MITRE PRE-ATT&CK Framework:

- Tactics a technique is found in
- Actor or Group(s) identified as using this technique

PreAttckTechnique Class

3.11.5 MobileAttck

This documentation provides details about the MobileAttck class within the `pyattck` package.

The MobileAttck class provides detailed information about data within the MITRE Mobile ATT&CK framework

Each of the `main` properties can return a json object of the entire object or you can access each property individually. An example of this is here:

```
from pyattck import Attck

attack = Attck()

# accessing techniques and their properties
for technique in attack.mobile.techniques:
        # if you want to return individual properties of this object you call them␣
↪directly
        print(technique.id)
        print(technique.name)
        print(technique.alias)
        print(technique.description)
        print(technique.stix)
        print(technique.platforms)
        .....
```

The following is only a small sample of the available properties on each object and each object type (actors, malware, mitigations, tactics, techniques, and tools) will have different properties that you can access.

- Every data point has exposed properties that allow the user to retrieve additional data based on relationships:
  - Actor
    - Relationship Objects
      - Tools used by the Actor or Group
      - Malware used by the Actor or Group
      - Techniques this Actor or Group uses
    - External Data
      - Retrieve a logo for an actor using ascii_logo properties
      - country which this actor or group may be associated with (attribution is hard)
      - operations
      - attribution_links
      - known_tools
      - targets
      - additional_comments
      - external_description
  - Malware
    - Actor or Group(s) using this malware
    - Techniques this malware is used with
  - Mitigation
    - Techniques related to a specific set of mitigation suggestions
  - Tactic
    - Techniques found in a specific Tactic (phase)
  - Technique
    - Relationship Objects

- Tactics a technique is found in
- Mitigation suggestions for a given technique
- Actor or Group(s) identified as using this technique

* External Data
  - command_list - A list of commands from multiple open-source tools and repositories that contain potential commands used by a technique
  - commands - A list of property objects that contain the `Name`, `Source`, and`Command‘ dataset
  - queries - A list of potential queries for different products to identify threats within your environment by technique
  - datasets - A list of the datasets as it relates to a technique
  - possible_detections - A list of potential detections for different products (e.g. NSM rules) as it relates to a technique
  - For more detailed information about these features, please view the following External Datasets

– Tools

* Relationship Objects
  - Techniques that the specified tool is used within
  - Actor or Group(s) using a specified tool
* External Data
  - additional_names for the specified tool
  - attribution_links associated with the specified tool
  - additional_comments about the specified tool
  - family of the specified tool

Below shows you how you can access each of object types and their properties. Additionally, you can access related object types associated with this selected object type:

```
from pyattck import Attck

attack = Attck()

for actor in attack.mobile.actors:
    print(actor.id)
    print(actor.name)

    # accessing malware used by an actor or group
    for malware in actor.malwares:
        print(malware.id)
        print(malware.name)

    # accessing tools used by an actor or group
    for tool in actor.tools:
        print(tool.id)
        print(tool.name)

    # accessing techniques used by an actor or group
    for technique in actor.techniques:
        print(technique.id)
```

(continues on next page)

(continued from previous page)

```
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

# accessing malware
for malware in attack.mobile.malwares:
    print(malware.id)
    print(malware.name)

    # accessing actor or groups using this malware
    for actor in malware.actors:
        print(actor.id)
        print(actor.name)

    # accessing techniques that this malware is used in
    for technique in malware.techniques:
        print(technique.id)
        print(technique.name)

# accessing mitigation
for mitigation in attack.mobile.mitigations:
    print(mitigation.id)
    print(mitigation.name)

    # accessing techniques related to mitigation recommendations
    for technique in mitigation.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

# accessing tactics
for tactic in attack.mobile.tactics:
    print(tactic.id)
    print(tactic.name)

    # accessing techniques related to this tactic
    for technique in tactic.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

# accessing techniques
for technique in attack.mobile.techniques:
```

(continues on next page)

(continued from previous page)

```
    print(technique.id)
    print(technique.name)
    # you can also access generated data sets on aa technique
    print(technique.command_list)
    print(technique.commands)
    print(technique.queries)
    print(technique.datasets)
    print(technique.possible_detections)

    # accessing tactics that this technique belongs to
    for tactic in technique.tactics:
        print(tactic.id)
        print(tactic.name)

    # accessing mitigation recommendations for this technique
    for mitigation in technique.mitigations:
        print(mitigation.id)
        print(mitigation.name)

    # accessing actors using this technique
    for actor in technique.actors:
        print(actor.id)
        print(actor.name)

# accessing tools
for tool in attack.mobile.tools:
    print(tool.id)
    print(tool.name)

    # accessing techniques this tool is used in
    for technique in tool.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

    # accessing actor or groups using this tool
    for actor in tool.actors:
        print(actor.id)
        print(actor.name)
```

MobileAttck Class

MobileAttckActor

This documentation provides details about MobileAttckActor class within the `pyattck` package.

The `MobileAttckActor` class provides detailed information about identified actors & groups within the MITRE Mobile ATT&CK Framework. Additionally, an `MobileAttckActor` object allows the user to access additional relationships within the MITRE Mobile ATT&CK Framework:

- Tools used by the Actor or Group

- Malware used by the Actor or Group
- Techniques this Actor or Group uses

You can also access external data properties. The following properties are generated using external data:

- country
- operations
- attribution_links
- known_tools
- targets
- additional_comments
- external_description

You can retrieve the entire dataset using the `external_dataset` property.

An additional fun feature is that you can now retrieve a logo for a actor. Currently, a limited set of logos are provided. If a logo is not provided, then `pyattck` will generate one using ascii art.

You can access these logos using the following properties:

- ascii_logo

MobileAttckActor Class

MobileAttckMalware

This documentation provides details about the `MobileAttckMalware` class within the `pyattck` package.

This class provides detailed information about identified malware used by actors or impact specific techniques within the MITRE Mobile ATT&CK Framework. Additionally, a `MobileAttckMalware` object allows the user to access additional relationships within the MITRE Mobile ATT&CK Framework:

- Actor or Group(s) using this malware
- Techniques this malware is used with

MobileAttckMalware Class

MobileAttckMitigation

This documentation provides details about the `MobileAttckMitigation` class within the `pyattck` package.

This class provides detailed information provided by MITRE to help mitigate specific techniques within the MITRE Mobile ATT&CK Framework. Additionally, a `MobileAttckMitigation` object allows the user to access additional relationships within the MITRE Mobile ATT&CK Framework:

- Techniques related to a specific set of mitigation suggestions

MobileAttckMitigation Class

MobileAttckTactic

This documentation provides details about the `MobileAttckTactic` class within the `pyattck` package.

This class provides information about the tactics (columns) within the MITRE Mobile ATT&CK Framework. Additionally, a `MobileAttckTactic` object allows the user to access additional relationships within the MITRE Mobile ATT&CK Framework:

- Techniques found in a specific Tactic (phase)

MobileAttckTactic Class

MobileAttckTechnique

This documentation provides details about the `MobileAttckTechnique` class within the `pyattck` package.

This class provides information about the techniques found under each tactic (columns) within the MITRE Mobile ATT&CK Framework. Additionally, a `MobileAttckTechnique` object allows the user to access additional relationships within the MITRE Mobile ATT&CK Framework:

- Tactics a technique is found in
- Mitigation suggestions for a given technique
- Actor or Group(s) identified as using this technique

Each technique enables you to access the following properties on the object:

- command_list - A list of commands associated with a technique
- commands = A list of dictionary objects containing source, command, and provided name associated with a technique
- queries = A list of dictionary objects containing product, query, and name associated with a technique
- datasets = A list of raw datasets associated with a technique
- possible_detections = A list of raw datasets containing possible detection methods for a technique

MobileAttckTechnique Class

MobileAttckTools

This documentation provides details about the `MobileAttckTools` class within the `pyattck` package.

You can also access external data properties. The following properties are generated using external data:

1. additional_names
2. attribution_links
3. additional_comments
4. family

You can retrieve the entire dataset using the `external_dataset` property.

You can also access external data properties from the C2 Matrix project. The following properties are generated using C2 Matrix external data:

- HTTP
- Implementation
- Custom Profile
- DomainFront
- Multi-User
- SMB
- Kill Date
- macOS
- GitHub
- Key Exchange
- Chaining
- Price
- TCP
- Proxy Aware
- HTTP3
- HTTP2
- Date
- Evaluator
- Working Hours
- Slack
- FTP
- Version Reviewed
- Logging
- Name
- License
- Windows
- Stego
- Notes
- Server
- Actively Maint.
- Dashboard
- DNS
- Popular Site

- ICMP
- IMAP
- DoH
- Jitter
- How-To
- ATT&CK Mapping
- Kali
- Twitter
- MAPI
- Site
- Agent
- API
- UI
- Linux

You can retrieve the entire dataset using the `c2_data` property.

This class provides information about the tools used by actors or groups within the MITRE Mobile ATT&CK Framework. Additionally, a `MobileAttckTools` object allows the user to access additional relationships within the MITRE Mobile ATT&CK Framework:

- Techniques that the specified tool is used within
- Actor or Group(s) using a specified tool

MobileAttckTools Class

3.11.6 ICS (Industrial Control Systems)

This documentation provides details about the ICSAttck class within the `pyattck` package.

The ICSAttck class provides detailed information about data within the ICS MITRE ATT&CK framework

Each of the `main` properties (above) can return a json object of the entire object or you can access each property individually. An example of this is here:

```
from pyattck import Attck

attack = Attck()

# accessing techniques and their properties
for technique in attack.ics.techniques:
        # if you want to return individual properties of this object you call them␣
↪directly
        print(technique.id)
        print(technique.name)
        print(technique.alias)
        print(technique.description)
        print(technique.stix)
        print(technique.platforms)
```

(continues on next page)

(continued from previous page)

```
        print(technique.permissions)
        print(technique.wiki)
        .....
```

The following is only a small sample of the available properties on each object and each object type (malware, mitigations, tactics, and techniques) will have different properties that you can access.

- Every data point has exposed properties that allow the user to retrieve additional data based on relationships:
  - Malware
    - Techniques this malware is used with
  - Mitigation
    - Techniques related to a specific set of mitigation suggestions
  - Tactic
    - Techniques found in a specific Tactic (phase)
  - Technique
    - Relationship Objects
      - Tactics a technique is found in
      - Mitigation suggestions for a given technique
    - External Data
      - command_list - A list of commands from multiple open-source tools and repositories that contain potential commands used by a technique
      - commands - A list of property objects that contain the `Name`, `Source`, and`Command` dataset
      - queries - A list of potential queries for different products to identify threats within your environment by technique
      - datasets - A list of the datasets as it relates to a technique
      - possible_detections - A list of potential detections for different products (e.g. NSM rules) as it relates to a technique
      - For more detailed information about these features, please view the following External Datasets

Below shows you how you can access each of object types and their properties. Additionally, you can access related object types associated with this selected object type:

```
from pyattck import Attck

attack = Attck()

# accessing malware
for malware in attack.ics.malwares:
    print(malware.id)
    print(malware.name)

    # accessing techniques that this malware is used in
    for technique in malware.techniques:
        print(technique.id)
        print(technique.name)
```

(continues on next page)

(continued from previous page)

```
# accessing mitigation
for mitigation in attack.ics.mitigations:
    print(mitigation.id)
    print(mitigation.name)

    # accessing techniques related to mitigation recommendations
    for technique in mitigation.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

# accessing tactics
for tactic in attack.ics.tactics:
    print(tactic.id)
    print(tactic.name)

    # accessing techniques related to this tactic
    for technique in tactic.techniques:
        print(technique.id)
        print(technique.name)
        # you can also access generated data sets on aa technique
        print(technique.command_list)
        print(technique.commands)
        print(technique.queries)
        print(technique.datasets)
        print(technique.possible_detections)

# accessing techniques
for technique in attack.ics.techniques:
    print(technique.id)
    print(technique.name)
    # you can also access generated data sets on aa technique
    print(technique.command_list)
    print(technique.commands)
    print(technique.queries)
    print(technique.datasets)
    print(technique.possible_detections)

    # accessing tactics that this technique belongs to
    for tactic in technique.tactics:
        print(tactic.id)
        print(tactic.name)

    # accessing mitigation recommendations for this technique
    for mitigation in technique.mitigations:
        print(mitigation.id)
        print(mitigation.name)
```

ICSAttck Class

Control

This documentation provides details about Control class within the `pyattck` package.

The `AttckControl` class provides detailed information about compliance controls related to a techniques within the MITRE ATT&CK Framework.

> Currently the `AttckControl` class only supports data for NIST 800-53

AttckActor Class

Malware

This documentation provides details about the `AttckMalware` class within the `pyattck` package.

This class provides detailed information about identified malware used by actors or impact specific techniques within the MITRE ICS ATT&CK Framework. Additionally, a `AttckMalware` object allows the user to access additional relationships within the MITRE ATT&CK Framework:

- Techniques this malware is used with

AttckMalware Class

Mitigation

This documentation provides details about the `AttckMitigation` class within the `pyattck` package.

This class provides detailed information provided by MITRE to help mitigate specific techniques within the MITRE ICS ATT&CK Framework. Additionally, a `AttckMitigation` object allows the user to access additional relationships within the MITRE ICS ATT&CK Framework:

- Techniques related to a specific set of mitigation suggestions

AttckMitigation Class

Tactic

This documentation provides details about the `AttckTactic` class within the `pyattck` package.

This class provides information about the tactics (columns) within the MITRE ICS ATT&CK Framework. Additionally, a `AttckTactic` object allows the user to access additional relationships within the MITRE ICS ATT&CK Framework:

- Techniques found in a specific Tactic (phase)

AttckTactic Class

Technique

This documentation provides details about the `AttckTechnique` class within the `pyattck` package.

This class provides information about the techniques found under each tactic (columns) within the MITRE ICS ATT&CK Framework. Additionally, a `AttckTechnique` object allows the user to access additional relationships within the MITRE ICS ATT&CK Framework:

- Tactics a technique is found in
- Mitigation suggestions for a given technique

Each technique enables you to access the following properties on the object:

- command_list - A list of commands associated with a technique
- commands = A list of dictionary objects containing source, command, and provided name associated with a technique
- queries = A list of dictionary objects containing product, query, and name associated with a technique
- datasets = A list of raw datasets associated with a technique
- possible_detections = A list of raw datasets containing possible detection methods for a technique
- data_sources = A list of raw datasets containing data sources listed for the technique

AttckTechnique Class

Index